

1990 Annual Meeting Proceedings

TeX Users Group

Eleventh Annual Meeting

Texas A&M University, June 18-20, 1990

TUGBOAT

COMMUNICATIONS OF THE T_EX USERS GROUP

TUGBOAT EDITOR BARBARA BEETON

PROCEEDINGS EDITOR LINCOLN DURST

VOLUME 11, NUMBER 3

PROVIDENCE

•

RHODE ISLAND

•

SEPTEMBER 1990

•

U.S.A.

Production Notes

The $\text{T}_{\text{E}}\text{X}$ source code for each article in this issue of *TUGboat* was transmitted via network or floppy diskette to the editors, who then used PC $\text{T}_{\text{E}}\text{X}$ running on a Zenith H-386 and an IBM PC-compatible 386 to generate dvi files. These files were then shipped to the American Mathematical Society via modem, and final copy was produced on the Society's APS μ -5.

A few items were produced locally by authors and then mailed to the editors for incorporation in articles. Personnel at the Math Society cut these figures and pictures into place. The output devices used for these items were:

- **HP LaserJet:**
 - Vulis, *V $\text{T}_{\text{E}}\text{X}$ enhancements to the $\text{T}_{\text{E}}\text{X}$ language*, appendix only.
- **Apple LaserWriter II:**
 - Beck and Siegel, *TransFig: Portable graphics for $\text{T}_{\text{E}}\text{X}$* , figures only.
- **Linotype L200P PostScript (1270dpi):**
 - Adams, *Problems on the $\text{T}_{\text{E}}\text{X}$ /PostScript/graphics interface*, pictures only.
- **Shaken SAPLS-Laura laser phototypesetter (1626dpi):**
 - Hamano, *Vertical typesetting with $\text{T}_{\text{E}}\text{X}$* , page from *Descartes' Dream* only.

Trademarks

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

APS μ 5 is a trademark of Autologic, Inc.

DOS and MS/DOS are trademarks of MicroSoft Corporation

LaserJet, PCL, and DeskJet are trademarks of Hewlett-Packard, Inc.

METAFONT is a trademark of Addison-Wesley Inc.

PC $\text{T}_{\text{E}}\text{X}$ is a registered trademark of Personal $\text{T}_{\text{E}}\text{X}$, Inc.

PostScript is a trademark of Adobe Systems, Inc.

$\text{T}_{\text{E}}\text{X}$ and $\mathcal{A}\mathcal{M}\mathcal{S}\text{-}\text{T}_{\text{E}}\text{X}$ are trademarks of the American Mathematical Society.

UNIX is a trademark of AT&T Bell Laboratories.

Other Conference Proceedings

Europe

Proceedings of the First European Conference on $\text{T}_{\text{E}}\text{X}$ for Scientific Documentation. Dario Lucarella, ed. Reading, Mass.: Addison-Wesley, 1985. [16-17 May 1985, Como, Italy.]

Proceedings of the Second European Conference on $\text{T}_{\text{E}}\text{X}$ for Scientific Documentation. Jacques Désarménien, ed. Berlin: Springer-Verlag, 1986. [19-21 June 1986, Strasbourg, France.]

$\text{T}_{\text{E}}\text{X}88$ Conference Proceedings. Malcolm Clark, ed. Chichester, England: Ellis Horwood, 1990. [18-20 July 1988, Exeter University, Exeter, England.]

North America

Conference Proceedings: $\text{T}_{\text{E}}\text{X}$ Users Group Eighth Annual Meeting. Dean Guenther, ed. *$\text{T}_{\text{E}}\text{X}$ niques* No. 5. Providence, Rhode Island: $\text{T}_{\text{E}}\text{X}$ Users Group, 1988. [24-26 August 1987, University of Washington, Seattle, Washington.]

Conference Proceedings: $\text{T}_{\text{E}}\text{X}$ Users Group Ninth Annual Meeting. Christina Thiele, ed. *$\text{T}_{\text{E}}\text{X}$ niques* No. 7. Providence, Rhode Island: $\text{T}_{\text{E}}\text{X}$ Users Group, 1988. [22-24 August 1988, McGill University, Montréal, Canada.]

Conference Proceedings: $\text{T}_{\text{E}}\text{X}$ Users Group Tenth Annual Meeting. Christina Thiele, ed. *TUGboat* 10, no. 4. Providence, Rhode Island: $\text{T}_{\text{E}}\text{X}$ Users Group, 1989. [20-23 August 1989, Stanford University, Stanford, California.]

President's Introduction

Nelson H. F. Beebe

The 1990 TUG meeting held at Texas A&M University in College Station, Texas, was our eleventh, and these *Proceedings* are the second to be published as a regular issue of *TUGboat*. Texas A&M is a big institution, with over 2200 faculty and 41000 students, 7000 of whom are graduate students. It is one of the few land, sea, and space grant universities, and also has the largest campus of any American university.

About 150 TUG members showed up, and almost all lived at the same hotel, so there were discussions going on from about 6:30 in the morning until well past midnight each day. I certainly enjoyed myself, and I would like to thank all of those who attended for contributing.

The first morning was devoted to status reports about the new T_EX 3.0 and METAFONT 2.0, the exciting work on the new L^AT_EX 2.10 implementation, reports from a number of international representatives, site coordinator reports, the first draft of a level 0 DVI driver standard, and archives and bulletin boards. In the afternoon, we were treated to Frank Mittelbach's insightful comments about the weaknesses of T_EX, with ideas for future extensions.

I would like readers who have stretched T_EX's limits to think seriously about writing up their experiences for *TUGboat* publication. While we *do* want to keep T_EX stable for some time, we risk killing it if we do not simultaneously plan for its peaceful evolution, because competing commercial desktop publishing systems are also evolving.

Frank's talk was followed by two presentations about the use of T_EX in Japan. The AutoLayouter system by Y. Miyabe and colleagues is a good example of an evolutionary direction that T_EX implementations might follow, combining workstation-based document preparation with a set of tools for structured document editing and management.

The second morning began with vendor presentations, and it was good to hear that several already have implemented T_EX 3.0. David Kellerman of Northlake Software showed some interesting slides illustrating the relative amounts of work that go into producing a solid commercial product—more than two-thirds is 'value-added', which clearly demonstrates that commercial, as well as public-domain, implementations of T_EX are needed if it is to survive. The remainder of the day was devoted to discussions of macro writing (see particularly Amy Hendrickson's valuable ideas, and Andrew Marc Greene's expression parser), portable graphics, and the real world of book production.

The third day began with talks about using other tools to support T_EX document preparation, the teaching of T_EX, and the annual T_EX help session hosted by Barbara Beeton. The problems and answers from this are finally going to make it into *TUGboat*, and I hope that this topic can be a regular column. The meeting wrapped up with the afternoon session on fonts. Michael Vulis has done some very interesting things with extensions to T_EX, which I believe we should study as a pilot implementation of some ideas for T_EX's evolution. The extensions can be suppressed by a command-line option to get a standard T_EX which passes the *trip* test. Alan Hoenig's METAFONT implementation of the Dürer alphabet will, I trust, encourage others to consider similar projects with classic fonts.

Of course, a TUG meeting would not be complete without numerous social activities. We are very grateful to ArborText, Blue Sky Research, Computer Composition Corporation, Kinch Computer Co., Micro Programs, Northlake Software,

Personal T_EX, T_EXnology Inc., the Bryan/College Station Convention Center & Visitors Bureau, and the College Station Hilton for their contributions to making the Eleventh Annual Meeting a big success.

Our sincere thanks go out to all those who worked so hard to deliver such an excellent meeting, in particular, the program committee, staff members of the Texas A&M Computer Science Department, and the TUG staff—all of whom dedicated a considerable amount of time and effort over the past 12 months.

And lastly, our sincere appreciation to Lincoln Durst, whose diligent efforts have resulted not only in the production of a *Proceedings* of the highest quality, but in its timely appearance—within 10 weeks of the meeting.

Hope to see you in September at T_EX90 in Cork!

◇ Nelson H.F. Beebe
Center for Scientific Computing and
Department of Mathematics
South Physics Building
University of Utah
Salt Lake City, UT 84112
USA
Tel: (801) 581-5254
Internet: Beebe@science.utah.edu

1990 Program Committee: The Program for the 1990 Annual Meeting of the T_EX Users Group was organised by the following people, whose efforts to make the 11th Annual Meeting a memorable start to the new decade should not go unnoticed: Lincoln Durst, our new editor; Regina Girouard of the American Mathematical Society; Tom Reid, the on-site contact person (Texas A&M University, College Station); and Christina Thiele, Program Coordinator (Carleton University, Ottawa, Canada).

E-TeX: Guidelines for Future TeX Extensions

Frank Mittelbach

Electronic Data Systems (Deutschland) GmbH

Eisenstraße 56 (N 15), D-6090 Rüsselsheim, Federal Republic of Germany

Tel. +49 6142 803267

Bitnet: pzf5hz@drueds2

Abstract

With the announcement of TeX 3.0, Don Knuth acknowledged the need of the (ever growing) TeX community for an even better system. But at the same time, he made it clear, that he will not get involved in any further enhancements that would change *The TeXbook*.

TeX started out originally as a system designed to typeset its author's own publications. In the meantime it serves hundreds of thousands of users. Now it is time, after ten years' experience, to step back and consider whether or not TeX 3.0 is an adequate answer to the typesetting requirements of the nineties.

Output produced by TeX has higher standards than output generated automatically by most other typesetting systems. Therefore, in this paper we will focus on the quality standards set by typographers for hand-typeset documents and ask to what extent they are achieved by TeX. Limitations of TeX's algorithms are analyzed; and missing features as well as new concepts are outlined.

1 Introduction

Last year at Stanford we celebrated the tenth birthday of the TeX project. Up to now, TeX has served thousands of users well and we expect it will continue to do so in the future. The longevity of TeX lies in

- the quality of its output
- its universal availability
- and its stability.

In the last few years, more and more users brought TeX from the universities into industry where it was challenged by new applications [33]. But time does not stand still, and what was at the top of its profession yesterday might prove to be obsolete tomorrow. TeX is still state of the art for the tasks it was designed to accomplish, but, with the growing understanding from several years' usage, we can now see where it will fail in high quality typesetting.

As a result of user pressure [27], Don Knuth announced a new version of TeX at Stanford, acknowledging the fact, that he did not foresee the need for 8-bit input [19]. At the same time, he made it clear, that he had decided to retire from this project and return to his long delayed topic "The Art of Computer Programming".

So TeX is finally frozen, and any further development will result in a different system no longer maintained by Knuth. The main purpose, therefore, of this paper is to give an overview of high quality typesetting requirements (covered and not covered by TeX 3.0) thereby, we hope, channeling future developments so that we do not end up with several incompatible "TeX-based systems", but rather with one system that will provide the same characteristics (i.e., quality, portability, and availability) as the current program.

TeX was designed as a low-level formatter, a stable kernel, of a typesetting system where extensions at both ends would be possible to take into account developments in printing technology (back end) and in user interfaces (front end) [14]. Thus, complaints about user unfriendliness of TeX are uncalled for, since such requirements can be handled by front ends either written in the TeX language itself like L^ATeX and, therefore, fully portable, or in an external language like ArborText's Publisher, or VAX Document, etc. These systems use TeX or a TeX-based system as the ultimate formatter but provide a user-friendly interface [32].

When we discuss missing features, we must distinguish carefully between things which can and should be handled by a front end system and things

that are truly tasks for a formatter and cannot be handled in \TeX 3.0. In the following sections we analyze features required for high quality typesetting, discussing whether they can be handled by \TeX primitives or by a suitable front end, or both. If they cannot be handled, we attempt to find ways to achieve the desired results. Finally, in section 12, we switch our attention to the concepts of the \TeX language itself, outlining some ideas on how a language for a new system could describe the underlying concepts more clearly.

2 Line breaking

\TeX 's line breaking algorithm is clearly a central part of the \TeX system. Instead of breaking a paragraph line by line, the algorithm regards paragraphs as a unit and searches for an 'optimal solution' based on the current values of several parameters. Consequently, a comparison of results produced by \TeX and other systems will normally favour \TeX 's methods.

Such an approach, however, has its drawbacks, especially in situations requiring more than block style text of a fixed width. The final line breaks are determined at a time when information about the content of the current line has been lost (at least for the eyes of \TeX , i.e., its own macro language), so that \TeX provides no sort of post-processing of the final lines based on their content.

Furthermore, there is no way to influence the paragraph shape with regard to the current position on the page, since this information is not known *a priori*. See section 4 for further discussion of this topic.

The use of only four categories (tight, decent, loose, very loose) to distinguish different glue-settings in adjacent lines seems somewhat inadequate. The number of categories should be increased. In addition, a more global approach (even beyond paragraph borders in certain circumstances), taking the overall variation of glue-setting into account might produce better results.

2.1 Line breaking parameters

While the algorithm provides a variety of parameters to influence layout, some important ones for quality typesetting are missing. There is no way to deal with vertical stripes produced by interword gaps falling into the same vertical position. A similar problem involves identical words one above the other, especially at the beginning of a new line. Both problems are distracting to the eyes of the reader and will destroy any effort to produce a beautifully broken paragraph. A good example is shown at the beginning of the third paragraph of section 4

where "...breaking algorithm ..." is repeated on two lines.

Another aspect of fine print is the assurance that the last line of a paragraph will not be too short. This is especially important in layouts which use paragraph indentation, where an undesired gap would be produced if the last line of one paragraph is shorter than the indentation of the next paragraph. Unknown to most \TeX users, this can be prevented by a special setting of \TeX 's line breaking parameters as shown in example 1 in section 14. While other parts of this paper use this setting, this paragraph shows the undesired effect.

Hyphenation of consecutive lines is handled for up to two lines (`\doublehyphendemerits`), but there is no possibility of avoiding paragraphs like the current one and the next one, in certain circumstances. As one can easily observe, the number of hyphens in these paragraphs is artificially forced by setting some of \TeX 's line breaking parameters to unusual values. But in non-English languages (with longer word lengths on the average), such situations present real-life problems.

Another problem is the discrepancy between the first and later lines of a paragraph, produced by the implementation of the paragraph indentation. This is especially crucial in layouts with zero indentation, because space at the beginning of the first line (for example, from `\mathsurround`) will not vanish into the margin because of the implicit `\hbox` representing the indentation (even if not visibly present), while such space will be removed at the beginning of later lines. This will result in strange starting gaps.

3 Spacing

When block text is to be produced, it is necessary to change the interword or the intercharacter spacing, or both. Since variable intercharacter spacing is frowned upon by the experts (except in rare circumstances), a line breaking algorithm has to stretch or shrink the interword space starting from an optimal value given by the font designer until the final word positions are determined. Again, \TeX has a well designed algorithm to take such stretchability into account. Additionally, each character has a so called `\spacefactor` assigned to it which will influence a following space, so that it is possible to enlarge or reduce the interword space after certain characters. As an example, compare the spacing after punctuation characters in this paragraph with other paragraphs.

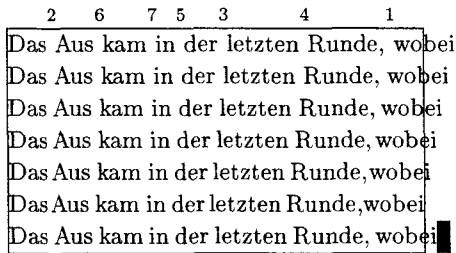


Figure 1: Interword spacing

The interword spaces are numbered in a way so that higher numbers denote spaces which should shrink less using the rules given by Siemoneit [28]. The last line shows the resulting overfull box which would be produced by standard T_EX in this situation.

There is no provision, however, for influencing the interword gaps in relation to the current characters on both word boundaries. If it is necessary to shrink a given line, not all gaps should shrink by the same amount. Instead, it is best to shrink more after a comma, for example, than between ‘then it’ because of the different shape of the characters. There is no way to achieve such fine tuning in T_EX except by manually adding `\hskip` in lines, which is intolerable. An example of this approach is shown in figure 1. Such a mechanism is clearly font dependent, and an implementation would, therefore, change both T_EX and METAFONT, since the best place to store this information is in the TFM file. But even tables similar to `\sfcode` (fixed by the format or a macro) would be a big improvement, since most fonts in use tend to have similar shapes.

In T_EX’s concept for glue-setting an important distinction is made between stretchable and shrinkable glue: while the latter is only allowed to shrink to a fixed minimum (i.e., the natural width minus the shrink component), any given amount of stretchable glue is automatically allowed to stretch arbitrarily far.¹ The reason for this behavior is that it allows the line breaking algorithm to achieve ‘emergency results’ if no suitable line breaks are otherwise found. But this is undesirable in most circumstances, so that either the stretching should be bounded similarly to shrinking in all cases (resulting in some changes to the line and page breaking algorithms), or another class of glue should be added, for which the amount of stretching can be determined individually.

Don Knuth [18, pp. 394–395] gives an example of how to achieve hanging punctuation (together with special fonts, as he noted). Since this, too, is a sign of good quality typesetting, it is questionable whether such a scheme (that will make the ligature mechanism partly unusable, along with other side ef-

fects) is advisable or whether this should be a direct feature of a future program.²

4 Page breaking

A major problem with T_EX is its page breaking algorithm. Page breaking is handled asynchronously by moving things at certain times from the list of recent contributions onto the “current page” until this list is filled with more items than will fit on the page in final form. The final page break is chosen by weighing badness (how full the page is if we break here) and penalties (how expensive it is to break here). Such penalties will be placed after some of the lines either by the line break algorithm or during macro expansion.

But good page layout usually requires taking pairs of facing pages into account as they will be seen by the reader. This is in itself not a real restriction, because one can view a double page as a huge case of two-column format, provided, of course, that both pages can be held simultaneously in memory. But, unfortunately, none of T_EX’s internal mechanisms can handle multi-column layout properly, so that such an approach has to avoid all internal features for page breaking like `\insert`, etc. Good examples that also show the limitations of T_EX in this regard are the output routine of L^AT_EX [22] and implementations of multi-column layout [4, 24], all bordering on the impossible.

But, even more important, the line breaking algorithm in conjunction with the page breaking algorithm pose unsolvable problems. When the final page break is chosen by T_EX, all paragraphs which were once candidates for the current page are already divided up by the line breaking algorithm, and this division cannot be undone for text carried over to the next page, since some of the necessary information (space at the line breaks, for example) is lost. This makes it impossible to change the page layout at a fixed place, e.g., at the top of a new page, to leave room for a small figure surrounded by text. Only in very restricted circumstances can a solution be found inside T_EX [9], but documents of moderate complexity cannot be handled this way. A general solution to this problem can be included in the current T_EX in an upward compatible manner. A prototype was designed at the University of Mainz shortly after the conference at Stanford [25].

¹ Under normal circumstances, however, this is prevented by the badness function.

² Starting with the next section, this article uses hanging punctuation. The change in quality is clearly visible although improvement is still possible by making subtle adjustments to all characters (e.g., move the ‘r’ a tiny bit out, etc.) to reach a perfect alignment.

It is an open question whether we should follow \TeX 's page breaking algorithm at all, since it was stopped short because of the space and time constraints of the computers available at the time of its development. In his PhD thesis [26], M. Plass considered several global optimization strategies, using a two pass system. His results open a wide field for future research work. Some of his ideas seem to be used in the Type & Set system [3].

The main contribution of \TeX 82 to computer based typesetting was the step taken from a line-by-line paragraph breaking algorithm to a global optimizing algorithm.³ The main goal for a future system should be to solve the similar, but more complex, problem of global page breaking.

5 Page Layout

For the tasks of page makeup, \TeX provides the concept of output routines together with insertions and marks. The concepts of insertions and marks are tailored to the needs of a relatively simple page layout model involving only one column output, footnotes, and at the most simple figures once in a while.⁴

The mark mechanism provides some information about certain objects and their relative order on the current page, or more specifically, information about the first and last of these objects on the current page and about the last of these objects on any of the preceding pages. Such information is necessary to construct certain kinds of running heads, e.g., one with the name of the current chapter or with information about the first and last word explained on the page, etc.

This is a global mechanism, however, so that only one class of objects can take advantage of the whole mechanism. If more than one class is implemented, some of the features of the mechanism are lost within one class.⁵ As a consequence of this deficiency, one should extend the mark mechanism to a system of independent marks which can be allocated separately by a macro package.

The insertion mechanism seems to be derived from 'footnote applications', and later extended to allow for some simple kinds of floating insertions.⁶ But placement of floating objects needs more than simply storing them in a huge box which is split at a certain point when the output routine is called. Floats are accompanied by captions and the like, which require differing treatment depending on their final placement on the page. Floats may vary in width even if they belong to the same class. On the other hand, deferred floats in one class may influence or even prohibit the placement of floats in other classes.

Some classes of insertions, like marginal notes,

cannot be handled by the primitives at all. To provide such features \LaTeX , for example, defines its own memory management for floating objects. Naturally, such a mechanism is slow and space consuming. Additionally, the quality of page breaks is further reduced because it is difficult to maintain, for free, all the information provided by the insertion concept.

Another problem is the design decision that the page breaking mechanism is, at least in its crucial parts, available only in outer vertical mode; thus, for example, space for insertions is not taken into account when splitting a `\vbox`.

For a designer, \TeX 's model of interline glue determination is very unfamiliar because it does not allow specification of baseline to baseline spacing in a page-spec without using lengthy and complicated internal computations (see figure 2 on the next page). This also means that it is nearly impossible to implement grid-oriented specs, i.e., where (nearly) all baselines fall into predetermined positions. This article uses a grid-oriented spec (which was partly hand-prepared) to show this aspect of high quality typesetting. Details are given in example 3.

The design of suitable primitives for this complex must go hand in hand with a new algorithm for page breaking, comprising probably the most drastic changes to the \TeX system.

6 Penalties — measurement for decisions

Line and page breaks in \TeX are determined chiefly by weighing the "badness" of the resulting output⁷ and the penalty for breaking at the current point. Such penalties are either inserted directly by the user (during macro expansion) or added later by certain \TeX formatting routines.

The main problem posed by the implicit penalties is that they cannot be removed. If, for example,

³ It should be noted that a similar algorithm was developed independently by J. Achugbue [2]. A comparison might lead to further enhancements.

⁴ The term 'one column output' means that all text is assembled using the same line width. Problems with variable line width are discussed in section 4. Of course, this already covers a wide range of possible multi-column layouts, e.g., the footnote handling in this article. But a similar range of interesting layouts is not definable in \TeX 's box-glue-penalty model.

⁵ The \LaTeX implementation provides an extended mark mechanism with two kinds of independent marks with the result that one always behaves like a `\firstmark` and the other like a `\botmark`. The information contained in the primitive `\topmark` is lost.

⁶ This is only a guess from studying [17].

⁷ This is in some sense a measure of the difference between the optimal and actual amount of white space on the line or page in question.

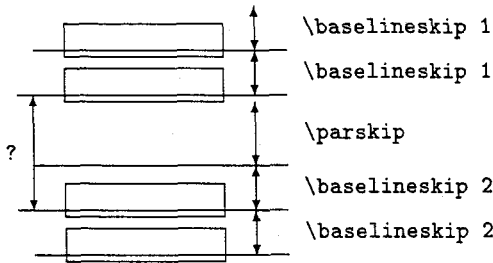


Figure 2: Baseline to baseline spacing

To implement a baseline to baseline dimension, for example between a paragraph and a heading (denoted by the question mark), the value for `\parskip` has to be determined depending on the `\baselineskip` of the second paragraph. Unfortunately the value of `\baselineskip` used will be the one current at the end of the second paragraph while the `\parskip` has to be computed at its beginning.

the line breaking algorithm decides to put a penalty after a line (e.g., from `\widowpenalty`), there is no way to prohibit a page break at this place via macro expansion except, of course, by setting the penalty in question to infinity. This is the result of T_EX's algorithm that consecutive penalties p_1 and p_2 behave like $p_3 := \min(p_1, p_2)$.

Local changes to the offending penalty parameters are error prone and time consuming, since after each change the following page breaks might fall in different places. Additionally, future editions of the document are made more difficult because every correction of this sort might produce undesired results after a single change.

If we think in terms of the current T_EX (i.e., assuming all major algorithms are unchanged), it would be better to adopt a different strategy in the case of consecutive penalties, either $p_3 := \max(p_1, p_2)$ or $p_3 := \frac{1}{2}(p_1 + p_2)$. For both functions, the boundary cases $p_i = \pm\infty$ would need special care. Breaking the chain of penalties could be performed as usual by grouping or `\kernOpt` or similar procedures as is already done with ligatures, etc.

As we mentioned, this is a solution within the framework of T_EX82. If a totally different algorithm for page breaking is designed, the concepts of local penalties should be reconsidered, too, and probably be replaced by a different strategy.

7 Hyphenation

When typesetting text, especially in narrow columns, hyphenation is often inevitable in order to avoid unreadable, spaced out lines. But readability has many faces; one of the golden

rules says [28] “Avoid more than two hyphenated lines in a row.” As we mentioned in section 2, this cannot be specified in T_EX (unless one disables even two consecutive hyphens).

Another problem is hyphenation of words at places which are allowed but which distort the meaning:

Stiefel-tern	Stief-eltern
Spargel-der	Spar-gelder ⁸

One should probably always forbid such problematic hyphens by choosing appropriate patterns for Liang's algorithm [23]. But readability is also distorted by the hyphenation of very short syllables which give no or almost no information about the word hyphenated and, therefore, slow down the reading process considerably. With T_EX 3.0 it is now possible to adjust the minimal number of letters to the left and right of a hyphen. This is necessary for many languages which often have long words and, for example, many two-letter syllables like German. But there is no provision for assigning weights to hyphenation points, i.e., it is a simple yes or no situation. One possible solution to this problem would be to add another class of demerits which could be applied via

$$\frac{\text{users value}}{\text{length of broken part}}$$

or a similar function. Of course, one probably has to distinguish between pre-break and post-break text (and/or length) in the formula.

Since the quality of a certain breakpoint also depends on the word (i.e., the meaning of the word-parts), we should consider whether such information could be provided by the hyphenation algorithm itself.

8 Box Rotation

T_EX's concept of document representation is strictly horizontal and left to right oriented. Beside the problem of processing documents containing right-left or top-down oriented languages (which can be handled to some extent by special versions of T_EX [21]), this also poses unnecessary restrictions in standard applications. Except by using `\special` (for POSTSCRIPT devices) it is impossible to rotate certain parts of the document. While arbitrary rotation is indeed next to impossible for most output devices, rotation by 90° can be handled in a simple manner by rotating the character cells. It should be easy to include some sort of `\rotate` primitive to T_EX's lan-

⁸ The meanings of the words are ‘step-parents’ and ‘savings’, but the first parts of the words in the left columns mean ‘boot’ and ‘asparagus’, respectively.

guage which would allow rotating hboxes and vboxes by multiples of 90 degrees.⁹ This would allow inclusion for example, of landscape tables, etc., in a document, without the need to use real glue and scissors to add the page number or the running head.

9 Font Information

The ISO Draft Standard [1] contains hundreds of properties describing a font resource. While some of them are accessible through TeX via `\fontdimen`, the majority are not. It seems advisable to add more of them to the set of TeX's parameters (for example, a recommended `\baselineskip`), in order to be able to make font family changes in documents more easily.

9.1 Virtual Fonts

Use of font families in TeX, which differ in character position, etc., from the defaults in the Computer Modern family, is difficult but can be achieved as proved in several projects [7, 35]. The proposed use of virtual fonts [20] may help to simplify matters in this regard. If it is possible to agree on standards for the position and the method of access for certain accented characters for the most common Latin alphabet languages, it should be possible to typeset multilingual documents (using the new features of TeX 3.0) without introducing unnecessary variants of standard fonts differing only in the availability of certain accented characters as 'real' letters.¹⁰ A good survey of accented characters in Latin alphabet languages and a proposal for their access via ligatures is given by Haralambous [8].

9.2 Ligatures and Kerns

Unfortunately, ligatures as well as kerns differ from language to language. Take, for example, the 'ffl' ligature which is not used in traditional German documents. On the other hand, such documents contain 'ch', 'ck' and 'ft' ligatures to obtain a better script. The following examples shows the difference:

Druckschrift	Druckschrift	(standard)
Druckschrift	Druckschrift	(German ligatures)

Since these special ligatures do not involve new letter shapes (at least not in most font families) it is possible to achieve the desired results simply with kerning. In the Computer Modern font family [15], both 'ch' and 'ck' are contained in kerning programs, but only for serif fonts. Other font families show similar deficiencies. Thus, for typesetting German documents, one either needs special physical fonts (or at least virtual fonts), or a way to manipulate ligature and kerning programs from within the TeX program. For reasons of portability, a controlled ac-

cess to the ligature/kerning programs during font loading seems preferable.

10 Tables

Well-designed tables are difficult to typeset even for experienced hand composers. TeX's primitives `\halign` and `\valign` do a marvellous job in this respect, even in complex situations. There is one important subclass of tables, however, which cannot be handled at all, except with hand tailoring. It is not possible to specify combinations of horizontally and vertically spanned columns, e.g., an open curly brace spanning several rows in one column while the row structure is maintained on both sides.

Another feature often desired is the ability to specify tables spanning several pages. While this is difficult to achieve, since TeX's table primitives normally read the whole table before determining the column width, etc., it does not pose unsolvable problems with the advanced features of TeX 3.0.

11 Math

Mathematical typesetting is one of TeX's major domains where no other automatic typesetting system has been able to catch up. But even in this area several things could be improved.

The source code of $\mathcal{A}\mathcal{M}\mathcal{S}\mathcal{T}\mathcal{E}\mathcal{X}$ [30] shows many interesting examples where Spivak circumvents limitations of TeX's formatting rules by introducing complex code to define functions that should perform standard tasks in mathematical typesetting. A detailed analysis of these problems (double accents, under accents, placement of equation numbers, etc.) would easily fill several pages; some comments can be found in Spivak's documentation [29].

While TeX's spacing rules for math are quite good, it seems at least questionable that many of them are hardwired into the program instead of being accessible through parameters. The table for spacing between different math-atoms is probably the most important example of this sort.

Another problematical feature of TeX's math typesetting routines is that sub-formulas are always boxed at natural width even if the top level math-list is subject to stretching or shrinking. This might produce ugly results in certain circumstances. The concept of boxing sub-formulas has the additional disadvantage that such parts of a formula cannot be broken across lines. Therefore programming constructs like `\left...\right` which automatically de-

⁹ This would also require changes to the dvi language and thus changes in all driver programs.

¹⁰ The use of the accent primitive of TeX is not recommended for standard accents of a language [18, p. 54] since it disables the hyphenation facility.

termine the height of variably sized delimiters, cannot be used in complex displays.

12 \TeX 's language

The language of \TeX is divided into two parts which are described as the mouth and the stomach of \TeX [18]. This distinction is crucial in many applications since the output produced by \TeX 's gastrointestinal routines cannot be fed again into its mouth, i.e., its scanner. Actually, constructed boxes are post-processable to a limited extent (via `\lastbox`, `\unpenalty`, etc.) but arbitrary constructions cannot be handled this way because primitives for manipulating things like characters, rules, etc., are missing. In general, this distinction is the reason for many obstacles in \TeX -programming, which sometimes prevent any solution at all. A new system should remove this two-class society of internal commands.

Another severe problem of the language is its incompleteness regarding standard programming constructs (such as certain conditionals, an acceptable arithmetic parser, etc.), as well as special constructs suitable for typesetting. To determine, for example, the length of the last line in a paragraph (which is done automatically by \TeX when typesetting displays), one has to use a complicated and lengthy computation, as shown in example 2, below. This example also shows one of the inconsistencies of \TeX 's language: `\prevgraf` has to be advanced using a scratch register because the direct use of `\advance` is forbidden. Many problems of this sort can be found by looking at appendix D of the *The \TeX book* [18, pp. 373–401] which is entitled “Dirty Tricks”. Actually nine out of ten examples therein are used in the implementation of \LaTeX [22], which shows that these examples are far less exotic than the preface to this appendix suggests. As examples of missing programming constructs, conditionals, such as `\ifmathopen`, for determining math atoms should be mentioned.

Such problems explain the fact that general application software written in \TeX (like \LaTeX) easily takes up more than a third of the available memory without typesetting even a single letter. For better and more stable front ends one needs a language where such tasks can be specified in a more elegant manner.

Some of \TeX 's restrictions in the language are due to the representation of dimensions in most \TeX installations as ‘real numbers’, which are machine-dependent.¹¹ To make \TeX nevertheless machine-independent, Knuth tried to prevent machine-dependent results generated in \TeX 's stomach from creeping into parts accessible to the scan-

ner, or to influence internally any decisions about line or page breaks. As a result of this strategy, the use of the code in example 2 together with a finite `\parfillskip` (as in example 1) will nearly always produce the value `\maxdimen` instead of a decent one.¹² For the same reason, Knuth said in a conversation with the author at Stanford that he cannot permit the removal of arbitrary items in a constructed list since this would allow access to floating-point arithmetic. But there exists another way to achieve machine-independence, which would also eliminate the restrictions mentioned, namely to change from floating-point to fixed-point arithmetic¹³ which can be done in a straightforward way, as Knuth himself has acknowledged [16, p. 46].

\TeX is a macro-language with all the advantages and disadvantages. Anyone who ever wrote a relatively long application in \TeX knows that debugging is extremely difficult. Transparent programming, as proposed by the author of \TeX [10], is next to impossible: it is no problem to write three lines of \TeX code that cannot be understood even by \TeX perts without a second and third look. But it is much more difficult to write \TeX code that performs a desired function, and is, at the same time, understandable to the average user. The examples given in section 14 are good test cases; they are all straightforward \TeX -coding, but their precise meanings are difficult to understand without explanatory text.

Many problems arise from design decisions based on totally different semantic constructs, which have similar or identical syntactical structure. The most important examples are the curly braces and the dollar sign.¹⁴ The curly braces are used both for delimiting arguments during macro expansion, as well as for the start and end of block structures that define the scope of certain declarations. In math mode they have the additional meaning of delimiting the scope of a sub-formula. Two consecutive dollar signs normally start or end a display formula, but in restricted horizontal mode they simply denote an empty math formula. Such concepts should be unraveled for the sake of clarity.

\TeX 's language is suitable for simple programming jobs. It is like the step taken from machine code (of the formatter) to assembly language. For

¹¹ Actually, this only applies to internal dimensions representing stretching or shrinking of glue, computed kerns for accents, and some others.

¹² The reason is given in module 1148 of the \TeX program [16, p. 470].

¹³ Perhaps, always using the same floating-point algorithm (either available in the compiler library or simulated by the program) would be even better.

¹⁴ To be more exact, the three characters with `\catcode` one, two, and three.

complex programing tasks of general application software, especially from the viewpoint of logically tagged documents [5], a more powerful language with well-defined concepts for variable-bindings, procedures, etc., is preferable. While this aspect can be achieved in a front end programming language (which compiles into the TeX language) it is better to include it, for the sake of portability, in the TeX kernel. In the author's opinion, an ideal language should combine the advantages of procedural languages with the goodies of interpreter features.¹⁵ As a side effect, such a language would be partly compilable.

13 Conclusion

The current TeX system is not powerful enough to meet all the challenges of high quality (hand) typesetting. The author shares Knuth's dream of a stable, low-level formatter which is able to produce documents of highest quality. But, unlike Knuth, he views the current TeX only as a very good prototype on the way to reach this goal.

As outlined in this paper, many important concepts of high quality typesetting are not supported by TeX 3.0. Further research is necessary to design a typesetting language which can handle these tasks properly.

The TeX user community needs an open mind for new developments that keep the 'TeX-System' the state of art in the field of computer typesetting. As Knuth is no longer involved in research on typography it is important for TUG to find an identity in *supporting* and *maintaining* 'the best typesetting program' and not only promoting the program that Knuth has given to the world. If we don't strike for even further quality our large community might fall back to insignificance.

One important step for TUG would be to initiate and (when advisable) support further research projects which will take up the challenges posed by the Stanford project.

14 Examples

Example 1 To avoid nearly empty lines at the end of a paragraph, the following code could be used:

```
\parfillskip \columnwidth
\advance \parfillskip -1.5\parindent
\advance \parfillskip 0pt minus \parfillskip
\advance \parfillskip 0pt minus -1em
```

This setting was used throughout this article, for example, which led to some changes in section 2. With the standard setting (e.g., 0pt plus 1fil), the first and third paragraph therein would have ended with the word part 'ods' (from 'meth-ods') and the word 'topic' respectively. Unfortunately,

this solution is not perfect either, since it produces somewhat funny results with lines consisting of two very short words.

Example 2 The following code determines the length of the last line of the preceding paragraph, using a feature of TeX built into mathematical displays. This code can be used to determine, for example, the amount of white space before an itemized list, or something similar. The example of code given is not really suitable for direct applications of this sort, since it simply displays the value found on the terminal. But it could easily be extended.

```
\def\getlastlinewidth{\ifhmode $$$%
\predisplaypenalty\@M \postdisplaypenalty\@M
\abovedisplayskip-\baselineskip \belowdisplayskip\z@
\abovedisplayshortskip\abovedisplayskip
\belowdisplayshortskip\belowdisplayskip
\showthe\predisplaysize
$$\count@\prevgraf \advance\count@-\thr@@
\prevgraf\count@ \else\typeout{*Not hmode*}\fi}
```

This example illustrates several important things. First, there is no elementary way to compute such important information. Second, it is one of the (not unusual) cases where information about the typesetting process can only be got by introducing undesired (since space-consuming) penalties, glues, and null-boxes in the output.

Example 3 To introduce a grid-oriented spec all flexible glue on the page has to be disposed of (except for \skip\footins) and the \vsize must be adjusted. Titles are set with 8pt + 4pt = \baselineskip leading and we have to ensure that the above space is kept after a page break. Lists are set with 6pt + 6pt so the inner lines are halfway off. Page breaks insides lists would need special treatments, e.g., by increasing \topskip to keep the subgrid. Figures and examples in different type sizes are measured and necessary kerns added to keep the surrounding material in line. Again this approach only works if no page break intervenes, which happens to be the case for this article. To use the badness calculation of TeX for determining page breaks a stretchable \topskip can be used. During the output routine this extra stretch must then be canceled again.

References

1. *Information Processing — Font Information Interchange, ISO/IEC JTC 1/SC 18/WG8 N1036*, February 1990.

¹⁵ Some sort of LISP-like system, but with primitives suitable for typesetting.

2. Achugbue, James O. "On the line breaking problem in text formatting." *Proc. of the ACM SIGPLAN/SIGOA*, 2(1, 2), 1981.
3. Asher, Graham. "Type & Set: T_EX as the engine of a Friendly Publishing System." Pages 91–100 in *T_EX applications, uses, methods*, Malcolm Clark [6].
4. Benson, Gary, Debi Erpenbeck, and Jannet Holmes. "Inserts in a multiple-column format." Pages 727–742 in *1989 Conference Proceedings*, Christina Thiele [31].
5. Bryan, Martin. *SGML: an author's guide to the standard generalized markup language*. Addison-Wesley, Woking, England; Reading Massachusetts, second edition, 1988.
6. Clark, Malcolm, editor. *T_EX applications, uses, methods*, Chichester, West Sussex, England, 1990. Ellis Horwood Limited. Exeter conference July 1988.
7. Conrad, Arvin C. "Fine typesetting with T_EX using native autologic fonts." Pages 521–528 in *1989 Conference Proceedings*, Christina Thiele [31].
8. Haralambous, Yannis. "T_EX and latin alphabet languages." *TUGboat*, 10(3):342–345, November 1989.
9. Honig, Alan. "Line-Oriented Layout with T_EX." Pages 159–184 in *T_EX applications, uses, methods*, Malcolm Clark [6].
10. Knuth, Donald E. "Literate programming." *The Computer Journal*, 27:97–111, 1984. An expository introduction to WEB and its underlying philosophy.
11. Knuth, Donald E. *Computers & Typesetting*. Addison-Wesley, Reading, Massachusetts, 1986. Consists of [18, 16, 13, 12, 15].
12. Knuth, Donald E. METAFONT: *The Program*. Volume D of *Computers & Typesetting* [11], 1986.
13. Knuth, Donald E. *The METAFONTbook*. Volume C of *Computers & Typesetting* [11], 1986.
14. Knuth, Donald E., September 1987. Talk given at Gutenberg Museum Mainz.
15. Knuth, Donald E. *Computer Modern Typefaces*. Volume E of *Computers & Typesetting* [11], July 1987. Reprint with corrections.
16. Knuth, Donald E. *T_EX: The Program*. Volume B of *Computers & Typesetting* [11], May 1988. Reprint with corrections.
17. Knuth, Donald E. "The errors of T_EX." Technical Report STAN-CS-88-1223, Stanford University, Department of Computer Science, Stanford, California 94305, September 1988.
18. Knuth, Donald E. *The T_EXbook*. Volume A of *Computers & Typesetting* [11], May 1989. Eighth printing.
19. Knuth, Donald E. "The new versions of T_EX and METAFONT." *TUGboat*, 10(3):325–328, November 1989.
20. Knuth, Donald E. "Virtual Fonts: More fun for Grand Wizards." *TUGboat*, 11(1):13–23, April 1990.
21. Knuth, Donald E. and Pierre MacKay. "Mixing right-to-left text with left-to-right text." *TUGboat*, 8(1):14–25, April 1987.
22. Lamport, Leslie. *latex.tex*, February 1990. L^AT_EX source version 2.09.
23. Liang, Franklin Mark. *Word Hy-phen-a-tion by Com-put-er*. PhD thesis, Stanford University, Department of Computer Science, Stanford, CA 94305, August 1983. Report No. STAN-CS-83-977.
24. Mittelbach, Frank. "An environment for multi-column output." *TUGboat*, 10(3):407–415, November 1989.
25. Mittelbach, Frank. Letter to Don Knuth, September 1989. Suggestions for the T_EX 3.0 release. Published by R. Wonneberger in [34].
26. Plass, Michael Frederick. *Optimal Pagination Techniques for Automatic Typesetting Systems*. PhD thesis, Stanford University, Department of Computer Science, Stanford, CA 94305, June 1981. Report No. STAN-CS-81-970.
27. Rynning, Jan Michael. "Proposal to the TUG meeting at Stanford." *T_EXline*, 10:10–13, May 1990. Reprint of the paper that triggered T_EX 3.0.
28. Siemoneit, Manfred. *Typographisches Gestalten*. Polygraph Verlag, Frankfurt am Main, second edition, 1989.
29. Spivak, Michael. *amstex.doc*, 1990. Comments to [30].
30. Spivak, Michael. *amstex.tex*, 1990. $\mathcal{A}\mathcal{M}\mathcal{S}\mathcal{T}\mathcal{E}\mathcal{X}$ source version 2.0 (without comments).
31. Thiele, Christina, editor. *1989 Conference Proceedings*, volume 10#4 of TUGboat. T_EX Users Group, December 1989.
32. Wittbecker, Alan E. "T_EX enslaved." Pages 603–606 in *1989 Conference Proceedings*, Christina Thiele [31].
33. Wonneberger, Reinhard. "T_EX in an industrial environment." In Brüggemann-Klein, Anne, editor, *1989 EuroT_EX Conference Proceedings*, 1990. To appear.
34. Wonneberger, Reinhard. "T_EX yesterday, today, and tomorrow." *T_EXhax*, 90(5), January 7 1990.
35. Youngen, R. E., W. B. Woolf, and D. C. Lattner. "Migration from computer modern fonts to times fonts." Pages 513–519 in *1989 Conference Proceedings*, Christina Thiele [31].

Vertical Typesetting with T_EX

Hisato Hamano

6-12-1 Minami Aoyama, Minato-ku, Tokyo, 107-24, JAPAN
+81 3 486-4518. hisato-h%ascii.co.jp@uunet.uu.net

Abstract

ASCII Corporation, as technical publishers, ourselves, has long felt a need to introduce into the Japanese market a truly Japanese technical documentation system. As a first step, three years ago we developed a Japanese version of T_EX capable of handling *kanji*.

This paper introduces our second step in producing an even more sophisticated T_EX system — the addition of a vertical typesetting function.

Japanese — The Language

The history of the language. Near the beginning of the third century, a man by the name of Wani came to Japan from the nation of Kudara, located in the eastern part of the Korean peninsula. With him he brought volumes of The Analects of Confucius and *Senjimon*, a Chinese textbook for studying *kanji*, or Chinese characters.

This was the introduction to Japan of Chinese characters developed in the 14th century B.C. but it was not until the 4th and 5th centuries, when trade volume between the two nations increased, that *kanji* really got its beginning.

The Japanese language originally developed without a form of written expression, so it remained oral and employed professional narrators called *kataribe* to relay news of important events when necessary.

Those descendents of the original Chinese immigrants to Japan worked as official recorders, transcribing the ancient Japanese language, called *yamatokotoba*, into *kanji* and providing Japan with its first form of written expression.

Japanese characters. Unlike the English alphabet which is made of phonograms, *kanji* are ideograms, that is, symbols representing things or ideas. As is the case with hieroglyphics, *kanji* began as drawings of natural objects.

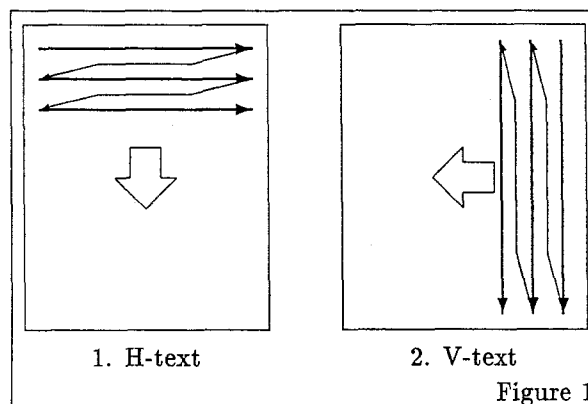
JIS (Japan Industrial Standard) recognizes 6,353 characters in the level one and level two categories used by computer manufacturers. Most PCs now have level two capability due to the availability of inexpensive memory. The Ministry of Education recognizes 1,945 *joyo kanji* as a minimum requirement for education.

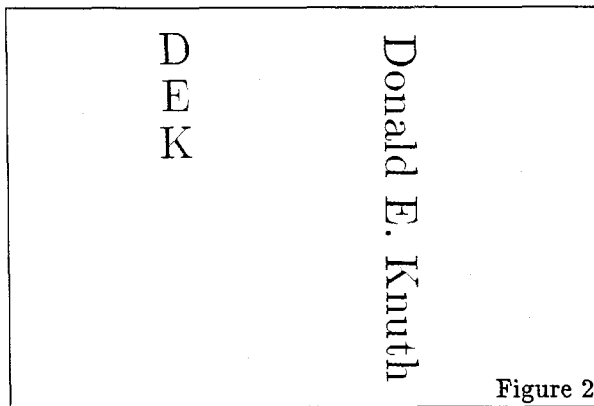
The Japanese language also has two phonetic alphabets called *kana* collectively and divided into *hiragana* and *katakana*. While *katakana* is used mainly to express words which are non-Japanese, *hiragana* forms a link between *kanji* and Japanese grammar.

Japanese typesetting.

V-text and H-text. Japanese sentences can be written in two ways (see Figure 1):

1. the form familiar to Western language speakers, starting from the top lefthand corner of the paper, writing horizontally to the right, with the next line starting under the previous line (hereafter referred to as “H-text”) and
2. the traditional form in Japan which was in use before the introduction of H-text, starting from the top righthand corner of the paper, writing downwards to the bottom, with the next line starting to the left of the previous line (hereafter referred to as “V-text” or vertical text).





As Japanese characters are essentially those of China, the basic form of writing is vertical, although at times numbers and English words appear horizontally (H-text).

Primary school textbooks are written in V-text for Japanese language studies and social studies but are H-text for mathematics and science. Outside of those texts for the sciences most textbooks appear in V-text format.

The line break rule (*kinsoku*). There are no spaces in Japanese sentences between words, so line breaks in the middle of words are not only possible but quite acceptable, whereas in English there are restrictions on where line breaks should occur.

The Japanese line break rule is called *kinsoku* and states that line breaks should not occur immediately before or after a symbol. For example 「 」 are used in Japanese as quotation marks and line breaks should not occur between an opening quotation mark and the character after it or between a closing quotation mark and the character before it.

Justification. As there are no word spaces characters are justified across the full line.

Handling non-Japanese in V-text. Short forms in English such as “DEK” (Donald E. Knuth) appear one letter at a time, vertically, but full spellings such as “Donald E. Knuth” appear written sideways (rotated 90°) in a manner similar to that seen on the spine of a book (see Figure 2).

Japanese computer files. As the need for business users of personal computers grows, the need to use Japanese *kanji* at the computer level grows also.

There are several problems involved in handling Japanese on computers that are not present in English.

Two-byte codes. Japanese characters are expressed using two-byte codes. However, one-byte

code English words are mixed in the same sentence at times with the two-byte *kanji* codes.

There are presently three coding schemes for mixing one and two-byte characters: (1) JIS (Japanese Industrial Standard), (2) Shift-JIS, and (3) EUC (Extended Unix Code).

In the JIS system, an escape sequence is used to switch between one and two-byte characters. Both Shift-JIS and EUC use an eighth bit to make such switches. In EUC the eighth bit of the JIS code is set at 1 only, while Shift-JIS employs a different method. For communications, JIS is used; for personal computers, Shift-JIS is used; and for UNIX, EUC is most common.

Typesetting H-text with T_EX

Once the problems of two-byte code usage and the line break rule are solved, H-text can be typeset. The Japanese T_EX used here is not NTT’s J_TE_X (*TUGboat* 8, no. 2) but one independently developed by ASCII Corporation. We call this pT_EX or Publishing T_EX.

Font switching for the two types of coding. We have prepared two current fonts. Computer Modern is used for the one-byte current font and a Japanese font for the two-byte current font, with the selection depending on the coding method employed. For mixing, we can use JIS, Shift-JIS, or EUC.

Line break rule. A small amount of *glue* is used between each character to make line breaks and justification possible, and where line breaks are not possible a penalty is imposed. This penalty is automatically inserted and can be adjusted for imposing penalties before or after characters. Although there are many characters to deal with, we have not used a lookup table because it would take too much memory. Instead, we used a 256 entry hash table, as there is a restricted number of cases in which penalties would apply.

Typesetting V-text with T_EX

We have tried using T_EX to do H-text typesetting and found that it rivals the traditional methods of typesetting. This lead us to consider using it to do some actual publishing; however, as V-text is still the most common form of official printing in Japan the inability to typeset vertically would confine such a system to a very restricted market. We therefore decided to extend T_EX to enable it to handle V-text typesetting.

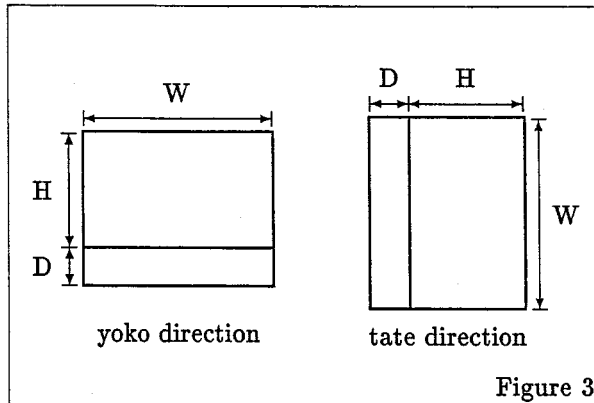


Figure 3

The essence of T_EX typesetting. The major problem to be overcome is determining whether the text being handled is to be printed as H-text or V-text.

The basis for T_EX typesetting is to combine characters to form a line, combine the lines to form a page and combine the pages to form a document. Forming lines of characters is called *hmode* in T_EX, while forming pages of lines is called *vmode*. This means that if *hmode* is used to line up characters vertically and *vmode* is used to line up the lines right to left, V-text typesetting becomes a possibility.

Direction. In Japanese publications, V-text and H-text are used together. For example, many books have body text in V-text with the page numbers in H-text. In other words, it is necessary to be able to use both V-text and H-text in any one document.

To solve this problem, ASCII has employed the idea of **direction**. The **directions** available are **tate** or vertical and **yoko** or horizontal. If the **direction** is **yoko**, T_EX behaves in the regular manner. In other words, while in *hmode* the elements are lined up from left to right, while *vmode* allows for formation from top to bottom. When using **tate direction** and *hmode* the elements are lined up from top to bottom, while *vmode* allows for formation from right to left.

The **direction** default is **yoko**. Text can be switched between V-text and H-text when necessary, but only when the `hlist` or `vlist` involved is empty. In the `\tate` primitive, the **direction** is set as **tate** while in the `\yoko` primitive it is set as **yoko**.

Boxes with direction.

In T_EX, lines and pages are all *boxes* with parameters expressed in *W*(width), *H*(height) and *D*(depth). Each box has a *Bline*, or baseline, from which *W*, *D*, and *H* are measured. In *hmode*, *Bline*

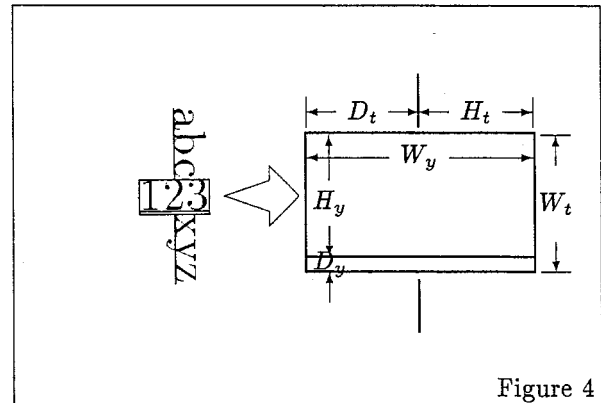


Figure 4

will line up boxes parallel to the direction of the text.

In the **yoko direction**, the *Bline* is horizontal. *W* is the length of the *Bline* and *H* is the length above the *Bline* while *D* is the length below the *Bline*.

In the **tate direction** the box *Bline* is vertical. There is a 90° difference between characters lined up in *hmode* and lines done in *vmode* and so the size of the box is expressed in terms revolved 90°(see Figure 3).

As explained before, it is possible to change **direction** in the middle of a document. In other words, a *box* formed in the **tate direction** can be lined up in **yoko direction**. The opposite is also possible.

```
(tate direction, hmode)
abc
\hbox{\yoko 123}
xyz
```

In this example, `\hbox` is formed in a **yoko direction** but `abc`, the *box* itself, along with `xyz` are in **tate direction** using *hmode* (vertical).

The direction of the *Bline* and the value of *W*, *H* and *D* for this `\hbox` differ for **tate** and **yoko directions**. When the box is made in the **yoko direction** the *Bline* is horizontal and $(W, D, H) = (W_y, H_y, D_y)$. When the box is actually used in the **tate direction** the *Bline* becomes vertical and $(W, H, D) = (W_t, H_t, D_t)$. The relationship between (W_y, H_y, D_y) and (W_t, H_t, D_t) is $W_t = H_y + D_y$ and $H_t = D_t = W_y/2$.

As shown in the illustration, the directions of the *Bline* inside and outside the box are different (see Figure 4).

Fonts for V-text. Japanese fonts for use in H-text and V-text are different. There are some differences

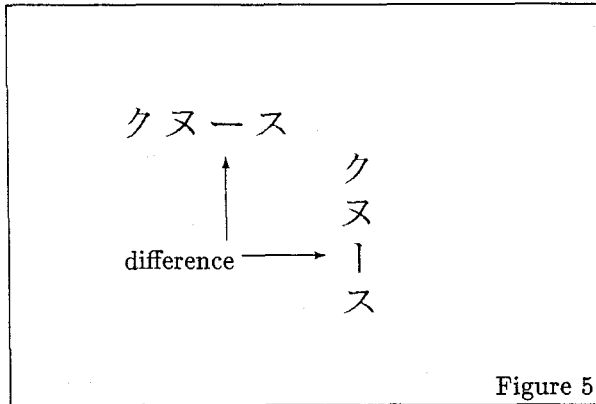


Figure 5

in the symbols used and in the information needed for typesetting (see Figure 5).

The baseline for H-text fonts is set horizontally while the baseline for V-text fonts is set vertically. In order to make vertical typesetting possible, the two fonts used for H-text (one-byte English and H-text Japanese) are supplemented by a third font for vertical Japanese text. When one-byte characters are used in **tate direction** they are rotated 90°.

Implementation

It is now necessary to explain how to implement this form of T_EX.

Using two-byte code. In standard T_EX the character field for *char_node* or *token* has 8 bits only. When using two-byte code, two *char_nodes*, or two *tokens* are linked to form one character with the two-byte code information added to the *info* field in the second node.

In the *char_node* we can check the *font* field to determine if the character is a one-byte or two-byte character. (see Figure 6)

For *tokens*, the category code tells us if the character is in one or two-byte code. If the category

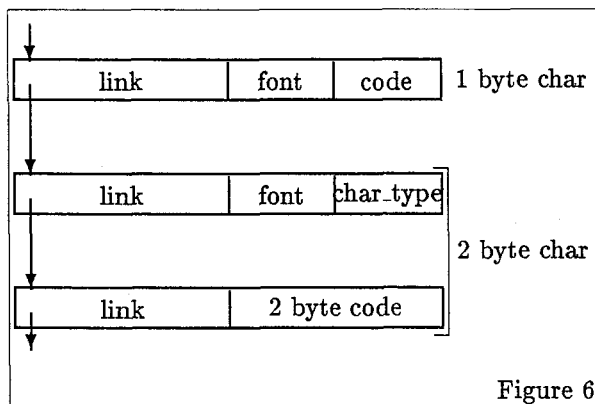


Figure 6

code is 16(kanji), 17(kana) or 18(other two-byte character) it is a two-byte character.

Boxes of Different Directions. When **yoko direction** boxes are linked to **yoko direction** lists, and **tate direction** boxes are linked to **tate direction** lists, it is possible to link *vlist_nodes* and *hlist_nodes* directly to the list as in original T_EX.

However, when a **tate direction** box is linked to a **yoko direction** box or vice versa the *dir_node* is used. The *dir_node* has the same structure as the *hlist_node* and *vlist_node*. For example, the result of the sample "Boxes with direction" is as shown in Figure 7.

In the *width*, *height* and *depth* fields of the *hlist_node* the $(W, H, D) = (W_h, H_y, D_y)$ values, when the *hbox* was made, are entered. In other words, the structure of the *hlist_node* does not change and the routine for making the *hlist_node* is the one found in original T_EX.

In the *width*, *height* and *depth* fields of the *dir_node* the $(W, H, D) = (W_t, H_t, D_t)$ values of the box when actually used are entered. When a list containing abc, *dir_node*, and xyz is processed, the (W, H, D) of the *dir_node* can be typeset much as directly linked *hlist_nodes* and *vlist_nodes* can be typeset.

Japanese tfm file format (jfm format). In the *tfm* files to date only 256 characters could be registered. This would not allow use of Japanese characters so the *tfm* format has been extended and called the *jfm* format.

Fonts are not divided into sub-fonts and therefore a variety of fonts can be used in the same document.

If metric data for each character are added, the file would become too large, so we have endeavored to keep this file as small as possible. To do this we took groups of characters which enjoyed similar font metrics and called each of these groups a

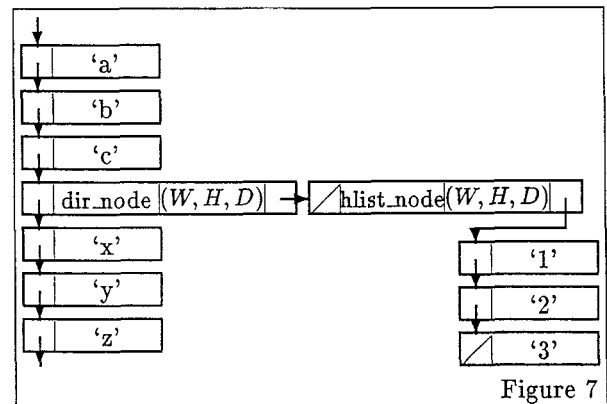


Figure 7

char_type. As almost all *kanji* have the same metrics we were able to put them in one group. A total of about 10 groups cover *hiragana*, *katakana*, and commonly used symbols.

In the *jfm* file there is a table showing in which **char_type** group each character belongs along with the metrics for that group. In this **char_type** table are listed the code and the **char_type** code. The code '0' has been assigned to the *kanji* group as it is the biggest and all codes not appearing in this table are considered to be '0', or *kanji*. This move has kept this table relatively small.

The metric information is provided in a format that is very similar to *tfm* although the *lig_kern* is somewhat different. As there are no ligatures in Japanese this area has been used for the *glue* mentioned previously. The *lig_kern* field has thus become the **glue_kern** field.

The difference between the Japanese *tfm* and the standard *tfm* can be found in the first half word. In the Japanese version, if the first half word uses a V-text font, it is given a value of 9. If using an H-text font, it is 11. In the standard *tfm* file the first half word is "length of the entire file, in words" and the standard *tfm* file is never less than 12 words.

The *tfm* for the Computer Modern font remains unchanged.

Dvi file format extension. In the *dvi* file we have used the *set2* command to express Japanese and have extended the *dvi* file format for use of V-text.

In the *dvi* driver there are modes for printing H-text and V-text. In the H-text mode the *dvi* driver remains unchanged. The beginning of each page is in H-text mode so *dvi* files can be printed out as they have been in the past.

In the print V-text mode the coordinate system for the *dvi* driver is different. Using commands such as *right*, *w*, *x*, *set*, *set_char*, *set_rule*, etc., the current point is moved in the vertical direction, and the commands *down*, *y*, *z* etc. are used to move the current point in the horizontal direction (see Figure 8).

A new command, **dir**, has been added to *dvi* to switch between H-text and V-text. 255 has been used as the code for **dir**.

As a new command has been added, new *dvi* files cannot be handled by the standard *dvi* driver. In order to distinguish between standard and new files, the preamble *id_byte* has been set at 2 and the postamble *id* at 3.

Programming. A 10,000 line Change File has been used to make all of the necessary changes from Knuth's original T_EX to our p_ET_EX.

The Printer Driver

Extensions made to the *dvi* format and the two-byte code mean it is not possible to print out p_ET_EX files using standard printer drivers. Also, the font file is another problem area since one font contains thousands of characters.

Recent Japanese printers include Japanese fonts in various sizes. Japanese T_EX fonts, unlike the Computer Modern font, use a common coding scheme so they can be used in place of fonts found on such popular printers as the Japanese Laser Writer II (NTX-j).

Japanese printing is done using the fonts that come with the printer and p_ET_EX printer drivers for various printers are available as public domain software.

At ASCII we use a Canon OEM machine at 480 dpi for printout work. This machine had no fonts so we had to make a new font file format.

Japanese fonts. Very few characters can be stored in the font file in formats such as *gf*, *pxl*, and *pk*, so a new format (*jxl* format) has been added.

jxl has code fields of two-bytes in *pxl* format. We use packing the same as that in the *pk* format for bitmap work.

We have produced *jxl* format bitmap fonts from outline font data we received from Dai Nippon Printing Company Limited.

Availability

p_ET_EX is public domain software and is enjoying wide distribution and use. There is also some p_ET_EX printer driver software in the public domain.

Dai Nippon Printing Co. Ltd. provides phototypesetting services and ASCII's Japanese version

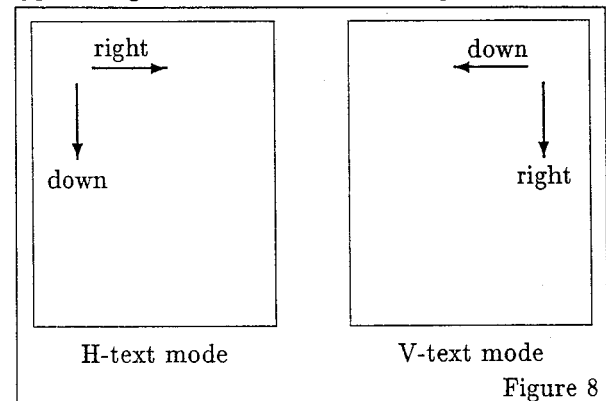


Figure 8

of Donald Knuth's *The T_EXbook* was typeset and printed using this system.

Acknowledgement

I would like to thank all of those people who have made our p_TE_X possible. Mr. Sagiya of DIT Co. Ltd. helped in the testing of our product while Mr. Enari of Dai Nippon Printing Co. Ltd. provided Japanese outline fonts.

Japanese T_EX, which forms the core of p_TE_X, was developed by Mr. Ohno and Mr. Kurasawa. Mr. Iseri and Mr. Tamura taught me the traditional aspects of typesetting and printing. Mr. Leach, who

helped in producing this paper, was one of many ASCII employees whose help was invaluable.

In p_TE_X we have been able to use many of the original T_EX routines. Original T_EX and the very elegant manner in which it was made we owe to the author of T_EX, Professor Donald E. Knuth.

References

- Saito, Yasuki. "Report on j_TE_X: A Japanese T_EX", *TUGboat* 8, no. 2.
- Kurasawa, Ryoichi. "Japanization of T_EX" (in Japanese), Japanese T_EX distribution tape.

Appendix

Sample Output by p_TEX

この世はすべて推計論か

ゆがみのない公平なコインを投げて、事を決める。コイン投げ自体は、その結果生じる事態とまったく無関係である。コイン投げには偏見の入る余地がまったくない。それ故に、コイン投げはもつとも倫理性の高い行為である。コイン投げは、結果として生じる事態にいつさい関知しない。つまり、物事の意味が完全に切り捨てられている。それ故に、コイン投げはもつとも倫理性の低い行為である。この一見矛盾した性格故に、コイン投げは新たな意味を獲得する。

※

いま、学問の世界では「stochastic (推計論的、確率的)」という言葉がさかんに使われる。偶然的、無作為的で混沌とした事象を表現する言葉である。そこで、偶然性(あるいは、確率)を客観的事実と見なして、この世界を構成する基本的要素の一つと考える姿勢が登場する。世界を推計論的にとらえる姿勢である。数理統計学や確率論は、単独では予測のつかない混沌とした事象を対象とし、混沌とした状態にある程度予測可能なパターンに帰結させることを目的とする理論で

あるが、そうした数学理論の手法を応用することもまた、推計論的世界観の表われである。stochasticの《反対》は「決定論的」ということになろう。しかし、いまや私たちは、推計論的でありながら同時に決定論的でもある世界に生きている。となれば、両者の関係は《反対》というよりも、《相補的》というほうが当たっている。

「統計論的」といえばよきそんなものを、なぜわざわざ「推計論的」などという新語を使うのか。どこが違うのだろうか、と疑問に思う向きもあろう。現在の用法では、「統計」とは大量のデータを収集して、そこから推論を導くことをいう。一方、「推計論」という言葉はもつと意味が広く、偶然性の問題を哲学的、方法的にとらえた理論と手法の総体を指す。

毎日の新聞にあふれる数字も、推計論を土台にしたものが少なくない。ニューヨーク市の全世帯のなかで、子どものいない世帯はおよそ何パーセントか。フロリダ州オーランドに住む四人構成の世帯は、平均何台車を持っているか。これこれの臓器移植手術が成功する確率はどのくらいか。ニック某の競馬予想。ある地域のハンバーガーチェーン店の月間総売上予測。好ましい状況が続いたために、保険料率が月間一〇〇ドルにつき0.82ドル引き下げられたというニュース。いずれの場合も、この数字にもとづいて何らかの方策なり、行動なりをとるべきだという意味合いが暗に含まれている。ネブラスカ州の十学年生の国語の成績がこれこれ、アイオワ州

デカルトの夢 256

「デカルトの夢」(Philip J. Davis, Reuben Hersh 共著 棕田直子訳 アスキー発行)より

Structured Document Preparation System

AutoLayouter

Yoshiyuki Miyabe, Hiroshi Ohta, and Kazuhiro Tsuga

Matsushita Electric Industrial Co., Ltd., 1006 Kadoma, Kadoma-shi, Osaka 571 Japan
+81-6-906-4600. CSNET: miyabe%isl.mei.co.jp@uunet.uu.net

Abstract

We have developed a structured document preparation system *AutoLayouter*, which consists of an easy-to-use structured editor and a Japanese L^AT_EX based formatter.

Not only have we designed better user interfaces, but we have introduced a simple document structure. A document produced with *AutoLayouter* is a one-dimensional list with each node corresponding to a logical component of the document. This use of the simple structure largely contributed to making the editor easy to use but powerful enough both for editing the document structure and its contents, which may contain finer substructures.

Since we use the simple structure, we had to append various macros to complement the differences between our structure and the L^AT_EX begin-end environments. We also developed a device driver which converts dvi files to Kanji PostScript files.

Introduction

In Japan, most of the commonly used document processing systems are Japanese word processors, designed originally to produce beautiful documents without having to resort to hand writing. The most serious problem in early Japanese word processors was to make Japanese input efficient, easy, and fast. This problem has been almost solved by the development of efficient Kana (Japanese alphabet) to Kanji (Chinese character) conversion algorithms. Now we face the second step in document processing of Japanese.

From the beginning Japanese word processors have been designed with some layout facilities, using the fact that Japanese characters usually have the same width; thus spaces and tabs may be used to align them. In addition, Keisen characters, special line characters, were created to solve other alignment problems and to create tables of any shape.

One direction for the advancement of Japanese word processors is to augment the layout facilities, in order to format documents more flexibly, as English desk top publishing systems do.

On the other hand, our analysis of the Japanese word processor market led us to conclude that

Japanese word processors are used with a great variety of documents and, therefore that some objective other than just the beautification of documents is indicated.

Among computer software people, Japanese versions of T_EX and L^AT_EX are being used as replacements for Japanese word processors. But due to the complex syntax and the scarcity of supporting tools, it is extremely difficult for the typical users of word processors to take advantage of T_EX's automated layout.

Basic Concepts of *AutoLayouter*

We developed *AutoLayouter* as a T_EX based document preparation system whose objective is to give T_EX's power to the users of Japanese word processors, and to induce them to write documents in more logical ways, similar to those used in making documents with L^AT_EX.

If the main task is to print documents, it is reasonable to use paper-oriented document preparation systems such as Japanese word processors or desk top publishing systems; but when the object is to manage documents, other means are required. Even in this case, if the size of a document is too large for it to be developed by only one person, one has to look for a better way to produce it.

Furthermore, when requirements of managing and delivering documents increase, working with such paper-oriented document preparation systems can cause serious problems:

Document file contains layout information mixed with text. In the word processor file, formatting information is mixed in with the text. For instance, special codes are used to specify the size and location of titles. But when manipulating the file in order, for example, to search the contents or reuse them, these codes interfere and must often be removed so that the text can be retrieved.

Document file format is formatting system dependent. When delivering a document to another person to be reworked, file format codes must be compatible for both sender and receiver. Maintaining such compatibility often creates impediments to further development of the system itself.

The *AutoLayouter* project represents an attempt to change the document preparation style in Japan. We focused first on logical contents of documents, and put layout of the documents aside temporarily.

Architecture

Figure 1 shows the architecture of *AutoLayouter*.

We put structured document files in the kernel of the system, and in the near future, we shall develop a document database to manage them. To create the document files, we developed a structured editor. The editor checks the document structure

whenever it is modified, and warns the user of invalid operations. So the document structure is always properly guaranteed.

The structured files are processed by a formatter that uses the Japanese version of \LaTeX . We also developed various device drivers including a previewer and a PostScript converter for dvi files. The whole system was developed on Panasonic's Unix workstation BE using X Window System.

Structured Editor

Problem of structure-driven editing. Mark-up languages such as SGML and \LaTeX , when used with a typical text editor, have the following advantages:

Document portability. Users may select any text editor to make documents, so they can edit the documents on any machine.

Editing efficiency. Since mark-ups are natural extensions of the process of inputting the text, users can edit marked-up documents almost as efficiently as normal documents.

Unfortunately, the use of these mark-up systems may result in the creation of documents containing fatal syntax errors. Because the syntax rules are too complicated for many users, error correction may require more time than is reasonable.

Since our aim was to develop *AutoLayouter* for users of Japanese word processors, rather than for programmers, we developed a structured editor which may be used without any knowledge of mark-up languages; and, best of all, it never produces syntax errors.

Two kinds of structured editors may be considered:

Hypertext type. This type displays the document structure and the document contents in separate windows. So the editor normally consists of two parts, one of which handles the tree structure of the document while the other is concerned with the text contents of each tree node. Although one can view the document structure easily, it is difficult to read the contents of the entire document smoothly.

Tag embedded type. Roughly speaking, this type extends the character set to include mark-up tags which specify the document structure. Usually all the tags look similar, which makes it difficult to distinguish an important tag from unimportant ones. To solve this problem, some systems use different fonts for contents with different structures or they align the important tags outside the text field.

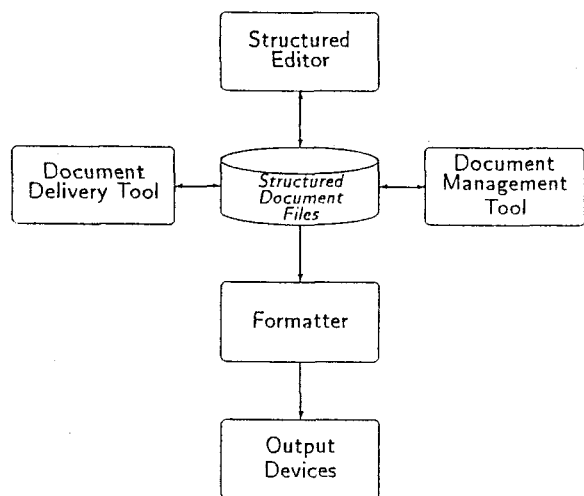


Figure 1: Architecture of *AutoLayouter*

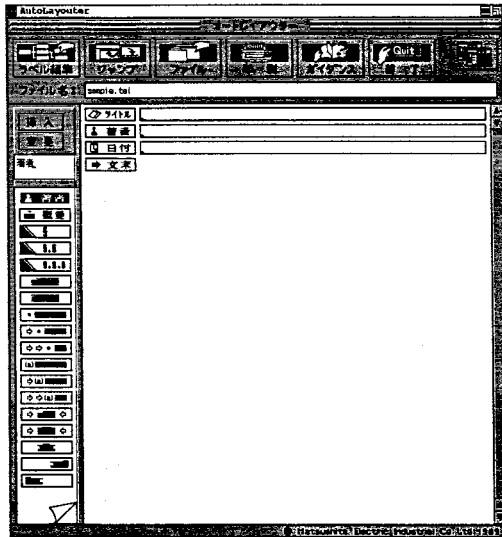


Figure 2: Snapshot of editor screen (in Japanese)

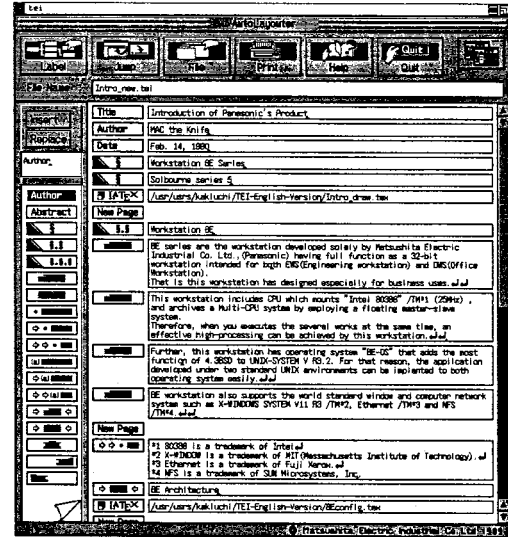


Figure 3: Snapshot of editor screen (in English)

Document structure. We first designed a framework of document structures. We took the approach that the document structure provides ways to view the contents; and that different views may make it possible to perform appropriate processing. One view may show the logical or semantic structure of the document, and another its layout structure. These two structures must be distinct. The next design requirement is that the document structure should be simple enough to be understood by users of Japanese word processors.

Based on these initial ideas, we made a hierarchy of document structures as follows:

- 1) Logical structures indicate the semantics of the text. That is, logical structures may be used not only in formatting the document but searching its contents from a database and translating between different documents. The editor displays logical structures as iconic labels, outside the text field.
- 2) Structures specifying layout information only, such as indentation or font changes, may be embedded in the text as mark-ups. We modified a text editor so that it can handle these mark-ups as normal characters.
- 3) Footnotes and references are exceptional logical structures embedded in text, but should be distinguished from the layout structures.

Snapshots of the editor screen are shown in Figure 2 and Figure 3.

Table 1: Logical structure component attributes

Classification	Attributes
Component Id.	Label Name
Rules	Max. Occurrences Min. Occurrences Contents Type
Display Mode	Priority Levels Default Lines

Structure definition. The structure definition of a document type is quite simple when compared with a full-scale SGML.

An entire document is a one-dimensional list of logical structure components such as title and section, where the logical structure is specified using restricted regular expressions. Each component of the structure has attributes shown in Table 1. In the table, Maximum and Minimum occurrences for a component give the restrictions that apply to the regular expressions for components of the structure. Contents type can be text, file name, integer, PostScript, and I^AT_EX.

Substructures, such as layout structure to indicate indentation, font changes, and so on, are specified separately from the logical structure. The substructures are embedded in the contents text of the logical components in mark-up form. Mark-ups for the layout structures have a type that is one of quasi-character, begin-end, and toggle. Quasi-character type inserts layout objects such as skips and arrows. Begin-end type locally replaces a property of the characters between the marks,

and toggle type changes a current property after the mark. Type information for the marks provides the formatter with instructions for recovering from mark-up errors.

Architecture. We show the system diagram of the structured editor in Figure 4, components of which are explained in detail below.

Structure definition parser. This parses structure definition files of the specified document type, and generates rules to be used in the structure editor.

Structure rule checker. When the user edits a structure component, the structure editor always asks the structure rule checker if the desired editing operation may produce an illegal structure. If the answer is affirmative, the structure rule checker returns the reason to the structure editor, and the editor displays a panel which explains why the desired operation is rejected.

Structure editor. Each component of the logical structure is displayed as an iconic label outside the text field, corresponding to its text contents. One can edit labels with mouse operations. For instance, to insert a label, the user clicks the insert button on the label panel, and selects the desired label from the label palette. Then the user specifies a label on the editor screen, and the

desired label is inserted. This design is similar to that of a hypertext editor, mentioned above.

Contents text editor. The contents text editor is a typical text editor customized using X Window System's text widget. Kana-to-Kanji conversion, which is managed in a front end processor, is performed in a special window, and the converted text is inserted by the text widget. Mark-ups specifying fine structures such as layout structures and references are inserted from a pop-up panel. Editing operations for the mark-ups can be restricted to maintain legal nesting of begin-end type mark-ups.

User interface manager. The editor's menus and panels may be changed to suit the type of document being edited. This is done using the on-line manual, which explains how to use labels and tags of the type of document currently being edited. We parameterized all the menus and panels which depend on the document type, and stored the interface definitions in the definition file. Each time the document type changes, the manager consults the definition file and resets the menus and panels appropriately.

Features. Our structured editor supports several editing styles:

Top-down editing. When the editing process consists of filling in the blanks in an existing form, the labels corresponding to the items on the form appear on the screen as default labels. When articles are being edited, the default structures include title, author, date, and abstract. Users may construct any document structure simply by using the mouse to insert or delete labels and to specify the field to which the new ones apply.

Bottom-up editing. Marking up a document is an example of bottom-up editing. We support replace, merge, and the splitting of labels by selecting these options from a pull down menu. Other useful forms of bottom-up editing involve cut-and-paste of text strings.

Outline editing. Outline editing is not really an editing feature, it is more like a different mode for viewing a document. When editing, users may select one of three display modes: normal, one-line, and selective. One-line mode displays only the first line of the text corresponding to each label, which allows the user to scan items throughout the document. Similarly, selective mode displays only important labels, such as sections, and their contents, where the labels that are important are so designated in the label definition file.

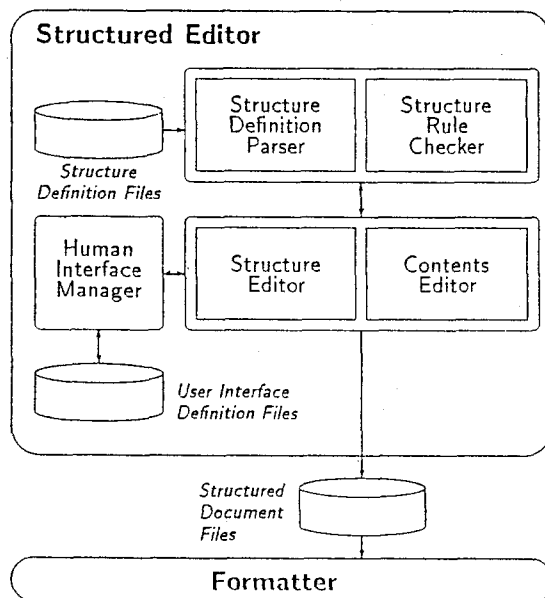


Figure 4: System diagram of the structured editor

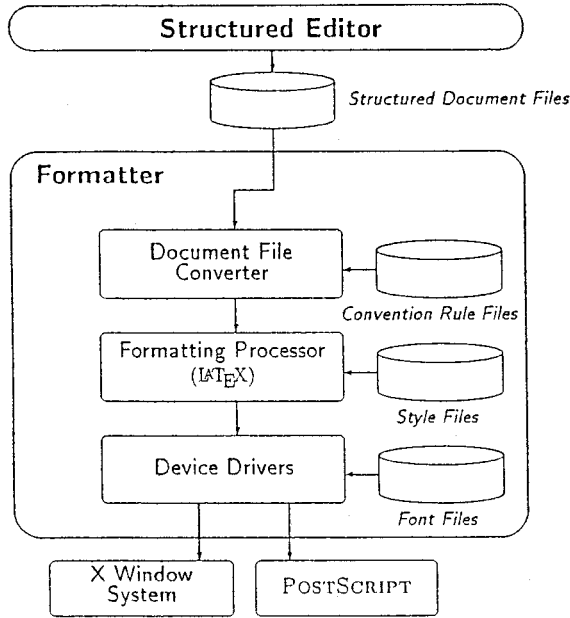


Figure 5: System diagram of the formatter

Formatter

Architecture. The system's block diagram of the formatter is shown in Figure 5. A detailed explanation of the components in the figure is as follows.

Formatting processor. We use the Japanese version of L^AT_EX as a formatting processor. T_EX provides basic language facilities, such as macro definitions, as well as a simple formatting model which recursively constructs component boxes.

L^AT_EX appends various parts and environments that are convenient for writing articles, that simplify and speed up the preparation of style files.

Structured document file. The file consists of labels and their text contents as described in the document structure section.

Document file converter. This converter reads the structured document file, and scans each label and its contents. The conversion rules are stored in the conversion rule file of the document type. As described above, each label is converted to its corresponding macro, and its contents become the arguments of the macro. When the conversion is performed, contents are checked to verify that they satisfy the conditions required for their label, for example, kind of text type (text, integer, file name, reserved word, and so on).

Figure 6 shows how the document file converter works. A typical line of the structured document file is

@BTag - Name@DArgument@E

The document file converter reads the conversion rules written in the conversion table and makes a L^AT_EX file.

Style file. On the basis of the document type, the formatter selects a corresponding style file. On the whole, conversion from logical structure to layout structure is a one-to-one mapping. One exception is represented by sequential restraints involving logical labels such as the requirement that the label "caption" must always be followed by the label "table". In such a case, style parameters will be modified. In order to allow for such conditional layout, we created a function to trace the sequence of input labels.

Device drivers. We developed the following device drivers to output dvi files to screen and printers:

Previewer. The previewer displays printer images of the formatted document on a X Window System. Since the physical resolution of the CRT differs greatly from that of current laser printers, we display the characters on the screen using fewer dots, scaling the same font used by the printer. When scaling the font, we maintain the quality of the display font by using an anti-aliasing technique, in which the gray scale of each dot is calculated in accordance with the number of black points in the sampling area of the original

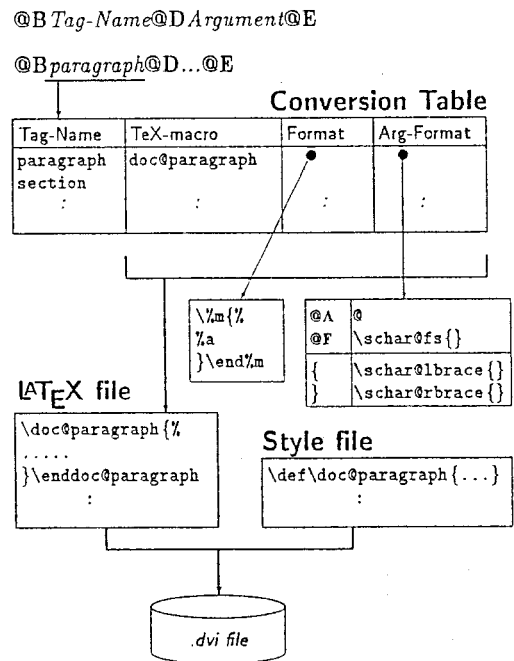


Figure 6: Document file converter

font. Furthermore, the previewer displays a small magnifying window on the preview window, so that a finer image of the formatted document is also available.

Printer drivers. In addition to supporting raster image printers, we developed a converter from a dvi file to PostScript source code. With this converter, graphics files written in Encapsulated PostScript can be imported. In order to supply various fonts to the raster image devices, mentioned above, we also developed a font manager based on Japanese outline fonts. For English fonts, we use those supplied with T_EX.

Concluding Remarks

Development of *AutoLayouter* is the first step in the construction of our new Japanese document processing system. The kernel of the system involves structured document files and, for this, we have developed an easy-to-use structured editor and a L^AT_EX-based formatter.

In the future we plan to construct a document database in which structured documents are stored. We will also need a style file editor. At this time, style files are programmed directly using L^AT_EX or T_EX, which may prevent users from changing or creating their own styles; one possibility being considered is a WYSIWYG editor to specify layout.

Acknowledgements

The authors would like to thank T. Ohno and R. Kurasawa, who developed Japanese T_EX, and Lincoln Durst for his suggestions on this paper.

Bibliography

- Adobe Systems Incorporated. "Encapsulated PostScript Files", Specification Version 2.0. Mountain View, California, 1989.
- ISO 8879. "Information Processing—Text And Office Systems—Standard Markup Language (SGML)". Geneva ISO, 1987.
- Kurasawa, Ryoichi. "Japanese T_EX at ASCII Corporation" (*in Japanese*). Proceedings of T_EX Users Group Japan, TX-97-5 (September 1987).
- Lamport, Leslie. *L^AT_EX: A Document Preparation System*. Reading, Massachusetts: Addison-Wesley, 1983.

Getting T_EXnical: Insights into T_EX Macro Writing Techniques

Amy Hendrickson

T_EXnology Inc., 57 Longwood Avenue, Brookline, MA 02146

617-738-8029 Internet: amyh@ai.mit.edu

Abstract

Most of us understand the basic form of T_EX macros but that understanding alone is often inadequate when we need to solve certain problems. We need additional insight to be able to develop methods of passing information, moving text with changed catcodes, preserving blank lines, and more. Writing a large macro package brings in a new set of issues: how to avoid bumping into implementation restrictions, e.g., constraints of hash size, string size, and others; how to make a pleasant user interface; how to make your code as concise as possible.

Some of the techniques to be discussed here include making a macro with a variable number of arguments; changing catcodes in macros, defining a macro whose argument is intentionally never used; conserving hash size by using counters instead of newifs; c_sname techniques and non-outer dynamic allocation; and table making techniques. Finally, some suggestions are included on methods to use when developing new macros.

A Quick Review of Some Important T_EX Primitives

Expandafter. `\expandafter`, a T_EX primitive often used in this article, affects the timing of macro expansion. Macro expansion is that step in T_EX's processing which changes a control sequence to whatever that control sequence is defined to represent. `\expandafter` is usually followed by calls to two macros. It expands the *first* macro following it only after it has expanded the *second*. Thus, `\expandafter` makes it possible for the first macro to process the pieces of the second macro as if the second macro were written out, not represented by a control sequence.

Here is an example: If we define `\letters` and `\lookatletters`,

```
\def\letters{xyz}
\def\lookatletters#1#2#3{First arg=#1,
  Second arg=#2, Third arg=#3 }
```

and follow `\lookatletters` with `\letters ? !`, `\lookatletters` takes the whole definition of `\letters` as the first argument, `?` as the second argument, and `!` as the third. Thus

```
\lookatletters\letters ? !
```

produces

```
First arg=xyz, Second arg=?, Third arg=!
```

But if we use `\expandafter`, `\lookatletters` will be able to process the contents of `\letters` for each argument:

```
\expandafter\lookatletters\letters ? !
```

produces

```
First arg=x, Second arg=y, Third arg=z ? !
```

String. `\string` is a T_EX primitive which causes the control sequence following it to be broken into a list of character tokens in order to print the control sequence or to process it with another macro. `\tt\string\TeX` will produce `\TeX`. (What is the `\tt` doing in there? It makes the backslash print as backslash (`\`) when it would otherwise print as a quote mark (`"`). If you are curious about this, look up `\escapechar` in *The T_EXbook*.)

C_sname. `\csname ...\endcsname` is an alternative way to define and invoke a T_EX command. Its function is the inverse of that of `\string`. `\string` takes a control sequence and turns it into tokens; `\csname... \endcsname` takes tokens and turns them into a control sequence.

Commands called for with `\csname` produce the same results as the backslash form (e.g., `\csname TeX\endcsname` and `\TeX` are equivalent) but the `\csname` construction combined with `\expandafter` allows you to build and invoke a control sequence dynamically at the time the file is processed, as opposed to knowing its name at the time the macros are written. This technique has many interesting and useful applications, as we will soon see.

If and ifx. Since both `\if` and `\ifx` are conditionals used to compare tokens, the T_EX user may well wonder when to use `\if` and when to use `\ifx`. When we understand how each of these conditionals works, we may conclude that the answer is to use `\if` *only when comparing single tokens* and to use it with care.

How ‘if’ works. `\if` expands whatever immediately follows it until it arrives at two unexpandable tokens. It then compares them to see if their charcodes match. This test is useful to see if a given letter is upper- or lower-case and in some other instances where we need to test a single token.

The two tokens that are compared are the first that appear after `\if`, *even if they are both* found inside the same macro following it. Understanding that principle makes sense of these samples which would otherwise be mystifying.

```
\def\aa{ab}
\def\bb{ab}
\if\aa\bb
```

tests false, because `\if` expands `\aa` and compares ‘a’ with ‘b’. Whereas

```
\def\aa{aa}
\def\bb{bb}
\if\aa\bb
```

tests true, because T_EX compares ‘a’ with ‘a’ in the macro `\aa`. `\if` doesn’t process `\bb` since it has already found two unexpandable tokens and in this case will cause the letters ‘bb’ to print since the conditional is set to true and `\bb` is found in the true part of the conditional.

There is another problem to consider. Since `\if` expands a control sequence to its bottom level, meaning every control sequence that is found in the definition of a command being expanded will itself also be expanded, it may generate an error message if a control sequence is expanded that contains an `@` in its name.

This problem arises because Plain T_EX commonly includes `@` as part of macro names, with the catcode of `@` set to that of a letter. The catcode of `@` is set to ‘other’ in normal text so that when a Plain

T_EX command of this sort is expanded in text the `@` is no longer understood as a letter, and T_EX will give the user an error message about an undefined control sequence. For example,

```
\if\footnote X Yes\else No\fi
```

produces this error message:

```
! Undefined control sequence.
\footnote #1->\let \@sf
\empty \ifhmode...
```

How ‘ifx’ works. `\ifx`, on the other hand, will not have this problem since it only expands to the first level of macro expansion. If `\dog` is defined by `\def\dog{\cat}`, for instance, `\ifx` will expand `\dog` as far as `\cat` but will not expand `\cat` to use its definition.

This means that when we want to compare control sequences, and to supply one control sequence as an argument to a macro, we can use the `\ifx` conditional to look at the name of the macro supplied without having to worry about macros that may be contained in its definition.

For example, we can define `\def\aster{*}` so that we can use it to compare with another macro. Inside the macro where we want to make the comparison, we can write

```
\def\sample#1{\def\one{#1}\ifx\one\aster...
```

making both the argument to `\sample` and `*` be defined as macros.

When `\sample` is used, `\ifx` causes only one level of expansion. If the argument given to `\sample` is `\footnote`, as in the `\if` example above, `\one` will be defined as `\def\one{\footnote}`. `\ifx` will expand `\one` to find ‘`\footnote`’ but will not expand it any further, and will not give an error message.

Another reason to use `\ifx` to compare control sequences is that `\ifx` will pick up both control sequences following it and compare them. When we try the same samples with `\ifx` that we did with `\if`, we will find that we get results opposite to those we got with `\if`—and we will get the results we would want when comparing control sequences.

```
\def\aa{ab}
\def\bb{ab}
\ifx\aa\bb
```

tests true, because `\aa` and `\bb` match each other in their first level of expansion, whereas

```
\def\aa{aa}
\def\bb{bb}
\ifx\aa\bb
```

tests false because the first level of expansion of `\aa` and `\bb` do not match.

Picking Up Information

Defining a macro that will pick up and process a variable number of arguments. There are many instances where you might want to allow a variable number of arguments. Table macros are one such case, in which the user might supply the width of each column as an argument, and the number of columns may well vary from table to table. A table alignment macro that determines whether each column in the table should be aligned to the right, left, or center, is another case where processing a variable number of arguments is necessary.

There is a general method for constructing a macro that will accommodate a variable number of arguments. This method is to pick up all the arguments as one unit and then take that unit apart as a second step. For example, `\table 1in 2.3in 4in*` can be the command to start a table using dimensions to specify the width of each column. When `\table` is defined as `\def\table#1*{...}` we can pick up all the dimensions as the first argument, since the first argument ends with `*`, then use a second macro to process each dimension as its argument. The second macro will call itself again after each dimension is processed until all the dimensions have been used.

Here, in a sample macro, we define `\pickup` as `\def\pickup#1*{...}` to use it to pick up everything between `\pickup` and the `*` as its first argument. Then we use `\expandafter` to allow `\lookatarg` to process the contents of the first argument.

```
\def\pickup#1*{\expandafter\lookatarg#1*}
```

The definition of `\lookatarg` contains a looping mechanism: It is a conditional that tests to see if its argument is equal to `'*`'. It will keep calling itself (recursing) until its argument is `*`. It calls itself by redefining the command `\go` within the conditional, and calling for `\go` outside the conditional. (`\go` must be placed outside the conditional. If it were to be used inside the conditional it would take the `\else` or the `\fi` as its argument and massive confusion would result.) When `\lookatarg` sees `'*`' as the argument, it will define `\go` as `\relax` and thus will not call itself again.

First we define `\aster` so that we have a command to use with `\ifx` to compare with the argument of `\lookatarg`:

```
\def\aster{*}
```

Now we can compare the argument of `\lookatarg` with `\aster`. Thus, with

```
\def\lookatarg#1{\def\one{#1}
\ifx\one\aster\let\go\relax
\else Do Something \let\go\lookatarg
\fi\go}
```

if we use the `\pickup` macro as follows

```
\pickup abc def*
```

the results would be:

```
Do Something Do Something Do Something Do
Something Do Something Do Something
```

`\lookatarg` has been invoked 6 times since it picked up 6 tokens before it found the `*`. We can substitute some other command for 'Do Something' and build a more useful macro.

Here are two applications of the technique demonstrated in `\lookatarg`; a macro to underline every word in a given section of text, and a macro to process a given section of text to imitate the small caps font.

First, we define `\underlinewords`, which picks up the whole body of text to be underlined:

```
\long\def\underlinewords #1*{
\def\wstuff{#1 } \leavevmode
\expandafter\uword\wstuff * }
```

Here `\leavevmode` asks T_EX to go into horizontal mode. Since each word will be placed in a box, we need this command to prevent the boxes from stacking vertically, as they would in vertical mode.

Now we define `\ulword` which will unpack the text picked up, word by word, put each word in a box, and provide a horizontal rule under each:

```
\long\def\ulword#1 {\def\one{#1}%
\ifx\one\aster\let\go\relax
\else\vtop{\hbox{\strut#1}\hrule \relax}
\let\go\ulword
\fi\go}
```

The space given after the argument number in the parameter field will allow us to pick up one word at a time, since the collection of the argument will be completed only when `\ulword` sees a space. Here we use `\underlinewords`:

```
\underlinewords
non-outer dynamic allocation*
```

which results in:

```
non-outer dynamic allocation
```

The macro `\fakesc` is another construction using this technique. It lets you set text in large and small caps, imitating the 'small caps' font. Its arguments are, in order, the font for the larger letters, the font for the smaller letters, and the text that is to be set in small caps.

When we use `\fakesc` we need to declare the two fonts to be used:

```
\font\big=cmr10
\font\med=cmr8
```

and then

```
\fakesc\big\med Here are Some Words to be
Small Capped. NASA, Numbers, 1990*
```

will result in:

```
HERE ARE SOME WORDS TO BE SMALL
CAPPED. NASA, NUMBERS, 1990
```

The macro starts by defining the two fonts and the text to be processed; there is a space after #3 in `\def\stuff{#3 }` because `\pickupnewword` needs a space to complete its argument when the last word is found as `\stuff` is expanded:

```
\def\fakesc#1#2#3*{\def\bigscfont{#1}%
\def\smcfont{#2}\def\stuff{#3 }%
\expandafter\pickupnewword\stuff *}
```

`\pickupnewword` picks up one word at a time, in order to preserve the space between words. If we just asked `\pickupnewlett` to process the entire third argument of `\fakesc`, the space between words would be thrown away as irrelevant space appearing before the next character being looked for as the argument of `\pickupnewlett`. Here, then, is the definition of `\pickupnewword`:

```
\long\def\pickupnewword#1 {%
\expandafter\pickupnewlett#1\relax}
```

Once `\pickupnewword` has picked up the word, `\pickupnewlett` is used to test each letter to determine whether it should be capitalized. If so, it uses the larger size font; otherwise, the smaller. `\pickupnewlett` tests to see if the argument is uppercase by using the first argument to define `\letter`, `\def\letter{#1}`, and then defines `\ucletter` in an uppercase environment.

```
\uppercase{\def\ucletter{#1}...}
```

Now it uses the `\if` conditional to compare `\letter` and `\ucletter`. If they match `\pickupnewlett` makes the current letter or number be printed in the larger font; otherwise the smaller font is used. Note that we can use the `\if` conditional here since we are only comparing single letters. So, finally, the definition of `\pickupnewlett`:

```
\def\pickupnewlett#1{\def\letter{#1}%
\if\letter*\unskip\let\go\relax
\else%
\if\letter\relax{\bigscfont\ }%
\let\go\pickupnewword
\else\uppercase{\def\ucletter{#1}%
\if\letter\ucletter%
{\bigscfont#1}\else{\smcfont#1}
\fi}%
\let\go\pickupnewlett
\fi\fi\go}
```

When to pick up text as an argument, and when to pick up text in a box. The correct timing of catcode changes is an issue of concern to

the macro writer. Picking up text as an argument will usually be the right way to provide information for the macro, but will fail if you need to change catcodes, since catcodes are irrevocably assigned at the time `TEX` reads each character. Thus, by the time `TEX` has picked up an argument, the catcode of all the tokens in the argument are set, and no amount of fiddling with the argument within a macro will change this.

Even a catcode change asked for in the body of an argument will not effect a catcode change because the catcodes of the tokens will already be set by the time `TEX` expands the request for the catcode change.

There are two ways to solve this problem. In simple cases, one can build a macro containing the desired catcode changes and then invoke a second macro within the first, i.e.,

```
\def\changeecat{\bgroup
\obeylines\pickupchanged}
\def\pickupchanged#1\endchange{%
\setbox0=\vtop{\hsize=1in#1}%
\centerline{xx\vtop{\unvbox0}yy}%
\egroup}
```

In `\changeecat` a catcode change is produced by `\obeylines` which changes the catcode of the end-of-line character, `^M`, to 13 ('active') so that it can be defined as `\par`. Once that catcode change is made, `\pickupchanged` is invoked. Its argument has the end-of-line character set to category 13 at the time the argument is picked up. Notice the `\bgroup` command in `\changeecat` is matched with the `\egroup` command in `\pickupchanged` to confine the catcode change.

Used:

```
\changeecat
What
Happens
Here?
\endchange
```

```
xx      What      yy
        Happens
        Here?
```

But, though this example will work for a catcode change set within the `\changeecat` macro it will not allow `\changeecat ... \endchange` to pick up an argument that contains a catcode change, for instance, an argument containing macros to produce verbatim text, as we could do with the following technique.

The two-part macro defined below will build a box, starting it in `\pickupcat` with `\setbox0\vtop\bgroup`. Any material found between it and `\endpickup` will be expanded, and finally the box will be

completed with the `\egroup` found in `\endpickup`, a construction that allows a catcode change between the first and second part of the macro.

```
\def\pickupcat{\global\setbox0
\top\bgrouphsize=1in\obeylines}
\def\endpickup{\egroup%
\centerline{XXX\top{\unvbox0}YYY}}
```

`\pickupcat... \endpickup` is shown using a previously defined verbatim environment, `\beginverb... \endverb`, to pick up and move verbatim text in a box. Once we see this principle, we can see that a revision of this macro would allow us to place verbatim text in a figure environment, or in a table, or in another environment where it would normally be difficult to introduce material with changed catcodes. For one example:

```
\pickupcat
\beginverb
  Test of
%$_~#@\
  Verbatim

  text.
\endverb
\endpickup
```

produces:

```
XXX  Test of  YYY
      %$_~#@\
      Verbatim

      text.
```

Looking ahead at end of line to preserve blank lines. Since T_EX normally ignores blank lines between paragraphs and in some cases we might want to maintain blank lines, we need to develop a way to test for blank lines and provide vertical space when one is present. In this case, we are not interested in picking up text but in picking up information. What comes after each end-of-line character?

As previously mentioned, `\obeylines` changes the end-of-line character, `^^M` to `\par`. You can define `^^M` to do other things as well. For instance, you can define it to be a macro that will supply a `baselineskip` when the next line is blank or a `lineskip` when the next line is not.

In this example, `^^M` will be defined as `\lineending`, a macro that includes `\futurelet` to look ahead in the text. If the character that it sees is *itself* (`\lineending`), the next line is blank, since there is nothing from one end-of-line character to the next one. The macro `\looker` will then supply a `baselineskip`. If it does not see itself, indicating that the next line is not blank, `\looker` will supply a `lineskip`:

```
\def\looker{%
\ifx\next\lineending%
\vskip\baselineskip\obeyspaces\noindent%
\else%
\ifx\next\endgroup\else%
\vskip\lineskip\obeyspaces\noindent%
\fi\fi}
```

```
\def\lineending{\futurelet\next\looker}
```

```
{\obeylines
\gdef\saveblanklines{\bgrouphsize=1in\obeylines%
\let^^M=\lineending}}
```

```
\def\endsavelines{\egroup}
```

Example:

```
\saveblanklines
Here is

a blank line,
and a non-blank line.
\endsavelines
```

which produces

```
Here is

a blank line,
and a non-blank line.
```

Passing Information: When Counters Can be More Advantageous than Newif's

Hash size, the size of that part of T_EX's memory in which it stores control sequence names, is usually not something about which the macro writer has to be concerned. When building a large macro package, however, hash size can be exceeded, making the number of control sequences defined an important issue. One way to economize on the number of definitions in a package is to use counters to pass information rather than using `\newifs`.

When the number of control sequences is not important, `\newif` can be used to create a conditional. This conditional can then be set to true or false in one macro, and tested to see if it is true or false in another as a way of passing information from one macro to another.

However, every time a `\newif` declaration is used, three new definitions are generated. If saving hash size is an issue, we can use `\newcount` instead, and only one new definition is generated.

We can use `\newcount` to allocate a counter and assign it a name, e.g., `\newcount\testcounter`. Then, instead of setting a conditional to true or false, i.e., `\global\titletrue`, and testing for it, i.e., `\iftitle ... \else ... \fi` we can test for the value of the counter. For example,

```
\global\testcounter=1
```

could be the equivalent of `\global\titletrue`. We can then use this test:

```
\ifnum\testcounter=1 ...\else ...\fi
```

Counters have the additional advantage of allowing you to test for a range of numbers, i.e.,

```
\ifnum\testcounter>1 ...\else ...\fi
```

so you can write more compact code when testing for a number of options.

For instance, if we were to write a macro that allowed the user to choose to fill a box by pushing the text to the left, center, or right, we could assign a numerical value to each of the options. If we assigned a value to `\testnum` according to the plan, `left=0`, `center=1`, `right=2`, we could test for a range of numbers when another macro was determining which way to fill the box. The test could look like this:

```
\hbox to\hsize{\ifnum\testnum<1
  %% if text is to be pushed to the left
\else
  %% if text is to be either centered or
  %% pushed to the right, do \hfill
  \hfill\fi
  <text>
\ifnum\testnum>1
  %% if text is to be pushed to the
  %% right, don't do \hfill
\else
  %% or text is either centered
  %% or pushed to the left
  \hfill\fi}
```

This same principle can be used in more complicated cases as a way of reducing great masses of nested conditionals to a test of the range of the value of a particular counter.

Methods of Conserving Hash Space

As mentioned earlier, using counters to pass information rather than `\newif\thinspace s` is one way to help prevent the hash size from being exceeded. Here are some others.

Input separate macro files on demand. To reduce the number of macros in a macro file, break up the complete macro package into a general macro file and a number of secondary macro files. Within the general macro file definitions can be made that read in the secondary files only when the user calls for a macro for a particular function. For instance, a file containing all the table macros will only be read in if the user uses the general table macro. This principle can be used for listing macros, indexing macros, and any other sort of macro that will not necessarily be used for every document.

Using non-outer dynamic allocation. Dynamic allocation is the way macro writers are able to access the next available number of a dimension, box, or counter at TeX processing time and assign a symbolic name to it. `\newdimen`, `\newbox` and `\newcount` are the commands that allocate these numbers dynamically. It is safer to use dynamic allocation in a macro than to use a particular numbered box, counter, or dimension, since it prevents accidental reallocation.

Unfortunately, all of the commands in this useful set are `\outer`, which means that they cannot be declared *within* a macro. By making these dynamic allocation macros non-outer, we can then include them inside macros and only declare new counters or new boxes or new dimensions when they are needed.

Here is how to make these commands non-outer. Simply copy the original definition, supply a new control sequence name and define them without the `\outer` that originally preceded the definition. For example, the definition of `\newbox` was originally

```
\outer\def\newbox{%
  \alloc@4\box\chardef\insc@unt}
```

Here are the new versions, `\nonouternewbox`, etc.:

```
{\catcode'\@=11
\gdef\nonouternewbox{%
  \alloc@4\box\chardef\insc@unt}
\gdef\nonouternewdimen{%
  \alloc@1\dimen\dimendef\insc@unt}
\gdef\nonouternewcount{%
  \alloc@0\count\countdef\insc@unt}
\catcode'\@=12}
```

In the next section we will see these non-outer commands being used in a table macro, only making named boxes or dimension when needed. Macro writers may find other uses for this technique as well.

Fun with Cname

One of the really useful features of `\cname` is that control sequences can be expanded within the body of the `\cname... \endcname` construction:

```
\expandafter
\def\cname\testmacro\endcname{%
  <definition>...}
```

Counters can be used:

```
\expandafter
\def\cname\testcounter\endcname{%
  <definition>...}
```

Counters with roman numerals can be used:

```
\expandafter
\def\csname\romannumeral\testcounter%
\endcsname{<definition>...}
```

You can even make a definition out of numbers or other symbols that ordinarily are not allowed for a control sequence name:

```
\expandafter\def\csname 123&#\endcsname{<definition>...}
```

The only thing to remember here, is that any control sequence made with `\csname` that contains anything other than letters must be invoked as well as defined with `\csname`: `\csname 123&#\endcsname` is the way to use this macro.

Using `\csname` with `\expandafter` makes it possible to do all sorts of things that would not otherwise be possible. Some examples will be found in the following text.

Macros that define new macros using data supplied. One example involves having one macro define another macro where the name of the second macro depends on data supplied in the text.

In the example constructed below, `\usearg` takes the first two words as arguments #1 and #2, reverses their order and uses them to make a control sequence name. This control sequence is then defined to be the complete name and address of the person whose name was used to form the control sequence name.

The order of the name is reversed so that the names of the new macros can be sent to an auxiliary file and be sorted alphabetically. The Appendix illustrates how a more elaborate form of this set of macros may be used to manipulate mailing lists.

`\obeylines` below changes the catcode of the end-of-line character (`^^M`) to 13 so it can be used as a argument delimiter in the definition of `\usearg`; `\obeylines` also defines `^^M` as `\par` so that every line ending seen on the screen is maintained when the text is printed:

```
{\obeylines
\def\usearg#1 #2^^M#3^^M^^M{%
\expandafter\gdef\csname #2#1\endcsname
{#1 #2\par #3}}
```

With this definition,

```
\usearg George Smith
21 Maple Street
Ogden, Utah 68709

\SmithGeorge
}
```

produces

```
George Smith
21 Maple Street
Ogden, Utah 68709
```

The Appendix contains a macro subsystem for processing and sorting address labels; it demonstrates this technique and many of the others discussed in this paper.

Macros that define new macros using a counter. Here is another use of `\csname`: In this case, it is used to define a macro that will itself define a new macro every time it is used, with the name of the new macro determined by a counter whose number is represented with roman numerals.

This can be used to construct a series of macros that expand into areas of text to be reprocessed at the end of a document. For instance, this technique could be used to produce a set of slides from given portions of the text of a document.

In the following example, each time `\testthis` is called it will define a new control sequence. It makes the name of the new control sequence by advancing `\testcounter` which is operated on by `\romannumeral` to produce a new set of letters. These letters will appear in the name of the new macro.

Each control sequence is then sent to an auxiliary file, embedded in code to make the slide. (The slide formatting code is represented here as `[[[]]]`). At the end of the document the auxiliary file containing all the definitions can be input, to produce a set of slides.

```
\newcount\testcounter
\testcounter=501
%% just to start with a large
%% number to make into roman numerals
\newwrite\sendtoaux
\immediate\openout\sendtoaux
\jobname.aux %% opening a file to write to
\def\testthis#1{%
\global\advance\testcounter by1
\expandafter\gdef\csname%
\romannumeral\testcounter XYZ\endcsname{%
[[[#1]]]}
\immediate\write\sendtoaux{%
\noexpand\csname\romannumeral\testcounter
XYZ\noexpand\endcsname}}
```

Here is an example of `\testthis` being used:

```
\testthis{This is the first bit of text...}
\testthis{This is the second...}
```

The code above writes the following lines into the .aux file:

```
\csname diiXYZ\endcsname
\csname diiiXYZ\endcsname
```

and when the .aux file is input, these commands produce

```
[[[This is the first bit of text...]]] [[[This is the
second...]]]
```

Designing generic code with `\csname`. Another really important use for `\csname` constructions is a way of making compact code for a section of a macro that repeats many times with a small variation each time. Table macros are often examples of this kind, since they tend to have repeating sections, one for each column.

We consider below some parts of a set of macros for table construction showing several ways that `\csname` can be used. The form adopted for the `\halign` command line, using a `&` immediately after the `\halign{`, will allow the specifications for this column to be used for each column in the table.

Using `\csname` with a counter allows this set of commands to be defined only once. Each new column entry will cause the `\colcount` counter to advance making the otherwise similar column definition use a new counter value inside the `\csname... \endcsname` constructions.

Non-outer dynamic allocation is used to name only those counters, dimensions, or boxes that are needed when the table is made up. A new set is declared for each column of the table. Since we don't need to guess ahead of time how many columns are going to be used, only those dimensions or boxes that are needed will be declared. In addition, `\ifdefined` (below) tests to see if the particular set of dimensions and boxes has been used in a previous table, and will only declare a new set if they have not been defined previously.

`\xtab` and `\dtab` involve counters only, so they can be used later in a `\csname... \endcsname` construction where it doesn't matter if the expansion will produce numbers as part of the control sequence. `\gtab` and `\vtab`, on the other hand, need to be used as ordinary control sequences which is the reason for the `\romannumeral` command that will produce letters instead of numbers when the `\gtab` and `\vtab` are expanded.

`\asizetab` and `\finishasizetab` will use these boxes and counters to actually set the table entries.

Here, finally, are some definitions for table construction:

```
\def\multipagetable{\global\firstcoltrue
\halign\bgroup%
&\global\advance\colcount by1\relax%
\ifdefined{\the\colcount tab}{-}{%
\edef\xtab{\expandafter\csname
\the\colcount tab\endcsname}%
\edef\dtab{\expandafter\csname
\the\colcount tabwide\endcsname}%
\edef\gtab{\expandafter\csname
\romannumeral\colcount
gapped\endcsname}%
\edef\vtab{\expandafter\csname
\romannumeral\colcount
```

```
vlinewd\endcsname}%
\expandafter\nonouternewdimen\vtab%
\expandafter\nonouternewbox\xtab%
\expandafter\nonouternewdimen\dtab%
\expandafter\nonouternewcount\gtab%
}%
\ifdefined{align\the\colcount tab}{-}{%
\edef\atab{\expandafter\csname
align\the\colcount tab\endcsname}%
\expandafter\nonouternewcount\atab}%
\asizetab##\finishasizetab\cr}
```

A `\csname... \endcsname` construction defined using one counter can be invoked *using a different counter*, if that proves useful. Another part of the code for multipage tables uses a second counter to invoke macros defining boxes containing the column heads, used when the table continues over page breaks. Even though the original definition used `\colcount` as the counter to name the boxes, `\contcolcount`, another counter, can be used in another macro to invoke the same definition. When \TeX expands `\csname... \endcsname` construction it produces a number as the replacement for the counter, so the name of the counter used doesn't affect the result. This might be helpful in cases where you don't want to change the value of one counter, but still wish to use a `\csname` construction that contains it.

Tips on Table Macros

`\everycr` is a \TeX primitive for a token list. It functions similarly to `\everypar` or `\everymath` in that its definition will be used every time the named environment is present, in this case after every `\cr`. By setting `\everycr` equal to some definition we can insert a set of commands after every line in a table, since every line will end with a `\cr`. A simple example is this:

```
\everycr={\noalign{\hrule}}
```

which will insert a horizontal rule automatically after every `\cr` in the table. Once this possibility is discovered, the macro writer may realize that there are many other things that can be done with `\everycr`, such as including a set of conditionals that will call for horizontal lines with breaks in them, double horizontal lines, thicker horizontal lines, thicker lines under some columns but not under others, and so on.

You can include a counter which is advanced every time `\everycr` is called, and use that counter to determine how many lines have been used in the table, in order to stop and restart the table, making it possible to have a table that will continue

for hundreds of pages without T_EX running out of memory.

Moreover, you can call for a small vertical skip in the `\everycr` definition which will allow the table to break over pages. If you use the following construction, your table will break over pages and a horizontal line will appear both at the bottom of the previous page and at the top of the new page, without the user having to know ahead of time where the table will break.

```
\everycr={\noalign{\hrule\vskip-1sp\hrule}}
```

When Doing Nothing is Helpful

The usual form of a macro with an argument is (in its most basic form) `\def\example#1{#1}`. There are cases in which **not using** the argument can be helpful when you want to get rid of something: `\def\example#1{}`.

You can use this principle to prevent large sections of text from being processed by T_EX.

```
\long\def\ignorethis#1\endignore{}
```

Thus

```
\ignorethis
Here is some text that will be ignored...
\endignore
This is where TeX starts printing text...
```

will produce

```
This is where TeX starts printing text...
```

You might want to use this macro in the process of debugging a document you are working on. All text between `\ignorethis` and `\endignore` will be ignored, making it possible for T_EX to print only the part of the document in which you are interested. T_EX will run out of memory after about 6 pages of text is picked up by the `\ignorethis` macro, depending on the implementation of T_EX being used, but if you want to ignore more than 6 pages of text you can end the first `\ignorethis` with `\endignore` and enter a second `\ignorethis` `...\endignore`.

A slight improvement, however, is needed to prevent T_EX from complaining if an `\outer` command is found in the argument of `\ignorethis`. This is the error message which we would like to avoid:

```
! Forbidden control sequence found
while scanning use of \ignorethis.
```

We can avoid it by changing the catcode of the backslash to be that of a letter. Now there will be no commands processed until `\ignorethis` encounters `\endignore` and the catcode changes are turned off.

```
\long\gdef\ignorethis{\bgroup
\catcode'\=12 \catcode'\^=12 \finish}

{\catcode'\|=0 |catcode'\|=12
|long|gdef|finish#1\endignore{|egroup}%
}
```

Note that here, too, `#1` is never used in the replacement part of the macro.

Getting rid of backslashes. Here is another example of an argument that is thrown away:

```
\def\stripbackslash#1#2*{\def\one{#2}}
```

which only uses the second argument, throwing away the first argument, in this case stripping away a backslash from a control sequence supplied by the user. `\stripbackslash` can then be used in another macro which needs a control sequence without its backslash to work correctly, for instance:

```
\def\newdef#1{\expandafter
\stripbackslash\string#1* \one}
```

When this is used,

```
\newdef\testmacro
produces
testmacro
```

Instead of simply printing the control sequence without the backslash, `\newdef` can be rewritten to test to see if a given macro has already been defined. In this example, `\newdef` tests to see if the expansion of the control sequence `\curname\one\endcurname`, (where `\one`, was defined by `\stripbackslash` to be the control sequence supplied by the user minus its backslash) is equal to `\relax`. This takes advantage of the T_EX convention that a previously undefined control sequence invoked in a `\curname...\endcurname` environment will be understood to be equal to `\relax`, whereas an already defined control sequence will not:

```
\def\newdef#1{%
\expandafter\stripbackslash\string#1*
%% \stripbackslash defines \one
\expandafter
\ifx\curname\one\endcurname\relax
%% \one is expanded to be the
%% control sequence the user supplied
%% minus the backslash.
%% If curname construction equals
%% \relax, do nothing
\else %% Else, give error message:
{\tt Sorry, \string#1 has already been
defined. Please supply a new name.}
\fi}
```

In the test below, notice that we do not get an error message for `\cactus` which hasn't been previously defined, but we do get a message for T_EX, which is defined:

```
\newdef\TeX
\newdef\cactus
```

produces

Sorry, \TeX has already been defined.
Please supply a new name.

Not using boxes. Similar to not using arguments, there are times when setting a box and then **not** using it can be useful.

When writing a macro to make text to wrap around a given figure, we might want to use a test box to put a given amount of text in, say, a paragraph, which has been picked up as an argument to a macro. We can then measure the box to see if it will exceed the depth of the figure. If it does not, the box can be used as it is, but if it does, the box can be ignored and the argument re-used, with changed `\hangindent` and `\hangafter`, to allow the text to fit around the figure neatly. This works because text picked up as an argument to a macro does not yet have its glue set, so it can accommodate different line widths.

Another use for a box that is never printed is to use it as a container in which to expand a macro having symbols in the parameter field. For example, if the macro `\splittocentry` is defined by

```
\def\splittocentry#1-#2-#3{\gdef\one{#1}
\gdef\two{#2}\gdef\three{#3}}
```

we can use it in a another macro to process an argument which may or may not include the hyphens, i.e.,

```
\setbox0=\hbox{\expandafter
\splittocentry#2-{}-{}}
```

The hyphens that are necessary to complete the use of `\splittocentry` are supplied in the box but they will not print if the replacement for #2 turns out to supply the hyphens already. Since `\one`, `\two`, and `\three` are globally defined (`\gdef`), their definition will be understood outside the box.

Some General Macro Writing Tips

There are several commands that can make the process of macro writing easier.

`\show` is a T_EX primitive that will cause the definition of the macro it precedes to appear on your screen when you run T_EX on a file that contains it. `\show\samplemacro` will cause the definition of `\samplemacro` to be appear on your screen, for example. `\show` can be temporarily included inside a macro to let you see what is being picked up as arguments. For instance, if

```
\def\test#1,#2{\def\one{#1}\def\two{#2}
\show\one\show\two...}
```

then

```
\test some, stuff
```

will help you see what is being picked up as argument #1 and #2. In this example the results are obvious, but there are more complicated situations. For example, when one macro is looking at the contents of another macro, a test like this can quickly help you understand what T_EX sees when it picks up an argument, a helpful debugging tool. It also has the advantage of giving you information at the time you T_EX your file, saving you the steps of either previewing or printing the .dvi file.

`\show` will also send the definition of the macro that it precedes to the .log file, a feature which you can take advantage of when you are interested in redefining a Plain T_EX macro. If you write `\show\raggedright`, for example, in a test file and run T_EX on that file, the definition of `\raggedright` will appear in the .log file. You can then move those lines of code from your .log file to your macro file and you will have saved yourself the trouble of looking up the command in *The T_EXbook* and copying it into your file. Now you are ready to make changes to the original macro.

A related command, `\showthe`, will give you the current value of a token list, like `\everypar`. Including `\showthe\everypar` in a test file can tell you what T_EX sees as the current value of `\everypar` at that point in the file. You can also use `\showthe` to get the current value of a counter or dimension. You may want to include a `\showthe` temporarily in a macro you are developing, similarly to `\show`, as a debugging tool.

Finally, using `\message` in a conditional while working on a macro can give you helpful information. You could put this code in a headline, for instance, to be able to see the state of a particular conditional in the headline,

```
\headline={...%
\iftitle
\message{SEES TITLE, WIN}
\else
\message{NO TITLE, LOSE}
\fi...}
```

or include a similar construction in the body of a macro while you are testing it. When you T_EX the file you can quickly see if you are getting the results you were expecting.

Appendix

Code to Alphabetize an Address List

These macros demonstrate many of the techniques discussed in this paper. The macros process an existing an address list by taking the first line of each address, re-ordering the name with last name first, then turning the name into a control sequence which is sent to an auxiliary file. The user must alphabetically sort the auxiliary file. The resulting sorted file is then input back into the originating file and the whole address list will be transposed and printed in alphabetical order.

The user enters `\alphalist` at the beginning of an address list, and a blank line and `\endalphalist` at the end. `\alphalist` picks up the name, then makes a macro using the name (last name first) as the control sequence. This control sequence is sent to auxiliary file with the same name as the originating file and with an `.alf` extension. The file `filename.alf` must be sorted to produce `filename.srt`, using a sort routine on the user's system. If DOS, write

```
sort < filename.alf > filename.srt
```

`\endalphalist` checks to see if `filename.srt` exists, and if so, will `\input filename.srt`. The sorted list of control sequences will produce an alphabetized address list.

First we name dimensions and counters and set them to arbitrary sizes.

```
\newdimen\heightofentry \heightofentry=.75in
\newdimen\widthofentry \widthofentry=.3\hsize
\newcount\namenum
```

`\alphalist` makes every new paragraph start with the command `\look`. `\obeylines` will maintain the same line endings as seen on the screen.

```
\def\alphalist{\bgroup\obeylines
\global\everypar={\look}}
```

First we discuss the definition of `\look`, then we will consider the macros used in its definition.

`\look` picks up the entire name. It then defines it as `\test`. `\test` is placed in `\box0` and expanded after `\throwawayjr` which defines `\fullname`. Then `\fullname` is expanded after `\takeapart` to define `\nameinrev`. `\nameinrev` is the name in reverse order; it is used as the name of a control sequence that defines the entire name and address. `\nameinrev` in a `csname` environment is also sent to an auxiliary file so that it can be sorted alphabetically. Here is the definition of `\look`:

```
{\obeylines
\gdef\look#1^^M#2^^M^^M{\def\test{#1}
\setbox0=\hbox{%
```

```
\expandafter\throwawayjr\test, {}}
\global\namenum=0
\expandafter\takeapart\fullname
\obeylines
\everypar={}
\expandafter%
\gdef\csname\nameinrev\endcsname{%
\vtop to\heightofentry{\parindent=Opt
\vfll\hsize=\widthofentry
#1
#2
\vfll}}%
\immediate\write\alphafile%
{\noexpand\csname\nameinrev%
\noexpand\endcsname}%
\global\everypar={\look}}
}
```

Now we consider the commands used in the definition of `\look`.

To make the last name appear first in the command sent to the auxiliary file, we count the number of parts to the name ("Mr. R. G. Greenberg" has four parts, for example) and use `\ifcase` to select the correct order. After `\nameinrev` (for 'name in reverse order') is defined, it will then be used in the `\look` macro to create a control sequence by being expanded within a `csname`.

```
\def\makerightdef{\ifcase\namenum\or
\or\gdef\nameinrev{\one}
\or\gdef\nameinrev{\two\one}
\or\gdef\nameinrev{\three\one\two}
\or\gdef\nameinrev{\four\one\two\three}
\or\gdef\nameinrev{\five\one\two\three}
\four}
\fi}
```

`\makedef` gives a control sequence name to the argument of `\takeapart` according to the number of times `\takeapart` is invoked:

```
\def\makedef#1{\ifcase\namenum
\or\gdef\one{#1}
\or\gdef\two{#1}
\or\gdef\three{#1}
\or\gdef\four{#1}
\or\gdef\five{#1}
\fi}
```

In order to make an `\ifx` comparison, we set

```
\def\aster{*}
```

`\takeapart` loops until it sees the `*`, which will be supplied in the `\throwawayjr` macro:

```
\def\takeapart#1 {%
\global\advance\namenum by1
\def\onex{#1}
\makedef{#1}
\ifx\onex\aster
\makerightdef\let\go\relax
\else
\let\go\takeapart
\fi\go}
```

We want to alphabetize according to the last name, and not mistakenly use 'Jr.' as the last name. The first argument ends when `\throwawayjr` sees a comma, which would normally occur before a Jr. or Sr. following a name. The second argument is never used, which is how Jr., or Sr., or III, are thrown away:

```
\def\throwawayjr#1, #2{%
  \gdef\fullname{#1 * }}
```

`\throwawayjr` is used inside a box that is never used, so we can supply the comma that ends argument #1, in case there is no comma in the name given. If a name is used that contains a comma, that comma delimits the first argument. Since the extra comma is in a box that is never invoked, the extra comma is never printed.

Here is code to open an auxiliary file whose name is the same as the file containing `\alphalist`, but with an `.alf` extension:

```
\newwrite\alphafile
\immediate\openout\alphafile=\jobname.alf
```

Now we have finished describing the commands needed to define the names and address and to send their macro names to the auxiliary file, and it is time to input the sorted list.

`\endalphalist` turns off the `\everypar` that was established with `\alphalist` and inputs the `.srt` file if it exists. Since all the definitions precede `\endalphalist`, when the `.srt` file is brought in with the `csname` control sequences in it, each control sequence will produce its defined name and address:

```
\def\endalphalist{\egroup
  \global\everypar={}
  \openin1 \jobname.srt
  \ifeof1 %
    \message{<<Please sort \jobname.alf
      to produce \jobname.srt >>}
  \else
    \immediate\closein1
    \input \jobname.srt
  \fi}
```

Example:

```
\alphalist
George Smith
21 Maple Street
Ogden, Utah 68709
```

```
Jacqueline Onassis
Upper East Side
NYC, NY
```

```
Mr. W. T. C. Schoenberg, Jr.
Travesty Lane
Culver City, Iowa
```

```
\endalphalist
```

This writes the following lines in the file `test.srt` after `test.alf` is sorted:

```
\csname OnassisJacqueline\endcsname
\csname SchoenbergMr.W.T.C.\endcsname
\csname SmithGeorge\endcsname
```

which will transpose the original list to print the names and addresses in alphabetic order.

The complete address list code.

```
\newcount\namenum
\newdimen\heightofentry \heightofentry=.75in
\newdimen\widthofentry \widthofentry=.3\hsize
\def\alphalist{\bgroup\obeylines
  \global\everypar={\look}}
{\obeylines
\gdef\look#1^^M#2^^M^^M{\def\test{#1}
\setbox0=\hbox{%
\expandafter\throwawayjr\test, {}}
\global\namenum=0
\expandafter\takeapart\fullname
\obeylines \everypar={} \expandafter%
\gdef\csname\nameinrev\endcsname{%
\vtop to\heightofentry{\parindent=0pt
\vfill\hsize=\widthofentry
#1
#2
\vfill}}%
\immediate\write\alphafile%
{ \noexpand\csname\nameinrev%
\noexpand\endcsname}%
\global\everypar={\look}}}}
\def\makerightdef{\ifcase\namenum\or
\or\gdef\nameinrev{\one}
\or\gdef\nameinrev{\two\one}
\or\gdef\nameinrev{\three\one\two}
\or\gdef\nameinrev{\four\one\two\three}
\or\gdef\nameinrev{\five\one\two\three}
\four}\fi}
\def\makedef#1{\ifcase\namenum
\or\gdef\one{#1}
\or\gdef\two{#1}
\or\gdef\three{#1}
\or\gdef\four{#1}
\or\gdef\five{#1}\fi}
\def\aster{*}
\def\takeapart#1 {%
\global\advance\namenum by1
\def\onex{#1} \makedef{#1}
\if\onex\aster \makerightdef\let\go\relax
\else \let\go\takeapart
\fi\go}
\def\throwawayjr#1, #2{%
\gdef\fullname{#1 * }}
\newwrite\alphafile
\immediate\openout\alphafile=\jobname.alf
\def\endalphalist{\egroup
\global\everypar={}
\openin1 \jobname.srt
\ifeof1 \message{<<Please sort \jobname.alf
to produce \jobname.srt >>}
\else
\immediate\closein1
\input \jobname.srt\fi}
```

Where's the Greek Shift Key?

S. A. Fulling

Mathematics Department, Texas A&M University, College Station, TX 77843

409-845-2237. Internet: fulling@sarastro.tamu.edu Bitnet: saf8613@tamstar

Abstract

Experienced typists of mathematical formulas soon tire of typing out the names of Greek letters; they adopt short macros, such as `\za` for `\alpha`. A standard correspondence between Greek and Latin letters is proposed, in which phonetic resemblance is given precedence over typographical. The resulting macros enable any Greek letter to be typed in three keystrokes (or two, if `\z` is assigned to a function key).

The title is one of the first questions I asked when I began using `TeX` in 1985. I couldn't believe that one had to write out the names of the Greek letters in mathematical formulas. Later I learned to appreciate `TeX`'s design, where control sequences are given self-explanatory, comprehensible names, and it is the user's responsibility to speed things up by giving shorter names to the most frequently used ones.

Ideally, our keyboards would have a Greek shift key, so that holding down that key and typing a would enter `\alpha`. Lacking that, we could just `\define \a as \alpha`, and similarly for other letters — except that there are already some single-letter control sequences in plain `TeX` (e.g., `\b`, `\d`, `\H`, `\i`, ...). A check of *The TeXbook*'s index reveals, however, that there are no *two-letter* control sequences beginning with `\z`. Thus we can `\define \za as \alpha`, and so on, enabling any Greek letter to be typed in three keystrokes as soon as a correspondence between Greek and Latin letters is established.

In fact, the keystrokes can be reduced to two if `\z` can be assigned to a single key. This is even better than a shift key! One situation where this can be done is an IBM PC or compatible running a keyboard macro program such as SuperKey. My solution (which will not fit everyone's typing style) is to program the 5 key in the numeric keypad to enter `\z`. (This is the only numeric key that does not already have a cursor or editing function assigned to it.)

All this discussion is preliminary to the main problem: There is no well-defined correspondence between Greek and Latin letters. Indeed, there are 26 Latin letters and only 24 Greek ones. Clearly, α should be assigned to a and β to b, but should γ be

represented by c or by g? I would choose the latter; but then what should we do with letters like θ and ψ , which have no Latin equivalents?

Looking at the *Symbol 12* type element ("math golf ball") of my old IBM Selectric typewriter, I find α and β treated as expected, but then γ is assigned to q, δ to w, θ to r, ϕ to d, and so on. I have never taken a course in Greek, but I think that every reasonably well educated person can see that something is wrong here.¹ Let us try to construct a correspondence that is less arbitrary.

Of course, ultimately such a correspondence is arbitrary, in the sense that some design decisions are necessary. My first decision has already been hinted at: Instead of simply matching up the alphabets in order ($\gamma \mapsto c$), we should try to assign each Greek letter to the most closely corresponding Latin letter ($\gamma \mapsto g$). With this understanding, the following Greek letters have obvious Latin equivalents: α , β , γ , δ , ϵ , ζ , ι , κ , λ , μ , ν , \omicron , σ , τ , υ , ϕ . This is 16 of the 24.

Beyond this point, however, there is room for controversy. There is a temptation to assign ρ to p, since the letters look alike. But I prefer to give phonetic resemblance precedence over typographical resemblance; ρ is r, and p is π . For the same reason, χ represents ξ , not χ . After all, χ is nicely handled by q, a phonetically similar and otherwise useless letter. However, typographical resemblance takes over for η ($\mapsto h$) and ω ($\mapsto w$), because their closest phonetic counterparts have already been used for ϵ and o .

¹ Unfortunately, this irrational transliteration scheme is "common in most word processing systems", according to Hoover [1989, page 557 (cf. Figure 4)].

Finally, it is stretching the truth only slightly to say that there *is* a Latin phonetic equivalent to θ : There was an old English letter “thorn”, which resembled a y and is still often represented by y on the signs of small businesses with names of the form “Ye Olde ...”.

The only Greek letter left is ψ . It is natural to assign it to c, since both are somewhat superfluous sibilants. The two unused Latin letters are j and v.

I have been using macros implementing this transliteration scheme for over four years. After submitting the abstract for this presentation, I learned that similar systems had already been introduced by Silvio Levy [1988] for typesetting Greek *text*, and by John Collins [1990] in a T_EX-compatible WYSIWYG editor for PCs. To my great relief, their choices² are almost identical to mine:

Latin	Fulling	Collins	Levy
c	ψ	ψ	[ς]
j	unused	unused	θ
v	unused	θ	unused
y	θ	unused	ψ

Clearly we are close to a consensus; let us all work to establish one.

In addition to lower-case letters, one will define macros for the capital Greek letters when necessary (e.g., $\backslash zC$ for Ψ). But there are 13 capital letters plus the lower-case omicron that are typographically identical with Latin letters. (E.g., we do not need $\backslash zQ$ for X, the capital chi.) The Latin letters thereby freed, along with both cases of j and v and the 10 numerals, can be used with $\backslash z$ to define additional macros at the user’s discretion. (E.g., $\backslash zQ$ might be defined as $\backslash subseteq$.) In effect, $\backslash z$ acts (almost) as a second escape character. Examples are listed below.

Listing

The foregoing discussion is summarized and implemented by these macros:

```
%% greek.mac %%
\def\za{\alpha}      % alpha  -> A
\def\zb{\beta}       % beta   -> B
\def\zc{\psi}        % psi    -> C
\def\zC{\Psi}
\def\zd{\delta}     % delta  -> D
\def\zD{\Delta}
\def\ze{\epsilon}   % epsilon -> E
\def\zf{\phi}       % phi    -> F
\def\zF{\Phi}
```

² More precisely, the treatment of σ and ς described here is a slight modification of Levy’s, described by Haralambous and Thull [1989].

```
\def\zg{\gamma}      % gamma  -> G
\def\zG{\Gamma}
\def\zh{\eta}        % eta    -> H
\def\zi{\iota}       % iota   -> I
% (available for another purpose) -> J
\def\zk{\kappa}      % kappa  -> K
\def\zl{\lambda}     % lambda -> L
\def\zL{\Lambda}
\def\zm{\mu}         % mu     -> M
\def\zn{\nu}         % nu     -> N
% (implicitly assigned to omicron) -> O
\def\zp{\pi}         % pi     -> P
\def\zP{\Pi}
\def\zq{\chi}        % chi    -> Q
\def\zr{\rho}        % rho    -> R
\def\zs{\sigma}      % sigma  -> S
\def\zS{\Sigma}
\def\zt{\tau}        % tau    -> T
\def\zu{\upsilon}    % upsilon -> U
\def\zU{\Upsilon}
% (available for another purpose) -> V
\def\zw{\omega}      % omega  -> W
\def\zW{\Omega}
\def\zx{\xi}         % xi     -> X
\def\zX{\Xi}
\def\zy{\theta}      % theta  -> Y
\def\zY{\Theta}
\def\zz{\zeta}       % zeta   -> Z
```

Examples of possible ways to fill up the table are

```
\def\zI{\infty}     \def\zN{\emptyset}
\def\zj{\quad}      \def\zJ{\quad}
\def\zK{\subset}    \def\zQ{\subseteq}
\def\zo{\oplus}     \def\zO{\otimes}
\def\zv{\partial}   \def\zV{\nabla}
\def\zZ#1{\ifcase#1 }\or \displaystyle
\or \textstyle \or \scriptstyle
\or \scriptscriptstyle \fi}

\def\z#1#2{\ifcase#1 {\overline {#2}} %\z0
\or{\if #2i {\tilde\imath}}
\else\if #2j {\tilde\jmath} %\z1
\else {\tilde #2} \fi\fi}
\or{\if #2i {\hat\imath}}
\else\if #2j {\hat\jmath} %\z2
\else {\hat #2} \fi\fi}
%% \or ... % more numerals
\fi}
```

Bibliography

- Collins, John. “ET, Version 1.05.” Illinois Institute of Technology, 1990.
- Haralambous, Yannis, and Klaus Thull. “Typesetting Modern Greek with 128 Character Codes.” *TUGboat* 10(3), pages 354–359, 1989.
- Hoover, Anita Z. “Using WordPerfect 5.0 to Create T_EX and L^AT_EX Documents.” *TUGboat* 10(4), pages 549–559, 1989.
- Levy, Silvio. “Using Greek Fonts with T_EX.” *TUGboat* 9(1), pages 20–24, 1988.

TransFig: Portable Graphics for T_EX

Micah Beck

Department of Computer Science, Cornell University, Ithaca, NY 14853
(607) 255-8597. Internet: beck@cs.cornell.edu

Alex Siegel

Department of Computer Science, Cornell University, Ithaca, NY 14853
(607)-255-1165. Internet: siegel@cs.cornell.edu

Abstract

The TransFig software package defines a portable description language for technical graphics. Translations are provided from this language to commonly used graphics description formats, which can then be included in typeset documents. TransFig includes a particularly convenient framework for including figures in L^AT_EX. The graphics language defined by TransFig facilitates the interchange of structured, modifiable graphics between applications. In this paper, we review our experience with TransFig to argue the need for a standard *application level* graphics language, and suggest guidelines for its design.

Fig and TransFig

The Fig graphics editor was originally developed by Supoj Sutanthavibul at the University of Texas. Fig was designed to produce output in the language of the PIC graphics preprocessor for Troff, although it uses an editable intermediate file format which is quite independent of the output language. This *Fig code* format consists of a simple dump of Fig internal data structures. Fig was distributed from the University of Texas with two translators: from Fig code to PIC and to PostScript.

TransFig. Neither of the output forms supported by Fig allowed inclusion of Fig graphics in T_EX documents in the operating environment of the Computer Science Department at Cornell University. To make such inclusion possible, Micah Beck developed a translator from Fig code to P_IC_TE_X macros [Wichura]. Frank Schmuck, also at Cornell, developed a translator to L^AT_EX picture environment macros; the generality of this translation was restricted by limitations of the target language. These two translators, together with those developed at Texas, and a translation to the EPIC and EEPIC macro packages developed by Conrad Kwok at the University of California, Davis, were combined to create a single package for *Translating Fig* code [Beck].

TransFig was developed with two high level goals:

1. to define a useful graphics intermediate form with a clear interpretation which can be implemented in any reasonably expressive graphics language.
2. to create a framework for the convenient inclusion of figures in T_EX documents with no user customization due to the choice of graphics language.

In order to create a widely used intermediate form quickly, it was decided to define a standard interpretation for the Fig intermediate format. A reference manual was developed which defines Fig code and its interpretation [Beck]. While this interpretation was derived from the Fig editor, it is independent of that implementation. The second goal was addressed in the UNIX computing environment by the Transfig program which is described in a later section.

Goals

TransFig should be evaluated in light of its specific goals; we will therefore look more closely at what TransFig does and does not attempt to achieve.

Expressiveness. The most important parameter in the design of TransFig is the class of graphical figures which is to be expressed. These figures,

which we call *technical graphics*, are combinations of *graphical primitives* with embedded text; bitmaps are not included. Primitives are simple lines and curves, with properties such as dotted or dashed lines, shading, and arrow heads; text properties include font and size.

Technical graphics are typically used to illustrate some idea or example. The content of such figures is transmitted mainly through the shape and labelling of primitives and their placement relative to one another. This should be considered in contrast to the pixel-level precision required to produce highly detailed or realistic images (For examples, see Appendix A).

Restricting our interest to technical graphics limits the possible uses for TransFig. On the other hand, it allows us to give a less precise interpretation to Fig code than is required for a general purpose graphics language such as PostScript. A less exact interpretation in turn eases the task of producing a correct implementation using a wide variety of output languages.

TransFig does not attempt to model the expressiveness of PostScript, its most flexible output form. The goal of portability leads TransFig to a level of expressiveness closer to the least common denominator of its output forms. This has led to a reluctance among some developers of Fig to maintain compatibility with the TransFig interpretation of Fig code.

Portability. Portability of graphics is a goal which underlies many other choices in the design of TransFig. We have mentioned portability as a limiting factor on the precision and expressiveness of the interpretation of Fig code; it also rules out local or non-standard interpretations. In this context, portability means that a document, including figures, can be moved between operating environments.

To illustrate this point, consider the specification of bitmap patterns for area fill. It would be possible to increase the flexibility of the area fill specification by using a local configuration file to map logical names of area fill types to actual bit patterns. This would, however, also reduce the portability of the resulting Fig code. For this reason the list of area fill patterns defined by TransFig is not locally extensible; the intent is for this list to be extended at the discretion of the developers of TransFig.

Ease of inclusion. One goal of TransFig is to allow the user to specify the location of a figure within a \TeX document with a simple command which

requires no information about the figure except the name of the Fig code file. This means that the \TeX file produced by TransFig must include all the spacing information required for the proper placement of graphics relative to the surrounding text; the bounding box of the figure must be known.

The details of how to include figures described in Fig code in a document will be discussed later. The problem of calculating the bounding box points up one of the main problems of the definition of Fig code. Formatted text embedded in figures is not handled properly by TransFig, since the bounding box of the text is known only *after* it has been formatted.

Implementation

The current implementation of TransFig is a compromise; it meets some of the above goals, and meets others only partially. Further discussion of the current implementation can be found in the TransFig manual [Beck].

The Fig2dev program. All Fig code translation programs are derived from F2p, the original program written by Supoj Sutanthavibul to translate Fig code to PIC. The TransFig translators were named Fig2pic, Fig2ps, Fig2tex, Fig2latex, and Fig2epic to differentiate them from the original versions.

Recent releases of TransFig have combined these five translation programs into a single program called Fig2dev. This program consists of a common control structure which uses a standard subroutine interface to produce a specific output form. A specific translation is then implemented as a set of subroutines meeting this interface, much like an operating system device driver.

Output languages. The translations currently implemented by Fig2dev are from Fig code to the following output languages:

$\Pi\text{CTE}\text{X}$, a general picture environment for TEX which uses only native TEX facilities [Wichura].

$\text{L}\text{A}\text{TE}\text{X}$ picture environment, a restricted graphics facility that uses special fonts which are a standard part of $\text{L}\text{A}\text{TE}\text{X}$ [Lamport].

EPIC (Extended Picture Environment), a more flexible extension of $\text{L}\text{A}\text{TE}\text{X}$ picture environment [Podar].

EEPIC (Extended EPIC), a generalization of EPIC which uses an extension of TEX 's DVI output format [Kwok].

PostScript, a general graphics description language often proposed as an industry standard [Adobe].

PIC, a graphics language designed for the Troff typesetting program [Kernighan].

None of these output languages can be used to include any figure in all operating environments; taken together they provide a translation compatible with most environments. T_EX has no native graphics facility, so each output language must strike a balance between generality and adherence to standards.

P_TC_TE_X draws lines using a text character, usually the period, as a pen. This strategy, together with the implementation of all calculations using T_EX integer registers, allows graphics to be generated using only standard features. Formatting complex figures, however, is slow and can require a very large internal T_EX memory.

L^AT_EX picture environment uses special drawing fonts which are a standard part of L^AT_EX; however, the class of figures which can be represented is quite restricted; slopes of lines are restricted to a small set, curves and area fill are not implemented at all.

EPIC is an extension of L^AT_EX picture environment which can represent a broader, but still restricted, class of figures using the same L^AT_EX drawing fonts.

EEPIC is a reimplement of EPIC which uses a graphics extension of the DVI output format (tpic specials), and therefore requires non-standard software support.

PostScript is a very general graphics description language which requires non-standard software (and often hardware) support.

PIC figures require non-standard software support to be included in T_EX documents (tpic specials).

The Transfig program. The goals of generality and portability are addressed by the Fig2dev program; the Transfig program provides ease of graphics inclusion, at least in the UNIX operating environment. Each figure in a document is represented by a separate Fig code file. In order to create a printable document, these figures must be translated to some T_EX-compatible output language, and appropriate commands must be inserted in the T_EX document. These commands will, in general, depend on the choice of output language. The Transfig program hides these details from the user by automating them.

The mechanics of including a set of figures expressed in a given graphics language can be divided into two parts: certain definitions required by all figures, and a particular set of commands for each figure. To allow the automatic generation of

the initial definitions, the user must `\input` into the document the file `transfig.tex`, which will be created by Transfig. For each Fig code file named `figure.fig`, the user must input into the document the file `figure.tex`, which will also be created by Transfig.

The Transfig program takes as arguments an output language and the list of Fig code files. It creates an initial file of definitions `transfig.tex`, and it creates a Makefile which, when processed by the UNIX Make facility, invokes Fig2dev to translate each Fig code file into an appropriate T_EX file.

The `transfig.tex` file generally inputs style or macro files specific to a given output language. The file `figure.tex` may be a large file of graphics commands. Some output forms, notably PostScript, require the creation of an additional file, which is given an appropriate suffix such as `figure.ps`. The file `figure.tex` will then contain T_EX commands which make reference to the PostScript file.

TransFig compatibility. The most powerful aspect of TransFig is that it defines a non-proprietary *application level* language for the description and transfer of technical graphics. By application level, we mean a language which describes graphics primitives at a level high enough to be edited by users or conveniently translated to other forms. In contrast, PostScript is a description level language; it is impossible to recover the higher level primitives from PostScript, particularly the text formatting commands. TransFig is non-proprietary in the sense that it is not under the exclusive control of the developer of any particular software tool. It is based on the Fig graphics editor, but has a definition and interpretation of its own.

Many application level description languages have been defined; every structured graphics editor defines an intermediate format for storage of figures, and ultimately translates it to a printable form. Since such a storage format is seen only as a utility for one graphics editor, there is generally little attention paid to its design. The definition of the format is encoded in the programs which use it, and can change with every release. These proprietary graphics formats are not useful for interchange of graphics between applications.

Fig code is derived from the proprietary graphics format of the Fig graphics editor. In fact, recent developers of Fig have defined PostScript-oriented extensions to the format which are not compatible with the standard TransFig interpretation. To distinguish the TransFig definition of Fig code, we refer to it by its version identifier TFX (for TransFig

eXtension). TFX is a language with a fixed syntax and interpretation, albeit somewhat loosely specified. This makes it appropriate as a target language for other graphics applications.

The most flexible version of the Fig graphics editor currently available is Fig 1.4.FS, or Fig-FS, which supports all TFX features. Fig-FS is a version of Fig Version 1.4 Release 2, the last release distributed from Texas, enhanced by Frank Schmuck of Cornell, and runs under the SunView windowing system.

XFig 2.0 is the most recent version of Fig which runs under the X Windowing System. XFig has been developed by several people; Brian Smith of the Lawrence Berkeley Laboratory has made the most recent improvements. XFig supports one of two Fig code dialects for use as an intermediate language, TFX and its own 2.0 format; the choice is made at compile time. Fig code 2.0 is a PostScript-oriented extension to Fig code; the PostScript driver in recent versions of the Fig2dev program supports both dialects.

Several programs are currently compatible with TransFig and with one another through their use of TFX. These include:

- Gnuplot, a numerical plotting program, which can produce output in TFX;
- Pic2fig, which translates PIC into TFX;
- Plot2fig, which translates the UNIX plot file format to TFX.

TransFig is a flexible and widely used tool for the portable exchange and inclusion of graphics. This success has come in spite of serious shortcomings which the definition of TFX has inherited from the initial implementation of Fig. In any case, it is worthwhile asking what the most appropriate application level graphics format for technical graphics in T_EX documents would be.

Intermediate Languages

Fig code has serious shortcomings as an application level graphics description language. These problems include:

1. an unreadable syntax,
2. an ad hoc integration of text with graphics,
3. limited facilities for the creation and use of composite graphics objects.

The least important problem is the syntax; it is mainly troublesome to software developers. In order to be easily parsed using the C language I/O library, Fig code consists almost solely of numbers; strings are used only to represent text objects. A

more readable syntax similar to that of PostScript would be preferable. The other points are more troublesome, and the first question to address is: why not settle on PostScript as a standard graphics language for T_EX?

PostScript. It is commonly held that one graphics language will suffice for all document description needs, and that PostScript is the appropriate standard. As we have pointed out, the level at which graphics are described in PostScript is too low to be useful as an application level representation. The function served by Fig code is simply different from that served by more primitive languages.

If PostScript were accepted as the standard graphics language for T_EX, no higher level standard would be needed to provide portability. On the other hand, PostScript is very general and a full implementation places a substantial and unnecessary burden on users of technical graphics. A simpler extension to the DVI format would suffice for that purpose.

A simple interface designed to meet a specific purpose can insulate a software system from changes in technology. This is an important function of non-proprietary document description languages like Fig code or the T_EX DVI format. It is possible that PostScript will be superseded by another popular page description language; the T_EX community should not be tied to a single low level interface.

Note that the choice of PostScript or some other language of equivalent power is not important for our purposes. The complexity of technical graphics does not change rapidly; like technical prose, figures which convey ideas are best expressed with simple constructs. The full flexibility of PostScript is not required, and may in fact distract the technical writer.

Embedded text. Integrating text into graphics turns out to be more of a problem than integrating graphical figures into text documents. The problem is that a graphical interpreter may not be able to deduce what the size and shape of the text will be after formatting. On the other hand, it is necessary to allow formatting of text, in order to give unity to the appearance of the text and figures, and because technical graphics often require complex equations to be embedded.

Because of this problem, the TransFig interpretation of Fig code does not allow any embedding of formatting commands in text objects. This strict interpretation is, however, unacceptable to users, and so formatting of text objects is a necessity. To illustrate the problem posed by mixing of formatted

and unformatted text, consider how the curly brace character (`{}`), which has a special significance to T_EX, is handled when it appears in a text object.

When the curly brace character appears in unformatted text, it must be escaped with a backslash (`\{`) to indicate that it is not to be interpreted as a control character by T_EX. In formatted text, however, control characters are not escaped; the translation program must know which type of text is being handled.

Since Fig code does not provide a means for making the distinction, TransFig takes a heuristic approach: text that has either the font or size field set to a non-default value is assumed not to include formatting commands; a text object which has both text properties set to the special default value is allowed to include such commands. A better way to deal with this problem is by differentiating between three distinct types of text: plain, formatted, and special.

Plain text contains no formatting commands. Properties such as font and size are specified as properties of the text object, but do not appear in the text itself. This treatment of text is the most common in graphics editors.

Formatted text contains simple formatting commands in a language which is part of the definition of the intermediate language. An appropriate choice might be a subset of L^AT_EX.

Special text can be expressed in any formatting language; it is not interpreted but is passed through to the output language unchanged. The specific formatter used can be specified as a property, or omitted.

The properties of plain or formatted text must be part of the definition of the language. Furthermore, a reasonably powerful graphics editor should be able to display formatted text. A very sophisticated editor might actually invoke a formatting program to generate output for special text. To allow sufficient space to be left in the containing document, however, the bounding box of special text must be explicitly specified as a property of the text object.

Intermediate language design goals. The design of an intermediate graphics representation is a complex matter, with many trade-offs to consider. We can, however, list a number of design goals for such a language:

Fast load and store: since this language will be used by an editor as the main form of graphics storage, it must be very efficiently loaded from and stored to a file.

Fast display: graphics previewers and editors must be able to interpret the language with minimal computational overhead.

Easy conversion to other formats: translation of this language to other languages such as PostScript will be very common. Unusual drawing primitives can be a major impediment to this conversion.

Extensibility: there must be a well defined facility for efficiently adding primitives or attributes to the language. It must be possible to parse extensions to the language without necessarily interpreting the extension.

User readability: users must be able to understand and edit the language. This is valuable for software debugging and for making manual adjustments to pictures.

Intuitive interpretation: there must be a direct correspondence between intuitive concepts and language constructs. Unusual constructs lead to bugs, misunderstandings, and lengthy documentation.

Density: pictures will be archived for long periods of time in this language. The most common constructs should be expressed with as few wasted characters as possible. The judicious use of macros and abbreviations is very helpful in this respect.

Composition: it must be convenient to create compound objects from more primitive objects, and to manipulate these compound objects.

The most important shortcomings of Fig code and PostScript can be understood in terms of these goals. For example, Fig is unreadable and not conveniently extensible; PostScript suffers from high computational overhead due to high language complexity and low density due a verbose style of punctuation.

ApGraph

Experience with Fig code has demonstrated the usefulness of an application level graphics description language to the T_EX community. In spite the shortcomings of Fig code, the use of Fig and TransFig is increasing along with the popularity of the applications which use it. This is the appropriate time to stop and redesign TransFig; code based on preexisting designs and minimal resources is not a sound basis for future development.

At Cornell, Alex Siegel is developing ApGraph, an intermediate language for *Application Graphics*, which is intended to replace Fig code eventually. A new graphics editor based on the X Windowing System will support ApGraph as well as T_EX, and

TransFig will continue to support both. Further development based on Fig code will, however, cease.

The T_EX community has much to gain by adopting some application level graphics language as a standard. We hope that ApGraph will be an attractive option, as it is a simple language oriented towards the technical graphics most needed by the T_EX community. Of course, other standards are possible, including the ANSI/ISO standard Computer Graphics Metafile (CGM) format. The best choice of language requires further study and consideration.

Conclusions

We have defined a set of goals for an application level intermediate form for technical graphics in T_EX documents. We have seen how the definition of a standard interpretation for Fig code has allowed it to serve the function of an intermediate graphics language. The TransFig package, which implements this standard interpretation of Fig code, is gaining increasing popularity and acceptance in the T_EX community, in spite of Fig code's serious shortcomings as a graphics language. We have argued the value to the T_EX community of adopting a standard language for application level description of technical graphics, and outlined some design requirements of such a language. ApGraph is being developed as Cornell in the hopes of influencing the definition of such a standard.

Software Availability

Most of the software described in this article is available without charge from the archive server at Clarkson University (Internet: sun.soe.clarkson.edu). Access is through anonymous FTP or by mail. Many packages, including the most recent version of TransFig, are also available for FTP from Cornell University (Internet: svax.cs.cornell.edu). FTP sites for packages not available from Clarkson are listed below. This information is subject to change.

GnuPlot is available from duke.cs.duke.edu.

Pic2fig is not available for anonymous FTP. Contact author Micah Beck for distribution.

Plot2fig is available from qed.rice.edu.

Xfig is available from expo.lcs.mit.edu as a contributed client.

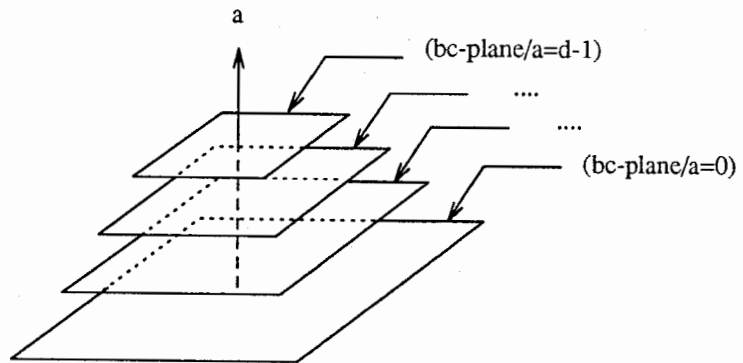
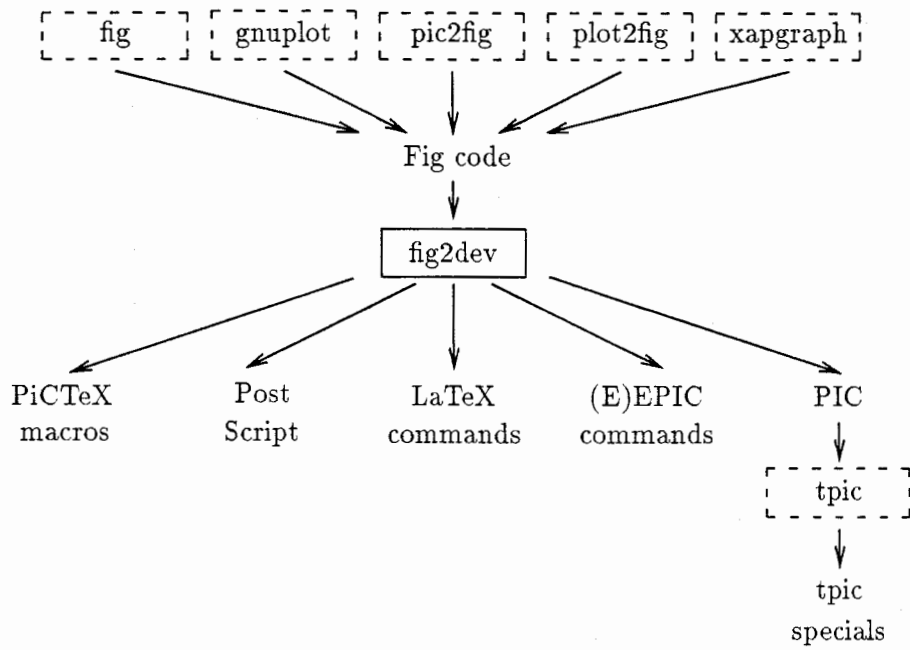
Acknowledgements

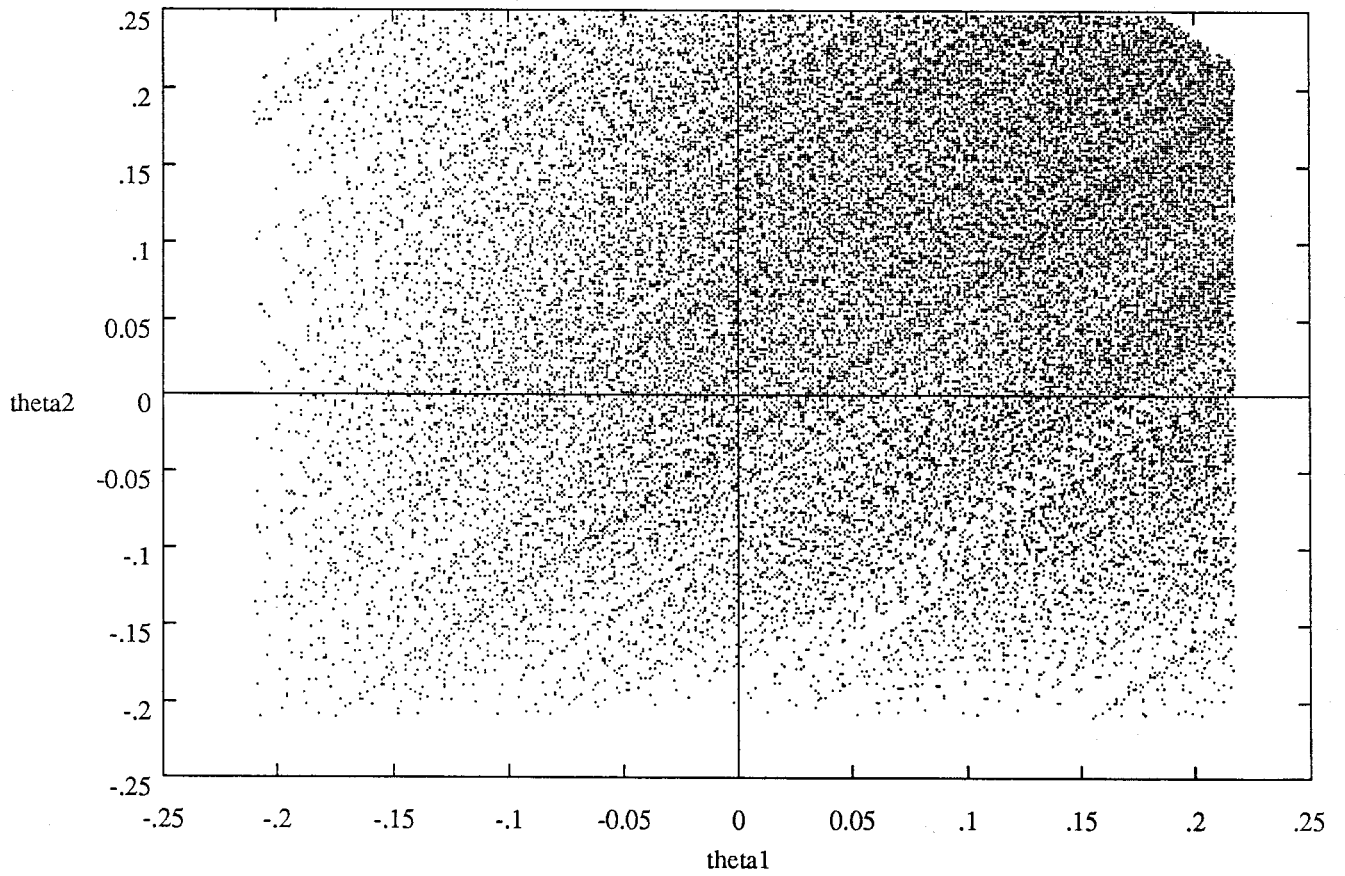
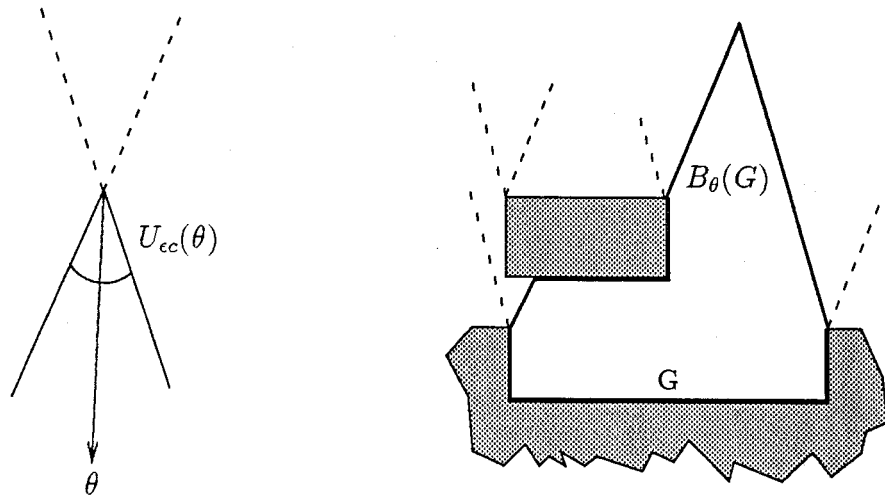
The development of TransFig has been made possible by the collaboration of a group of individuals on the network too numerous to list here. The most important individual contributions were made by Frank Schmuck and Conrad Kwok. Ajei Gopal first introduced Micah Beck to Fig, and Alex Aiken patiently tested the Fig-to-P_TC_TE_X translator while writing his thesis. A special acknowledgement goes to Supoj Sutanthavibul, whose original implementation of Fig has lived on in many incarnations and from which all versions of Fig and TransFig are derived.

Bibliography

- Adobe Systems Incorporated. *PostScript Language Reference Manual*. Reading, Mass.: Addison-Wesley, 1985.
- Beck, Micah. "TransFig: Portable Figures for T_EX" Cornell University Dept. of Computer Science Technical Report #89-967, (February 1989)
- Kernighan, B. W. "PIC—A Language for Typesetting Graphics", *Software Practice and Experience*, 12(1), pp. 1-21 (January 1982)
- Kwok, Conrad. "Extensions to EPIC and L^AT_EX Picture Environment." Software documentation. University of California, Davis, Dept. of Computer Science. (July 1988)
- Lamport, Leslie. *L^AT_EX: A Document Preparation System*. Reading, Mass.: Addison-Wesley, 1986.
- Podar, Sunil. "Enhancements to the Picture Environment of L^AT_EX." State University of New York at Stony Brook, Dept. of Computer Science Technical Report #86-17 (July 1986)
- Wichura, Michael. "The P_TC_TE_X Manual." Software Documentation. The University of Chicago. (November 1986) Second printing, by The T_EX Users Group, as *T_EXniques*, Number 6, 1987.

Appendix





BASIX — An Interpreter Written in T_EX

Andrew Marc Greene

MIT Project Athena, E40-342, 77 Massachusetts Avenue, Cambridge, MA 02139
617-253-7788. Internet: amgreene@mit.edu

Abstract

An interpreter for the BASIC language is developed entirely in T_EX. The interpreter presents techniques of scanning and parsing that are useful in many contexts where data not containing formatting directives are to be formatted by T_EX. T_EX's expansion rules are exploited to provide a macro that reads in the rest of the input file as arguments and which never stops expanding.

Introduction

It is a basic tenet of the T_EX faith that T_EX is Turing-equivalent and that we can write any program in T_EX. It is also widely held that T_EX is “the most idiosyncratic language known to us.”¹ This project is an attempt to provide a simple programming front end to T_EX.

BASIC was selected because it is a widely used interpreted language. It also features an infix syntax not found in Lisp or POSTSCRIPT. This makes it a more difficult but more general problem than either of these others.

The speed of the BASIX interpreter is not impressive. It is not meant to be. The purpose of this interpreter is not to serve as the BASIC implementation of choice. Its purpose is to display useful paradigms of input parsing and advanced T_EX programming.

Interaction with T_EX

Associative arrays. Using `\csname` it is possible to implement associative arrays in T_EX. Associative arrays are arrays whose index is not necessarily a number. As an example, if `\student` has the name of a student, we might look up the student's grade with

```
\csname grade.\student\endcsname
```

which would be `\grade.Greene` in my case. (In the case of `\csname`, all characters up to the `\endcsname` are used in the command sequence regardless of their category code.)

¹ Ward, Stephen A. and Robert H. Halstead, Jr., *Computation Structures*, MIT Press, 1990

BASIX makes extensive use of these arrays. Commands are begun with C; functions with F; variables with V; program lines with /; and the linked list of lines with L. This makes it easy for the interpreter to look up the value of any of these things, given the name as perceived by the user.

One benefit of `\csname... \endcsname` is that if the resultant command sequence is undefined, T_EX replaces it with `\relax`. This allows us to check, using `\ifx`, whether the user has specified a non-existent identifier. This trick is used in exercise 7.7 in *The T_EXbook*. We use it in BASIX to check for syntax errors and uninitialized variables.

Token streams. The BASIX interpreter was designed to be run interactively. It is called by typing `tex basix`; the file ends waiting for the first line of BASIC to be entered at T_EX's * prompt. This also allows other files to `\input basix` and immediately follow it with BASIC code.

We cannot have the scanner read an entire line at once, since if the last line of `basix.tex` were a macro that reads a line as a parameter, we'd get a “File ended while scanning use of `\getline`” error. Instead, we use a method which at first blush seems more convoluted but which is actually simpler.

We note that T_EX does not make any distinction between the tokens that make up our interpreter and the tokens that form the BASIC code. The BASIX interpreter is carefully constructed so that each macro ends by calling another macro (which may read parameters). Thus, expansion is never completed, but the interpreter can continue to absorb individual characters that follow it. These characters affect the direction of the expansion; it is this behaviour that allows us to implement a BASIC interpreter in T_EX.

Category codes. Normally, TeX distinguishes between sixteen categories of characters. To avoid unwanted side effects, the BASiX interpreter reassigns category codes of all non-letters to category 12, “other”. One undocumented feature of TeX is that it will never let you strand yourself without an “active” character (which is normally the backslash); if you try to reassign the category of your only active character (with, *e.g.*, `\catcode'\=12`), it will fail silently. To allow us to use every typeable character in BASiX, we first make `\catcode31=0`, which then allows us to reassign the catcode of the backslash without stranding ourselves. (Of course, the BASiX interpreter provides an escape back to TeX which restores the normal category codes.)

We also change the category code of `^^M`, the end-of-line character, to “active”. This lets us detect end-of-line errors that may crop up.

Semantics

Words. A *word* is a collection of one or more characters that meet requirements based on the first character. The following table describes these rules using *regular expression* notation.² These rules are:

First Character	Regular Expression	Meaning
[A-Z, a-z]	[A-Z, a-z][A-Z, a-z, 0-9]*\$?	Identifier
[0-9]	[0-9]+	Integer
"	"[^"]*"	String
Other	.	Symbol

An end-of-line at the end of a string literal is converted to a “.”.

Everything in BASiX is one of these four types. Line numbers are integer literals, and both variables and commands are identifiers.

The `\scan` macro is used in BASiX to read the next word. It uses `\futurelet` to look at the next token and determine whether it should be part of the current word; *i.e.*, whether it matches the regular expression of the current type. If so, then it is read in and appended; otherwise `\scan` returns,

² In this notation, [A-Z, a-z] means “any character falling between A and Z or between a and z, inclusive.” An asterisk means “repeat the preceding specification as many times as needed, or never.” A plus means “repeat the preceding specification as many times as needed, at least once.” A question mark means “repeat the preceding specification zero or one times.” A dot means “any character.”

leaving this next token in the input stream. The word is returned in the macro `\word`.

The peculiar way `\scan` operates gives rise to new problems, however. We can’t say

```
\def\Cgoto{\scan\lineno=\word}
```

because `\scan` looks at the tokens which follow it, which in this case are `\lineno=\word`. We need some way to define the `goto` command so that the `\scan` is at the end of the macro; this will take the next tokens from the input stream. We therefore have `\scan` “return” to its caller by breaking the caller into two parts; the first part ends with `\scan` and the second part contains the code which should follow. The second macro is stored in `\afterscan`, and `\scan` ends with `\afterscan`. As syntactic sugar, `\after` has been defined as `\let\afterscan`. This allows `\Cgoto` to be coded as

```
\def\Cgoto{\after\gotoPartTwo\scan}
\def\gotoPartTwo{\lineno=\word}
```

(Actually, the `goto` code is slightly more complicated than this; but the scanner is the important point here.) This trick is used throughout the BASiX interpreter to read in the next tokens from the user’s input without interrupting the expansion of the TeX macros that comprise the interpreter.

Expressions. An *expression* is a sequence of words that, roughly speaking, alternates between *values* and *operators*. Values fall into one of three categories: literals, identifiers, and parenthesized expressions. An operator is one of less-than, more-than, equality, addition, subtraction, multiplication, division, or reference. (Reference is an implicit operator that is inserted between a function identifier and its parameters.)

Expressions are evaluated in an approach similar to that used in the scanner. A word is scanned using `\scan` and its type is determined. “Left-hand” values are stored for relatives, additives, multiplicatives, and references. Using TeX’s grouping operations the evaluator is reentrant, permitting parenthesized expressions to be recursively evaluated.

In order to achieve a functionality similar to that of `\futurelet`, we exit the evaluator by `\expandafter\aftereval\word`, where the macro `\aftereval` is analogous to `\afterscan`. Since `\word` will contain neither macros nor tokens whose category codes need changing, this is as good as `\futurelet`.

Structure of the Interpreter

The file `basix.tex`, which appears as an appendix to this paper, defines all the macros that are needed to run the interpreter. The last line of this file is `\endeval`, which is usually called when the interpreter has finished evaluating a line. In this case, it calls `\enduserline`, which, in turn, calls `\parseline`.

The `\parseline` macro is part of the *Program Parser* section of the interpreter. It starts by calling `\scan` to get the first word of the next line. If `\word` is an integer literal, it is treated as a line number and the rest of the input line is stored in the appropriate variable without interpretation. If `\word` is not an integer literal, it is treated as a command.

Each command is treated with an *ad hoc* routine near the bottom of `basix.tex`; however, most of them call on a set of utilities that appear earlier in the file.

Character-string calls. There is a simple library of macros that convert between ASCII codes and character tokens, test for string equality, take subsections of strings, and deal with concatenation.

Debugging definitions. The macro `\diw` is a debugging-mode-only `\immediate\write16` (hence the name `\diw`). It is toggled by the user commands `debug` and `nodebug`.

Expression evaluation. The expression evaluator has a calling structure similar to that of the scanner. Calling routines are split in twain, with

```
\def\foo{\after\fooPartTwo\expression}
\def\fooPartTwo{\etc}
```

being a prototype of the calling convention. The evaluator will `\scan` as many words as it can that make sense; in contrast to the scanner, however, it evaluates each instead of merely accumulating them. This process is described above.

Linked list. The BASIX interpreter maintains a linked list of line numbers. The macro

```
\csname L{current line number}\endcsname
```

contains the next line number. These macros will follow the linked list (for the `list` command); they also can insert a new line or return the number of the next line.

Program parser. This section contains a number of critical routines. `\evalline` is the macro that does the dispatching based on the user's command. `\mandatory` specifies what the next character must be (for example, the character after the identifier

in a `let` statement must be `=`). `\parseline` has already been described.

Syntactic scanner. This is the section containing `\scan` and its support macros, which are described above.

Type tests. These routines take an argument and determine whether it is an identifier, a string variable, a string literal, an integer literal, a macro, or a digit. The normal way of calling these routines is

```
\expandafter\if\stringP\word
```

These predicates expand into either `tt` (true) or `tf` (false). Syntactic sugar is provided in the form of `\itstrue` and `\itsfalse`. `\ifstring\word` doesn't work because of the way T_EX matches `\if` and `\fi` tokens — only `\if`-style primitives are recognized.

User Utilities. This is the section of the interpreter in which most of the user commands are defined. Commands are preceded by `\C` (*e.g.*, `\Clist` is the macro called when the user types `list`). Functions are preceded by `\F`.

Limitations of this implementation

BASIX is a minimal BASIC interpreter. There are enough pieces to show how things work, but not enough to do anything practical. Here is a description of the capabilities of this interpreter, so that the reader can play with it. Error recovery is virtually non-existent, so getting the syntax right and not calling non-existent functions is critical.

Entering programs. Lines beginning with an integer literal are stored verbatim. Lines are stored in ascending order, and if two or more lines are entered with the same number, only the last is retained.

Immediate commands. Lines not beginning with an integer are executed immediately. Colons are not supported, so only one command may appear on a line. (When a program line is executed, its line number is stripped and the remainder is executed as though it were an immediate command.)

Commands. The following commands are implemented in some form: `goto`, `run`, `list`, `print`, `let`, `if`, `debug`, `nodebug`, `rem`, `system`, `exit`, and `stop` (but not `cont`). The interpreter is case sensitive (although with an appropriate application of `\uppercase` it needn't be; I was lazy), so these must be entered in lowercase.

The following tables list the commands with no parameters, the commands that take one parameter,

and the two commands (`let` and `if`) that take special forms.

The commands with no parameters are:

<code>run</code>	Starts execution at the lowest line number. Does <i>not</i> clear variables.
<code>stop</code>	Stops execution immediately.
<code>list</code>	Lists all lines in order.
<code>debug</code>	Useful information sent to terminal.
<code>nodebug</code>	Stop debugging mode.
<code>rem</code>	Rest of line is ignored.
<code>system</code>	Exit T _E X.
<code>exit</code>	Exit B _A S _I X to T _E X.

The commands with one parameter are:

<code>goto</code>	Starts execution at the given line number.
<code>list</code>	Lists the given line.
<code>print</code>	Displays the given argument.

(Any of these arguments may be an expression.)

The `let` and `if` commands take special forms. Variable assignments require an explicit `let` command:

`let` *<identifier>* = *<expression>*

Conditionals do not have an `else` clause, and `goto` is not implied by `then`:

`if` *<expression>* `then` *<new command>*

The new command is treated as its own line.

Expressions. Expressions are defined explicitly above. The operators are `+`, `-`, `*`, `/`, `<`, `=`, and `>`. Parentheses may be used for grouping. Variables may not be referenced before being set. (Unlike in traditional BASIC, variables are not assumed to be 0 if never referenced, and they aren't cleared when `run` is encountered).

Functions are invoked with

<function name>(*<param>*, *<param>*, ...)

The parameters are implicitly-delimited expressions that are passed to `\matheval` (which is simply called `\eval` in the table below to save space). The following functions are defined:

<code>len(string)</code>	Returns number of characters in the string.
<code>chr\$(expr)</code>	Returns the character with the given ascii value.
<code>inc(expr)</code>	Returns <code>\eval(expr) + 1</code> .
<code>min(expr1, expr2)</code>	Returns the lesser of two <code>\evals</code> .

Generalization

The B_AS_IX interpreter can easily be generalized to serve other needs. These other needs might be to interpret Lisp or POSTSCRIPT code [Anyone want to write a POSTSCRIPT interpreter in T_EX?];

or to take source code in a given language and pretty-print it.

The definition of a "word" can be changed by modifying the `\scan` macro. It selects a definition for `\scantest` based on the first character; `scantest` is what determines if a given token matches the selected regular expression. The `\scantest` macro is allowed to redefine itself.

The evaluation of expressions can be extended or changed by modifying the `\math` code. Floating-point (or even fixed-point) numbers could be dealt with, although the period would need to pass the `\digitP` test in some cases and not in others.

The method of dealing with newlines is easily removed for languages such as POSTSCRIPT or Lisp for which all whitespace is the same.

Obtaining copies of basix.tex

The source code to this paper and the B_AS_IX interpreter are available by anonymous ftp from `gevalt.mit.edu`, which is at IP address 18.72.1.4. I will also mail out copies to anyone without ftp abilities.

Summary

Using a number of T_EX tricks, some more devious than others, a BASIC interpreter can be written in T_EX. While T_EX macros will often be less efficient than, for example, `awk` paired with T_EX, solutions using only T_EX will be more portable. A less general macro package than B_AS_IX could be written that uses these routines as paradigms and that is very efficient at parsing a specific input format.

Listing of basix.tex

```

1. ---basix.tex begins here.
2. %
3. %   BaSiX (with the emphasis on SICK!) by Andrew Marc Greene
4. %
5. %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
6. %
7. %   Andrew's Affiliations
8. %
9. %   Copyright (C) 1990 by Andrew Marc Greene
10. %   <amgreene@mit.edu>
11. %   MIT Project Athena
12. %   Student Information Processing Board
13. %   All rights reserved.
14. %
15. %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
16. %
17. %   BaSiX's Beginnings
18. %
19. \def\flageol{\catcode13=13}
20. \def\endflageol{\catcode13=5}
21. \def\struncat{\catcode'\$=12}
22. \def\strcat{\catcode'\$=11}
23. \flageol\let\eof
24. \endflageol
25. \newif\ifresult
26. %\newcount\xa\newcount\xb
27. \def\iw{\immediate\write16}
28. \def\empty{}
29. \def\gobble#1{}
30. \def\spc{ }
31. \def\itstrue{tt}
32. \def\itsfalse{tf}
33. \def\isnull#1{\resultfalse}
34. \expandafter\ifx\csname empty#1\endcsname\empty\resulttrue\fi}
35. \newcount\matha\newcount\mathb
36. %
37. %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
38. %
39. %   Character-string Calls
40. %
41. \newcount\strtmp
42. \def\ascii#1{\strtmp'#1}
43. \def\chr#1{\begingroup\uppercase{\gdef\tmp{A}}\endgroup}
44. \def\strlen#1{\strtmp-2% don't count " " \iw tokens
45.   \expandafter\if\stringP #1\let\next\strIter\strIter #1\iw\fi}
46. \def\strIter#1{\ifx\iw#1\let\next\relax\else\advance\strtmp by 1\relax
47.   \fi\next}
48. \def\Flen{\expandafter\strlen\expandafter{\Pa}\return{\number\strtmp}}
49. \strcat
50. \def\F$chr${\expandafter\chr\expandafter{\Pa}\return{\tmp}}
51. \struncat
52. % first char only:
53. % \def\Fasc{\expandafter\asc\expandafter{\Pa}\return{\number\strtmp}}
54. %
55. %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
56. %
57. %   Debugging Definitions
58. %
59. \def\debug{\tracingmacros=2}
60. \def\diw#1{}
61. \def\Cdebug{\let\diw\iw\tracingmacros=2 \endeval}
62. \def\Cnodebug{\def\diw##1{}\endeval}
63. %

```

```

64. %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
65. %
66. % Expression Evaluation
67. %
68. \def\expression{\let\afterexpression\afterscan\math}
69. %
70. % (math is a misnomer and should -> expr)
71. %
72. \newcount\parens\newcount\mathParams
73. \def\math{\parens=0\mathParams96\mathInit\matheval}
74. \def\mathRecurse{\advance\parens by 1\relax\mathParams96\mathInit\matheval}
75. \def\mathInit{\begingroup
76. \let\mathAcc\empty
77. \let\mathOpRel\empty
78. \let\mathOpAdd\empty
79. \let\mathOpMul\empty
80. \let\mathlhRef\empty}
81. %
82. \def\matheval{\after\mathbranch\scan}
83. %
84. \def\mathbranch{\diw{EXPRESS:\expandafter\noexpand\word:}
85. \let\next\matherr
86. \ifx\empty\word\let\next\mathHardEnd\else % Expr. end?
87. \expandafter\if\numberP\word\let\next\mathLiteral\diw{@@}\fi % Number?
88. \expandafter\if\stringP\word\let\next\mathLiteral\fi % String literal?
89. \expandafter\if\identifierP\word\let\next\mathIdentifier\fi % Identifier?
90. \expandafter\if\stringvarP\word\let\next\mathIdentifier\fi % :- (
91. \expandafter\if\macroP\word\let\next\mathMacro\fi % Macro?
92. \ifx\word\Oleft\let\next\mathRecurse\fi % Open paren?
93. \ifx\word\Oright\let\next\mathEndRecurse\fi % Close paren?
94. \ifx\word\Ocomma\let\next\mathComma\fi % Comma?
95. %
96. % Operator?
97. %
98. \ifx\word\Oplus\let\next\mathOp\diw{!+}\fi
99. \ifx\word\Ominus\let\next\mathOp\diw{!-}\fi
100. \ifx\word\Otimes\let\next\mathOp\diw{!*}\fi
101. \ifx\word\Odiv\let\next\mathOp\diw{!/}\fi
102. \ifx\word\Olt\let\next\mathOp\diw{!<}\fi
103. \ifx\word\Oeq\let\next\mathOp\diw{! =}\fi
104. \ifx\word\Ogt\let\next\mathOp\diw{!>}\fi
105. %
106. \fi\next}
107. %
108. \def\Oleft{(\}\def\Oright{)}\def\Ocomma{,}
109. \def\Oplus{+}\def\Ominus{-}\def\Otimes{*}\def\Odiv{/}
110. \def\Olt{<}\def\Oeq{=}\def\Ogt{>}
111. %
112. % There's got to be a better way to do the above....
113. %
114. \def\mathLiteral{\diw{MLIT}}\ifx\empty\mathAcc\diw{ACCUMUL:\word:}
115. \expandafter\def\expandafter\mathAcc\expandafter
116. {\expandafter\expandafter\expandafter\empty\word}
117. \else
118. \diw{ACC has :\mathAcc: and word is :\word:}
119. \errmessage{Syntax Error: Two values with no operator}\fi\matheval}
120. %
121. % Operator stuff: (Need to add string support / error checking)
122. %
123. \def\mathAdd{\advance\matha by \mathb}
124. \def\mathSub{\advance\matha by -\mathb}
125. \def\mathMul{\multiply\matha by \mathb}
126. \def\mathDiv{\divide\matha by \mathb}
127. \def\mathEQ{\ifnum\matha=\mathb\matha-1\else\matha0\fi}

```

```

128. \def\mathGT{\ifnum\matha>\mathb\matha-1\else\matha0\fi}
129. \def\mathLT{\ifnum\matha<\mathb\matha-1\else\matha0\fi}
130. %
131. \def\mathFlushRel{\mathFlushAdd\ifx\empty\mathOpRel\else
132.   \matha=\mathlhRel\relax\mathb=\mathAcc\relax\mathOpRel
133.   \edef\mathAcc{\number\matha}\let\mathOpRel\empty\fi}
134. %
135. \def\mathFlushAdd{\mathFlushMul\ifx\empty\mathOpAdd\else
136.   \matha=\mathlhAdd\relax\mathb=\mathAcc\relax\mathOpAdd
137.   \edef\mathAcc{\number\matha}\let\mathOpAdd\empty\fi}
138. %
139. \def\mathFlushMul{\mathFlushRef\ifx\empty\mathOpMul\else
140.   \matha=\mathlhMul\relax\mathb=\mathAcc\relax\mathOpMul
141.   \edef\mathAcc{\number\matha}\let\mathOpMul\empty\fi}
142. %
143. \def\mathFlushRef{\ifx\empty\mathlhRef\else
144.   \mathParam
145.   \mathlhRef\let\mathlhRef\empty\fi}
146. %
147. \def\mathOp{%
148. \if\word+
149.   \mathFlushAdd\let\mathlhAdd\mathAcc\let\mathOpAdd\mathAdd\fi
150. \if\word-
151.   \mathFlushAdd\let\mathlhAdd\mathAcc\let\mathOpAdd\mathSub\fi
152. \if\word*
153.   \mathFlushMul\let\mathlhMul\mathAcc\let\mathOpMul\mathMul\fi
154. \if\word/
155.   \mathFlushMul\let\mathlhMul\mathAcc\let\mathOpMul\mathDiv\fi
156. \if\word=
157.   \mathFlushRel\let\mathlhRel\mathAcc\let\mathOpRel\mathEQ\fi
158. \if\word>
159.   \mathFlushRel\let\mathlhRel\mathAcc\let\mathOpRel\mathGT\fi
160. \if\word<
161.   \mathFlushRel\let\mathlhRel\mathAcc\let\mathOpRel\mathLT\fi
162. \let\mathAcc\empty
163. \matheval}
164. %
165. \def\mathIdentifier{%
166. \expandafter\ifx\csname C\word\endcsname\relax
167. \expandafter\ifx\csname F\word\endcsname\relax
168. \expandafter\ifx\csname V\word\endcsname\relax
169. \let\next\matherr\diw{LOSING:\word:}
170. \else\let\next\mathVariable\fi
171. \else\let\next\mathFunction\fi
172. \else\let\next\mathCommand\fi\next}
173. %
174. \def\mathVariable{\expandafter\edef\expandafter\word\expandafter
175. {\csname V\word\endcsname}\mathbranch}
176. \def\mathCommand{\expandafter\mathHardEnd\word}
177. \def\mathFunction{\expandafter\let\expandafter\mathlhRef
178. \csname F\word\endcsname\matheval}
179. %
180. \def\mathParam{\advance\mathParams by 1\relax\chr\mathParams
181.   \diw{PARAM:\tmp:\mathAcc:}
182.   \expandafter\edef\csname P\tmp\endcsname{\mathAcc}}
183. \def\mathComma{\mathEnd\mathParam\mathInit\matheval}
184. \def\mathEndRecurse{\mathEnd\advance\parens by -1\matheval}
185. \def\mathEnd{\diw{MATHEND: ACC=\mathAcc:}\mathFlushRel
186.   \xdef\mathtemp{\mathAcc}\endgroup\edef\mathAcc{\mathtemp}}
187. \def\mathHardEnd{\ifnum\parens>0\errmessage{Insufficient closeparens.}\relax
188.   \let\next\endeval\else\let\next\mathFinal\fi\next}
189. \def\mathFinal{\mathEnd\let\value\mathAcc\endexpression}
190. \def\matherr{\errmessage{Syntax error: Unknown symbol \word}}
191. \def\endexpression{\afterexpression}

```

```

192. %
193. %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
194. %
195. %   Linked List
196. %
197. \def\gotofirstline{\edef\lpointer{\csname L0\endcsname}}
198. \def\foreachline#1{\ifnum\lpointer<99999\edef\word{\lpointer}#1%
199. \edef\lpointer{\csname L\word\endcsname}\foreachline{#1}\fi}
200. %
201. %   gotopast{#1} where #1 is a line number, will set \lpointer to
202. %               the least value such that L(lpointer)>#1
203. %
204. \def\gotopast#1{\def\lpointer{0}\def\target{#1}\gotopastloop}
205. %
206. \def\gotopastloop{\edef\tmp{\csname L\lpointer\endcsname}%
207. \ifnum\tmp<\target%
208. \edef\lpointer{\csname L\lpointer\endcsname}%
209. \let\next=\gotopastloop\else\let\next=\relax\fi
210. \next}
211. %
212. \flageol\def\addLineToLinkedList#1#2
213. {\def#1{#2}\diw{Just stored #2 in \noexpand #1}%
214. % now put it into linked list...
215. \expandafter\ifx\csname L\word\endcsname\relax% if it isn't already there,
216. \gotopast{\word}% \def\lpointer{what-should-point-to-word}
217. \expandafter\edef\csname L\word\endcsname{\csname L\lpointer\endcsname}%
218. \expandafter\edef\csname L\lpointer\endcsname{\word}%
219. \fi\endeval
220. }\endflageol
221. \expandafter\def\csname L0\endcsname{99999}
222. %
223. %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
224. %
225. %   Program Parser
226. %
227. \def\evalline{%\iw{EVALLINE : \word:}%
228. \csname C\word\endcsname} %error-checking? :-)
229. \def\evalerror{\errmessage{Unkonwn command. Sorry.}}
230. %
231. %   \mandatory takes one argument and checks to see if the next
232. %   non-whitespace token matches it. If not, an error is generated.
233. %
234. \def\mandatory#1{\def\tmp{#1}\mandatest}
235. \def\mandatest#1{\def\tmpp{#1}\ifx\tmp\tmpp\let\next\afterscan\else
236. \let\next\manderror\fi\next}
237. \def\manderror{\errmessage{\tmpp\spc read when \tmp\spc expected.}}%
238. \afterscan}
239. %
240. %   \parseline gets the first WORD of the next line. If it's a line
241. %   number, \scanandstoreline is called; otherwise the line is executed.
242. %
243. \def\parseline{\after\firsttest\scan}
244. \def\firsttest{\expandafter\if\numberP\word
245. \let\next\grabandstoreline\else\let\next\evalline\fi\next}
246. \def\grabandstoreline{\diw{Grabbing line \word.}}%
247. \expandafter\addLineToLinkedList\csname/\word\endcsname}
248. %
249. %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
250. %
251. %   Syntactic Scanner
252. %
253. %   The \scan routine reads the next WORD and then calls \afterscan.
254. %
255. %   As syntactic sugar, one can write \after\foo to set \afterscan to

```

```

256. % \foo.
257. %
258. % Here are the rules governing WORD. Initial whitespace is
259. % discarded. The word is the next single character, unless that
260. % character is one of the following:
261. %
262. % A-Z or a-z: [A-Z,a-z][A-Z,a-z,0-9]*\ $?
263. % 0-9: [0-9]+
264. % ": "[~"]*"
265. % <,>: [<=>][<=>]? (one or two; not the same if two)
266. %
267. % Note that the string literal ignores spaces but may be abnormally
268. % terminated by an end-of-line. (I wasn't sure how to express that
269. % as a regexp).
270. %
271. %
272. \newif\ifscan % shall we continue scanning?
273. %
274. \def\scan{\def\word{}\futurelet\q\scanFirst}
275. %
276. \def\scanFirst{% Checks the first character to determine type.
277. \let\next\scanIter
278. \expandafter\if\spc\noexpand\q % Space -- ignore it
279. \let\next\scanSpace\else
280. \if\eol\noexpand\q % End of line -- no word here
281. \let\next\scanEnd\else
282. \ifcat A\noexpand\q % Then we have an identifier
283. \let\scanTest\scanIdentifier\else
284. \expandafter\if\digit\q
285. \let\scanTest\scanNumericConstant\else
286. \if"\noexpand\q
287. \let\scanTest\scanStringConstant\else
288. \expandafter\if\relationP\q
289. \let\scanTest\scanRelation
290. \else
291. \let\scanTest\scanfalse
292. \fi\fi\fi\fi\fi\fi\next}
293. %
294. \def\scanIter#1{\expandafter\def\expandafter\word\expandafter{\word #1}
295. \futurelet\q\scanContinueP}
296. \def\scanContinueP{\scanTest\ifscan\let\next\scanIter
297. \else\let\next\scanEnd\fi\next}
298. %
299. \def\scanSpace#1{\scan}% If the first char is a space, gobble it and try again.
300. \def\scanIdentifier{\ifcat A\noexpand\q\scantrue\else
301. \expandafter\if\digit\q\scantrue
302. \else\if$\noexpand\q\scantrue
303. \expandafter\def\expandafter\word\expandafter{\expandafter$\word}
304. \let\scanTest\scanfalse\else
305. \scanfalse\fi\fi\fi}
306. \def\scanEndString{\scanfalse}
307. \def\scanNumericConstant{\expandafter\if\digit\q\scantrue\else\scanfalse\fi}
308. \def\scanStringConstant{\scantrue\if"\q\let\scanTest\scanfalse\fi}
309. \def\scanRelation{\if<\q\scantrue\else\if>\q\scantrue\else\if=\q\scantrue
310. \else\scanfalse\fi\fi\fi}
311. %
312. \def\scanEnd#1{\relax\diw{SCANNED:\word:}
313. \afterscan #1}% dumps trailing spaces.
314. \def\after{\let\afterscan}%
315. %
316. %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
317. %
318. % Type Tests (Predicates for type determination)
319. %

```

```
320. \def\relationP#1{tf} % for now, only single-char relations
321. \def\identifierP#1{\expandafter\identifierTest #1\}
322. \def\identifierTest#1#2\{\ifcat A#1\itstrue\else\itsfalse\fi}
323. \def\stringvarP#1{\expandafter\stringvarTest #1\}
324. \def\stringvarTest#1#2\{\if$#1\itstrue\else\itsfalse\fi}
325. \def\stringP#1{\expandafter\stringTest #1\}
326. \def\stringTest#1#2\{\if #1"\itstrue\else\itsfalse\fi}
327. \def\numberP#1{\expandafter\numberTest #1\}
328. \def\numberTest#1#2\{\expandafter\if\digit #1\itstrue\else\itsfalse\fi}
329. \def\macroP#1{\expandafter\macroTest #1\}
330. \def\macroTest#1#2\{\expandafter\ifx #1\relax\itstrue\else\itsfalse\fi}
331. %
332. % \digit tests its single-token argument and returns tt if true,
333. % tf otherwise.
334. %
335. %
336. \def\digit#1{%
337. \if0\noexpand#1\itstrue\else
338. \if1\noexpand#1\itstrue\else
339. \if2\noexpand#1\itstrue\else
340. \if3\noexpand#1\itstrue\else
341. \if4\noexpand#1\itstrue\else
342. \if5\noexpand#1\itstrue\else
343. \if6\noexpand#1\itstrue\else
344. \if7\noexpand#1\itstrue\else
345. \if8\noexpand#1\itstrue\else
346. \if9\noexpand#1\itstrue\else\itsfalse
347. \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
348. % 9 8 7 6 5 4 3 2 1 0
349. }
350. %
351. %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
352. %
353. % User Utilities. These are the commands that are called by the
354. % user. We could really use a better section name. :-)
355. %
356. % List (one line or all lines, for now)
357. %
358. \def\Clist{\after\listmain\scan}
359. \def\listmain{\isnull{\word}\ifresult\let\next\listalllines
360. \else\let\next\listoneline\fi\next}
361. \def\listline{\iw{\word\spc\csname/\word\endcsname}}
362. \def\listalllines{\gotofirstline\foreachline{\listline}\endeval}
363. \def\listoneline{\listline\endeval}
364. %
365. %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
366. %
367. % Different degrees of ‘stop execution’
368. %
369. \def\Csystem{\end} % exits to the system
370. \def\Cexit{}% \endflageol} % exits to TeX
371. \flageol%
372. \def\Cstop#1
373. {\iw{Stopped in \lineno.}\cleanstop}%
374. \def\cleanstop{\diw{CLEANSTOP}\let\endeval\enduserline\endeval
375. }\endflageol}
376. %
377. %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
378. %
379. % The command ‘rem’ introduces a remark
380. %
381. \def\Crem{\endeval}%
382. %
383. %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```

384. %
385. % The ‘let’ command allows variable assignments
386. %
387. \def\Clet{\after\letgetequals\scan}
388. \def\letgetequals{\after\letgetvalue\mandatory{=}}
389. \def\letgetvalue{\after\letdoit\expression}
390. \def\letdoit{\expandafter\edef\csname V\word\endcsname{\value}%
391. \endeval}
392. %
393. %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
394. %
395. % The ‘print’ command takes a [list of] expression[s] and displays
396. % it [them].
397. %
398. \def\Cprint{\after\printit\expression}
399. \def\printit{\iw{\value}\endeval}
400. %
401. %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
402. %
403. % The ‘if’ command takes an expression, the word ‘then,’ and
404. % another command. If the expression is non-zero, the command is
405. % executed; otherwise it is ignored.
406. %
407. \def\Cif{\after\getift\expression}
408. \def\Cthen{\errmessage{Syntax error: THEN without IF}}
409. \def\getift{\after\getifh\mandatory t}
410. \def\getifh{\after\getife\mandatory h}
411. \def\getife{\after\getifn\mandatory e}
412. \def\getifn{\after\consequent\mandatory n}
413. \def\consequent{\ifnum\value=0\let\next=\endeval\else\let\next=\evalconsq\fi
414. \next}
415. \def\evalconsq{\after\evalline\scan}
416. %
417. %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
418. %
419. % Functions
420. %
421. % Functions may read the counter \mathParams to find out the number
422. % of the top parameter. Parameters are in Pa Pb Pc etc.
423. %
424. \def\return{\expandafter\def\expandafter\mathAcc\expandafter}
425. %
426. \def\Finc{\matha=\Pa \advance\matha by 1
427. \return{\{\mnumber\matha}}
428. %
429. \def\Fmin{\ifnum\Pa<\Pb\return{\Pa}\else\return{\Pb}\fi}
430. %
431. %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
432. %
433. % Program execution control
434. %
435. \def\Crun{\let\endeval\endincrline\def\lineno{0}\endeval}
436. \def\Cgoto{\let\endeval\endgotoline\after\gotomain\scan}
437. \def\gotomain{\edef\lineno{\word}\endeval}
438. %
439. \flageol%
440. \def\execline{%\message{Executing line \lineno...}}%
441. \edef\theline{\csname/\lineno\endcsname}%
442. %\message{THE LINE\theline}%
443. \let\endeval\endincrline\after\evalline\expandafter\scan\theline
444. }\endflageol
445. %
446. % Different varieties of what to do at the end of a command:
447. %

```

```
448.% get new line from user (enduserline)
449.% get next line in order (endincrline)
450.% get line in \lineno (endgotoline)
451.% keep parsing current line (endcolonline tbi)
452.%
453.\flageol%
454.\def\endincrline#1
455.{\diw{ENDINCRLINE}\edef\lineno{\csname L\lineno\endcsname}\execnextline}
456.%
457.\def\endgotoline#1
458.{\diw{ENDGOTOLINE}\let\endeval\execincrline\execnextline}%
459.%
460.\def\execnextline{\diw{Ready to execute line \lineno...}}%
461.\ifnum\lineno<99999\let\next\execline\else\let\next\cleanstop\fi\next}%
462.%
463.\def\enduserline #1
464.{\diw{ENDUSERLINE}\parseline}\endflageol
465.%
466.\let\endeval\enduserline
467.%
468.%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
469.%
470.% Start your engines!
471.%
472.\iw{This is BaSiX, v0.3, emphasis on the SICK! by amgreene@mit.edu}
473.\flageol
474.\catcode32=12
475.\endeval
476.
477.---basix.tex ends here. The blank line at the end is significant.
```


A Noddy's Guide to using T_EX for Text Production: From Manuscript to Bromide

Helen Gibson

Wellcome Institute for the History of Medicine 183 Euston Road, London NW1 2YN, England

Internet: gibson%euclid.ucl.ac.uk@nss.cs.ucl.ac.uk

Abstract

The purpose of this paper is to give a practical example of a PC system setup for obtaining high quality typeset output. A brief discussion of the Wellcome Institute's publication requirements will be followed by detailed explanations of the hardware and software configurations, how to output to PostScript and the utilisation of electronic communications for sending laser printer proofed text to a Linotronic 300 phototypesetter. The value of employing WordPerfect macros, alternative keyboard layouts and style files as an interface for secretarial inputting is also demonstrated.

Introduction

Many non-UK residents may wonder who or what is Noddy. He is a children's storybook character, a doll, created by children's author Enid Blyton. Basically he's young and cute, but pretty naïve. He gets himself into all sorts of sticky situations from which he is invariably extricated by his older, wiser companion, Big Ears. In England the term "a Noddy's guide" is synonymous with "basic", thus this paper is aimed at people who are new to electronic publishing and T_EX. We are all Noddys at some stage. Some of us get to be Big Ears.

In this paper I use T_EX as a generic term, taking L^AT_EX under its umbrella.

Background Information

The Wellcome Institute for the History of Medicine exists to provide library resources and research and teaching facilities for all persons with serious interests in the history of medicine and the allied sciences. It collects, maintains and makes available materials in the history of medicine from all cultures and from all periods ranging from primitive man to the present day. Its collections of books, manuscripts, periodicals, paintings, prints and photographs total upwards of 850,000 items. Its teaching and academic staff, its librarians and associated Research Fellows explore and disseminate the wealth of its collections to the world-wide academic community through lectures, seminars,

symposia and *publications*. It is because of this requirement to communicate through publications that T_EX has become a useful tool to the Institute.

T_EX Arrives

It was almost by accident that T_EX found its way into the Institute through the auspices of Dr Dominik Wujastyk, the Associate Curator for the Oriental collection. In this tale he is Big Ears to my Noddy. In February 1986, attracted by an article on processing strings in SNOBOL4, he bought an issue of *BYTE* magazine. His attention was soon grabbed, however, by an article from the pen of Pierre MacKay of the University of Washington entitled *Typesetting Problem Scripts*. Dominik's interest in string processing in SNOBOL4 was not prompted by idle curiosity, but stemmed from the fact that he definitely has *problem* scripts to typeset in providing a descriptive catalogue of the South Asian collections of the Wellcome Institute. In 1985 he had published the first volume of a Handlist of Sanskrit and Prakrit Manuscripts, of which there are over 6000. The text included many Latin transliterations of the original Devanāgarī script used in Hindi and Sanskrit which includes numerous diacritical marks (Figure 1, Appendix).

Having despaired of the nightmare task of proofing externally typeset galleys, Dominik was attempting to manipulate the IBM DisplayWriter and daisywheel printer technology available at the

Institute at that time. In his efforts to overcome the compromises imposed on non-Latin letters by the limitations of the IBM extended character set, he had obtained a customised daisywheel. As the amount of inputting was far too great to tackle without secretarial help, he devised a system for the secretary to type, for example, $d\backslash\$$ which would then be mapped to the d position on the daisywheel. As you have seen, the results were not ideal, but they were published.

Pierre MacKay's paper promoted \TeX as the solution to the typesetting of problem text. Not only were the facilities for dealing with diacriticals available, but Pierre also held out the promise of being able to typeset Devanāgarī script. This was like manna from heaven to Dominik, who subsequently lobbied for the purchase of a couple of IBM PCs, a Toshiba P1340 dot matrix printer, for which there was a \TeX driver, and a copy of PC \TeX . I'm not sure that there can be many examples of \TeX being the cause of an Institute becoming computerised, but this could be cited as one of them.

The PCs, however, could not be solely dedicated \TeX . They were there for Institute work in general. Their arrival coincided with the commencement of two major projects which it was proposed to *computerise*. Senior staff at the Institute were also realising the benefits of their work being word-processed as opposed to typewritten. It was decided to expand the computing provision in the Institute, and at the same time it was recognised that a *professional* computing person was required to oversee those changes. This is where I came on the scene in September 1986, fresh from the business information systems world of DBASE, LOTUS and, thankfully, DISPLAYWRITE3 (DW3). So, you could say in some roundabout way, that I have \TeX to thank(!) for my present job.

It may appear ungrateful but I have to admit that my priorities were not centred on \TeX . I was evaluating the overall PC requirements of the Institute as far as text and data processing were concerned, and dealing with the anxieties of secretaries who viewed the new technology with the suspicion and fear of the over-worked.

Initially, DW3 was chosen as the PC word processing package primarily because of its relationship to the DisplayWriters, which we expected would facilitate the translation of existing documents and ease the learning curve. The arrival of the PCs and faster printers speeded up the word processing output and consequently created a greater demand. The DisplayWriters and associated

daisywheel printers were deemed obsolete and were replaced. The Toshiba's print quality was inadequate for the Institute's requirements, therefore IBM Quietwriters were chosen. The secretaries, to their relief, were able to dispense with their ear-plugs. So far nothing but improvements in conditions.

Top priority was to take the load off the two secretaries who were carrying the bulk of the word processing load by persuading the primary producers of text, the academics, that it would be a good thing for them to have one of these dreaded things on their desks. Document production would be more direct and editing less onerous. Secretaries of long standing had to be weaned off their typewriters. Not only that, but we were introducing databases as well. You can imagine the learning curves and conceptual mysteries we were struggling with. And then, to top it all, there was \TeX .

In the midst of all the hardware and software changes, Dominik was enthusiastically pursuing \TeX with some splendid results. Fellow academics saw, admired and desired the same effects, but were unwilling or unable to invest the corresponding time and effort in their achievement. The burden then fell on the secretaries. You could say that we could have ignored the demand, but if a system is there to allow an academic to publish a work which would be prohibitively expensive by any other means, then it should be made available. With hind-sight though, in our enthusiasm, we did jump in at the deep end. However, we learnt many lessons which I hope this account may help you to avoid.

Our first \TeX task outside of Dominik's domain was a *book* for our Arabist, Dr Lawrence Conrad, to commemorate the work of a recently deceased colleague. There was a tight dead-line for its publication, and \TeX 's ability to deal with diacritical markings promised Dr Conrad release from the typesetter/proofing ordeal.

Lesson number one—the obvious—do not start with anything longer than a couple of pages. Having said that though, Donald Knuth does describe \TeX as “a new typesetting system intended for the creation of beautiful books” [*The \TeX book*, Preface].

Lesson number two—a must—get your authors to proof and edit the content of their text *before* \TeX ing. Unfortunately, because of the nature of the book's content and a need to prove that \TeX could do the job, this was a lesson we learnt the hard way. There is nothing more guaranteed to drive a secretary to the brink of insanity than having to alter her slaved over text and re-sit through

all that interminable TeX processing and printing procedure, which is not even in background mode! Our secretaries did a valiant job, with the end result being a superb 599 page book *The Formation and Perception of the Modern Arab World: Studies by Marwan R. Buheiry*, complete with 49 plates. This book sells for \$24. Dr Conrad informs me that if the book had been produced in the conventional manner it could not have sold for less than \$90. I mention this to encourage perseverance. It does pay off.

Lesson number three — if you are an applications developer, get your act together and provide an efficient system in order to prevent your secretaries deserting in droves. This is not a trivial point. In a city such as London, secretarial jobs are plentiful and well-paid.

To introduce TeX effectively into your organisation, it has to be accepted that it is not for the masses to learn. Conceptually, it is likely to be different from any other text processing package a secretary has come across. I think of TeX as a programming language embedded in the text, commanding the layout and design of the whole document. Often, complex nested macros are required to create the ultimate design. The first processing run is synonymous with de-bugging. This task is easier for a highly motivated person, with programming skills. Ideally, there should be someone employed with specific responsibility for TeX development. In a circumstance where we were required to respond to an urgent demand I was fortunate in that Dominik was there to design the macros and style files. This is where the hard work lies.

The hardware and software chosen for our bulk text processing had to have some coherence with TeX, or be able to be manipulated to be so, without isolating a TeX system from the main stream of the Institute's requirements. Our word processing package had to change. DW3 was really cumbersome. Text had to be converted to ASCII format before it could be processed by TeX. Often the conversion was problematic. EBCDIC control characters would be retained, grinding processing to a halt and taking an age to find. Fortunately, several major developments in the outside world helped to advance the integration of the TeX cause. We were progressing into the age of the laser printer and the secretaries were keen to take advantage of this fast, quiet technology. IBM was notoriously bad at providing drivers for non-IBM printers. For a short time we used PC-WRITE to overcome these problems particularly for the book inputting, but it

did mean an added learning burden all round. The change had to be to a mainstream package which would provide the secretaries with a marketable skill and ourselves with a ready supply of able operators. The choice came down to two packages, WordPerfect and Microsoft Word, both of which were becoming industry standards. I chose the former because, as well as being a powerful package, it was much easier to learn than Microsoft Word, and WordPerfect Corporation provided excellent telephone technical support. There is an advantage in choosing industry standards in that everyone writes for them. You will never be stranded out on a limb.

The same can be said for choosing hardware. When it came to laser printers I opted to go with Hewlett-Packards. Their documentation is clear and informative on all levels and they have good customer support. As we are not an Apple site the LaserWriter was not an option. I would stress that I started buying nearly three years ago. There have been great changes since those days. Today, I would recommend that you go for a PostScript printer, because it is the advent of PostScript and TeX's adaptation to its environment which has made really high quality output possible.

Lesson number four — you have to convince the powers that be that the upheavals involved in the implementation of TeX are worth the effort invested. We have an Administrator whose task it is to ensure that the Institute runs efficiently and all staff are relatively satisfied with their working conditions. Rumblyings of discontent from the ranks were threatening to rock the boat and, to mix metaphors, he was in the firing line. He rightly asked why we should upset an established tradition of external typesetting which, as far as he was concerned, worked well. The Administrator had never had to bear the consequences of an individual author's frustrated typesetting resubmissions, the authors having previously suffered in silence, yet he was bearing the brunt of the secretaries' complaints. To him TeX brought no improvements.

The argument to use is cost effectiveness. We needed to identify areas where TeX would be of benefit to the production of main-stream Institute publications. A major vehicle in the TeX PR battle was the publication of a comprehensive catalogue of the Wellcome Library's Tibetan collection entitled *Tibetan Manuscripts, Xylographs and Thankas in the Wellcome Institute Library*. The work had commenced in 1978 under the auspices of the collection's former curator, Miss Marianne Winder, and had been continued by her after her retirement.

The project was being funded by the Trustees of the Wellcome Institute, but there was some fear that the funding would disappear if the catalogue was not brought to publication before the end of the 1989 financial year, which fell in September.

We took this on as a \TeX project in the summer of 1988. In many ways we had to start practically from scratch. Text had started to be input on the DisplayWriter by Miss Winder herself and had subsequently been transferred into DW3. We converted all the DW3 text into WordPerfect in order to be able to use various inputting techniques I had devised to speed up the procedure. I can tell you that at this point Miss Winder was nearly having kittens: she had spent years painstakingly typing in all the entries, devising a form of codification that she hoped a typesetter would transform into Tibetan script and here we were practically stripping the whole thing bare.

The ability to sit down with the author and discuss the desired layout is an essential production process. We are fortunate to benefit from having a professional artist, Huw Geddes, on our staff as Exhibitions Designer experienced in catalogue layout, who was able to advise us in this area. The format for the manuscripts and xylographs section is more complex than that of the thankas, banners and paintings. Therefore a crucial stage in the undertaking of this project was in the creation of the different style files. This is where someone with an detailed knowledge of \TeX is essential, in this instance Dominik. Using his splendid \LaTeX style files, the results illustrated in Figures 3 and 4 (Appendix) were produced.

Using this catalogue as an example, I shall outline to you the steps I then took to make the \TeX inputting procedure less of a burden for the secretaries.

Firstly, PC \TeX provides a simple menu facility which allows you to switch easily between your editor, the \TeX program itself, the printing program and the screen previewer. I strongly recommend installing a screen previewer as it is useful for checking page-breaks and formatting integrity. We use MaxView. This is not a particular recommendation for or against this previewer as I have not undertaken a detailed evaluation of previewers; suffice it to say that it serves our requirements.

The PCTEX.CFG file, which sets up the menu, can be customised to use a particular editor or word processor, along with certain options. I configured the file to run WordPerfect with the /m- macro command line option. When you use this option as you start WordPerfect, the program immediately

executes a macro name, which you specify after the /m-. The main \TeX text file for the manuscripts section of the catalogue was called TIB.TEX. I created a WordPerfect macro, TIB.WPM similar to the file name, consisting of the following key strokes:

```
{DISPLAY OFF} {Text In/Out}12
d:\winder\mscript\tib.tex
{Enter} {Setup}6s{Enter}
```

As one wishes to work with ASCII files for \TeX this macro retrieves the file called TIB.TEX in DOS text mode. The macro then goes on to change to an alternative keyboard layout called SANSKRIT. Here the Sanskrit and Tibetan diacritically marked characters as well as various frequently used words such as

```
{it}={A}rya\vajra\cchedik\={a}%
praj\~{n}\={a}\p\={a}ramit\={a}}
```

for *Āryavajracchedikāprajñāpāramitā*, were mapped onto CTRL-key combinations. The ALT-key combinations had WordPerfect macros relating to the \TeX formatting macros, e.g., `\begin{physical}` `\end{physical}`, assigned to them, thereby retaining a conceptual consistency in macro calling for the secretary. I should just like to bring to your attention the usefulness of using the pause feature within the macro for paired commands or commands requiring more than one argument. The WordPerfect Save key F7 had another macro called DOS.WPM assigned to it. When the secretary came to save the document the macro changed the keyboard layout back to the original, used the CTRL-F5 DOS Text save with a pause to confirm the document name including a .TEX extension, retained the prompt to confirm overwriting, then used F7 to exit the document without saving it as a WordPerfect file.

Within the PCTEX.CFG file:

```
%E=wp /m-%s
%C=tex &lplain %s
%V=view %s
%P=makeps %s
%T:Print:
%T=makeps %s
%C:LaTeX:
%T:Print:
/PT=d:\pctex\textfms
/PF=d:\pctex\textfms
/PI=d:\pctex\texinput;d:\pctex\latex
/K
```

the %s stands for the "string" file name that you would normally type after wp. To run the PC \TeX menu the user types `pctex` and the file name at the DOS prompt. Therefore, in this case, the command `pctex tib` ran WordPerfect with the TIB.WPM macro with the results described above. By utilising these features the secretary remains

within a familiar WordPerfect environment and has far fewer keystrokes to input. Alternatively, if you do not wish to install PC T_EX on the secretarial PCs then invoke the macro with `wp tib`.

The onerous burden, as she saw it, of T_EXing and printing the text is removed from the secretary and centralised — with me. This does seem to make the procedure of going through the PC T_EX menu redundant. The important concept to remember is that if the secretary wishes to become more involved, e.g., by using the previewer, then she has that choice. The more people on site with T_EX competence gained through motivation, the better. Any reasonably sized document should be split into logically manageable chunks, usually by chapter, for inputting. The whole, or its parts, can then be processed and printed by using a main text file (Figure 2, Appendix), which initiates the document style and uses the `\input` command for the relevant chapter. By centralising the processing and printing one can justify the expense of investing in a very fast machine with lots of memory, thereby saving valuable time and freeing secretarial PCs for inputting. For instance, I have a DELL 386 with a 90MB drive and 2MB RAM at the moment. It also means that there is a focal point for all queries and developments, of which there are many. The developer of the application should be aware of the output. Centralisation also encourages standardisation in error correction procedures. If the macros are well designed, there should not be many problems arising from the inputting, particularly with that old chestnut — the missing command parameter.

Having gone through this procedure, we sent off the 300 dpi laser printer camera ready copy to our printer, the result being a hardback catalogue which sells for £10 (approximately \$16) per copy. Because of the complexity of the typesetting, it is unlikely that the catalogue would have sold for less than £50 using conventional methods. I think the powers that be were suitably impressed, and Marianne Winder was ecstatic that she was able to hold in her hand the culmination of a lifetime's work. I would challenge any other 'DTP' package to turn out work of this calibre.

By now we had proven cost effectiveness and had convinced some of our authors that T_EX was a good thing, with the immediacy of correction and production virtually under their noses saving them time and anxiety. We had even produced catalogues for the various exhibitions held within the Institute, but The Powers That Be were still not convinced that T_EX could be a viable alternative

to external typesetting. The problem lay in the quality of the output. We were dependent on laser printed camera-ready copy which, to be honest, at a resolution of 300 dots per inch looked distinctly sploidy to the eyes of those more accustomed to commissioning *professional* publications. And then there was the Computer Modern Roman font. To our academics, librarians and curators steeped in tradition it was not universally popular. They wanted Times Roman at the very least.

Fortunately, along came PostScript which, with its ever extending font families, became another de facto standard and opened up new horizons for in-house electronic publishing. Linotron then put PostScript fonts onto their phototypesetters. This means that if you have a PostScript laser printer you can design and proof your document at 300dpi, then send the file to the phototypesetter confident in the knowledge that you will receive a bromide at 1270dpi with the exact same line and page breaks. The fact that we had Hewlett-Packard printers which are not PostScript was not too great a setback as we were able to purchase JetScript which is a PostScript enhancer for the HP Series II. However, I believe that the HP Series III will soon be provided with a PostScript cartridge.

We were keen to take advantage of this new potential, particularly as the University of London Computer Centre had set up a phototypesetter service using a Linotronic 300. The T_EX world did not ignore these developments either. Several DVI to PostScript drivers began to appear. We use one, DVItOPS, designed by James Clarke which we obtained via the Aston Archives (detailed below). There are also commercially available drivers. The two I know of are from PC T_EX and ArborText.¹

DVItOPS has a file `dvitops.fnt` in which you substitute the name of the font as it is known to T_EX (with the extension `.tfm` removed) with the name of the font as it is known to PostScript, for example Times-Roman. DVItOPS also permits the inclusion of PostScript graphics in the document.

Because of the Institute's links with University College London we have access to their EUCLID system. Through the simple expediency of purchasing a modem (a Hayes Smartmodem 2400) and a good communications package (PROCOMM), I am, from my PC, able to link into JANET (the Joint Academic NETWORK) and thus the whole world. More specifically I am able to link to the ULCC Phototypesetter

¹ Philip Taylor in *ImageSetting*, the Phototypesetter User Group magazine of the ULCC, lists several drivers.

service. As KERMIT is bundled with PROCMM I am able to log into EUCLID, activate KERMIT on my PC, call up the ULCC Typesetter service via one of EUCLID's PADS, activate KERMIT at that end and have it transfer my laser printer proofed PostScript files to the ULCC VAX. The DVIToPS program has already enabled me to specify the required 1270dpi resolution for the typesetter output as opposed to 300dpi for the laser printer. ULCC then run my file through the Linotronic 300 and within two days I receive the bromides by post at a cost of approximately £1 per page. In Figure 4 (Appendix), you will see reproductions of identical catalogues we produced for two exhibitions at the Institute. The 1987 version used laserprinter camera ready copy with Computer Modern Roman fonts, the 1989 version used 1270dpi bromide camera ready copy with a Times Roman font. We were able to use the same input file with minor modifications. The beauty of T_EX is that you struggle once, then use the fruits of your labour over and over again.

WordPerfect Specifics

I have not gone into great detail describing how to create keyboard layouts and macros, but refer you to Anita Hoover's excellent paper on this subject in last year's proceedings. I have given an example of the use of an alternate keyboard layout above. There are, however, a few procedures which I feel can be usefully elaborated.

Firstly, a brief definition of the terms *alternate keyboard layout*, *macro* and *style*. With an alternate keyboard layout it is possible to change action performed by keys or combinations of keys. Moreover, you can create several keyboard definitions for different purposes. A macro is a file you create to represent a whole series of keystrokes. You can create a macro to perform nearly any task that you could accomplish with a series of keystrokes. This *series* can then be assigned to one key only in an alternate keyboard definition. The activation of the alternate keyboard definition can also be assigned to a macro. Style is a powerful tool for controlling the format of an individual document or a group of documents. You define and name a style, and then when you want to use that style, you select it from a list. Again, macros can be used to search for and replace a particular style with another. The combination of these three features allows an incredible amount of flexibility for T_EX conversion. One example is to create a macro in which you create a paired style for the `{\bf` and `}` formatting codes. This

macro can then be assigned to the standard WordPerfect function key for activating bold, F6, under an alternate keyboard layout designed for T_EX inputting. A macro can be designed to activate that keyboard whenever a document is to be in T_EX. In this way any secretary will be using familiar keys to activate a bold command, but the output will be a T_EX command.

The complex nature of the documents I have used in my examples so far has required the necessary T_EX command sequences to be input with the text. For straightforward content it is possible to allow the secretary to input using WordPerfect's formatting codes but involving the use of WordPerfect's *style* and *macro* features. As I have mentioned, a feature of *style* is the ability to change the formatting codes within a document. Again using the bold example, the beginning BOLD command is replaced with `{\bf` and the end bold with `}`. To create a paired style like this you go into a Paired Codes screen where your cursor is placed before a comment box. This box represents the text that is surrounded by the on and off conditions for the style. You enter the control sequences for the beginning of the style before the comment box, `{\bf` in this example, then move the cursor down to below the box and type in the sequences for ending the style `}`. An Alt macro can also be created to activate this style.

By using Block you can then replace existing codes in your document with this new *style*. To do this you complete the following steps:

1. Place the cursor at one end of the block you want to define.
2. Press Block (Alt-F4, or F12).
3. Highlight the block.
4. Press Style (Alt-F8).
5. Use the cursor keys to highlight the style you want.
6. Select On (1).

All of these steps can, of course, be stored in a single key macro.

In the academic world many documents contain footnotes. With T_EX the footnote text is included in the main body of the text, which can often cause problems with the authors when they are proofing and editing. Footnotes change their reference points or are taken out altogether, thus altering the relative numbering. WordPerfect's footnoting feature offers the secretary a relatively painless means of accomplishing this task. This is how to replace a WordPerfect footnote with a L^AT_EX footnote. Go to the top of the document as WordPerfect macros are

created by example. Use `Ctrl-F10` to create the macro. It can be called `Alt-F`. Use `F2` to search forward for the footnote option code `Ctrl-7`. Enter 1 for Footnote and 1 again for Note. At this point the prompt indicates `Search~ [Footnote]`. Press `F2` to activate the search. The cursor will be placed to the right of the footnote number, therefore move it one space to the left over the number. Type in `\footnote{` which will go to the left of the number. Then press `Ctrl-F7`, 1 for the footnote option and 2 and `Enter`. This takes you into the text of the footnote number on which the cursor is currently placed. Use WordPerfect's `Home Home \downarrow` combination to take you to the end of the text no matter how long it is. Type in the `}`. Use `Home Home \uparrow` to go to the beginning of the text, just to the left of the number. `Alt-F4` to start blocking the footnote text, `Home Home \downarrow` to highlight the blocked text, then `Ctrl-F4`, 1, 1 to move the block. At this point press `F7` to exit the footnote text back to the main body. Press `Enter` to retrieve the text, which will be inserted just after the `{` of `\footnote{`. Move the cursor one space to the left and press `Delete`, `Y` in order to delete the WordPerfect footnote number. To cause the macro to repeat itself until no further footnotes are found press `Alt-F` again. Ignore the `ERROR message: ALTF.WPM not found`. Press `Ctrl-F10` to exit the macro definition. To run the macro press `Alt-F`, and away it goes. Note that in the \LaTeX environment there is no need to insert footnote numbers. These are incremented automatically.

Conclusion

Before beginning to learn \TeX , I would recommend the apprentice \TeX er (or perhaps \TeX an) should use \LaTeX , a macro package developed by Leslie Lamport, as a starting point. If you are happy to stick with his design specifications, then beautifully turned out documents can be produced relatively painlessly. \LaTeX can give you the basic feel of \TeX and as you become more experienced and confident in its use you can tweak it by adding macros of your own. I should warn you to expect a few glitches at this point.

I would advocate investing in back copies of *TUGboat*. There has been a wealth of \TeX and \LaTeX experimentation, experience and developments over the past few years which have been documented in this publication. If you are able to link into an academic email network then subscribe to \TeX hax, and/or UK \TeX in the UK, where user problems and solutions are aired. These bulletin

boards are not for the real beginner as all levels of problems are intermixed, often with some prior knowledge being assumed.

I have always been concerned that nonacademic users are often cut off from developing \TeX 's potential because of the difficulty in obtaining information if one is not on the network. When we first began with \TeX we were in this situation. We were, and are, an IBM-PC compatible environment, therefore we bought PC \TeX , thus gaining a commercial company's technical support (admittedly somewhat stretched at times across the Atlantic) to help us overcome initial implementation problems. To nonacademic users I would say look at the commercial options in the PC or MAC worlds. It is not absolutely necessary to be linked to electronic mail. Peter Abbott at Aston University holds a repository of all \TeX related developments which can be obtained on disk as well as downloaded. He will also send you printed copies of UK \TeX .

I confess that I am not fully aware of all the latest developments in the \TeX world as other responsibilities unfortunately demand my attention. I am sure that more sophisticated set-ups can be achieved. In fact, I hope that people will stand up and tell us what other gems are available. My intention in this paper has been to prove by my example that any Noddy, with the help of Big Ears, can achieve quite a lot. Your Big Ears is this \TeX community, which has never hesitated in sharing its hard-earned experience. Don't be afraid to ask.

Bibliography

- Hoover, Anita Z. "Using WordPerfect 5.0 to Create \TeX and \LaTeX Documents" *TUGboat* 10(4), pages 549-559, 1989.
- Knuth, Donald E. *The \TeX book*. Reading, Mass.: Addison-Wesley, 1984.
- Lamport, Leslie. *\LaTeX : A Document Preparation System*. Reading, Mass.: Addison-Wesley, 1986.
- MacKay, Pierre A. "Typesetting Problem Scripts". *BYTE* 11(2), pages 201-218, 1986.
- Stewart, et al. *Using WordPerfect 5*. Carmel, Indiana.: Que, 1988

Appendix

Works on Bhakti

Dvādaśākṣarīkośakārikā. -- AD 1839 Serial no.1
leaves 7r-8v: paper. -- In Sanskrit. -- Copied by Suphagaṇa(?).
-- Date of copying: 14 kṛṣṇapakṣa of Bhādrapada, saṃ 1896. -- Copied
in Indraprastha, Jasapatavāida(?). -- Bibliography: not in NCC; not
the same text as MS ABC 199, nos.6431-6435. -- Complete in 12
verses. Explains that the meaning of the letters
n,m,bh,g,v,t,v,s,d,v,y is namo bhagavate vāsudevāya. -- With i)
Bhūtabhaviṣyatipraśna. -- Devanāgarī script.
Shelved at α 971 (ii).

Figure 1: Excerpt from first published handlist

```
% This is TIBET.TEX, the main text file for Marianne Winder's
% catalogue of the Tibetan manuscripts in the Wellcome's Oriental
% Collections.
\documentstyle[tib]{book}
\pagestyle{myheadings}
%Working title page information
\title{A Catalogue of Tibetan manuscripts and Xylographs, \
and a Catalogue of Thankas and other Paintings and Drawings\
in the Library of\
Wellcome Institute for
the History of Medicine}
\author{by Marianne Winder}
\date{{\large{\tt Draft proof of \today}}}}
%% The document itself
\begin{document}
\maketitle %Comment *in* for final run
\markboth{{Manuscripts and Xylographs}}{Manuscripts and Xylographs}
\input{tibtitle}
\input{copyright}
\input{forewrd}
\input{mstit2}
\input{mstoc}
\input{intro}
\input{abbrevtn}
\input{principl}
\input{tib}
\input{bib}
\input{mstitls2}
\input{shelf}
\end{document}
```

Figure 2: Main text file for T_EXprocessing

Incantations against evil and diseases

Wellcome Tibetan 11b

- 1 Manuscript; 11 × 34(7 × 29) cm.; ff. 30; 5 lines to a page; dbu can; gold and silver (i.e., yellow and white) writing on dark blue paper; painted boards 11½ × 34 cm. and leather strap.

Ff. 8–12: **rdo rje rnam par 'joms pa'i gzuñs,**
Vajravīdāraṇānāmadhāraṇī, “Incantation of all conquering indestructible reality”, [religion, ritual, incantation]. *Tripitaka* 406,8
Tripitaka 574,11 translated by Jinamitra, Dānaśīla and Ye-śes-sde.

Purchased at Sotheby's, 31.10.1933.

Wellcome Tibetan 21

- 2 Xylograph; 78½ × 51½ cm.; broadsheet; 7 lines to a page; dbu can; three woodcuts.

stag señ om ā hūṃ ... om ma ṇi pad me hūṃ hrī khyuñ 'brug,

“Tiger, lion, om ā hūṃ, om ma ṇi pa dme hūṃ hrīḥ, ... garuḍa, dragon”, [ritual, mantras].

Print used on prayer flags. Tiger, lion, garuḍa and snake are the four conquerors of evil forces located in the four directions. — Previous owner L. A. Waddell.

— Purchased at Sotheby's, 29.11.1920.

Wellcome Tibetan 36

- 3 Manuscript; 9 × 24½(7 × 22) cm.; ff. 27; 8 or 7 lines to a page; dbu med, & can; black and red ink on white paper, diagrams; strong brown paper covers 9 × 24½ cm.

Incipit in centre of f. 1v: **sgal tshigs gser gyi,**
 “The golden spine”, [ritual text with mantras].

Folios sewn together except f. 25 which is separate. — The illegible beginning of the MS is on the brown paper cover. — Folio 24, before *dbu can* script begins, is blank. — Previous owner Kohser Temple, Lahore, 1871.

— Purchased at Stevens', 31.5.1907.

Wellcome Tibetan 37

- 4 Manuscript; 6 × 22½(4½ × 19) cm.; ff. 113; 6 lines to a page; dbu med; black and red ink on white paper; wooden, slightly carved boards 8 × 23 cm.

Figure 3: Sample page

Case 6

ORIENTAL COLLECTION

The collection of oriental manuscripts and printed books – comprising over 11,000 manuscripts and some 3,000 printed books in 43 different languages – is one of the most important in Europe. While medical history is central to the collection, many cognate topics are represented. Variety of subject matter and language is matched by diversity of medium. Besides paper and vellum, the collection includes manuscripts written on bamboo, bone, ivory, metal, tree bark and palm leaf. This small display indicates something of the diversity and variety of the collection.

1. Amulets

Amulets were employed to protect man or his possessions from evil influences, including illness. The amulet is found in the East and in the West, among both tribal and settled peoples; and it exists to the present day. Assyrians and Egyptians, Greeks and Romans, Jews and Christians, fostered this ancient tradition – which, among the Jews, has a history of some three thousand years. Three Hebrew medical amulets are displayed:

- i. **Amulet for a fruitful marriage.** c.17th century; written in Italy in iron gall ink on paper.
- ii. **Amulet for the protection of Bela daughter of Rachel from plague.** c.18th century; vellum.
- iii. **Amulet for the protection of Moses David son of Esther from plague.** c.18th century; vellum contained in parchment case.

2. Medical notebook

This beautifully copied Hebrew manuscript, probably the notebook of a physician called Elhanan (f.11v), contains marginal annotations. Patients are named, including Moses, the writer's son (f. 10r), and Dulcita his wife (ff. 15v & 16v). The opening shown includes a remedy for pain in the ilium. Copied c.17th/18th century, in a fine Italian hand.

3. **Birkot ha-milah u-minhag wa-sepher ha-milah ke-phi ha-nahug ba-z'ot ha-kehillah.** London.

'Blessings of circumcision and the conduct and service of circumcision as it is led in this congregation ... London'. This finely executed Hebrew manuscript was copied by Isaac Luria in London during the late 18th or early 19th centuries: it lays out the form of service for the rite of circumcision to be followed by a London congregation.

Case 6

ORIENTAL COLLECTION

The collection of oriental manuscripts and printed books – comprising over 11,000 manuscripts and some 3,000 printed books in 43 different languages – is one of the most important in Europe. While medical history is central to the collection, many cognate topics are represented. Variety of subject matter and language is matched by diversity of medium. Besides paper and vellum, the collection includes manuscripts written on bamboo, bone, ivory, metal, tree bark and palm leaf.

1. Amulets

Amulets were employed to protect man or his possessions from evil influences, including illness. The amulet is found in the East and in the West, among both tribal and settled peoples; and it exists to the present day. Assyrians and Egyptians, Greeks and Romans, Jews and Christians, fostered this ancient tradition – which, among the Jews, has a history of some three thousand years. Three Hebrew medical amulets are displayed:

- i. **Amulet for a fruitful marriage.** c.17th century; written in Italy in iron gall ink on paper.
- ii. **Amulet for the protection of Bela daughter of Rachel from plague.** c.18th century; vellum.
- iii. **Amulet for the protection of Moses David son of Esther from plague.** c.18th century; vellum contained in parchment case.

2. Medical notebook

This beautifully copied Hebrew manuscript, probably the notebook of a physician called Elhanan (f.11v), contains marginal annotations. Patients are named, including Moses, the writer's son (f. 10r), and Dulcita his wife (ff. 15v & 16v). The opening shown includes a remedy for pain in the ilium. Copied c.17th/18th century, in a fine Italian hand.

3. **Birkot ha-milah u-minhag wa-sepher ha-milah ke-phi ha-nahug ba-z'ot ha-kehillah.** London.

'Blessings of circumcision and the conduct and service of circumcision as it is led in this congregation ... London'. This finely executed Hebrew manuscript was copied by Isaac Luria in London during the late 18th or early 19th centuries: it lays out the form of service for the rite of circumcision to be followed by a London congregation.

4. **Sharḥ Qānūnca.** 'Commentary on K. Qānūnca,' a resumé by al-Jaghminī of K. al-Qānūn.

K. Qānūnca, a once popular medical work written by Maḥmūd b. 'Umar al-Jaghminī [d. 1344]. The Arabic commentary shown here was written by 'Alī b. Kamāl al-Dīn Maḥmūd Muḥammad Ṭāhir of Constantinople. It is transcribed in the Naskh style and dedicated to the Ottoman Sulṭān, Bāyazīd Khan b. Muḥammad Khān b. Murād Khān.

Problems on the T_EX/PostScript/Graphics Interface

Robert A. Adams

Dept. of Mathematics, The University of British Columbia, Vancouver B.C. Canada V6T 1Y4
Bitnet: useradms@ubcmtsg, Internet: useradms@mtsg.ubc.ca

Abstract

This paper discusses several problems which arose in the process of using T_EX and PostScript together to produce two calculus textbooks. Three of these problems were particularly important. The first was getting a reasonable combination of PostScript (scalable) text and math fonts that looked “good” in 1270 dpi output from a Linotronic phototypesetter. The second was devising a practical method for getting suitable (and well aligned) two-colour separation for text and graphics. The third involved incorporating T_EX labelling in PostScript graphics. Solutions to these problems were largely dictated by the software available at the time the solutions were needed, about one and a half years ago.

Background

T_EX was designed to produce beautiful books, especially ones which contain mathematical formulas. It is therefore natural to choose T_EX to typeset a calculus book, but calculus books require numerous diagrams which themselves have mathematical formulas for labels. It is fairly easy to produce even very complex mathematical diagrams in PostScript, either directly or indirectly using high-level software which generates PostScript code. Therefore it is also natural to produce a calculus book in a PostScript environment.

During the past two years I have been involved in many aspects of the production of two calculus textbooks, *Single-Variable Calculus*, and *Calculus: A Complete Course*, both published in Canada by Addison-Wesley. Besides writing these books, I was responsible for all the typesetting, and the construction of all the macros necessary to implement a book design. Many (but not all) of the design elements were specified by a professional book designer.

Anyone who has ever authored a textbook using any system will know what a monumental job that can be. Knowing what information you want to present, and how you want to present it, is only a small part of the task. Getting a respectable typescript copy in the days before personal computers, word processors and computer graphics packages usually meant more hours at a typewriter or drawing board, or preparing and editing handwritten

copy for a typist and artist, than the author actually spent composing the material. Such was the state of affairs when I wrote the first edition of *Single-Variable Calculus* for Addison-Wesley in 1981–1982. It was my second book done by the *old method*, and I resolved at the time never to write another book! Then in 1984 my Editor sent me Addison-Wesley’s newly published MicroT_EX, and a copy of *The T_EXbook*, and my life was changed forever. He wanted a review of MicroT_EX. He got a review, and another book, *Calculus of Several Variables* (Addison-Wesley, 1987).

At that time we were still using Almost Modern fonts, and this author, at least, had never even heard of PostScript. The typesetting was successful enough, though Addison-Wesley (Canada) and I were both feeling our way as far as design was concerned. The problem of two colour separation came up, but was not adequately solved. In the end the production department got out the scissors and glue, and the separation was done *a posteriori* without the help of a computer. The diagrams were all redone by a graphics professional on a Macintosh from plotter copy I supplied, and they left a lot to be desired. Moreover, there were several serious colour alignment errors in the final book, which arose from the fact that the alignment of black and second-colour components for the figures were performed by someone who did not fully understand the devastating consequences of even a minor misalignment in a complicated two-colour

figure. I decided at that time that if I ever did another book, I would try to have a system in which I could produce most of the figures and integrate them directly into the production files myself.

By the time Addison-Wesley had raised the issue of a new edition of *Single-Variable Calculus*, (mid 1988), I had become reasonably familiar with PostScript and had acquired a 300dpi PostScript printer for my PC system. I had also developed a preliminary but useful version of a two- and three-dimensional mathematical graphics program MG which produced the kinds of figures I use in my books. (I had used that software to generate rough plotter copy for the several variable book.) There were, however, some problems which still had to be solved. Of these, the most important were

- getting suitable fonts to use with T_EX for doing mathematical typesetting in a PostScript environment.
- devising a simple system for getting two-colour separation in text and graphics.
- getting T_EX labels into my figures.

I will deal with each of these in turn.

The Font Problem

At the outset I should say that my T_EX setup two years ago consisted of an AT clone with colour EGA monitor, Addison-Wesley's MicroT_EX, and (version 4.0 of) ArborText's PREVIEW and DVILASER/PS driver programs. I had constructed a modified version of `plain.tex` called `psplain.tex` which used a hybrid of PostScript Times fonts for text mode material and Computer Modern math fonts for math mode material, with a few other minor modifications to clear up some problems which arise from the fact that T_EX manufactures some symbols such as “≠” using elements from text and math families.

Copy at 300 dpi resolution obtained from `psplain` looked fine to me, and to my editors. However, a sample generated on a 1270 dpi PostScript phototypesetter exposed the first serious problem. While the text mode material came out at 1270 dots per inch, the math mode material was still at 300 dots per inch, because DVILASER/PS had downloaded raster patterns for the `cmmi`, `cmex`, and `cmsy` fonts into the PostScript file. I suppose we could have tried to obtain 1270 dpi versions of those fonts, but I had no access to METAFONT.

About that time, I was given a copy of some PostScript (scalable) versions of these fonts produced on a Macintosh using the FONTOGRAPHER program, so we tried them. The combination again

looked good at 300 dpi, but at 1270 dpi a new problem became apparent. The CM math fonts have considerably less weight than the Times family of text fonts. Here is a sample formula involving characters from both families. It is magnified (to 20 points) to show the difference in weights.

$$\max\{a^j, x_k\} > \cos \psi$$

The combination would not do at all. At this point I would have given a good deal for a working version of John Hobby's *MetaPost* program [Hobby, 1989], or any other program that would produce PostScript outline fonts from METAFONT descriptions. I had a preprint copy Leslie Carr's paper [Carr, 1988] on converting METAFONT logfile output into a PostScript font description, but I was certain I was not a good enough programmer to implement it, at least not quickly.

A solution for this problem was finally found, and it was definitely a hack. The FONTOGRAPHER program generates the characters of a PostScript outline font in a coded format which is preceded in its output file by a PostScript prolog with definitions which enable the PostScript interpreter to understand the code and construct the character in the printer's memory. Being machine produced PostScript, even these definitions are a bit hard to read, but after some study I was able to conclude that the character outlines were merely being *filled* (with black) rather than *stroked* with a PostScript pen. Lines 10 and 11 and 22 in this prolog began

```
/Fill{{fill}Cfill}def
/Eofill{{eofill}Cfill}def
```

```
/StrokeWidth 0 def
```

I altered the definition of the `Fill` and `Eofill` operators being used so that in addition it *stroked* the outline with a pen of a prescribed thickness. After some experimentation, I determined that the thickness should be about 0.22 points for a nominal 10 point font. Thus, the PostScript prolog for the outline fonts `cmmi`, `cmsy`, and `cmex` was modified to become

```
/Fill{{gsave fill grestore stroke}
Cfill}def
/Eofill{{gsave eofill grestore
stroke}Cfill}def
```

```
/StrokeWidth 22 def
```

(The `StrokeWidth` variable is measured in thousandths of the nominal design size of the font.) Off went another test to the phototypesetter, this time successfully. Here is a sample of the output with the same mathematical formula shown earlier.

$$\max\{a^j, x_k\} > \cos \psi$$

I'm sure that experts in font design would find any combination of two such different typefaces as Times and Computer Modern aesthetically unsatisfactory, but the average mathematics student or instructor, and maybe even the average editor, does not. As so often happens in the real world, there was a problem which needed an immediate solution. While not ideal, an acceptable solution was found.

The Colour Separation Problem

Most textbooks these days use two or even four or more colours to achieve greater visual impact. Several queries about how to accomplish this with TeX have appeared in TExhax in the last few years (one was from me), and I have never seen an adequate response. Of course, in a sense S_LTeX has solved the multi-colour problem by using blank fonts (which have TFM files corresponding to those of printing fonts but themselves print only blank characters. It has, however, never been clear to me how to obtain (or construct) such blank fonts. There is also the TeX `\phantom` command, but I'm not sure how that would react to a pagefull of text to be blanked out. (I admit, I've never tried.)

Ideally, one would like to arrange the following situation for two-colour separation. There should be defined two control words, `\black` and `\red` say, so that you would insert one of these words in the TeX source code at points where you wanted to switch from red to black or from black to red. There should also be defined at the beginning of the TeX source two Boolean controls, `\printblack` and `\printred`, which should be set to *true* or *false* according to whether "black" output, "red" output, or both combined is desired. There remains, however, the problem of how to get the nonprinting colour to leave blank areas on the page exactly corresponding to the material that would appear if it were printing.

I still do not know how to solve this problem using TeX, but there are fairly easy PostScript solutions. PostScript has a `setgray` operator which determines the gray-level of printing. Thus 0

`setgray` causes printing in black; 1 `setgray` causes printing in white, i.e. no printing at all unless the background is not white. Numbers between 0 and 1 result in different levels of gray. Define `printblack` and `printred` as PostScript Boolean variables which you set to true or false according to whether you want either or both colours to print. Then have the TeX `\black` and `\red` commands insert PostScript operators `black` and `red` respectively, into the PostScript file via `\special` commands to the PostScript driver. The PostScript operator `black` could be defined as 0 `setgray` if `printblack` is true, and 1 `setgray` if `printblack` is false. A similar definition is made for `red`.

The above solution works well for text (e.g. headings, boxes and such items where black and red are never overlaid), and it is clearly generalizable to more colours. However, it poses problems for graphic material. PostScript is designed so that graphic elements plotted later always obscure ones plotted earlier in regions of overlap. For example, in a figure where a red curve (or pink shaded region) crosses or overlaps an earlier plotted black curve (or gray shaded region), the red element will blank out those parts of the black element where it overlaps. This is not what you want! In the final copy black ink is quite opaque, red less so, and light shades of pink or gray are not at all opaque and should not blank out one another.

The solution to this problem was to redefine the PostScript operators `black` and `red` so that, depending on the values of `printblack` and `printred`, each translates the PostScript origin some large distance in one direction or another. This causes printing of the undesired colour to occur well outside the boundaries of the physical page, and thereby leaves the printed elements intact. The PostScript driver DVILASER/PS can insert some PostScript prolog code of its own at the beginning of the PostScript output file it creates from a TeX dvi file. In my installation, that prolog code begins

```
% SET THE FOLLOWING BOOLEAN SWITCHES true
% FOR WHICHEVER "COLOUR" OF OUTPUT IS
% DESIRED. SET BOTH TO true TO PRINT
% BOTH COLOURS SIMULTANEOUSLY. DO NOT
% SET BOTH SWITCHES false AT THE SAME TIME
%
/printblack { true } def
/printred   { true } def
%
/rred { printred {0 setgray} {1 setgray}
      ifelse } def
/bblack { printblack {0 setgray}
          {1 setgray} ifelse } def
%
/black { bblack firstswitch
```

```

{/doingblack {true} def
printblack not {5000 5000 translate
/firstswitch {false} def} if }
{ doingblack not {5000 5000 translate
/doingblack {true} def } if }
ifelse } def
%
/red { rred firstswitch
{/doingblack {false} def
printred not {-5000 -5000 translate
/firstswitch {false} def} if }
{ doingblack {-5000 -5000 translate
/doingblack {false} def } if }
ifelse } def
%
/setoldgray { currentgray dup
/oldgray exch def } def
%
/restoregray { oldgray setgray } def
%
/fixblack { setoldgray pop 0 setgray } def
%
/maxgray { dup /newgray exch def
setoldgray ge {newgray} {oldgray}
ifelse setgray } def

```

These definitions are a little complicated, probably more so than absolutely necessary to achieve the desired effect. The definitions of `setoldgray`, `fixblack`, and `restoregray` are made to facilitate the printing of crop marks, registration crosshairs, and manuscript header information outside the margins of what will be the final trimmed page, on all printed pages regardless of the settings of `printblack` and `printred`. The operator `maxgray` is useful when colour separating shaded figures.

These PostScript definitions are accessed in the \TeX source code by means of the following control words defined at the beginning of the macro file containing all the macros for the book.

```

% SVC-PS.TEX
%
% Format for Calculus Book
% (PostScript Version)
% R. Adams revised 15 Dec 89
%
% first some defs to set up
% Postscript for two colours
\def\red{\special{ps:: rred }}
\def\black{\special{ps:: bblack }}
\def\fixblack{\special{ps::
  bblack fixblack }}
\def\restoregray{\special{ps::
  restoregray }}
\def\logo{\hbox to17pt{\special{ps::
  rred logo bblack}\hfil}}
\def\regmark{\hbox to60pt{\special{ps::
  regmark}\hfil}}

```

Here `logo` and `regmark` are PostScript procedures which produce a logo character for use in section headings in the book, and the registration crosshairs

mentioned above. To illustrate the use of `\red` and `\black` in the \TeX source, here is the definition of the macro `\examp` used to introduce examples in the book.

```

\long\outer\def\examp #1\par{\penalty-200
\vskip 12pt plus 2pt minus 2pt
\global\advance\itemno by1
\noindent\llap{\exampfont\red EXAMPLE
\itemlabel\hskip1pc\black}}#1}

```

The word “EXAMPLE” and its label number are printed in red in the left margin. The `\makeheadline` macro illustrates the use of `\fixblack` to ensure that the crop marks `\ulc` and `\urc`, and headline material outside the crop boundaries print regardless of which colour is printing. Exceptions are the words “black” which is printed only if black is printing, and “colour” which prints only if colour is printing.

```

\def\makeheadline{\vbox to Opt{%
\vskip-82pt\hbox to\pagewidth{%
\fixblack\kern-196pt\copy\ulc
\quad\raise12pt\hbox{%
\figfont ADAMS:
Single-Variable Calculus
Chapter \number\chapno\ -- page
\number\pageno\quad
\red colour \black black
\fixblack\quad\today}\quad
\raise12pt\regmark
\hfill\rlap{\kern42pt\copy\urc}
\restoregray}
\vskip26pt
\hbox to\pagewidth{\the\headline}\vss}
\nointerlineskip}

```

This system for colour separation works well. In PostScript code for figures, one inserts the PostScript operators `red` and `black` where colour changes are desired.

\TeX Labels on PostScript Graphics

In a calculus textbook mathematical graphics, both two and three dimensional, are a very important tool for presenting information and making it intelligible to the student. Most such graphics require labels involving mathematical formulas, sometimes almost as complicated from a typesetting point of view as the formulas appearing in the text. It is therefore very helpful to be able to use \TeX to label figures. Most commercial software programs which produce mathematical graphics do not support this capability yet. On the other hand, programs designed specifically for doing graphics within a \TeX environment are not of sufficient sophistication to produce the quality of mathematical graphics which can be generated by writing PostScript code.

I wanted the best of both worlds, and it seemed necessary in this instance to do some actual programming to produce a mathematical graphics package which would produce both PostScript output for the graphic and, simultaneously, T_EX labelling information. Over several previous years I had developed, using Turbo Pascal, a graphics program, MG, which produced a variety of two-dimensional plots of functions and equations as well as lines, vector fields, freehand spline curves and such, and was also able to produce three-dimensional diagrams of curves and surfaces ruled by families of curves. The program produced HPGL output, because I happened to have a Hewlett Packard plotter at the time. Such output was not of suitable quality for publication.

About two years ago my colleague, Dr. Robert Israel (Mathematics Department, The University of British Columbia) took over the MG project and redesigned the user interface, making the program much easier to use, and at the same time much more functional. Meanwhile, I altered the file output of the program to produce, instead of a single HPGL file, two files for each figure created, one a PostScript description of the graphic, and the other a text file containing labelling information in T_EX-readable form. All that was then needed was a T_EX `\figinsert` macro to pass on the PostScript file in a `\special`, and read the label file, typesetting the labels at the correct positions.

Specifically, the command

```
\figinsert{myfig}
```

carries out the following operations:

- First it opens the label file `myfig.lbl` and reads the first two lines, which contain integers giving the width and height of the graphic in points. It builds a `vbox` with those dimensions and `vfills` it so that the PostScript `currentpoint` (from the PostScript driver's point of view) is at the bottom left corner of that box.
- Next it passes the file `myfig.ps` to the driver with a `\special`. The PostScript code in `myfig.ps` (which is bracketed by a PostScript `gsave—grestore` pair) translates the PostScript origin to the current point and draws the figure.
- Finally the macro processes the remaining lines of the label file in groups of five. These are x and y coordinates of the position of the label in the `vbox`, two codes representing the horizontal and vertical justification or centering of the label, and finally the T_EX label itself.

- Reading of the label file terminates when a negative x coordinate is read. (MG inserts `-1` as the last line of the file.)

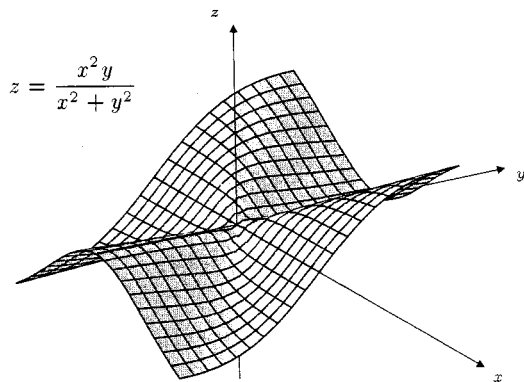
Here is a list of the `\figinsert` macro.

```
\newcount\pswidth
\newcount\psheight
\newcount\justx \newcount\justy
\global\justx=0 \global\justy=0
\newcount\vpos \newtoks\label
\newread\labelfile
\newcount\xcoord \newcount\ycoord
\newif\ifdoit \newbox\labox
%
\def\newfiginsert#1{\openin\labelfile=#1.LBL
\global\read\labelfile to\pswidth
\global\read\labelfile to\psheight
\vbox to\psheight pt{\vfill
\special{ps: /firstswitch {true} def }
\special{ps: plotfile #1.PS}
\special{ps: printblack not
{5000 5000 translate} if }
\vskip-\psheight pt\ninepoint%
\hbox to\pswidth pt{\hss}%
\parindent=0pt\offinterlineskip
\vpos=0
% read in label information
\loop
\global\read\labelfile to\xcoord
% test for end of labelfile
\ifnum \xcoord < 0 \doitfalse
\else\doittrue\fi
\ifdoit \global\read\labelfile to\ycoord
\global\read\labelfile to\justx
\global\read\labelfile to\justy
\global\read\labelfile to\label
\global\setbox\labox=\hbox{\label}
% insert the label, suitably justified
\advance\vpos by-\ycoord
\vskip-\vpos pt \vpos=\ycoord%
\hbox to\pswidth pt{\hskip\xcoord pt%
\hbox to 0pt{\ifnum\justx>0\hss\fi%
\vbox to0pt{%
\ifnum\justy<2\vss\fi%
\nointerlineskip\vbox
to\dp\labox{\vfil}
\nointerlineskip\copy\labox%
\nointerlineskip\vbox
to\ht\labox{\vfil}
\nointerlineskip%
\ifnum\justy>0\vss\fi}%
\ifnum\justx<2\hss\fi}%
\hss}
\repeat
\special{ps: printblack not
{-5000 -5000 translate} if }
\advance\vpos by-\psheight%
\vskip-\vpos pt}
\closein\labelfile}
```

The `\figinsert` macro is a low-level one. In practice, figures are inserted by more high-level macros which call `\figinsert` to place one or more

figures across a line or in a box, and which may also supply titles or headings for the figures.

The system is not perfect. MG takes no account of the actual size of a label, vertically or horizontally. (It does not attempt, for instance, to read any TFM files.) Therefore, each figure needs to be printed once after it is created to check on the positions of labels. Occasionally a label needs to be moved in one direction or another to avoid colliding with other elements in the figure. Such moving of labels is most easily accomplished by directly editing the label file to alter the coordinates of the label. All one needs is a pica ruler such as the handy plastic ones given out at T_EX Users Group meetings to advertise PCT_EX. Here is an example of a figure created by MG for *Calculus: A Complete Course*,



and here is its label file.

```

217
173
204
149
0
0
 $\ss$  x$
214
66
0
2
 $\ss$  y$
108
7
1
2
 $\ss$  z$
75
57
2
0
 $z=\frac{x^2 y}{x^2+y^2}$ 
-1

```

Here \ss is an abbreviation for \scriptstyle .

One final comment about producing PostScript graphics for inclusion with T_EX. There are numerous programs on the market which can be used to produce PostScript graphics output on an IBM PC, a Macintosh, or on other personal computers. (See J. T. Renfrow's paper [Renfrow, 1989] for methods of integrating such graphics into a T_EX document.) More and more of these programs are capable of generating what is called Encapsulated PostScript. This means that the PostScript code which defines the graphic is bracketed by interfacing code in a standard format which appears as ignorable comments to the PostScript interpreter, but which conveys necessary information (for example the size of the graphic) to external programs which must assimilate the graphic as part of a larger document. It is encouraging to see implementers of DVI-PostScript drivers such as ArborText are now taking note of this standard and providing for easy inclusion of Encapsulated PostScript in T_EX documents via \special commands.

Bibliography

- Carr, Leslie. "Of METAFONT and PostScript." *T_EXniques* 5, pages 141–152, 1988.
- Hobby, John D. "A METAFONTlike System with PostScript Output." *TUGboat* 10(4), pages 505–512, 1989.
- Renfrow, J. T. "Methodologies for Preparing and Integrating PostScript Graphics." *TUGboat* 10(4), pages 607–626, 1989.

TeX in Practice: Comments on a 4-Volume, 1400-page Series on TeX

Stephan v. Bechtolsheim

Computer Science Department, Purdue University, Computer Science Building, West Lafayette, IN 47907
Integrated Computer Software, Inc., 2119 Old Oak Drive, West Lafayette, IN 47906
(317) 463 0162, (317) 494 7802. Internet: svb@cs.purdue.edu

Abstract

This article discusses a four volume, 1400 page series, about TeX. It discusses two aspects: first of all I will discuss how processing a document of this size was organized. Second I will discuss extensions to TeX which I consider desirable.

A Short Introduction

In a previous draft of this article I had written about a page and a half about what pain it is to be an author, in particular the author of something as long hat and elaborate as the books under discussion; I still do not know whether there will be any real rewards since the fees paid to authors are mediocre, at best. But on the other hand: I could have quit at any time, and I decided stubbornly not to do that.

Yes, it was a frustrating activity, but now it's already the past!

Processing 1400 Pages of TeX Source Code

How did I manage a 1400 page TeX document? There are a number of additional programs that I used, which are described below.

The Computing Environment. Let me describe my environment briefly: I use a SUN 3/50 with an extra 4MB of memory (that's 8MB altogether) and a 327MB local disk. This machine is hooked up to the department's ethernet. 3/50s are not terribly fast machines so the real processing takes place on a departmental Sequent. The source code is copied to the Sequent using the `rdist` program (remote distribution) so that only those files which changed are copied.

I use the GNU Emacs editor, as far as I am concerned, the best editor around. I have written a small TeX mode for this editor which makes the editor much easier to use with a TeX document. I cannot repeat frequently enough how important a good editor is: you spend most of the time editing your text, and therefore the best editor is just good enough. See Bechtolsheim [1988] for details.

Some Statistics. Let me give some statistics about this series of books. The source code of this series is about 70000 lines of TeX code, which occupies about 2.2 MB of storage. All dvi files together are about 3MB long. The size of the directory in which the processing of the book takes place, is about 16 MB, around 20 MB if all PostScript files (I have a PostScript printer attached to my workstation) are also stored. The series is subdivided into 57 different files of source code files which I call *part source files*, some of which are of auxiliary nature, but most of them are one chapter of a volume of this series. The part source files also include parts belonging to a fifth volume that is not published, but contains information such as any matters pertaining to the publisher or shell scripts which I used for a variety of functions. There are 228 macro source files which are published with this series.

For the following please note that all volumes together are regarded as one unit, and they are processed as such. Therefore, cross-references across volumes are not really any different from cross-references within the same volume.

The Input Language, the Preprocessor Used. The input language is of course largely TeX, but I made some extensions. These are *not* extensions to TeX, but codes interpreted by a preprocessor, `pretex`, which I now discuss briefly. The tasks of the preprocessor are as follows:

1. Allow for the direct inclusion of macro source code. Originally, without the preprocessor, my set up was as follows: I would store the source code of a macro which I describe in the series, with comments, in an external file. I then used `\ListVerb` to read in such an external file to generate a verbatim listing of it. In case I

wanted to use this macro source file I would use `\input` to read in the macro source file.

I ended up with tons of macro source files, as you can imagine. Note that the previously given figure of 228 macro source files does not even include the source files of all examples! The correct figure is close to 500 files.

This was the main reason for the design of this preprocessor, which allows me to do two things with lines in the part source file:

- (a) Include macro source code files directly. The preprocessor writes this macro source code to external files *and* includes its verbatim listing in the output file generated by the preprocessor (this output file will later be processed by `TEX`).
 - (b) The preprocessor allows me to switch back and forth between writing a macro source file (and including its verbatim listing in the main output file) and writing comments, which appear as ordinary output in the book, but do not appear at all in the macro source file.
2. Maintain the makefile. I made extensive use of the UNIX utility `make` to process my book. The main idea behind using `make` is, of course, to let a program (rather than an unreliable human being) figure out which parts of the series need to be reprocessed after a change, and which do not.

Administering that part is quite difficult, which was another reason for writing the preprocessor.

3. Administer overlays. As you will see later, I used an overlay dvi file processor (or DFP for short). Again, certain functions are controlled by input to the preprocessor.
4. Administer the inclusion of log files. There are a great many log files included with the documentation. The generation of these log files is controlled by information written to the makefile generated by the preprocessor.

There is actually another program used in building makefiles, but this is beyond the scope of this article. For `pretex` and the utility just mentioned see Bechtolsheim [1990b] for details.

Note also that before `TEX` is actually executed to process a part, some other `TEX` executions may take place, for instance, to produce log files included in parts of the series, or to generate dvi files of figures overlaid in this series.

Running `TEX`. Running `TEX` is the easiest part in this context. I always process only one part at a time, and if you read *T_EX in Practice* (in particular

volume III) then you will find that the set-up is quite similar to that of `LATEX`: one part of the series is processed at one time. Also, during this step, an index file is written out for each part.

Because I process only one part at a time, at the end of every processing step (`tex main`, assuming the main source file is called `main.tex`), I would rename `main.dvi` and `main.log` appropriately as, say, `intro.dvi` and `intro.log`.

After `TEX`. After `TEX` has executed, the dvi processor which I mentioned previously (see Bechtolsheim [1990c]), is executed twice.

The main purpose of the first execution is to extract positional information for marginal notes. I generate marginal notes using the DFP because this allows me to separate the marginal note generation completely from the generation of the text itself. I use marginal notes for the following purposes:

1. Communication with the editor (I am, of course, talking about the “person” editor rather than the “program” editor). If I have a question, I simply put this question into the margin.
2. Addition of change bars. I found change bars an extremely useful feature. In case of a change to an already edited chapter, I could mark those changed areas easily so that my editor could have yet another look at it.
3. Print index terms in the margin. To develop an index is considerably simplified, if the index information is written into the margins of the document. This way when the index is being developed it is immediately visible which index terms refer to a specific page.

The second execution of the DFP does the following:

1. Puts the date, time, version number, and file name on every page.
2. Extracts information about which fonts were used in each part and store this information. This allows the generation of a table listing all the fonts used.
3. Overlays other dvi files. There are a number of instances where output generated by separate `TEX` runs must be glued into the main document. This function is performed at this point.

Extensions of `TEX`

In the remainder of this article I discuss possible extensions of `TEX`. Theoretically, `TEX` can be made to do anything, but this is not really true in

practice. Therefore, let me discuss what I would like to see added to TeX.

Operating system and interface related extensions. I would like to see a more flexible interface with the operating systems on which TeX runs. I am thinking of features such as opening and closing dvi files, asking whether or not certain tfm files are accessible, writing all characters to external files (including characters such as tabs and returns, I am thinking of a `\writechar` primitive analogous to `\char`). In TeX 3.0 the current line number is accessible, which is something I would have listed here if that were not the case. I would also like to be able to invoke other other programs, and I realize that TeX source code using this feature would be restricted in their portability.

Graphics extensions. I have *no desire* for any graphics extensions. The inclusion of PostScript generated figures works just fine.

Insertions and output routines. Because of the size of L^ATeX's output routine and the fact that insertions are *not* used for figures and tables (only for footnotes), I would like to see an extension to TeX's insertion mechanism. It should be made more powerful to allow one to specify, for instance, the number of insertions that can be permitted on one page: both a maximum and a threshold vertical length which, if exceeded by insertion material, will prevent other material from being printed on that page. This is a very short description of what I have in mind; L^ATeX has a whole set of style-file related parameters, which really should be built-in parameters of the insertion mechanism of TeX.

What I envision is a set-up in which the insertion queues are accessible to the user, so that the user can write TeX code which inquires about the number of elements in an insertion queue, the length of individual insertion elements, and so forth.

Also, when TeX's page-breaking algorithm completes a page and puts it in box register 255, the glue and penalty information around that break-point is essentially lost (with the exception of the setting of `\outputpenalty`). It is thus impossible, from within the output routine, to restore an old page in its entirety.

Paragraph computations of TeX. Typesetting specifications by publishers may prohibit a page break just following a heading or in the following two or three lines of text. Therefore an `\afterclubpenalty` should be introduced, and maybe one should generalize this penalty business even further.

The `\everypar` register is evaluated after the `\parskip` glue has been sent to the vertical list with the current page or vbox. This makes it difficult to use `\everypar`. Therefore, I would like to have a built-in token register `\everyvpar` which is evaluated as soon as TeX decides it is time to begin a paragraph but before TeX gets around to doing anything about it.

There should be a `\parskippenalty` as well as a `\baselineskippenalty` and a `\lineskippenalty`.

Expansion, grouping. I would like to have access to the current level of grouping in the form of a read-only counter register. This would allow me to determine at the time a heading is encountered whether all preceding groups have been terminated (that is, I would like to be able to set up TeX in such a way that groups cannot extend beyond certain subdivisions of a document).

A boolean data type and boolean operations (`\not`, `\and`, `\or`, and so forth) should be added. It should be possible to write "real conditionals".

Doing any type of arithmetic in TeX is a bit of a pain, so I would like to see something which would allow me to write, for instance:

$$\text{\dimen0} = 1.3 * \text{\baselineskip} - \text{\wd0}$$

Relational operators \neq , \geq , \leq should be made available for register arithmetic.

Box computations. It should be possible to access the badness of a box stored in a box register. Furthermore, it should be possible to *access and manipulate each element* of the horizontal or vertical list of a box on an individual basis. In other words, I would like to see a generalization of primitives like `\lastskip` and `\lastpenalty`. For instance, if `\lastpenalty` is zero, then this means either that the last item was a penalty of zero, or was not a penalty. I would like to see, therefore, a reliable way to learn what each item is, not just the last one.

In particular, I would add primitives which allow access to the dimensions of the lists and list elements of boxes. One reason the insertion of change bars with a dvi file processor is so easy (see Bechtolsheim [1990c]), but so difficult in TeX, is that there is no way to access this information.

I personally would remove the restriction which permits `\vcenter` to be used in math mode only.

Math mode. Believe it or not: I found someone who wanted more than three different fonts per font family in math mode. I am not sure I concur with this.

Concluding remark

TEX is a *great product*. It is so wonderful, powerful, and flexible, it's worth all the effort required to learn it.

Bibliography

- Bechtolsheim, Stephan v. "Using the Emacs Editor to Safely Edit TEX Sources", *TEXniques* 7, pages 195 - 202, 1988.
- Bechtolsheim, Stephan v. *TEX in Practice*. New York: Springer, 1990a
- Bechtolsheim, Stephan v. *A TEX Preprocessor and a make related Utility*. West Lafayette: Integrated Computer Software, Inc., Report 90-1, 1990b
- Bechtolsheim, Stephan v. *A dvi File Processor*. West Lafayette: Integrated Computer Software, Inc., Report 90-2, 1990c
- Knuth, Donald E. *The TEXbook*. Reading, Mass.: Addison-Wesley, 1984.

Textbook Publishing — 1990 and Beyond

Mimi L. Lafrenz

Electronic Technical Publishing, 2906 N. E. Glisan St., Portland, Oregon 97232
(503)234-5522

Abstract

Production of college texts and reference books is a natural for \TeX , yet the commercial publishers are slow to understand, and therefore accept, the power and versatility of the program. Some book publishers are utilizing authors' \TeX files for creation of camera-ready pages, while others will rarely consider the possibility. The resistance of commercial users to divulge macros, tricks, and techniques has impeded the acceptance of \TeX by commercial publishers.

We will explore the training and support of authors and publishers, and the impact of openness in the technical realm. Only through shared objectives will we be able to create the best environment for \TeX to flourish in this decade.

Introduction

The process of publishing textbooks, especially college-level, scientific, or technical textbooks is rapidly changing. Advances in technology — laser printers, standard page-description languages and available fonts — are largely to thank for the broader use of computing, or electronics, in publishing. With the emergence of computing as an integral part of the production of book pages, broader issues are brought to the surface — issues that will ultimately shape the way book production is handled.

Before looking to the future, we consider the past and present, to see how they determine our perspective. The future of \TeX , and all technology, lies in its roots. To understand the differences between scientific and commercial perspectives, is to understand the future of both. The company we are building comes from the scientific side; however, we are convinced of the long-term commercial benefits.

Throughout this paper we have referred to \TeX in the general sense, including all derivatives such as $\mathcal{A}\mathcal{M}\mathcal{S}\text{-}\text{\TeX}$, $\mathcal{L}\mathcal{A}\mathcal{T}\mathcal{E}\mathcal{X}$, and so on.

History

ETP is a venture that began when a group of key people from a former company launched a typesetting service bureau, exclusively accepting electronic manuscripts. `troff` was the focus, and the production group consisted of two programmers who spent most of their time cranking out the

same UNIX documentation set for many of the large hardware manufacturers across the country.

ETP's entry into textbook composition came when Holt, Rinehart & Winston had an author under contract who was guaranteed that his electronic files, created with UNIX/`troff`, would be used in the production of the book. That was a difficult guarantee to keep in 1985, since there were few services offering `troff` programming and page formatting. The people at Holt went straight to the source — AT&T Bell Labs — for the name of a service that could bring this project to completion. We were the leading source for UNIX documentation composition; they called on us. We had no idea how complex good page makeup could be when we agreed to take on our first textbook project.

We had no formal training in the publishing and typesetting conventions related to the proper ways to create a page. Only through the publisher's patience and willingness to work with us were we able to complete the book. They helped us develop our understanding of the numerous details involved in producing high-quality technical type. The project took six months to complete, with much time spent building special characters to match editing marks that we were not familiar with, and similar fiascos. The book was published, and did well. It was in the second printing within one year, and continues to be a popular text.

With each successive textbook and exposure to dozens of publishers, we developed and honed the craft of technical typesetting. More importantly,

we dove deeper into the typesetting languages and programming aspects to harness the inherent power of troff.

The Move to T_EX

After a few years of experience with troff, an emerging publisher asked us to look at the possibilities of using T_EX. Thanks to good fortune, our technical director had studied under Pierre Mackay at the University of Washington. Dan Olson understood T_EX and gladly accepted the challenge, launching our first T_EX-textbook project in 1987.

Something was very different. Pages were being created faster and more beautifully with T_EX. There was hardly a comparison with troff when the hours for page-makeup were tallied. T_EX books went through the plant faster and had fewer mistakes. We were really onto something here, yet at times there were frustrations that made it hard to continue.

Learning without teachers. The first obstacle to overcome was lack of experience with the program and its documentation. Dan's exposure to T_EX made him the sole source in the company for training, which occupied his time for years to follow. As Dan discovered new techniques with T_EX, he shared them with the production group, and vice versa.

There were many times when it seemed impossible to meet the publishers' strict demands for page composition with this program. When hours of programming time were demanded for physically simple tasks, we would resort to the *x-acto macro* and have it done in minutes. Of course there had to be two or three projects with excessive programming hours to lead us to the necessity of a macro made of hardware. Building the company without a role model, and very few mentors in the field, made us resourceful. The addition of T_EX to our service-line could have been fatal, had we not just gone through similar experiences with troff. T_EX was a major step forward.

Documentation. The T_EXbook is a good reference manual, not a good user's guide. There are books and periodicals available, but they require research and reading. The common lack of desire or initiative to research a program has inspired us to build a resource manual, one chapter at a time, to guide the user in understanding T_EX.

Training. The development of internal training programs is an expensive, arduous task. The lack of internal training is even more expensive. As

the old saying goes 'Think education is expensive? Try ignorance.' We spent our energies on a variety of training programs before implementing the current program. The most efficient training methods have been created by production workers who have developed a depth of knowledge in special areas. The use of *A Gentle Introduction to T_EX*, by Michael Doob, has become an integral part of our early training for new users. (Thanks Michael!)

Better, Cheaper, Faster

Better, cheaper, faster. That is all the publishers expect from an electronic manuscript, so what's the problem? Well, let's address these desires individually.

Better. The quality of mathematics set with T_EX is indisputable. The quality of page formatting that can be achieved using T_EX, albeit with effort, is among the best. Dr. Knuth built in many features that are simply unavailable, in their complexity, with any other program or system.

The quality of math set in an Adobe font with T_EX is another story. We have seen the attempts to incorporate other fonts into T_EX files succeed and fail. The advance of composite fonts will change the look of T_EX math. A major development project will eventually be undertaken, fine tuning character widths to automatically give the beautiful spacing inherent in the use of Computer Modern fonts. The accepted approach, in 1990, is the commingling of Times Roman and Computer Modern Math Italic on one page. It does differentiate the math variables from any other italic, but still leaves a lot to be desired in the aesthetic quality of the page.

The basic improvements T_EX has brought to computer-aided publishing should not be overlooked. The final product is better because of features like automated page bottoming, kerning, and the extra care T_EX puts into every paragraph while formatting pages.

Cheaper. The general feeling is that the manuscript must be ready to typeset if the author has input all of the information, so the labor extended to format the pages will be minimal. The main point that the publishers and compositors often miss is the condition of the electronic files prior to the beginning of the composition process.

Quite often in college textbook preparation authors will employ students, clerical staff, even family members, to input the chapters as they are written. It is not unusual to see a 17 chapter book input by 5 or 6 people with different styles, macros,

and contradicting definitions. In fact, we have produced several books in which 3 or 4 programs were used to compose the original files. It was usually easier to strip the word-processing codes and convert the troff or other languages into T_EX, in order to produce a book consistent through each chapter and section. Reducing expense is difficult in this scenario. The publisher might have done better to send it overseas for keyboarding.

As we look into the future, we will see alternatives to this haphazard system of manuscript preparation.

Faster. In actual production turnaround, faster is relative to company size, skill, seasonality, and organization. The amount of time required to format a book can, in some cases, be significantly greater than the time required to keyboard and traditionally typeset the text depending on the quality of the programs used and on the attention to detail by the author. Publishers who understand the impact of preplanning on the schedule and total expense of a project will recognize the need to negotiate with vendors for better tools and training to simplify and improve the process of book production.

Communication is the Key

The ability to understand and evaluate information and make decisions is the key to a successful project. Developing an understanding between any two parties is the biggest challenge of all, and there are several links in the publishing communication chain.

First, the author and associated support group must set guidelines and procedures for the project which will eliminate waste in the final stages of pagination. For example; coding elements by content (`\example{...}`) rather than by appearance (`\bigskip {\bf...} \medskip`). Next, the author must supply the publisher with complete information about the manuscript, media, and system of creation. The publisher must understand the information being passed on to the compositor and/or artist, since decisions must be made at this crucial step which will affect the results. A progressive publisher, like Addison-Wesley, will research the tools and technicians and have informed individuals guiding authors and preplanning composition. This approach allows growth for all who share the ideas and thereby broaden their own experience. The publisher must then communicate with the compositor, artist, printer, bindery, and distributor. The jargon of book manufacturing has changed

little over recent decades, with the exception of composition, art, and prepress. These are areas of rapid advancement in technology, and only through continuing study can one be up-to-date on all of the current developments. A basic understanding can, however, easily be gained that will allow an individual to converse and make decisions based on the technical information being supplied.

A Fine Line

Quality communication is not the sole responsibility of the publisher or author. The vendors doing the actual production of the book can improve the entire process by accepting the challenge of opening communication channels with clients. There is resistance to openness in the commercial arena, for fear that sharing knowledge will take away a competitive edge.

A company works for years at understanding a product or method, and may believe that sharing the technology with the marketplace would be placing the company's future in jeopardy.

The only thing a company has to sell is its technology. Pieces, parts, and production can be copied, but the intangible understanding must be developed. This is the reason private-sector gurus will not divulge macro source code; it is their security. With careful consideration and planning, tools and information can be made available to the marketplace. Tools which enhance the entire process are being released now. Just as Knuth and the AMS developed the most powerful typesetting program and turned it over to the public, we should be open minded about how our technology is used. It is a fine line—between industry for profit and R&D for the advancement of science—the most exciting line to walk.

The Next Generation

Looking to the future, we see trends developing in publishing, technology, and services. Predicting the outcome of developing trends is risky, but this is the approach we have chosen to develop the most efficient and effective publishing system possible. T_EX is a good gamble.

Training. The training of authors, editors and publishers is a vital step in fully utilizing the tools and services available. But who is responsible for this training program, and exactly what should it cover? The answer is obvious, we are all responsible to teach each other everything possible. This is the investment required to bring results. Investing time

and materials now is essential to bringing greater understanding and opportunities in the future.

One of the objectives in our program for training authors and publishers has been to foster a basic comprehension of computing. The use of electronics in publishing will continue to grow. Having the users of any product of electronics understand how the product is derived and applied will accelerate the growth. The course needed to gain this grasp takes less than a day.

Technology. The evolution of tools used for publishing will continue. Hardware and software capabilities increase while the prices decrease. Tools are available to a greater number of people, and are easier to use.

The trend we see unfolding in the technology of composition is an increased involvement by the author in the initial formatting and edit updating. Some authors create elementary graphics, some excel in their comprehension of technical illustration and desktop graphics programs. A greater number of authors submit camera-ready pages each year, fully formatted in a variety of ways.

The problem of authors thinking they are designers is common today. With attempts to separate content from appearance, like SGML and ETPtex, authors are less likely to work on the design. Publishers provide macros fully capable of page formatting, when they can. This is a new development, it should have an impact on the way books are produced.

Quality. It is up to the publisher to control the quality, and it is often not easy to persuade the author to conform to editing marks or specifications. Herein lies the frustration most common to those currently involved at this level: What can be sacrificed or improved in maintaining good author relationships, budget, and schedule?

Quality is the only factor left. It suffers in most cases. But college textbook publishing is a market driven, scientifically exact process demanding the highest quality. The ancillary products, guides, manuals, and such, experience an improvement in quality when an author has utilized a desktop or computer aided publishing system, rather than dot matrix or typewriter. Ancillary products have historically been composed by the author.

Preplanning to optimize production. To provide authors with tools capable of meeting publishing specifications, without the burden of years of developing the understanding necessary to produce quality pages, we are developing macro programs

for publishers. These are built to common specs, allowing for minimal variation in the design.

In 1988 we first released ETPtex, a macro set designed to produce double spaced manuscript output, with special features like callouts (elemental names) in the margins and floating figures. ETPtex overlays (as a front end), all versions of T_EX and is simple to use. Macros are named to match the specifications, and conform to the specs and house standards — when run through the version of the program kept at ETP. Frequently, the design has not been decided during the writing phase of the book. The macros give authors freedom to write without concern for pagination. The involvement of Prentice-Hall was very important in this development project. They have supported our growth in many areas, providing guidance and authors to work with during every stage.

Services like ETP will broaden their range and be available for technical consultation, macro writing, production, and training. The liaison work between the author and publisher is being handled by total concept houses, or services that act in a freelance capacity to bring the editing, design, and production under one roof. A total concept house able to converse technically with the author and publisher, in their respective languages, will bridge the gap until open communication ripples through the industry and training is commonplace. This will happen much sooner than some anticipate.

Summary

The publishing of textbooks, reference books, and periodicals, is transforming into a process of communication. Pioneering companies and individuals realize the benefits of utilizing T_EX to improve quality, turnaround, and expense. With rapid advances in so many areas, we must realize that our understanding, support, and use, of these advances will shape the next generations of publishing. The use of T_EX will provide freedom not possible with any other platform.

We believe that helping each other will benefit all of us, as we approach a new century of information management.

Gratis

For a free copy of ETPtex, or other information, please write or call ETP Services at the address at the beginning of this paper.

Diagnosing T_EX Errors with a Preprocessor

David Ness

803 Mill Creek Road, Gladwyne, PA 19035
215-649-3522.

Abstract

T_EX finds our errors with ease; however, it sometimes reports them in ways that are hard to understand. Generally this is because we have confused it by unintentionally misrepresenting something. For example, if we do something as simple as forget an escape on a dollar sign T_EX will probably give us some obscure diagnostics about math mode. This paper discusses some preprocessors that can warn us about potential problems before we submit our files to T_EX. These programs may be of particular help to new users.

Purpose

T_EX errors can be difficult to diagnose, particularly for the new user. T_EX is *very* flexible and general. It provides a rich world for developing and describing typesetting processes. The very richness, however, of this world—and its flexibility and generality—seems to work against the new user, particularly by making it hard to see the cause and cure for problems. The *suite* of programs described in this paper attempts to deal with such problems.

The Overview

Instead of writing one (complex) program to help diagnose T_EX errors, we wrote a number of different programs, each of which could deal with one, more restricted, problem domain. The results of these separate analyses can be integrated into a single picture. By separating parts of the error analysis process we allow for independent evolution, and people who have only one particular problem need use only the module appropriate to it.

The modules that comprise this *suite* have the following functions:

TEXCHECK checks for some common T_EX errors, particularly those made by new users. For example, this module looks for unescaped dollar signs and percent signs (*i.e.*, \$, % instead of \\$, \%).

TEXBRACE takes the input file and keeps careful track of the use of left and right curly braces. If the left and right braces aren't properly balanced, the locations of those not matched are reported. The source file is also rewritten (temporarily) in an attempt to make brace problems easy to spot.

TEXLOG analyzes the log that results from running T_EX. This program, written in AWK-WEB, is still in a preliminary state and is not described further.

TEXERROR merges the results of running **TEXCHECK**, **TEXBRACE** and (when it is ready) **TEXLOG**, along with the original source file, into a new source file that can be edited to correct the mistakes and remove any error comments.

TEXFIND is an experimental program designed to help users relate their input source to the output obtained from a T_EX run. This aid is quite distinct from the others discussed here.

Organizing a T_EX Source File

Users, particularly new users, can find it helpful to adopt some discipline in organizing their source files. This will prove useful when diagnosing and fixing errors. As one gains comfort and familiarity with T_EX, this discipline can be relaxed, but at least in the early months it is wise to adhere to some simple principles.

Keep macro definitions in a separate file, to be incorporated by an `\input` command; this makes life much simpler. Modifications to macros as they are debugged represent an effort quite different from that required to modify the basic text. If these problems are isolated, it is easier to see what's going on.

Some of the diagnostic help provided by the programs in our *suite* is rendered more effective when the source is split into logical pieces. For example, the **TEXCHECK** program flags all occurrences of unescaped number signs (`#`). Generally these don't occur in normal text, but they are a regular

part of macro definitions. If the macros are in a separate file, which we don't pass through `TEXCHECK`, then no confusing diagnostics will appear.

`TEXCHECK` World-view

`TEXCHECK` was written to warn about potential problems. Since it is preferable to be warned too often than not often enough, occasionally warnings are generated about things that would be found legal if a more substantial analysis of the `TeX` source were made. The source file is not analyzed in a deep way, so complex things like mode shifts and macros will be missed.

The idea of creating a *Lint* for `TeX` was rejected because of the complexity of `TeX` syntax. After all, in `TeX` it is an easy matter to redefine nearly everything, and keeping track of all of this would rival writing the `TeX` processor itself!

We gave up on the idea of doing the job perfectly and may have gone to the other extreme. Our principal goal is simplicity, in particular we hope to share these ideas with others, so that feedback will help us to develop them further.

`TEXCHECK` warnings. `TEXCHECK` warns about a variety of possible errors. Most of the warnings are designed for new users, but even old hands at `TeX` may find a pass through `TEXCHECK` is worth the trouble, particularly if the source file was captured by someone not too familiar with `TeX`.

Here is a list of warning messages which `TEXCHECK` may issue:

Angle brackets probably need to be in `\tt` font.

Many `TeX` fonts place the upside down exclamation point and question mark in the ASCII table where the angle brackets are in the `\tt` font. This warns about all angle brackets in the text, unless they are preceded on their line by a percent sign (and thus are probably in a comment).

Something may be missing to avoid end-of-sentence spacing.

`TeX` has some sensible, but complex, rules about when it puts in end-of-sentence spacing. This warning indicates that `TeX` will put end-of-sentence spacing after a particular period, and `TEXCHECK` thinks it may be inappropriate (for example, on the period after 'Dr').

Perhaps there should be end-of-sentence spacing here, and there won't be.

We might also have a place where a capital ahead of a period blank might have suppressed end-of-sentence spacing when it shouldn't have. This checks for that situation too.

Em- and En-dashes generally about the words on either side.

English typesetting specification suggests that dashes about the words on either side. This warns about what appears to be contrary usage.

Number-signs are generally only in macro definitions.

Number signs are common in normal text so they may sometimes be entered without being properly escaped. Since they normally represent arguments to macros in `TeX`, diagnostics can be confusing. `TEXCHECK` warns about them indiscriminately, *i.e.*, it makes no attempt to see if the unescaped `#` is being used legally (for example in a macro definition).

Double quotes should go away.

`TeX` usage calls for two left quotes and two right quotes, which become left double quote and right double quote. While the typewriter double quote character will produce the `TeX` right double quote, it probably shouldn't be used at all for quotes in a `TeX` source.

Ampersands usually perform tab skips.

Ampersands generally represent tab stops in alignments. This warns about *all* unescaped ampersands because error diagnostics that result from ampersand misuse can be confusing.

Underscores and carets generally are sub- and super-scripts in math mode.

Sometimes underscores and carets creep into normal `TeX` text. They can generate confusing error messages there because they ordinarily represent sub- and super-scripts in math mode. This message will appear when underscores and carets are detected not following a dollar-sign that might indicate a previous shift into math on the line. In `TEXCHECK` no attempt is made to detect whether we are in math mode, which is complicated to determine.

'%' preceded by digits probably should be escaped.

We often forget to escape percent signs. This can cause text to disappear. This warning raises a question about situations where a number is followed by a percent sign (perhaps separated by blanks), without the percent sign being escaped.

Check to make sure that the thing following the '%' sign is a comment.

As an alternative, if the first thing following a percent sign (after some optional blanks) isn't an upper case alphabetic character (that might begin a comment), then we also raise a question.

The dollar sign indicates a shift to 'math'. Was that intended?

If a dollar sign happens to be followed by a number, it is possible that a real dollar amount was intended, and an escape forgotten. This message will, of course, improperly appear when something introduced in math mode begins with a number, but this is a smaller price to pay.

Running TEXCHECK. TEXCHECK can be executed with a number of switches. If none are specified it will, like the other modules described here, prompt for the appropriate inputs.

The switch `-I x` tells TEXCHECK to use 'x' as the input file. `-O x` names 'x' as the output file. If the switch `-F x` is used, then the input file is assumed to be 'x.TEX' and the output file will be 'x.CHK'.

TEXCHECK can issue reports at five different levels. The level of reporting is indicated with a `-R n`. Level 3 provides the greatest amount of descriptive information while level 0 provides the least. Level `-1` is used when the output of TEXCHECK is to be fed into TEXERROR.

TEXBRACE Functions

The purpose of TEXBRACE is to help us find errors in curly-brace structure. These pose particularly nattering problems for TeX because a missing brace will often cause TeX to misinterpret some element of the structure and can create obscure error messages.

TEXBRACE performs two functions. The easier to understand involves finding the lines on which unmatched left curly braces occur. A list of line numbers for unmatched braces is made by the program and as corresponding right braces occur this list is adjusted. If the list is not empty at the end of the file, it shows where the unmatched braces were. TEXBRACE will also report on excessive right curly braces if they occur, but this is generally a less difficult problem.

The other function of TEXBRACE involves creating a copy of the input file in a form that will emphasize its brace structure. When writing this copy, TEXBRACE replaces returns with blanks, thus producing (impossibly) long lines of text; however, each time a brace is encountered we drop to a new line and indent (for left braces) or outdent (for right braces). The file that results from this isn't good for anything but looking at brace structure, but any problems with this structure then turn out to be obvious.

Running TEXBRACE. TEXBRACE can be executed with a number of switches. The switch `-I x` tells TEXBRACE to use 'x' as the input file. `-O x` names 'x' as the output file. If the switch `-F x` is used, then the input file is assumed to be 'x.TEX' and the output file will be 'x.BRC'.

TEXBRACE can issue reports at two different levels. The level of reporting is indicated with a `-R n`. Level `-1` is used when the output of TEXBRACE is to be fed into TEXERROR. Level 0 provides output to be read by the user. At level 0 the entire text of the file is rewritten in a way that emphasizes brace structure. At level `-1`, only the error messages are written.

TEXERROR Functions

TEXERROR takes the output of TEXCHECK, TEXBRACE and (when ready) TEXLOG and merges them with the original source into a new copy of the file. Each of the routines identifies the line number on which potential errors have been reported and this module takes all messages appropriate for each line and places them in the output file just following the line in question.

The lines generated by these programs are in a format appropriate for TeX comments. TEXERROR also arranges to have the first line of a block of error messages begin with "%ERROR" and end with "%ERROR-MERGE End". This makes them easy to find with a text editor.

Running TEXERROR. TEXERROR can be executed with a number of switches. The switch `-I x` tells TEXERROR to use 'x' as the input file. `-O x` names 'x' as the output file. If the switch `-F x` is used, then the input file is assumed to be 'x.TEX', the brace error input file is 'x.BRC', the check error file is 'x.CHK' and the log error input file is 'x.ERL'. The output file will be 'x.NEW'.

TEXFIND

TEXFIND is an experimental program designed to act as a prototype for a TeX error facility that would allow the user to associate the input source file directly with what is seen in the output.

TEXFIND takes each piece of recognizable text in a document and follows it with a TeX call `\spc[m,n]` where `m` represents the line number and `n` the column number of the source file line that began the word in question. Since it is very difficult to know anything about the actual effect of a TeX macro without profound analysis, only first level text is recognized by TEXTFIND.

Once the `\spc[...]` markers have been placed in a source file, that file can be run through `TEX` with a definition of `\spc[...]` that will cause `TEX` `\special` commands to be written into the DVI file. This information is then available to display drivers able to use it to relate things being displayed back to their original locations in the source file.

An important modification needs to be made to a source file before it is sensible to run `TEXFIND` on it. There are situations (in the middle of a macro definition, for example) where inserting the `\spc[...]` markers might prove disastrous. For that reason a special form of `TEX` comment, `#!`, is used to toggle `\spc` generation on and off. `TEXFIND` begins operation with `\spc` generation off, so the source file should be modified by putting a `'#!'` on the line prior to the one on which the text begins. At the moment there are no drivers which will allow us to see the markers, so this represents an experiment in its earliest stages.

The Implementations

The programs described here are implemented in `C-WEBS`. The language we used is Norman Ramsey's implementation *via* `SPIDER`. The copyright on these `WEBS` has been assigned to TUG, the `TEX` Users Group, so that they may be freely exchanged in a community as wide as possible. We hope that feedback from this community will result in improvement of these programs.

Relationship to Text Editors

The output of `TEXERROR` can be processed by any ASCII oriented text editor. A good editor may deal effectively with the kind of messages that `TEXERROR` produces.

For example, using the old standby `PE2`, a definition like:

```
d a-e = [1/\%ERROR/] [mark line]
        [1/ERROR-MERGE End/] [mark line]
```

makes it possible to locate the next block of error messages and highlight them simply by typing `<ctrl>-E`. The normal editor function `<ctrl>-D` will then delete this block of error messages. Thus it is possible to page through the file with successive `<ctrl>-Es` and `<ctrl>-Ds`.

The TE.BAT File

The programs in this *suite* work together conveniently. One easy way is to construct a DOS `.BAT`

file that calls them in sequence. The following simple file `TE.BAT` does this:

```
@ECHO OFF
REM "<$TeX Error Analyzer - Ver (1)$>"
TEXBRACE -f %1 -r -1
TEXCHECK -f %1 -r -1
Echo TEXLOG doesn't exist yet
TEXERROR -f %1
DEL %1.BRC
DEL %1.ERL
DEL %1.CHK
```

Here the programs in the *suite* are executed in sequence, and the results are fed into the `TEXERROR` run where they are merged. Execution of `TE filename` results in a file `filename.NEW` which should be copied over the original `filename.TEX` after corrections have been made and it is decided that the new file is better than the original.

Experiences

A first, and rather pleasant, surprise was that these programs, particularly `TEXCHECK` and `TEXBRACE`, proved to be helpful to long time users of `TEX`, as well as to novices, since both groups still make mistakes in files which these programs isolate quickly and without much fuss.

The programs have also proved useful by allowing sophisticated `TEX` users to have texts typed by typists not very experienced with `TEX`. The rules about typing `\%`, `\$`, `\&` instead of `%`, `$`, `&` tend to be forgotten until these things have been typed many times. With these programs it doesn't seem to matter whether they are remembered, since it's so easy to find these mistakes and fix them.

Acknowledgements

The ideas presented here resulted from discussions involving S. Bart Childs of Texas A&M University, Alan Hoenig of John Jay College, and the author. Suggestions from many others with whom we have discussed this idea over the past six months are gratefully acknowledged.

Increased T_EX Efficiency Using Advanced EDT Editor Features

Linda Williams & Linda Hall

The University of Tennessee Space Institute, Tullahoma, Tennessee 37388
615-455-0631. Bitnet: williams@utsiv1 Bitnet: hall@utsiv1

Abstract

T_EX is implemented on various computer systems often with little or no consideration given to the time saving quality of an efficient editor. For both the novice and expert T_EX user, employing advanced EDT editor features greatly enhances the efficient use of T_EX. Because there are volumes describing these various tools, it becomes necessary to reduce this vast amount of material to a manageable subset. This paper describes selected EDT features, such as editor initialization files and other commands which, when mastered by the user, enhance the editing power of T_EX.

Introduction

This paper describes a text editor, EDT, supported by a VAX 11/785, running the VMS operating system. Under the name EDT+, this versatile editor will also operate under MS-DOS (2.0 or higher) or UNIX on Ardent, AT&T, Celerity, Convergent, DEC, Encore, Gould, Hewlett-Packard, IBM, Intergraph, MIPS, SUN, and many others.

Because skill levels in using EDT as well as other editors vary greatly, from novice to expert, advanced editor features are often overlooked. Although there are volumes of documentation on editors, with help files and computer-aided instruction, learning about these advanced features often occurs by passing information from user to user. It is the purpose of this paper to continue that tradition by describing three fundamental areas of EDT, namely, 1) EDT line-mode features, 2) EDT screen-mode keypad features, and 3) EDT initialization features. Other editors assign similar type names to their editor features; keyboard macros, learning keystrokes, or recording keystrokes. Whatever your editor, the message is still the same: when an efficient editor is mastered, knowledge of its features and capabilities give the user a fast and powerful method of editing.

EDT Line-mode Features

When working within a document, EDT offers the user two fundamental editing modes, line-mode and screen-mode. (Screen-mode will be explained in

the next section.) When the edit session begins, EDT starts by presenting *, the line-mode prompt. At this prompt it is possible to use many line-mode commands or to execute a specific line-mode command called CHANGE:

*c (*carriage return*)

This particular command allows for full screen editing capability (screen-mode). Other commonly used line-mode editor functions for creating T_EX and other data files are:

- 1) carriage return — present the current line,
- 2) TYPE [range] — display the specified lines on the terminal,
- 3) SUBSTITUTE — replace the next occurrence of an old string with a new string over a range of specified lines,
- 4) WRITE [range] — write a buffer or a segment of a buffer to a different disk file,
- 5) DELETE [range] — delete a line or range of lines,
- 6) EXIT — end the editor session and save a copy to the MAIN buffer,
- 7) QUIT — end the editor session without saving any changes,
- 8) CHANGE [line #] — change to screen mode at an optional location.

Using the key sequence Ctrl Z will evoke the line-mode option * anywhere within the text for use of the above commands. GOLD (PF1) and 7 (keypad), in screen-mode will also prompt for line-mode options. Specific words and numbers can be used as qualifiers in conjunction with SUBSTITUTE and DELETE. These options allow the user one more

level of choice to prevent possible damage to the file.

The command

```
*sub/Tex/\TeX\wh/query
```

causes the string of letters Tex to be replaced by the string \TeX\ throughout the whole (wh) file and prompts with a question mark to verify the operation on each line displayed. Possible responses to this option are yes (y), no (n), or quit (q).

The command

```
*del 121:131
```

deletes lines 121 through 131 of the file.

The CHANGE command makes rapid correction of errors much easier since T_EX error messages are given by line number. For example, if an error was detected at line 102 in the T_EX data file, it is possible to use the command

```
*c 102
```

to place the user at the point of the T_EX error.

Line-mode commands provide sufficient editing power to create most T_EX documents; however, by employing both line-mode and screen-mode features time-consuming procedures can be reduced. The next section will explain how screen-mode commands reduce the input for a given operation from a line-mode command word, possibly from 4 to 7 keystrokes to a keystroke sequence of one or two keys. This keystroke reduction enhances editing power.

EDT Screen-mode Keypad Features

PF1 GOLD	PF2 HELP	PF3 FNDNXT FIND	PF4 DEL L UND L
20	10	11	17
7 PAGE COMMAND	8 SECT FILL	9 APPEND REPLACE	18 DEL W UND W
7	8	9	18
4 ADVANCE BOTTOM	5 BACKUP TOP	6 CUT PASTE	19 DEL C UND C
4	5	6	19
1 WORD CHNGCASE	2 EOL DEL EOL	3 CHAR SPECINS	ENTER ENTER
1	2	3	
0 LINE OPEN LINE		• SELECT RESET	SUBS
0		16	21

The figure illustrates the EDT screen-mode keypad layout with associated keypad commands for VT100-type and PC enhanced-type keyboards. These EDT features are also available on other keyboards; specific key assignment sequences may vary.

Most keypad keys have two editing functions associated with them, primary function commands and alternate function commands. The primary function commands, not highlighted, use the indicated key only. The alternate function commands, highlighted, use the GOLD key (PF1) in conjunction with the indicated key to perform the desired commands.

These keypad functions can be broken down into three groups: 1) movement, 2) delete and recover, and 3) key definitions and redefinitions. The following three subsections discuss this in more detail.

Movement. Movement refers to three possibilities, 1) the general movement of the cursor, 2) repositioning the visual display to another segment of the file, for example, the beginning or the end of the buffer, and 3) the movement of the cursor to the location of a specific text string.

For general cursor movements, the 1 (WORD) on the keypad moves the cursor from the beginning of one word to the beginning of another. The 2 (EOL) on the keypad moves the cursor from the current cursor position to the end of a line.

For repositioning the screen display, the sequence of GOLD (PF1) and 4 (keypad) advances to the bottom of the text and GOLD (PF1) and 5 (keypad) advances to the top of the text. For moving the cursor to a specific text string, the command FIND uses GOLD and PF3. The command prompts the user with Search for: at which point the user types the desired string and hits 4 (BOTTOM) or 5 (TOP) to indicate the direction of the search. The cursor will then be moved to the first occurrence of the specified string in the indicated direction. Using the PF3 (FNDNXT) command will search for the next occurrence of the previously specified string.

Delete and recover. Delete and recover refers to two possibilities, 1) the complete removal of text, and 2) the movement of sections of text from one location to another. Deletion can be thought of in terms of the entities that the editor understands, the character, the word, the line, and a section of text specified by the SELECT command. In each case, the system associates a buffer with each of these types to hold the given removed entity — character, word, line, or section. Consequently, there is a delete

character buffer, a delete word buffer, a delete line buffer, and section buffer (specifically called the PASTE buffer). Since each associated buffer contains the last piece of text removed, this fact permits the recovery of that same text removed by that last command.

For the complete removal of text, there are three commands: 1) DEL C (keypad ,) — deletes the current character under the cursor, 2) DEL W (keypad -) — deletes from the current cursor position to the beginning of the next word, 3) DEL L (keypad PF4) — deletes text from the current cursor position to the end of the line. The main purpose in using these keypad commands to delete, instead of the delete key, is the previously stated advantages of the associated buffers. Using the GOLD key (PF1) in conjunction with the specific delete key undoes the specific operation. For example, GOLD (PF1) and PF4 recover the last deleted line.

For the movement of sections of text from one location to another, the CUT-and-PASTE operation is used. In conjunction with the SELECT operation, a section of text can be marked and either deleted permanently or moved to another portion of the buffer to be PASTED into the desired location. This is often useful in using similar T_EX constructs. Since CUT and PASTE can also be done between multiple buffers, this offers additional power in copying previously successful sequences of T_EX commands which have produced output in the desired form.

Key definitions and redefinitions. An extension of the power of EDT's screen-mode keypad features, key definitions and redefinitions enable the user to create, redefine, or relocate the operation of a previously defined or previously undefined key. Using two line-mode commands, SET and DEFINE KEY, the user is able to design complex, tailor-made keystroke sequences to perform the desired operation. The next section will illustrate the use of initialization files containing these complex sequences which can broaden keypad editing to encompass the entire keyboard.

EDT Initialization Files

When a user starts an editing session, EDT searches for a system-wide initialization file. If no system-wide command file is found, EDT then looks for a file (`edtini.edt`) in the default directory. Because initialization files are not required, this feature's potential power is often overlooked by the less experienced user.

The `edtini.edt` file is allowed to contain only line mode commands, previously stated above. The most common commands are SET and DEFINE KEY. These two basic commands allows users to customize their editing environments. The keyboard can be rearranged for specialized editing functions for individual preference reducing errors and allowing for more advanced application development.

A novice's initialization file may simply contain one line — SET WRAP 75 — forcing words hitting the 75 column to wrap to the next line. However, as expertise with the editor grows, so does the size and functionality of the initialization files. The following presents a fully commented initialization file which can be used in the creation of T_EX documents; these specific commands can be implemented independently or in conjunction with other initialization files:

```
! Start-up commands for editor EDIT/EDT
(VAX/VMS)
!
! Key definitions for EDT commands:
! set wrapping to 75 columns
SET WRAP 75
! set full-screen editing mode (KEYPAD MODE)
SET MODE CHANGE
! make searches distinguish between uppercase
and lowercase characters
SET SEARCH EXACT
! to use with slow modem at home
DEFINE KEY CONTROL P AS "EXT SET LINES 6; SET
CURSOR 1:4."
DEFINE KEY CONTROL @ AS "EXT SET LINES 22;
SET CURSOR 7:14."
! B -- back to main buffer
DEFINE KEY CONTROL B AS "EXT EXTERN."
! F -- find a file to be put in buffer
DEFINE KEY CONTROL F AS "EXT INCLUDE ?'Enter
file name: ' BUFFER EXTERNAL."
! includes header file into main buffer
automatically
DEFINE KEY CONTROL U AS "EXT INCLUDE
header.tex BUFFER MAIN."
! includes file typed at the prompt
automatically into main buffer
DEFINE KEY CONTROL I AS "EXT INCLUDE ?*
'Insert file:'."
! R -- returns to file in buffer
DEFINE KEY CONTROL R AS "EXT RETURN."
! set screen to 132 columns
DEFINE KEY CONTROL W AS "EXT SET SCR 132."
! set screen to 80 columns
DEFINE KEY CONTROL E AS "EXT SET SCR 80."
! change case of first letter of current
word (left of cursor)
DEFINE KEY 8 AS "-W +CHGCC -C +W."
! move cursor to next section (16 lines)
DEFINE KEY 3 AS "16L."
DEFINE MACRO EXTERN
FIND=EXTERN
I;FIND=EXTERNAL.
```

```

DEFINE MACRO RETURN
  FIND=RETURN
  I;FIND=MAIN.
  FIND=MAIN.

```

In the creation of initialization files, or when using features from other initialization files, some commands must be used with caution; others will require further careful study to understand and utilize their full potential. For example, this illustration contains no defined commands either to exit an edit session CTRL/Z *exit or to quit an edit session CTRL/Z *quit. The possibility of mistakenly QUITing an edit session, thus not saving edit changes, rather than EXITing and saving these specific changes, is too great to risk using a previously defined key which might be hit by mistake. Other users have kept these definitions in their initialization files and have mistakenly destroyed the results of an entire edit session, a very costly result. Initializations files may also contain TeX commands, such as

```

DEFINE KEY GOLD B AS "I{\bf }^Z -C."
DEFINE KEY GOLD C AS "I\centerline{}^Z -C."
DEFINE KEY GOLD D AS "I\baselineskip ."
DEFINE KEY GOLD F AS "I\footnote{}^Z -2C."
DEFINE KEY GOLD G AS "I{\it }^Z -C."
DEFINE KEY GOLD H AS "I\halign{}^Z -C."
DEFINE KEY GOLD I AS "I\item{}^Z -C."
DEFINE KEY GOLD L AS "I\line{}^Z -C."
DEFINE KEY GOLD M AS "I\magnification=."
DEFINE KEY GOLD N AS "I\hfill."
DEFINE KEY GOLD P AS "I\par \noindent."
DEFINE KEY GOLD Q AS "I\hskip ."
DEFINE KEY GOLD S AS "I\ss ."
DEFINE KEY GOLD V AS "I\$\vbox{}^Z -3C."
DEFINE KEY GOLD Z AS "I\bye."
DEFINE KEY GOLD _ AS "I\underline{}^Z -C."
DEFINE KEY GOLD , AS "I\matrix{\cr}^Z -4C."
DEFINE KEY GOLD ; AS "I{\hbox{}}^Z -2C."
DEFINE KEY GOLD : AS "I\pmatrix{\cr}^Z -4C."
DEFINE KEY GOLD ^ AS "I\eqalign{\cr}^Z -4C."
DEFINE KEY GOLD ~ AS "I\eqalignno{\cr}^Z -4C."
DEFINE KEY GOLD = AS "I\widehat ."
DEFINE KEY GOLD + AS "I\widetilde ."
DEFINE KEY GOLD \ AS "I\pmb{}}^Z -2C."
DEFINE KEY GOLD [ AS "I\overline{}^Z -C."
DEFINE KEY GOLD ] AS "I\cases{\cr}^Z -4C."

```

The DEFINE KEY command allows commonly used TeX commands to be inserted into a TeX data file using only the defined keystrokes, e.g., GOLD C instead of typing \centerline, etc. The command is placed after the "I (insert), ^Z signals the end of the command, the curser can be moved back by specifying spacing -4C, and ." indicating placement where the command is invoked. By defining commands in this manner TeX formatting

errors and typing errors can be reduced. A well-designed EDT initialization file can enhance EDT's power further in the creation of TeX.

Conclusion

This paper presents of three major topics: 1) EDT line-mode features, 2) EDT screen-mode keypad features, 3) EDT initialization files. As the paper progresses from topic to topic, each step has in effect reduced the number of keystrokes and increased the power of the editing tool. While it is sufficient to know only one editing modality, such as line-mode, familiarity with screen-mode will enhance editing even more. By adding the power of the third topic, time-saving advanced applications can be developed.

It is not within the scope of this paper to condense volumes of user manuals to just a few pages, but it is sufficient to say that there is no substitute for learning through real-life experience. Thus, the purpose of this paper is to acquaint the user with some often overlooked editing features. Efficient use of a sophisticated editor makes a fast and effective tool and its power should not be overlooked.

Acknowledgements

Without the help and patience of two individuals this paper would have been an impossible task. Our thanks goes to Harry Ferber II for his writing and presentation experience and Heinrich Senge for his ingenious .edt files.

Bibliography

- Boston Business Computing, Ltd. *EDT+*. Andover, Massachusetts, 1990.
- Digital Equipment Corporation. *Guide to VMS Text Processing*. Maynard, Massachusetts, April 1988.
- Digital Equipment Corporation. *VAX EDT Quick Reference Guide*. Maynard, Massachusetts, September 1984.
- Digital Equipment Corporation. *VAX EDT Reference Manual*. Maynard, Massachusetts, April 1988.
- The University of Tennessee Computing Center. *VAXCluster User's Guide*. Second Edition, UT Publication Number E01-9915-004-89, Knoxville, Tennessee, 1989.

TEX for TEXnical Typists

Charles R. Martin

National Biomedical Simulation Resource, One University Place Suite 250, Durham, NC 27707
(919) 383-2256. Internet: crm@nbsr.duke.edu

Abstract

Many TEX users are not programmers or mathematicians but technical typists, practitioners of a skilled craft. These users often find existing TEX texts intimidating and cryptic. *TEX for TEXnical Typists* is intended especially for these users. Each unit introduces a few concepts, then immediately reinforces those concepts with practical experience using a short document. Students see visible results immediately, which leads to rapid progress and greater confidence. The course encourages an experimental attitude that serves well in practice. *TEX for TEXnical Typists* appears to be an effective way to teach technical typists to create attractive documents.

Teaching TEX to nontechnical staff. People who prepare documents using TEX find themselves not just typing, but setting type. Doing so means that they have an opportunity to exercise considerably greater creativity and craft. It also means, however, that they must learn new skills; these skills require effort and time to learn. Mathematicians and programmers often find acquiring these skills to be relatively easy, because they have background knowledge on which to draw.

Many people use TEX who are neither programmers nor mathematicians. These people often have difficulty getting started in TEX. *TEX for TEXnical Typists* is an introductory course directed toward nontechnical users of TEX, who are neither programmers nor mathematicians, but who have experience with technical document preparation. These users are sometimes known as “technical typists”, “technical editors”, or “technical writers”. All of these groups have their own expertise and their own qualifications; for simplicity and because the course is intended to be as broadly applicable as it can be, we can consider the technical typists as our example audience.

TEX for TEXnical Typists is based on an operational approach to TEX. In this approach, the students are exposed to a very few concepts at one time; these concepts are then immediately reinforced with exercises. This quick reinforcement has two effects: it helps the students learn each concept, and reduces confusion; more important, each successful exercise reinforces the students' confidence. With greater confidence, the students are

more willing to experiment on their own, and more willing to brave the mystic realms of *The TEXbook*.

The audience: Technical typists. Document preparation is a craft in itself. As with most crafts, it has aspects of labor, skill, and art: the simple mechanical labor of keying a text; the skills needed to handle complexities of the language, of punctuation and capitalization; and artistic aspects like those of book design. The people we speak of as technical typists are the tool and die makers of the crafts of the secretary: they must not only be masters of the skills of typing manuscripts, they must be aware of all the other issues involved in preparing drafts, “clean copy” for submission, and camera ready copy for publication. Surprisingly often now, photo-offset printing means that these people also prepare the final designs and camera-ready copy for entire books.

Technical typists are usually accustomed to using complex terminology, and those who prepare scientific texts are skilled in using difficult symbol-ogy. We can expect them to have considerable skill with the mechanics of typing as well as aesthetic sense and a sense of good style. We can't expect them to be skilled programmers, or to be more than very slightly acquainted with mathematics.

Intimidation. Someone reading *The TEXbook* is immediately presented with stylish but dense prose, and confronted with the dread “double-bend” sign. From the first time I tried to teach a technical typist to use TEX, I have heard some variant of the sentence “I'm just too dumb to learn this.” This is a

sign that they are intimidated by the material. The course is based on the idea that this intimidation is the most important impediment to learning to use T_EX.

Why are they intimidated? In my opinion, there are several reasons:

- As a reference, *The T_EXbook* must present a lot of material; *The T_EXbook* at once presents a typesetting system, a tutorial on typesetting, and the reference for a complex programming language. The presentation is necessarily dense.
- *The T_EXbook* is not structured as a cookbook; rather than presenting recipes, it presents the material in a form more similar to mathematics texts. Learning the material requires more than one reading.
- The material is directed toward the particular typesetting problem Knuth found most troubling: typesetting mathematical text. Unlike T_EX's technical users, many technical typists are not mathematically skilled, and often they feel a certain distaste for mathematics in general.

The course is particularly intended to avoid intimidating the students at any time. The key to this is to structure the course always to instill confidence. Some part of this is attitude: how many people have taken courses in which the instructor's managed with every word and every sentence to imply how much smarter the instructor was? But beyond questions of attitude, there are many specific steps that can be taken to instill confidence in the students.

The instructor should always be careful to praise the students repeatedly, even though I think many people feel that repeated praise is somehow patronizing. As long as the praise is sincere — which it can be with a little thought on the instructor's part in order to recognize the praiseworthy parts of any sincere effort — then students are quite willing to accept it.

The dual of the exhortation to praise repeatedly is never to punish. The greater the intimidation level of the students to start, the more easily any punishment can be interpreted as a sign that competence in T_EX is beyond the student. Once students believe they can never become competent, that belief becomes prophecy. Similarly, the more often the instructor reinforces the belief that competency can be achieved, the more likely that belief will become self-fulfilling prophecy.

The general rule should be “to instill confidence, ensure success; to build competence, ensure success in successively more difficult problems.”

The course: Structure and philosophy. *T_EX for T_EXnical Typists* is organized around a series of small tasks, chosen so that each of the tasks is slightly more than trivial, but not much more. For example, the first task is to type two paragraphs of text with no font changes and no T_EX commands whatsoever, and process it through T_EX to a printed document. These small tasks I call “units” for want of a better term.

Each unit in the course is structured around a small number of concepts. The first unit, for example, is intended to ensure that the student understands the basics of using whatever editor and operating system is used in the course, and that the student has succeeded in producing a document. These are intentionally small goals; the size of the succeeding steps, and the number of concepts included, will increase, but a starting point this simple makes it certain that the students start the course with a success.

The concepts are introduced through a specific document that serves as the focus and assignment for the units. These documents are chosen (directly influenced by Knuth) in the attempt to be light and amusing. Since I don't trust my comedic writing skills, early texts are stolen from collections of jokes and small, humorous essays. Later texts, of course, must be chosen to illustrate specific skills such as typesetting multiple integrals; the number of jokes using multiple integrals being small, the examples are composed specifically for the course.

Each unit has a few minutes of lecture, followed immediately by an experiment; this experiment first uses the concept, then extends the concept in some way that requires the student to use some creativity and thought to reach a solution. Later units necessarily build upon earlier units; this can be extended to make certain that concepts from previous units are always re-used in some later unit.

I cannot emphasize enough that *each unit is directed toward success*; those students who have some problem with a step get help from the instructor and from each other until they successfully complete the unit. (As an aside, I believe that allowing and encouraging students to help one another has several benefits. It improves the environment for everyone, not least the harried instructor. It leads the better students to help bring the less-talented along, and it also improves the more-talented students' competence.)

This approach has a number of features that can be explained psychologically. First, the structure of the units means that there is a short cycle between the introduction of a concept and its actual use, and hence little separation between the first introduction of a concept and reinforcement in correct use of the concept. From the time of Skinner's pigeons, it's been well-known that this encourages quick learning.

Second, each unit leads to the successful completion of a task that the student perceives as a difficult one; the pleasure of doing this is hard to beat. This in itself reinforces the student in the concepts used, and makes the class rather a pleasure for both teacher and student. Since the student is generally motivated for one reason or another to learn the material, successively greater achievements are very rewarding for everyone. (There is little that is better than teaching a topic to happy students.)

Finally, when concepts introduced previously are reused, and these reuses are pointed out, the interactions between the concepts is emphasized. To ensure this for all concepts, it is necessary to include some review units at the end of a course in order to make several reinforcing uses of the final concepts. As more and more such associations are built up, the students are increasingly likely to build up their own associations. This leads to greater comprehension, as the students' cognitive models are expanded.

To summarize quickly, a successful unit is built using these guidelines:

- Don't be afraid of asking questions that are too easy. Any unit that introduces anything new leads to learning. People prefer getting the right answer to the pleasure of any noble effort that doesn't lead to a right answer.
- For that reason, target the units' contents to ensure that all students achieve success. Use the talents of any more-talented people in the class to help the less-talented, in order to balance the load and prevent boredom.
- Ensure that later units use concepts from previous units. (Even the very first unit can use concepts from the students' experience as typists or compositors.)
- Plan for several reinforcing units as the last units in a course, which introduce no new concepts or few new concepts, but which ensure that all previous concepts are reinforced at last.

Discussion and conclusions. I've now taught this course many times; much of the content of this

paper comes from self-examination of my own mistakes. (Much of the rest comes from remembering the most horrible experiences of my own checkered academic career: there is a lot to be said for remembering the things one hated most in courses, and trying *not to make the same mistakes!*)

Here are some of my own mistakes:

- Require that the prerequisites be met. One of my worst experiences in the course was the time I agreed to teach an experienced group an abbreviated version of the course, only to find that the students not only were *not* experienced with T_EX but were not in general computer literate. If one cannot ensure similar competence on the part of all students, one is reduced to starting at the level of the least competent member of the group. Nothing can keep the course from being boring for some part of the rest of the group.
- Don't do requests. The course I mentioned above was an amazingly productive source of mistakes. The other major one was agreeing to abbreviate the course. A one-day course is a much different thing from a four-day course, and must be planned as thoroughly. If it's worth doing, then plan it out as carefully and call it another course.

Similarly, any extensions can be troublesome. The worst of these was attempting to add serious macro programming to the course: students without a programmer's algorithmic skills find macros completely opaque, and these skills cannot be taught in an hour.

- Provide the notes. Any course in T_EX will necessarily include a lot of details of various coding and markup conventions. You can cover lots more material if you don't expect the students to take careful notes; instead provide copies of your slides or detailed copies of your notes for each unit.
- Insist that every student have a copy of *The T_EXbook*. The students *must* be able to look things up on demand. (Plus, they must become comfortable with the book, and introducing them to it in class again reduces the intimidation.) They will need their own copy eventually, in any case.
- Ensure sufficient resources. Without sufficient resources, the delay between concept and reinforcement is longer. As the delay grows longer, the benefits of the short concept-use cycle grow less. In particular, students *must* be free to make mistakes, which means that they must

be insulated from whatever cost there may be in producing many printed results. This is particularly true of environments where there is no adequate previewer available.

Conclusions. *TEX for TEXnical Typists* has proven to be effective and successful as a way to teach composition using TEX to students for whom *The TEXbook* and other texts are not very effective: students with little technical background. These students—particularly technical writers, technical editors, and technical typists—can be taught to be extremely effective TEXnicians; they bring to the job skills and taste that technically-educated users have never had reason to develop. But to become effective, they must be presented the material in a special way.

One objection that might be made to this paper is that it is simply an advertisement for the particular TEX course I have designed. It is even in part true: I am happy to teach the course, and do charge a fee. But I think the points that I am making are essential ones for anyone trying to teach TEX; I hope that the concepts in this paper will make it possible for others to construct effective courses along the same lines as *TEX for TEXnical Typists*. The difficulty of getting started in TEX is, in my opinion, one of TEX's major problems; if we don't solve it, the seductive ease of use of "what you see is what you get" may doom us to years of poorly designed and poorly typeset documents. As with so many other problems, the solution is education.

A fit of philosophy. Document processing is moving away from a separation between the author, the person keying the document, and the person composing the typeset page, and toward a separation between the roles of creator of the text and creator of the attractive text on a page. More and more, those who compose the text on the page will take on a more creative, and less mechanical role. We will eventually need to recognize these people as craftspeople. TEX is a part of this process; TEX skills allow these people greater creativity, as they exert greater control over the results of their work. More creativity and more control over the results means more power and inevitably more professionalism. Teaching TEX to "technical typists" means teaching these skilled craftspeople ways to enhance the parts of their craft that require the most skill and are most like an art.

Acknowledgement. The course *TEX for TEXnical Typists* grew from one I originally wrote for the Physiology Department (now Cell Biology) at Duke

University and its then-chairman, Edward A. Johnson, M.D. I've been able to continue to develop this course, and to present it, at least in part through the forbearance and assistance of J. Mailen Kootsey, Ph.D., of the Departments of Cell Biology and Computer Science at Duke University. Facilities for the preparation of this paper were provided through the kind assistance of the National Biomedical Simulation Resource, supported under NIH Division of Research Resources grant 5P41-RR01693.

V_TEX Enhancements to the T_EX Language

Michael Vulis

MicroPress, Inc., 67-30 Clyde Str., Forest Hills, NY 11375
718-575-1816. Bitnet: cscmlv@ccnyvme

Abstract

V_TEX enhances T_EX by providing support for scalable fonts and thus achieving true device independence. V_TEX turns T_EX into a compact system (less than 10% of the size of traditional T_EX), supports a printer driver definition language, supplements the T_EX system with a number of new high-quality scalable typefaces, implements a variety of font effects (compression, shade, outline, shadow). Support of scalable fonts necessitated certain changes to the T_EX program, syntax, and fonts; this article describes some of these changes. Since it is likely that other scalable T_EX implementations will follow, the author hopes that a scalable T_EX standard can be defined before appearance of a conflicting set of definitions.

The Aim of the V_TEX System

Device-independence. From the beginning, T_EX was designed as a device- and resolution-independent document processor; however, because of its reliance on raster fonts, T_EX has never achieved this aim. In a typical T_EX implementation, a user is confined to particular output devices and particular resolutions by the mere necessity to maintain a large volume of raster fonts. A typical T_EX with a reasonable collection of fonts requires between 10 and 15 MB of storage; support for a second device, or just another magnification step, adds a few megabytes of storage.

The V_TEX system is based on vector, rather than raster fonts. Instead of maintaining multiple copies of raster fonts at each resolution, V_TEX keeps only one version of each font. The algorithmic encoding of characters is expanded into raster images at run time, allowing instant access to any needed magnification. Use of vector fonts not only greatly increases the number of available fonts — it also shrinks the size of the T_EX system to under a megabyte.

Changes in T_EX Syntax

Dynamic fonts in V_TEX necessitated certain changes in T_EX syntax, most importantly in the `\font` command. These changes do not interfere with the TRIP test (so V_TEX is still a T_EX), and provide much greater flexibility in choosing fonts.

scaled and at. The `scaled` and `at` parameters work as in standard T_EX, with an important exception: *any* acceptable value that follows these keywords is fully supported by all device drivers.

slant. The `slant` keyword is used to specify the amount of slant in the font. Any V_TEX font can be dynamically slanted. The number that follows the keyword is the slant coefficient, multiplied by 1000. The `slant` keyword makes the existence of `cmsl`, `cmssi` and `cmti` fonts unnecessary, since they can be obtained from other fonts. For instance, `\font\sl=cmr10 slant 167` can be used to declare a *slanted* roman font.

aspect. The `aspect` keyword specifies the aspect ratio of the font. `aspect 1000` is the default, `aspect 500` defines a half-height font, while `aspect 2000` defines a font that is twice as high as the default font. The `aspect` keyword, in particular, makes `cmdunh` unnecessary, since `cmdunh` is fairly close to an “aspected” `cmr`.

smallcaps. The `smallcaps` keyword defines a font of caps and smallcaps. Any font can be used with the `smallcaps` option. This keyword makes `cmcsc` and `cmtcsc` unnecessary.

outline, shadow, gray, and fillpattern. These keywords implement standard font effects: outline, drop shadow, gray, and pattern shading. The width of the outline and the length and direction of shadow can be specified in resolution-independent

units. The `gray` option is followed by the percentage of gray; gray is reasonably device-independent. The `fillpattern` option implements non-uniform shading, such as horizontal or vertical stripes; `fillpattern` is inherently device- and implementation-dependent. (The current implementation simply enumerates the available fill patterns).

Changes in T_EX Internals

T_EX program. Additional features supported by the `\font` command forced deep changes in the V_TE_X program. Character dimensions (widths, height, depth, and italic correction) are computed dynamically whenever a character is used. The dimensions are adjusted when the font has `slant`, `aspect`, and `smallcaps`; but no adjustment is made for `shade`, `outline`, or `shadow`. The changes to the T_EX program generally fall into two groups: the font dimension computations and the `\aliasfont` primitive. The precise specifications for changes will be made available to interested T_EX users.

TFM format. V_TE_X retains the structure of `.tfm` files but alters the meaning of some fields. For instance, the italic correction values in V_TE_X `.tfm` files are stored for fonts slanted 45° (`slant 1000`). The actual italic correction is dynamically computed by V_TE_X.

Notice that if a font is defined with `slant 1000`, no adjustments are made to the italic correction specified in the `.tfm` file. This is used with those Symbol fonts (e.g., math extensions) that should never be slanted. These fonts should be defined with `slant 1000`; a flag in the font file informs the drivers that slant should be ignored.

DVI format. The DVI format has been enhanced to support vector fonts. To retain as much compatibility as feasible, extra font information is passed as a `\special` that immediately follows a `fontdef` command. Thus, drivers that use raster fonts can handle enhanced DVI files by ignoring the “tail” of the `fontdef` command.

Font names. As mentioned above, V_TE_X does not support those Computer Modern fonts that can be obtained as attributes of other fonts. To retain source compatibility with raster T_EX sources, V_TE_X implements the `\aliasfont` primitive. This command maps font names: specifying

```
\aliasfont cmsl10=mvr10 slant 167
```

forces V_TE_X to treat all references to the `cmsl10` font as references to the `mvr10` font with the `slant 167` parameter.

Fonts

V_TE_X supports vector analogs of Computer Modern fonts. For compatibility, V_TE_X uses the same metric files as raster-based T_EX. V_TE_X fonts were developed with the InstaFont program. InstaFont is a combination of the optimized METAFONT algorithms with a user-friendly interface. InstaFont makes font development a relatively trivial task—a complete font can be designed by a novice in less than a week. (For comparison, the Euler font project sponsored by AMS featured about character-per-day performance.) About one hundred fonts have been developed, including look-alikes of such traditional typefaces as Times Roman, Helvetica, and Avant Garde.

Font Layouts

Several changes were made to the font layouts, primarily to eliminate some inconsistencies in the original T_EX layouts. In particular:

- V_TE_X fonts contain all printable ASCII characters, including the greater and less signs and braces. The original T_EX approach of borrowing these characters from symbol fonts works poorly with bold or non-CM fonts.
- The " character (`\char34`) is a straight double quote, not the closing double quote. Closing double quote is still available as the ligature ”.
- Straight quote is available, since it may be preferred to the opening quote in some abbreviations.
- V_TE_X does not “cross” L or l. Instead crossed L and l are included as separate characters. In professional typefaces, the crossing line in these letters is not a straight line.
- V_TE_X fonts include a number of additional characters, that are essential for professional typesetting. Among them are the section, paragraph and dagger signs, pound and yen, single and double guillemets, crossed D and d. Ordinary T_EX lacks some of these characters, the others are “borrowed” from symbol fonts. This works poorly even with many CM fonts: the usual section sign does not blend well with CM Bold Sans-Serif text (§§1.1).

The changes in the layout are mostly transparent, since they are compensated for by changes in PLAIN macros. Thus, as long as the user does not unnecessarily refer to characters by their position and does not use the double quote character for closing quotes, V_TE_X stays compatible.

Printer and Screen support

In order to support the maximum number of printers without having to manufacture many different device drivers, V_TE_X uses printer definition files. These files are written in a brief Pascal-like language that is sufficiently flexible to describe most graphics printers. A Printer Information Compiler (PINC) compiles the definitions to a pseudo-code, which is interpreted by device drivers during printing. Since compiled printer information files (.PIN) are very small (100–200 bytes each), V_TE_X supports many printers in minimal space. On most printers more than one resolution is supported. For instance, on the NEC P-series, the output can be written at 120×240, 240×240, 180×180, 360×180, or 360×360 dots per inch. The variety of PIN files allows the user to choose the optimal tradeoff between time and quality. PINs for non-standard printers can be easily designed by the end-user with PINC.

In a similar fashion, V_TE_X supports Screen Information files (.SIN). The “open architecture” approach of V_TE_X resulted in many PINs and SINs developed by V_TE_X users.

Performance

Runtime scaling used by V_TE_X drivers does not seriously slow down printing or previewing. The following factors contributed to the performance:

- V_TE_X scaling algorithms are very fast (up to 500 times faster than those used by METAFONT).
- Similar to PostScript, V_TE_X drivers maintain a font cache and reuse characters.
- V_TE_X allows one to pre-generate commonly used fonts. If raster fonts are available, they would be used instead of vectors. Preliminary timing experiments show that in most cases it does not pay to pre-generate more than two or three of the most common fonts. On 386 class computers, pre-generation is completely unneeded.

The current implementation limits font scaling to approximately 120–150 point fonts (at 300 dpi). This is because V_TE_X font effects require drivers to actually generate the entire bitmaps in memory. By the time this paper appears, we expect to raise the limits to about 1000 points (for a 386 CPU).

A Dirty Trick: Find-a-Font

The availability of fonts at any size makes the following example meaningful: Assume that you want to fill a box of given width and height with a

given text set in a given font. It is simple to write a generic T_EX macro that will return the required magnification of the font. In a raster-based T_EX this macro would not be especially helpful—the chances are it will return a magnification that is not available. With V_TE_X, however, the computed magnification (and, if desired, the aspect ratio) can be used immediately for actually building the box.

The same approach can be used to build an adjustable \hat macro. T_EX provides the \hat in just three sizes. It is, however, possible to compute the dimensions of a \hat that would cover a given expression, and then construct a font that would contain a \hat of the correct size. The same approach can also work with extendable delimiters.

Not Yet Implemented

Single character scaling. The examples given above will work, as long as you do not use too many individually scaled characters at once. Defining an entire font to scale a single character is overkill, since there are limits on the number of fonts and the amount of font memory available. A possible mechanism would be to allow individual scaled characters with a syntax similar to

```
\char'\^ xscaled 4000 yscaled 1200
```

Such an extension would not be difficult and may be implemented in V_TE_X in the future.

Rotation. Vector font representation, used in V_TE_X, allows easy rotation of fonts. A 90° rotation may be particularly useful. A possible syntax for such an extension would be the `rotated` keyword on \hbox and \vbox commands. Again, changes to the T_EX program would not be too difficult.

Much more exciting is a possibility of incorporating an arbitrary-degree rotation. The algorithmic font representation used by V_TE_X makes generation of rotated fonts relatively simple. On the other hand, A. Hoenig demonstrated in a recent *TUGboat* article (volume 11, number 2, pages 183–190) a set of T_EX macros that position text along slanted lines and a circle. Hoenig’s approach relies on pre-generation of a large number of fonts via METAFONT; if a 50-letter sentence is to be positioned along a circle, 50 different fonts are to be generated. Combining his ideas with V_TE_X would remove the font pre-generation element. However, an unmodified T_EX engine will still have to allocate 50 fonts, which will be extremely memory consuming. Thus, additional changes will have to be made to T_EX internals to make Hoenig’s techniques truly useful.

Graphics. A natural complement to scalable fonts would be scalable graphics. We are currently evaluating several possible approaches and would very much appreciate input from the \TeX community.

Accent Positioning. $\text{V}\TeX$ currently follows \TeX 's centering approach to accent positioning. While experimenting with CM and non-CM fonts, we have discovered many glitches in positioning (\bar{L} for one). Ideally, every font should include a correction table that specifies the amount an accent should be shifted from the default position. This type of positioning is rather similar to kerning and, in fact, can and should be specified as part of the `.tfm` kern table. Changes to the \TeX program will be minimal.

Support for older implementations

$\text{V}\TeX$ typefaces can be used with raster-based \TeX . Doing this defeats the main goals of $\text{V}\TeX$ since with older \TeX implementations adding more typefaces makes the system much more bulky. However, this would be a solution for those outside of MS-DOS world ($\text{V}\TeX$ currently runs only under DOS), and those who are not yet ready to accept scalable fonts.

To install $\text{V}\TeX$ typefaces in other \TeX 's one would use the `PXLGEN` font scaling utility that creates raster fonts in `PXL` or `GF` format. `PXLGEN` supports most of font attributes described above. `PXLGEN` uses the same scaling kernel as $\text{V}\TeX$, so scaling is extremely fast. For instance, generating a `cmr10` variant at 300 dpi takes under 10 seconds on 386/16mhz machines.

`PXLGEN`'s companion program `TFMGEN` adjusts $\text{V}\TeX$ `.tfm` files for use with non-scalable \TeX s. This is necessary, since non-scalable \TeX s require different `.tfm` files each setting of `slant`, `aspect`, and `smallcaps` parameters.

Availability. If you are interested in trying `PXLGEN`, send a self-addressed stamped disk mailer to MicroPress.

Is $\text{V}\TeX$ a \TeX ?

The question whether $\text{V}\TeX$ is \TeX or a different program came up during the discussion of this paper at the TUGboat Annual Meeting. While the ultimate judgement lies with the \TeX community, I would like to begin the discussion with a few remarks:

- $\text{V}\TeX$ is \TeX since it supports a compatibility mode, where all enhancements are disabled. Documents (and `.dvi`) files created in the

compatibility mode are fully compatible with any other \TeX .

- $\text{V}\TeX$ is \TeX since $\text{V}\TeX$ enhancements cannot be detected by TRIP. (But one can surely design a special single-purpose TRIP that $\text{V}\TeX$ will fail).
- $\text{V}\TeX$ is not really \TeX , because the way one uses it is different. Many new macros can be written that would be $\text{V}\TeX$ -specific.
- $\text{V}\TeX$ is not really \TeX , because switching to $\text{V}\TeX$ is like following a one-way street. $\text{V}\TeX$ can read generic \TeX files, but the reverse is not always true.

And, finally:

- $\text{V}\TeX$ is not really \TeX , because ultimately it will not be. Historically, a successful program has a life span of only 3–5 years. \TeX just celebrated its eleventh birthday, which is in itself an unparalleled achievement. However, \TeX is losing ground to more modern packages (notably, WordPerfect). Long-time survival of \TeX requires substantial improvements to the program, and some compromises on compatibility. We would like to consider current $\text{V}\TeX$ as just a first tentative step in this direction, and (let us hope) not the last.

Acknowledgements

The author would like to thank Lin Tay, Donald Tsai, Denny Chen and others for their help in creating $\text{V}\TeX$, early $\text{V}\TeX$ users for struggling with embarrassing bugs but not giving up, and last, but not least, Lincoln Durst, for greatly improving the readability of this paper.

Appendix

Examples of VTeX Fonts

The fonts below are all at 30 points. The examples were printed on an HP LaserJet printer at 300 dpi.

THIS IS LIKE STENCIL

This is like Hobo

This is like Broadway

This is like Futura Black

This is like Park Avenue Script

This is like Blippo

This is like Palatino Regular

This is like Cloister (Old English)

This is like Korinna Regular

This is like Euler Fraktur Medium

This is like Times Roman

This is like Windsor Bold

This is like Brush Script

This is like Clarendon Bold

This is like Amer. Typewriter

This is like Handel Gothic

This is like Avant Garde Bold

Examples of V_TE_X Effects

The examples below were printed by V_TE_X on an HP LaserJet II printer at 300 dpi.

`\font\test=mvavnb scaled 3000 outline...`

This is an Outline font.

`\font\test=mvhlvb scaled 3000 shadow outline...`

This is Shadow with Outline.

`\font\test=mvtnsb scaled 3000 smallcaps fillpattern 3...`

THIS IS SHADED SMALLCAPS.

`\font\test=mvkorb scaled 3000 outline fillpattern 7...`

This is an Outline with Shade.

`\font\test=mvfrac scaled 3000 shadow fillpattern 12...`

This is a Shadow with Shade.

`\font\test=mvavnm scaled 3000 aspect 800...`

This is an expanded font.

`\font\test=mvavnm scaled 3000 aspect 1200...`

This is a compressed font.

`\font\test=mvpalr scaled 3000 slant 250...`

This is slanted right.

`\font\test=mvpalr scaled 3000 slant -250...`

This is slanted left.

`\font\test=mvssbx10 scaled 3000 slant 200 outline smallcaps fillpattern 6...`

SMALLCAPS, SHADED, SLANTED ...

A Constructed Dürer Alphabet

Alan Hoenig

John Jay College, City University of New York; Mailing address: 17 Bay Avenue, Huntington, NY 11743
516-385-0736

Abstract

The author used METAFONT to implement the design of a Roman uppercase alphabet by Albrecht Dürer. Although Dürer intended only a Roman alphabet, an attempt was made to create METAFONT programs to generate related fonts in a bold, sans serif, typewriter-like, slanted, and casual style. (The last of these is a style inspired by informal Roman letterforms designed by Sumner Stone and Neenie Billawala.)

Introduction

In the year 1525, the German artist Albrecht Dürer published a work on art techniques which included his specifications for an uppercase Roman alphabet. Dürer was one of a series of artists who, fascinated by the beauty of Roman capitals especially as they appeared on antique Roman monuments, sought to generate recipes for their construction. The recipes would only involve circular arcs and straight line segments. Knuth [1979] lists many of these efforts, and they have continued right till the present day (see Goines [1982]).

Published accounts of these alphabetic constructions reveal that, for the most part, these alphabets are dreary and lifeless. Dürer's alphabet is an exception, and there are two reasons for that.

1. Dürer was far superior an artist to any of these others, and
2. Dürer cheated. Many of his letters require curves other than circular arcs, and some other little flourishes and touches that add zest to the plates that accompanied his work are not specified at all in the printed description.

Nevertheless, I thought it would be an interesting project to use METAFONT to create the uppercase font for use by \TeX and other typesetting systems. The purpose of this presentation is to describe this project.

Dürer's Design Scheme

Dürer imagined each letter as inhabiting the inside of a square. Fractions of each side determine the linear dimensions of each letter. Let the length of each side be s . Then for example, all thick stems of letters (such as the verticals in I and H or the

thickest parts of the curves in S) are to be one-tenth of s , and all thin stems (such as the horizontal cross stroke in H) are to be one-third as broad as the thick stroke. All seriffed corners are constructed the same way, out of broad strokes and filled-in quarter-circles.

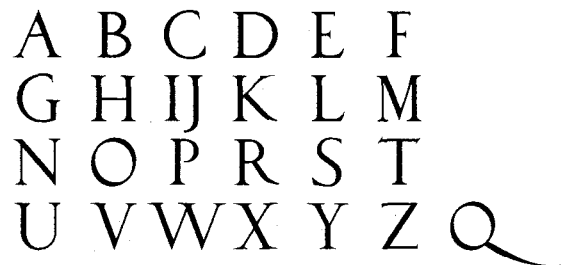


Figure 1: Dürer's uppercase Roman alphabet.

METAFONT proved adept at creating these letters, and at smoothing over those portions of the curve which are not strictly circular or straight. After all, one basic design goal of the METAFONT system is to create pleasing curves. However, when METAFONT was constrained to use circular arcs as much as possible, it took more time than usual to create each letter. Figure 1 displays this font.

It's helpful to examine the construction of one of the letters, such as the capital D, to see how Dürer's design scheme works (see Figure 2). Begin by laying out a square of side s . Next, mark two serifs at the upper and lower left corners. The serifs are framed by quarter-circles whose radius is $T = s/10$. Abutting the serifs, draw the vertical stem, whose thickness is also T .

Two short horizontal bars will connect the stem to the curved bowl. These bars are each of thickness $t = T/3$ and of length $(1/2 - 2/10)s$. The outermost outline of the curved lobe will be a

semi-circle whose radius is $s/2$. The thickest part of the lobe will be of thickness T , and will connect the upper and lower thin bars. This inner arc will have diameter $s - 2t$ and radius half that. The center for this circle will be at the point with coordinates $(1/2s - (T - t), s/2)$; the origin of this coordinate system is at the lower left corner of the bounding box. Round the lower inner corner by adding a serif-like quarter-circle, and, finally, fill in the entire outline. (In METAFONT, the construction technique is easier to implement than it is to describe.)

In the same way, Dürer presents descriptions for 22 other letters. (His account omits the J, U, and W.) Competent modern translations of Dürer's letter programs together with the illustrative plates are listed in the bibliography.

Generalizing the Design

Dürer's alphabet turns out to be surprisingly handsome, and great fun to implement with METAFONT. (See Figure 1 for specimen letters.) As I entered the terminal phases of this project, I was struck by one inadequacy. Although the alphabet, which I call Computer Dürer Roman (CDR), was a METAFONT font, it was not a true meta-font, because there were few parameters to vary. With properly designed meta-fonts, such as the Computer Modern or Pandora families, it's possible to twirl and twist software "knobs" to generate whole families of different (but compatible) fonts.

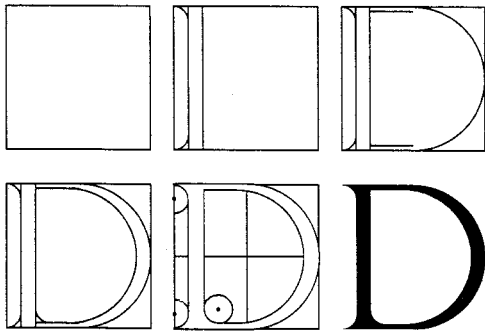


Figure 2: Dürer's construction of the letter D. In the penultimate frame, the D appears surrounded by the "scaffolding" as it appears in Dürer's plates.

In my opinion Dürer's design specifications themselves left little room for adjustment. In Figure 3, for example, we see Dürer's H together with several variants obtained by varying the radius of the serif quarter-circle, which is also the thickness of the vertical stems. There does not seem to be much room for parameterization in this design scheme, and varying the serif radius leads to designs

which aren't particularly pleasing. The bottle-neck to the metamorphosis to a true meta-font seems to be Dürer's construction of serifs, tied as they are to quarter-circles.

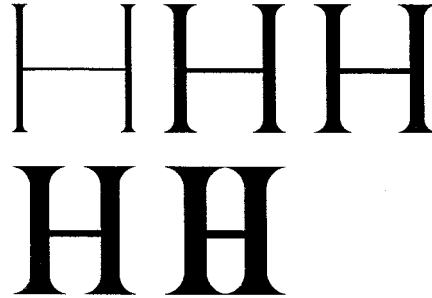


Figure 3: Varying the serif radius and stem thickness for the letter H. The second letter shows the H as Dürer wanted it. The third letter shows an H where the stem thickness is one-ninth the side, a fraction given by other letter designers contemporary with Dürer.

The key to successful parameterization of the Dürer font lies in rethinking the arcs that constitute the serifs and bowls. For example, it is possible to design a "circular," Dürer-like serif in the following way, motivated by techniques used both by Don Knuth and Neenie Billwala in their METAFONT constructions.

For this demonstration and discussion, imagine that we want a serif in the top left position of a stem, such as the upper serif of a D. Draw a pair of horizontal upper bars separated by a distance d ; this is the *slab*. In CDR, this value is $d = 0$, so the pair of slab strokes collapses into the single horizontal stroke we see in Dürer's serifs. Draw also the vertical bar that frames the serif; this is the *bracket*. Call the point where the slab and the bracket intersect the *corner point*.

Now construct an auxiliary point which lies midway between the endpoints of the slab and bracket. Hypothesize the existence of a *darkness* parameter whose meaning will shortly be explained. It will be a number between 0 and 1.

Create the curved serif path as one which starts at the end of the horizontal member, proceeds right, passes through a point which is *darkness* of the way between the auxiliary point and the corner point, and ends up travelling down at the end of the vertical member. Finally, connect the ends of the slab strokes by a smooth curve.

Now we have a serif whose exact appearance depends on four parameters:

1. d , the thickness of the slab

2. the length of the slab
3. the length of the bracket
4. the value of the *darkness* parameter

See Figure 4 for a diagram of a generalized Dürer serif.

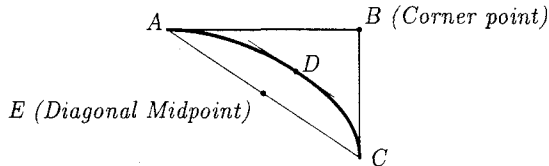


Figure 4: One way to draw the arc of a serif. The line at point *D* is parallel to the line connecting *A* and *C*. The heavy line traces the outline of the serif.

When $d = 0$, when the vertical and horizontal serif strokes are both equal to $s/10$, and when *darkness* = $1 - \sqrt{2}/2 \approx 0.29$, the serif that METAFONT draws is visually indistinguishable from a Dürer quarter-circle serif (although the equation of the curve that METAFONT draws is *not* that of a circle). Under this interpretation of a Dürer serif, it now becomes possible to hope that meta-descriptions of letters are possible that will reproduce Dürer's letters for one value of the parameters, and generate a series of related fonts for other values of the parameters (see Figure 5).

In order to exploit this insight, all the programs were redone in order to incorporate "the METAFONT way" in a better fashion. Figure 6 displays examples of non-Roman Computer Dürer fonts, specifically a boldface, slanted, sans serif, a monospaced typewriter-like font, and a *casual* font (together with CDR for purposes of comparison). The casual font was inspired by a family of fonts created by Sumner Stone of Adobe Systems and by some of the variant Pandora letterforms designed by Neenie Billawala [1989]. Stone's fonts are METAFONT-like in that a whole family of fonts was designed to be visually compatible; specimens will be found in recent issues of the Adobe Type Catalog, *Font & Function*. His Stone Informal fonts attempt to provide a font with an obvious connection to a formal Roman font, but which by looser joins of straight strokes, a more flowing sinuosity of the curves, and occasional omission of a serif communicates an informal, casual bearing.

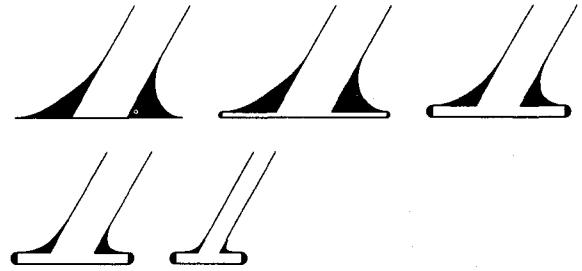


Figure 5: Examples of meta-serifs. The initial serif is that of Dürer.

Design Limitations

In my creation of a meta-design, I tried to restrict myself to the macros Knuth created for his Computer Modern types. There were several reasons for this.

1. I hoped to reduce the time for design by using these available tools. Creation of a set of robust METAFONT macros is far from trivial.
2. I felt I would learn more about type anatomy by trying to fit CDR into a CMR mold.
3. In any full-scale font design scheme, a close following of Knuth's macros may make it easy to adapt all the math fonts for the new family. Although that was not a concern here, I thought it best to cultivate this style.
4. With luck, I would be able to adapt existing members from the Computer Modern family for characters not designed by Dürer (see below).

I was not successful in this endeavor. METAFONT type design macros are not valid for any but a single family of type fonts. However, I was able to adapt pre-existing CM macros for many of the tasks I needed.

A More Useful Font?

A major disappointment in this project arose because of the limited uses of this font. Not only are there no lowercase letters, no punctuation marks, and no numerals (Dürer created no designs for them), but the uppercase alphabet is not even complete. It lacks the J, U, and W, which did not exist in those days. I was able to jury-rig a U applying the existing principals of this font, a W from two V's, and a purely ad hoc design for the J, but these letters suffer in comparison with their mates.

In an attempt to render this font more useful, I generated some CMR10 fonts with special values of the parameters and also some minor tinkering with the program files. Therefore, these fonts

are no longer members of the Computer Modern family. I replaced as many of Knuth's values in the file CMR10.MF by Dürer-like values to see if the resulting font was in any way compatible with the cap fonts CDR10. For example, I identified the side *s* of the square with METAFONT's *cap_height* parameter, and let *thick_stem* be *.1cap_height*. Minor adjustments to letter programs were needed to eliminate some visual clashes between Computer Modern fonts and the Dürer font. A sample of typesetting mixing the Dürer caps with the modified Computer Modern font appears in Figure 7 shown at a 10-point, magstep 3, size. For comparison, the same paragraph is shown in the corresponding Computer Modern font. It looks nicer than I expected, but the results should not be taken too seriously.

DUERER A I P U
DUERER A I P U
DUERER A I P U
DUERER A I P U
DUERER A I P U
DUERER A I P U

Figure 6: Members of the Computer Dürer family of fonts: bold, slant, sans serif, monospaced typewriter, and casual.

An additional lesson I learned from this project is that type design is a difficult craft in its own right, and while it is feasible to assign the task of rendering an existing font in METAFONT to someone like myself, it is *not* feasible to ask this person to design type.

ACKNOWLEDGMENTS. I am grateful for the financial support of the Research Foundation of the City University of New York, grant number 669243, for this project. I thank Barbara Beeton, David Ness, and Hermann Zapf for several helpful comments.

Bibliography

Billawala, Neenie. *Metamarks: Preliminary studies for a Pandora's Box of shapes*. Stanford, CA: Computer Science Department, Stanford University (report STAN-CS-89-1256), 1989. (Copies

obtainable from the T_EX Users Group, P. O. Box 9506, Providence, RI 02940-9506 USA.)

Dürer, Albrecht. *The Painter's Manual*. Trans. Walter L. Strauss. New York: Abaris Books, Inc., 1977.

Dürer, Albrecht. *Of the Just Shaping of Letters*. Trans. R. T. Nichol from the Latin text of the edition of 1535. New York: The Grolier Club, 1917; reprinted by Dover Publications, 1965.

Goines, David Lance. *A Constructed Roman Alphabet*. Boston: David R. Godine, Publisher, Inc., 1989.

Knuth, Donald E. *T_EX and METAFONT: New Directions in Typesetting*. Bedford, MA: The Digital Press, 1979.

Uppercase Easy Does It
Puerto Rico 574 Argyle Road
a b c d e f g h i j k l m n o p
q r s t u v w x y z

Figure 7: (a) CDR10 completed by means of a hacked-up CMR10 font.

Uppercase Easy Does It
Puerto Rico 574 Argyle Road
a b c d e f g h i j k l m n o p
q r s t u v w x y z

Figure 7: (b) For comparison, the second specimen is typeset in Computer Modern at the same design size.

Abstracts

Deutsche Kurzfassungen der *TUGboat*-Artikel [German Abstracts of *TUGboat* Articles]

Luzia Dietsche

E-TeX: Richtlinien für zukünftige Erweiterungen von TeX (F. Mittelbach, S. 337)

Mit der Ankündigung von TeX 3.0 erkannte Don Knuth das Bedürfnis der (immer mehr anwachsenden) TeX-Gemeinde nach einem verbesserten System an. Gleichzeitig machte er aber auch deutlich, daß er an keinerlei weiteren Entwicklungen, die Änderungen im TeXbuch mit sich bringen würden, beteiligt sein wird.

TeX war ursprünglich dazu entwickelt worden, die eigenen Veröffentlichungen des Autors zu setzen. Inzwischen verwenden es Tausende von Benutzern. Nach 10 Jahren Erfahrung ist es jetzt an der Zeit, sich zu überlegen, ob TeX 3.0 eine adäquate Antwort auf die Anforderungen von Setzern in den 90er Jahren sein kann oder nicht.

Eine Ausgabe, die mit TeX erzeugt wurde, hat einen höheren Standard als mit den meisten anderen Textsatzsystemen erzeugte Ausgaben. Deshalb werden wir uns in diesem Bericht auf den Qualitätsstandard, den Buchdrucker für "handsatzte" Dokumente aufgestellt haben, konzentrieren und fragen, inwieweit dieser Standard von TeX verwirklicht wird. Die Grenzen von TeX's Algorithmen werden analysiert, fehlende Besonderheiten sowie neue Konzepte werden umrissen.

Senkrecht Setzen mit TeX (H. Hamano, S. 346)

Als technischer Verlag hatte ASCII Corporation schon lange den Eindruck, daß auf dem japanischen Markt ein tatsächlich japanisches Textsatzsystem nötig wäre. In einem ersten Schritt entwickelten wir vor drei Jahren eine japanische Version von TeX, mit der es möglich war, *Kanji* zu verwenden.

Dieser Bericht stellt nun unseren zweiten Schritt vor, in dem wir ein erheblich komplizierteres TeX-System entwickelten — das Hinzufügen einer Funktion, mit der man senkrecht Setzen kann.

Ein strukturiertes System zum Erstellen von Dokumenten *AutoLayouter* (Y. Miyabe, H. Ohta, K. Tsuga, S. 353)

Wir haben ein strukturiertes System zum Erstellen von Dokumenten entwickelt (*AutoLayouter*), das aus einem einfach aufgebauten Editor und einem Formtierer besteht, der auf japanischem L^AT_EX basiert.

Dazu haben wir aber nicht nur eine bessere Benutzeroberfläche entwickelt, sondern auch eine einfache Dokumentenstruktur eingeführt. Ein Dokument, das mit *AutoLayouter* hergestellt wurde, ist eine Liste, in der jeder Knoten einer logischen Komponente des Dokuments entspricht. Diese Verwendung einer einfachen Struktur ermöglichte es, den Editor einfach in der Anwendung zu gestalten und dennoch mächtig genug zu lassen, um sowohl die Struktur des Dokuments als auch seines Inhalts, der zusätzliche feinere Untergliederungen enthalten kann, zu editieren.

Seit der Verwendung dieser einfachen Struktur mußten wir verschiedene Makros hinzufügen, um die Unterschiede zwischen unserer Struktur und den *begin-end* Umgebungen von L^AT_EX auszugleichen. Zusätzlich zum Editor haben wir einen Treiber entwickelt, der *dvi*-Files in *Kanji* PostScript Files umwandelt.

TeXnisch werdend: Einblicke in Techniken, mit denen man TeX-Makros schreiben kann (A. Hendrickson, S. 359)

Die meisten von uns verstehen die Grundformen von TeX-Makros; allein das bloße Verstehen ist oft ungenügend, wenn wir bestimmte Probleme zu lösen haben. Wir benötigen dann zusätzliche Einblicke, um Methoden entwickeln zu können, mit denen man Informationen weitergeben, Text mit verändertem Catcode verschieben, Leerzeile erhalten kann und vieles mehr. Schreibt man ein großes Makropaket, sind eine Reihe neuer Aspekte zu beachten: Wie kann man es vermeiden, über die Einschränkungen der diversen Implementationen zu stolpern, wie sie z.B. die Zwänge von *hash size*, *string size* und andere Größen darstellen? Wie macht man eine komfortable Benutzeroberfläche? Wie macht man die Eingabe so knapp wie möglich?

Einige der Techniken, die hier vorgestellt werden, zeigen, wie ein Makro mit variabler Anzahl von Argumenten geschrieben werden kann; wie man Catcodes in Makros ändern kann, indem man ein anderes Makro definiert, dessen Argument absichtlich nie benutzt wird; wie man den Wert von *hash size* gering halten kann, indem man Zähler anstelle

von `newifs` verwendet; Techniken mit `cname` und `non-outer` dynamischen Allokierungen; Techniken, um Tabellen zu setzen; einen Blick auf `output routines`. Zu guter Letzt folgen Arbeitsmethoden, die man beim Schreiben neuer Makros verwenden kann.

Wo ist der Umschalter für griechische Buchstaben? (S. A. Fulling, S. 371)

Erfahrene Setzer mathematischer Formeln ermüden bald beim Tippen der Namen von griechischen Buchstaben. Meist schreiben sie dafür einfache Makros wie z.B. `\za` für `\alpha`. Hier wird nun eine Standardübereinstimmung zwischen griechischen und lateinischen Buchstaben vorgestellt, in der phonetischer Ähnlichkeit Vorrang vor typographischer gegeben wird. Mit den daraus resultierenden Makros wird es möglich, beliebige griechische Buchstaben durch dreifachen Anschlag zu tippen (oder sogar durch zweifachen, wenn man `\z` auf eine Funktionstaste legt).

TransFig: Übertragbare Grafiken für T_EX (M. Beck, A. Siegel, S. 373)

Das TransFig Software Paket definiert eine übertragbare Beschreibungssprache für technische Grafiken. Es werden Übersetzungen von dieser Sprache in oft verwendete Grafik-Beschreibungsformate geliefert, die dann in eine Eingabe eingebunden werden können. TransFig beinhaltet außerdem einen ausgesprochen angenehmen Rahmen, um Figuren in L^AT_EX einzubinden. Die Grafik-Sprache, die mit TransFig definiert ist, ermöglicht den Austausch von strukturierten und modifizierbaren Grafiken zwischen verschiedenen Anwendungen. In diesem Bericht beschreiben wir unsere Erfahrungen mit TransFig, um die Notwendigkeit einer standardisierten Grafik-Sprache auf Anwendungsebene zu verdeutlichen und schlagen einige Richtlinien für deren Entwicklung vor.

B_AS_IX: Ein in T_EX geschriebener Interpreter (A. M. Greene, S. 381)

Ein komplett in T_EX geschriebener Interpreter für BASIC wurde entwickelt. Er stellt Techniken zum scannen und analysieren zur Verfügung, die oftmals da von Nutzen sind, wo Daten, die keine Formatierungsvorschriften enthalten, mit T_EX formatiert werden sollen. T_EX's Expansions Regeln werden ausgenutzt, um ein Makro bereit zu stellen, das den Rest der Eingabe als Argument einliest und nicht mehr aufhört, ihn zu expandieren.

Ein Anfängerleitfaden für die Benutzung von T_EX in einer Textproduktion: Vom Manuskript zum Bromid (H. Gibson, S. 393)

Die Absicht dieses Berichts ist es, ein Beispiel für eine PC-Anpassung zu geben, mit der qualitativ hochwertige Druckausgabe erzeugt werden kann. Einer kurzen Beschreibung der Anforderungen, die an Veröffentlichungen des Wellcome Institutes gestellt werden, folgt eine ausführliche Darstellung der Hard- und Software-Konfiguration, der Umlenkung der Ausgabe zu PostScript und der Verwendung von elektronischer Kommunikation, mit der auf Laserdruckern kontrollierter Text auf eine Linotronic 300 Lichtsatzmaschine geschickt werden kann. Außerdem wird der Wert der Verwendung von WordPerfect Makros, alternativer Keyboard Layouts und Style Files als eine Oberfläche für die Eingabe durch Sekretärinnen dargestellt.

Probleme mit der T_EX/PostScript/Grafik Oberfläche (R. A. Adams, S. 403)

In diesem Bericht werden die unterschiedlichen Probleme behandelt, die bei der gleichzeitigen Verwendung von T_EX und PostScript zur Produktion von zwei Formelsammlungen auftreten können. Drei solcher Probleme sind besonders wichtig. Das erste ist es, eine vernünftige Kombination von mit PostScript (skaliertem) Text und den mathematischen Formeln zu erhalten, die auch noch bei einer 1270 dpi Ausgabe auf einer Linotronic Lichtsatzmaschine gut aussehen. Das zweite Problem ist, eine brauchbare Methode zu ersinnen, um eine passende (und gut ausgerichtete) 2-Farben-Trennung von Text und Grafik zu erhalten. Das dritte bringt es mit sich, T_EX Marken mit PostScript Grafiken zu verbinden. Die Lösungen zu diesen Problemen sind bestimmt durch die Software, die zu der Zeit vorhanden war, als sie benötigt wurden (vor ca. 1½ Jahren).

T_EX in der Praxis: Kommentare zu einem 4-bändigen, 1400 Seiten starken Werk in T_EX (S. v. Bechtolsheim, S. 409)

Dieser Artikel handelt von einem 4-bändigen, 1400 Seiten starken Werk über T_EX. Zwei Aspekte werden behandelt: Als erstes werde ich zeigen, wie die Produktion eines Dokuments dieser Größe organisiert wurde. Als Zweites zeige ich Erweiterungen in T_EX, die ich dazu für wünschenswert hielt.

Veröffentlichung von Bücher — 1990 und darüber hinaus (M. L. Lafrenz, S. 413)

Die Produktion universitärer Veröffentlichungen und Handbücher mit \TeX ist etwas Selbstverständliches. Trotzdem verstehen, und dadurch akzeptieren, Verlage erst allmählich die Mächtigkeit und Vielfalt dieses Programms. Einige Verlage verwenden bereits \TeX -Files von Autoren, um Kamerafertige Seiten zu produzieren, während andere nur sehr zögernd diese Möglichkeit in Betracht ziehen. Der Widerstand von kommerziellen Anwendern, Makros, Tricks und Techniken auszuplaudern, verhinderte die Akzeptanz von \TeX durch die Verlage.

Wir werden die Schulung und Unterstützung von Autoren und Verlegern sowie die Einwirkung von Offenheit auf den technischen Bereich untersuchen. Nur durch die Aufteilung in verschiedene Gebiete werden wir in der Lage sein, die beste Umgebung zu schaffen, um \TeX in diesem Jahrzehnt zur Blüte zu bringen.

\TeX -Fehler mit einem Präprozessor diagnostizieren (D. Ness, S. 417)

\TeX findet unsere Fehler mit Leichtigkeit. Allerdings teilt es uns diese manchmal auf eine schwer verständliche Weise mit. Normalerweise kommt das dadurch, daß wir \TeX durch die unbegründete Verwendung von Irgendetwas irritieren. Ein einfaches Beispiel: Wenn wir vergessen haben, vor ein Dollar-Zeichen ein Befehlszeichen zu setzen, gibt uns \TeX im Normalfall eine eigenartige Meldung über den mathematischen Modus zurück. Der vorliegende Bericht handelt von einem Präprozessor, der uns vor potentiellen Fehlerquellen warnt, bevor wir unsere Files von \TeX bearbeiten lassen. Besonders für Neulinge kann dieses Programm von Nutzen sein.

Zusätzliche \TeX Leistung durch die Verwendung von erweiterten EDT Editor-Möglichkeiten (L. Williams, L. Hall, S. 421)

\TeX ist auf vielen Computersystemen installiert, allerdings oft mit wenig oder gar keinen Gedanken an die zeitsparenden Vorteile eines leistungsstarken Editors. Sowohl für Neulinge als auch für erfahrene \TeX er eröffnet das Arbeiten mit den erweiterten EDT Editor-Möglichkeiten den effizienten Gebrauch von \TeX . Da mehrere Bände dessen verschiedene Hilfsmittel beschreiben, wurde es nötig, diese entsetzliche Anhäufung von Material auf ein anwendbares Minimum zu reduzieren. In diesem Artikel werden einzelne EDT-Möglichkeiten vorgestellt, wie Editor-Initialisierungs-Files und andere Komman-

dos, die, wenn sie richtig angewandt werden, die Editor-Möglichkeiten von \TeX noch steigern.

\TeX für \TeX nische Sekretärinnen (C. R. Martin, S. 425)

Viele \TeX Benutzer sind keine Programmierer oder Mathematiker, sondern Fachsekretärinnen, Praktiker eines erlernten Handwerks. Solche Benutzer finden bereits vorhandene \TeX -Eingaben oftmals einschüchternd und geheimnisvoll. \TeX für \TeX nische Sekretärinnen ist gerade für solche Benutzer gedacht. Jede Einheit beschreibt einige Konzepte, wobei diese sofort durch praktische Anwendungen in einem kurzen Dokument vertieft werden. Schüler können sich die Resultate direkt ansehen, was zu schnellerem Fortschritt und größerer Zufriedenheit führt. Der Kurs fördert eine experimentelle Haltung, die sich in der Praxis gut bewährt hat. \TeX für \TeX nische Sekretärinnen scheint ein effizienter Weg zu sein, mit dem Fachkräften beigebracht werden kann, schöne Dokumente zu erstellen.

$\text{V}\text{\TeX}$ Steigerungen zur \TeX -Sprache (M. Vulis, S. 429)

$\text{V}\text{\TeX}$ erweitert \TeX um die Unterstützung durch skalierbare Fonts und damit um das Erlangen echter Treiber-Unabhängigkeit. $\text{V}\text{\TeX}$ bringt \TeX in ein kompaktes System (weniger als 10% seiner ursprünglichen Größe), unterstützt eine Druckertreiber Definitionssprache, vervollständigt das \TeX -System mit einer Anzahl von neuen, qualitativ hochwertigen Schriften und implementiert eine Reihe von Font-Effekten (verdichtete, schraffierte, umrißene, mit Schatten versehene Fonts). Die Unterstützung von skalierbaren Fonts erfordert bestimmte Änderungen im \TeX Programm, der Syntax und den Fonts; dieser Artikel beschreibt einige der Änderungen. Da es wahrscheinlich ist, daß weitere Anpassungen für ein skalierbares \TeX folgen werden, hofft der Autor, daß eine Standardisierung darüber gefunden werden kann, bevor Konflikte durch verschiedene Sätze von Definitionen entstehen.

Ein konstruiertes Dürer Alphabet (A. Hoenig, S. 435)

Der Autor benutzte METAFONT, um ein Albrecht-Dürer-Alphabet in Roman Großbuchstaben zu entwickeln. Obwohl Dürer nur an ein Alphabet in Roman-Schrift dachte, wurde der Versuch unternommen, ein METAFONT-Programm zu entwickeln, mit dem verwandte Fonts erzeugt werden können in einem bold, sans serif, typewriter-ähnlichen, slanted und casual Stil. (Das letzte ist ein Stil in Anlehnung an die formlose Roman Buchstabengestaltung, die

von Sumner Stone und Neenie Billawala entwickelt wurde.)

Luzia Dietsche
Rechenzentrum der Universität
Heidelberg
Im Neuenheimer Feld 293
D-6900 Heidelberg 1
X68@DHDURZ1

Status Reports

TeX 3.0 and METAFONT 2.0

Nelson H.F. Beebe

Introduction

During the last few years, use of TeX has spread to Chinese, Japanese, Korean, Coptic, Russian, Thai, Vietnamese, several Indian languages, Persian, Arabic, Hebrew, and all major European languages. These uses made some of the limitations of TeX 2.x more evident. The most serious of these are the seven-bit character set and uni-lingual hyphenation. Consequently, Donald Knuth announced at the 10th Annual Meeting of TUG, held at Stanford University in August, 1989, that a new version of TeX and METAFONT would be produced to address problems of multinational support; those versions were officially released on the ides of March, 1990. Don resisted calls for more sweeping changes, in the interests of (a) restricting the impact of the changes, and (b) getting back to writing *The Art of Computer Programming* books.

At the time of writing this report (late June 1990), the two new programs, and their related TeXware and METAFONTware, including BibTeX, have been successfully installed on numerous UNIX variants, using version 5.0c of the Web2C translator, as well as on the IBM PC, VAX VMS, and IBM VM/CMS. Several commercial vendors announced release of the new versions at the June TUG meeting. We hope that, by the end of 1990, all TeX users the world over will have had the opportunity to upgrade to the new versions.

What's New

The new features of TeX 3.0 and METAFONT 2.0 are described in detail in [3]. It is important to

emphasize that, apart from the obscure exceptions noted in section 12 of that paper, changes should not affect existing TeX and METAFONT files. In particular, *DVI files are unchanged, and old font files can still be used.*

Here is a list of the current support programs and their versions:

bibtex	0.99c	pltotf	3.2
dvitype	3.2	tangle	4.0
gftodvi	3.0	tex	3.0
gftopk	2.2	texinfo	3.0
gftype	3.0	tftopl	3.1
inimf	2.0	vftovp	1.0
initex	3.0	virmf	2.0
mft	2.0	virtex	3.0
pktofg	1.0	vptovf	1.0
pktype	2.2	weave	4.1

Hyphenation can be applied to input words coded using only 8-bit characters. This means that a user with a European keyboard can enter a word like *liberté* (the accented letter is a single character) without losing a hyphenation opportunity. However, if the word is entered using a control sequence, `libert\'e`, as for earlier versions of TeX, hyphenation will not be tried.

Michael Ferguson has graciously made available a change file for TeX 3.0 that translates input accented character sequences to the internal 8-bit form, and at DVI output time, translates them back again. This change makes it possible for words accented with control sequences to be hyphenated, and seems to be a very valuable extension.

The new ligature mechanism allows letters to change, based on their position within a word. This is of particular importance in Arabic, but may be used in other languages: observe that in normal handwriting, letter shapes may also vary according to position, so a handwriting font might usefully incorporate such ligatures.

`\emergencystretch` (section 7 of [3]) allows better control of excessive white space. L^AT_EX users in particular should note this new parameter. The L^AT_EX `\sloppy` command sets `\tolerance` to its maximum value of 10000, allowing all lines to be loosely spaced. Under TeX 3.0, a smaller `\tolerance` value can be chosen and `\emergencystretch` can be brought into play when TeX deems it necessary, thus eliminating excessive white space. The L^AT_EX bibliography environment by default invokes `\sloppy`; try a two-column bibliography first without `\emergencystretch`, then with it and a reduced `\tolerance` to see the difference.

The changes to the source code are extensive: Don Knuth reports that 218 of TeX's 1377 modules have been changed, and dozens of modules have been completely rewritten.

It was decided not to issue a new edition of the TeXbook and the METAFONTbook; instead, the new version of TeX will be described in a new printing of Volume A of *Computers and Typesetting*, identified by "Ninth printing" on the copyright page. The paperback version of the TeXbook will be updated in a few months; it will be marked "Seventeenth printing". Presumably, volumes B, C, and D will be similarly updated.

The finder's fee for bugs in the new code is \$10.24. If you discover a bug in the "old" parts of TeX while you're installing the new version, you win \$163.84.

Why You Should Upgrade

Many people adhere to the "if it ain't broke, don't fix it" rule, also known as the "if it is not necessary to change, it is necessary not to change" axiom. These apply to many things besides software. Consequently, some users will no doubt resist an upgrade, particularly if they see no immediate need for the multinational features in their own documents. Don Knuth views these changes as so important that he wrote [3]: *Let us root out and destroy the obsolete 7-bit systems, even though we were able to do many fine things with them.*

One of the strengths of TeX lies in its ability to produce the same output from the same input, no matter what the host computer is. Macro writers, and European TeX users in particular, will be quick to take advantage of the new features; if your system lags behind, you risk being unable to process new documents and macro files in the future.

Character Set Issues

We need to exercise great care in the use of eight-bit input in TeX documents that are intended to be exchanged with others in their original input form, because positions 128...255 in the character sets do not have universally accepted assignments. Without such an agreement, we seriously risk the loss of the wonderful TeX file portability that we have heretofore enjoyed. For detailed discussion, see [1, 2] and references cited therein.

Because of the gravity of the character set problem, TUG and several European groups have initiated a joint effort to rapidly produce a proposed character set assignment that can be used for the exchange of TeX documents. An initial progress re-

port is planned for the TeX90 meeting in Cork in September, 1990.

Composite Fonts

Eight-bit characters will simplify TeX input for many users, but what about fonts? Don Knuth certainly has no time to sit down and expand the Computer Modern and Concrete Roman families from 128-character fonts to 256-character fonts. Some commercial fonts already provide larger character sets, but one still has the problem of finding a DVI driver that can support them.

The solution to these is composite fonts [4], also known as 'virtual' fonts. I'd like to take this opportunity to issue a plea for a name change; 'virtual' is not very descriptive, while 'composite' is, and has the advantage of already being in wide use. Also, the term 'virtual fonts' has been in use in my DVI drivers since 1986 with quite a different meaning.

Composite fonts provide for the construction of new fonts, each character of which may be made up of pieces taken from one or more other (possibly composite) fonts, where the pieces are positioned by commands in the DVI language.

Composite font support requires either a DVI-to-DVI program that can convert a DVI file using composite fonts to one that does not, or addition of new code in DVI drivers. Since a lot of existing software is affected by this new feature, it is going to take considerably longer to get such programs developed and distributed.

References

- [1] Nelson H.F. Beebe. Character set encoding. *TUGboat*, 11(2):171-175, June 1990.
- [2] Janusz S. Bień. On Standards for Computer Modern Font Extensions. *TUGboat*, 11(2):175-183, June 1990.
- [3] Donald Knuth. The New Versions of TeX and METAFONT. *TUGboat*, 10(3):325-328, November 1989.
- [4] Donald E. Knuth. Virtual fonts: more fun for grand wizards. *TUGboat*, 11(1):13-23, January 1990.

◇ Nelson H.F. Beebe
Center for Scientific Computing and
Department of Mathematics
South Physics Building
University of Utah
Salt Lake City, UT 84112 USA
Tel: (801) 581-5254
Internet: Beebe@science.utah.edu

L^AT_EX 2.10

Frank Mittelbach

Remarks on L^AT_EX 2.10 + 3.0

At the TUG meeting in Texas, I was able to announce the availability of the new font selection scheme which will be incorporated in the new L^AT_EX and explained its features. Further work is ongoing, including the redesign of the internal style interface and a new attribute concept. A more detailed talk about this project will be given at the Cork meeting in September.

- ◊ Frank Mittelbach
Electronic Data Systems
(Deutschland) GmbH
Eisenstraße 56 (N 15), D-6090
Rüsselsheim, Federal Republic
of Germany
Tel. +49 6142 803267
Bitnet: pzf5hz@drueds2

International Reports**DANTE, Deutschsprachige
Anwendervereinigung T_EX e.V.**

Joachim Lammarsch

On April 14, 1989 in Heidelberg, DANTE (the Deutschsprachige Anwendervereinigung T_EX e.V.), was founded. I was elected chairman of the society, and we also elected a vice chairwoman (Mrs. G. Kruljac-Dronskowski, MPI Stuttgart), a treasurer (Mr. Friedhelm Sowa, Research Center of the University of Düsseldorf), and a secretary (Mrs. Luzia Dietsche, Computing Center of the University of Heidelberg). By May of this year, the user group had about 500 members in West Germany, Austria, Switzerland, East Germany, France, Netherlands, Luxemburg, Belgium, the USA, and Canada.

The principal aim of the society is to encourage advice and cooperation among German-speaking T_EX users. But this is not the only intention. The user group cooperates with other related national and international T_EX groups; and, as well, DANTE represents the interests of the German-language T_EX users to TUG. That is done in coordination with

other European T_EX groups and their national interests.

For its members, DANTE distributes a variety of software, including complete T_EX systems with common macro packages for PCs and Ataris, META-FONT, and drivers. In the future, T_EX for Macintoshes and Amigas will also be distributed. Anyone who has e-mail access can get some of the software via the FTP server in Stuttgart (129.69.1.12) or via LISTSERV@DHDURZ1.EARN.

Twice a year a general meeting (held together with the annual conference) takes place for all interested T_EXers. During these meetings some experienced members of DANTE hold courses for free for everyone who wants to attend. Another DANTE activity is to organise training and education for beginners. Together with this activity, DANTE supports its members with information about all that is going on in the T_EX world: because a lot of interested T_EXers have no e-mail access, DANTE distributes T_EXhax and UKT_EX via diskettes.

Last but not least, a quarterly newspaper for members is published, with articles about new macros, style files, dates, reviews, and so on.

In addition to this, there are a lot of other activities and plans to spread T_EX and the knowledge about it, for example, by publishing articles in the most widely circulated computer journals in Germany, or by bringing T_EX into high schools to students and teachers.

Institutions as well as individuals can become members: universities, publishers, computer companies, private persons, students, etc., to name but a few. Dues differ for the various categories.

For more information, please contact:

DANTE, Deutschsprachige
Anwendervereinigung T_EX e.V.
c/o Research Center
Im Neuenheimer Feld 293
D-6900 Heidelberg 1
Federal Republic of Germany
Bitnet: DANTE@DHDURZ1.EARN

- ◊ Joachim Lammarsch
Chairman of DANTE

News From and About GUTenberg

Bernard Gaulle

[Editor's note: The following report was read by Christina Thiele at the Texas A&M meeting as Bernard was unable to attend.]

As this is the first time I've been invited to talk about GUTenberg, it's perhaps necessary to explain why we decided to found GUTenberg two years ago. There were many reasons, but here are only the most important ones.

During the past few years we had organized more or less informal one-day meetings which were very much appreciated. People began asking for more and closer contacts; there were telephone calls frequently requesting help for access to a T_EX (or T_EX-related products) distributor. Though *TUGboat* contained many useful pieces of information, it seemed specially devoted to experienced T_EX users. Another important reason was of course that we felt that problems related to francophone users remained unsolved. And on other fronts, professionals were beginning to use T_EX and METAFONT. And so, we "officially" founded GUTenberg less than two years ago with many ideas but only two precise ideas: (1) to **create our own journal** (*Les Cahiers GUTenberg*), and (2) to **continue to organize meetings**.

In the meantime, we opened a list for discussions (GUT@FRULM11.BITNET) under LISTSERV, and we promoted Michael Ferguson's Multilingual T_EX, as well as the use of L^AT_EX by beginners.

Last year at our meeting in Paris, public domain PC diskettes were announced. We discovered at that moment the first French book on T_EX (*Le petit livre de T_EX* by Raymond Sérroul), which is excellent. This year we worked especially on the design of a French-speaking style. We are now ready to distribute "running" versions of MLT_EX for various UNIX systems and machines.

T_EX 3.0 brings us many useful new features (and we thank DEK for that), but it also raises again the problem of hyphenating accented words (words containing accented letters) which are intensively used in French. There are two solutions for this. One is to use virtual fonts — it's certainly THE solution for the future, but it now looks a little complicated. The second solution is to use the (transitional) bypass developed by Michael Ferguson (called Multilingual T_EX), which is simpler, ready to use, and will probably be valid for the next few years. GUTenberg is now examining these two ways for its own distributions.

One related problem to solve as quickly as possible is the place of accented letters within the second part of the ASCII code. Facing this particular problem (and also many others), GUTenberg wants very much to find (at least) a European solution. This procedure will always be adopted by GUTenberg when multilingual solutions are possible. More generally we prefer to work as closely together as possible with other T_EX user groups that share our problems, ideas, goals, etc.

GUTenberg memberships are growing rapidly (200 members the first year, 400 or more planned for the second, **right now: 350**). In spite of this good position, fewer than 20% of our members are also TUG members and only an extremely small portion of them volunteer to help. The lack of T_EX gurus seems to be specific to our group — or perhaps they are only unknown today?

Regarding courses, GUTenberg is still very young. We have only organised one-day tutorials, held before or after our annual meetings. There have been a few in-house courses, but none which were organised by GUTenberg.

GUTenberg is now at a crossroads: one way is to let things be done quietly; the other is more difficult. We have to move into higher gear, with more projects, more volunteers, more professionals, more young members, etc. This is a challenge that GUTenberg would be pleased to share (and win) with other T_EX user groups.

Thank you.

For more information, contact:

Association GUTenberg
c/o IRISA
Campus de Beaulieu
F-35-42 Rennes Cedex
France

◇ Bernard Gaulle
President of GUTenberg

NTG's Second Year

Kees van der Laan

[Editor's note: The following report was read by Barbara Beeton at the Texas A&M meeting as Kees was unable to attend.]

Organizational

NTG (Nederlandstalige T_EX Gebruikersgroep), the Dutch-language oriented T_EX users group, has existed legally since the fall of 1989. At the moment, we have 15 institutional members (representing 45 people) and some 50 individual members, making roughly a hundred members.

Members received minutes of the spring and fall meetings, with much information collected in appendices, as well as a printout of the renewed membership database. At the meetings, organizational aspects (budget, etc.) were discussed, and some presentations were also given (including Victor Eijkhout on "Unusual paragraph shapes", Nico Poppelier on "SGML and T_EX at Elsevier", Johannes Braams on "Various 'network' aspects"). Roughly 40 members were present at each meeting.

The listserver `TEX-NL@HEARN` has been heavily used for information exchange and for asking questions. The fileservers `TEX-NL@HEARN` contains codes useful for the Dutch-speaking community. The Universiteit van Utrecht (RUU, for short) has made a second (Internet) fileservers available. The NTG board can be reached via: `NTG@HEARN`.

The first year was characterized by getting started, the second by getting organized. Now we face the difficult task of organizing continuity of NTG: that is, to get a renewal process working for election of (two) board members every year, charged with preserving an active NTG. Of course the Working Groups are the basis and therefore they must be stimulated. Work of individuals also has to be encouraged. A structural activity for continuity is "teaching the teachers". It is hoped that contacts with other user groups will also encourage continuity.

Cooperation with other (European) T_EX user groups has resulted in:

1. inviting a representative of other users groups to the (open) meeting(s) at no (conference and one course) costs
2. the secretary of every user group is a member of the other user groups by default (information exchange is organized)

3. members of one group are considered members of other groups with respect to admission fees for meetings
4. sharing know-how (exchange of teachers, speakers, etc.)

With respect to TUG, we welcome the realization of the possibility of joint memberships.

Working Group Activities

Although we mainly operate in working groups — which help structure the discussion and stimulate cooperation — much work has been done on an individual basis. As an example of the latter one can think of the article "Typesetting Bridge via (plain) T_EX", which was published in *TUGboat* 11, no. 2, and was presented at GUTenberg '90 in Toulouse.

Education

A survey of (local) courses and courseware is maintained. Tools for preparing and maintaining (course) transparencies are in preparation. A set-up of courses has been made for the "NTG Days", with the idea of worldwide modules in mind. A contribution to Child's discussion of what every module should contain, has been made. Some Dutchies teach at other TUG meetings. For example, Kees van der Laan has taught the SGML class at both Stanford and the upcoming meeting at Cork; Victor Eijkhout will be giving the advanced T_EX course at Cork as well.

Guidelines for Authors

A report (*Journal Style Guidelines*, RC RUG Report 26), has appeared. It can be obtained by writing to: Rekencentrum RUG, Landlevem 1, NL-9700 AV Groningen, The Netherlands.

PCs

Some evaluations of public domain (PD) MS-DOS programs have been made. An Atari PD set is available upon request. It looks like this Working Group will become more active this year, which is very much appreciated. Cooperation with other user groups is bound to be beneficial.

Fonts

A chess font has been developed, but not published. Note: this work has possibly been superceded by the published work of Zalman Rubinstein, *TUGboat* 10, no. 3, 387-389.

NTG Days

The first "NTG Days" (June 1989) was organized by RUU. Roughly 80 people attended. This year the theme is SGML and T_EX. The organization is done by NTG and SGML Holland. The number of courses offered during these days has been increased from 2 to 7. We now have multi-day courses, as opposed to the one-day courses last year. This year's meeting will be on 31 August, with courses to be held the week before and the week after the meeting.

SGML T_EX

An introductory paper (appendix to minutes) was prepared and presented at the second SGML Holland seminar (October 1989, Amsterdam) and at GUTenberg 90 (May 1990, Toulouse).

Dutch Aspects

Various .sty files have emerged for *report*, *article* and *letter*. Moreover, articles have been prepared on A4, international L^AT_EX, and Babel. These articles have been submitted to *TUGboat*, and some have already appeared in *TUGboat* 11, no. 1 and no. 2.

Communication

Fileservers — TEX-NL@HEARN and RUU's internet server — are maintained. Problems of what to store, how (coded?) to store, and how to retrieve, are under study. We have a floppy distribution service: floppies with information (minutes and appendices) as well as programs (those available on the file servers), especially for those members who don't have access to electronic mail.

Summary

At the last meeting it was urged that NTG should pay more attention to public relation activities, e.g.: "helicopter talks" such as "What is T_EX all about (off the shelf)", and a real survey course on T_EX, L^AT_EX, METAFONT, Postscript, SGML and how these are related.

Furthermore, attention will be paid to public domain PC versions, not so much to develop them ourselves but to get the material, exercise it and organise dissemination. The idea of an NTG report series has also been proposed.

◊ Kees van der Laan
President of NTG

Report from the Nordic T_EX Users Group

Roswitha Graham and Jan Michael Rynning

The Nordic countries have a tradition of close cooperation, as well as having a common cultural background. Swedish, Danish and Norwegian are much alike, and Icelandic is by origin connected to the Norwegian languages. Finnish is entirely different but there is a large Swedish-speaking minority resident there.

The Nordic T_EX Users Group was officially formed in Copenhagen in 1988, covering the Nordic countries of Denmark, Iceland, Norway, Sweden and Finland. One informal Nordic meeting had already taken place beforehand in Stockholm in 1985, with the purposes of: discussing the future of T_EX and solving common Nordic problems as the input of national characters, how to use L^AT_EX when using national characters, correct hyphenation and proper placement of accents. The result was some more national versions of T_EX82 under VAX-VMS. T_EX is still frequently used on VAX-VMS.

When T_EX was ported to personal computers with output on laser printers, membership in the User Group increased. PostScript devices became the de facto standard, followed by HP.

T_EX is mainly used by staff and students in academic and research environments, and is well supported at all large universities. We can also report that there are also a few companies and private users of T_EX.

Networks, Distribution and Support

All universities in the Nordic countries are connected to the NORDUNET (Nordic University Network), centrally paid for and free for academic use, with communication over 64kbit/s leased lines, using TCP/IP and DECNET protocols. The TCP/IP network is part of the Internet. Stockholm/KTH is the central node with gateways to EARN/Bitnet, SPAN/HEPNET and UUCP. These networks give easy access to archives all over the world.

ftp.kth.se (120.237.72.201) is an FTP archive with a current copy of the labrea.stanford.edu archive plus other some other stuff from different archives and some local stuff. Other local archives exist at various departments and universities.

A mailing list (NordicTeX@kth.se) has recently taken over communication of news, problems and questions from different sites. T_EX support is decentralized to local systems groups at the different sites. There is a central register kept of T_EX experts or contacts at different sites. Local courses, hand-

outs and instructions are offered by the various sites and made available through the schools. \TeX for personal computers has mostly been distributed by dealers (PC \TeX , $\mu\TeX$, *Textures*), but we are also looking into what is available in the public domain. Special adaptations have been made at KTH/NADA to use *Textures* with direct entry of national characters.

Trends and News

An increasing interest in \TeX has been reported from math departments, while other desktop publishing software is taking over at other places. The number of members of the Nordic \TeX Users Group has been reported stable during the last year.

One trend seems to be that \TeX users are moving to UNIX workstations (SUN, DEC, HP, ...) from mainframes and micro computers.

Increases in book and journal production with \TeX have also been reported and typesetting services are now offered by the University of Oslo (Linotronic, Risø (IBX) and UNI·C/Copenhagen (Linotronic) in Denmark.

Most sites have upgraded to \TeX 3.0, and are now waiting for 8-bit fonts so they can make full use of it.

Wishes for 1990

The strongest wish is for an 8-bit font standard and Greek and math symbols fonts to match Times Roman. Best wishes go for the \LaTeX project. We wish to support by all means a world-wide \TeX .

◊ Roswitha Graham
President, Nordic \TeX Users
Group

◊ Jan Michael Rynning

\TeX in the UK

Malcolm Clark

It is readily apparent that only a portion of \TeX activity becomes "institutionalised". A great deal of \TeX ing goes on at an everyday, unexceptional, background level. Thus the description here is likely to be only a partial description of what has happened in the UK's portion of the \TeX world in the last year. If anyone feels aggrieved because they have been left out, I apologise. It was inadvertent.

UK \TeX Users Group

The group was founded in 1989 (see [2]) to be a focus for \TeX and \TeX -related activities in the UK. The term " \TeX " is used by the Group as a shorthand not only for the \TeX program itself, but also for the many other related pieces of software, such as \LaTeX , $\AMS\text{-}\TeX$, METAFONT, the Computer Modern typeface, WEB, output device drivers, ...

Among the aims of the Group was a desire to encourage the work at Aston on the \TeX archive and the UK \TeX bulletin, particularly in their efforts to distribute to people not on the academic networks. Equally encouraging is its wish to cooperate with TUG and with the other European groups such as GUTenberg, DANTE, NTG and the Nordic Group. Together with these other groups, the UK \TeX Users Group seeks to ensure adequate recognition of European needs at TUG. It also seeks to encourage the development of other European groups in Poland, Hungary and Czechoslovakia.

The Group has already organised several one-day meetings on a variety of topics. These have included contributions from renowned speakers, as well as from the UK \TeX community at large (note that these are not necessarily mutually exclusive groups). There were two meetings of the group in the period between the Stanford TUG Conference and this Conference.

The first of these, at Aston University (Birmingham), was entitled "Fonts: how they are created; how they can be used". Ten speakers introduced topics ranging from the creation of inscriptions in stone, right through to the esoteria of coercing METAFONT to produce PostScript. In her review of this meeting, Carol Hewlett [3] noted that ' \TeX didn't get mentioned by the speakers until after tea'. The meeting organizers have tended to take a very eclectic view of the topics which will interest \TeX users, perhaps feeling that an introduction to the wider world of publishing and typesetting would be no bad thing.

The other meeting was on “Bibliographies and Indexing”, and was held at the London School of Economics. This again followed the “policy” of inviting speakers who were specialists in their fields, but not necessarily T_EX- or L^AT_EX-wise, and attempting to balance them with the nitty gritty of more technical material (e.g. [4]).

Immediately after the TUG Conference, the Group held its first workshop, on “Fonts and font families”. This was held at the University of Windsor. As a workshop this was limited to a smaller number of participants, and approached its topics much more intensively. This time it was completely in the hands of the wizards (Chris Rowley and Dominik Wujastyk).

In order to expand the range of meetings, the Group proposes to continue the one-day general meetings and the one-day in-depth workshops, but to augment these with visits to, for example, publishers, book and journal producers, typesetting hardware manufacturers, typographers, and basically anyone who can help us increase our knowledge of all aspects of electronic publishing and document preparation. Next year will see a significant departure in meeting style with the introduction of a two-day conference on “Book production”.

The Group has also been active in creating specialist “Working Groups” with interests such as hyphenation, fonts and METAFONT, implementation issues, BIB_TE_X, the other L^AT_EX tools, PostScript, and standards. It is currently examining the production of a newsletter for the membership, but as an interim, all members of the group receive the *T_EXline* newsletter. Negotiations have taken place with publishers to enable the Group to supply T_EX-relevant books to the membership, at a discount.

The Aston Archive

It is difficult to separate the work of the Archive (see [1]) from other T_EX activities in the UK. The same people crop up actively in several different roles. The Archive seems to be maintaining its pre-eminent position as the largest T_EX-archive. Nevertheless it is still plagued by the vagaries of character translation at the Rutherford Gateway. At least there has been grudging acceptance by Rutherford that the problem is directly related to their Gateway. A small advance, but an advance. The last year has seen some reorganisation of the archive’s structure, but the sheer volume of material can make finding the right files rather difficult. Since the JANET network in the UK does not permit the “anonymous FTP” which the rest of the world seems to enjoy, this can be a stumbling block to the beginning archive

browser. One welcome innovation has been the introduction of a mail-server type service, which has an interface like lots of other mail servers. Thus file transfers can be accomplished over e-mail as well as NIFTP. Recently the requests to the Archive have been monitored to see what is going where. This confirmed that a great deal of material was being shipped out, but to everyone’s surprise (well, mine), a large proportion of the information was being shipped to Europe. So, despite the apparent Rutherford problems, the Archive is beginning to accomplish its destiny.

Of course, the success of the Archive could not be achieved without the devotion of the Archivists, Peter Abbott (himself an Archivist), and the many people around the world who submit material to the Archive and help to keep it at the forefront of T_EX software.

The UKT_EX digest is also an Aston activity, although in theory it has little to do with the Archive. Strangely enough, the Archivists seem to answer many of the queries. The digest continues to appear with welcome regularity (despite a few hiccoughs from time to time, common to all electronic distributions). The size of the digest seems to have stabilized, and the range of questions varies between the straightforward “does T_EX run on the PC yet?” to the esoteria of METAFONT and subtle L^AT_EX macros.

References

- [1] Peter Abbott. The UKT_EX archive at the University of Aston. *TUGboat*, 10(4):675–680, 1989.
- [2] Malcolm Clark. Olde worlde T_EX. *TUGboat*, 10(4):667–673, 1989.
- [3] Carol Hewlett. ukT_EXug meets again. *T_EXline*, 10:18–19, 1990.
- [4] David Rhead. Towards BIB_TE_X style-files that implement principal standards. *T_EXline*, 10:2–8, 1990.

◊ Malcolm Clark
Chairman of the UK T_EX Users
Group; Aston Archivist

TeX in Europe

Malcolm Clark

Great things have happened in Europe in the last year. It would be good to think that TeX might have played some small but significant role in the liberalisation of Europe. TeX's ability to put the power of the printing press back into the hands of the people gives it some of the hallmarks of the tools needed for democracy. But perhaps we hope for too much.

I was lucky enough to visit Poland in the autumn of 1989 at the invitation of Janusz Bień, and to be entertained both by him and by other TeX colleagues. There can be little doubt that there is a surprising amount of TeX activity in the country. This was reinforced by Marek Ryćko's paper at TeX89 in Karlsruhe, where he won the best paper award, and more recently by Bień's paper [2] in *TUGboat*. Although "traditional" computing power ("mainframes") is present, its use has tended to be rather controlled. The availability of personal machines however places comparable computing power into many hands. A more powerful limiting factor is output devices and their consumables. But anything written now is likely to be out of date very soon.

Elsewhere, an officially recognised TeX group has been established in Czechoslovakia (Československé sdružení uživatelů TeXu), as the result of the merger of two informal groups. In Hungary, another group — "HunTeX" — has come into being. This group is fortunate to have e-mail connection and regularly receives TeXhax. It is hoped that Hungary can be supplied electronically with much relevant public domain TeXware.

Recently news broke that Poland is also to join the electronic mail domain. Not only will this simplify communication, but again it will allow the transfer of TeXware. The re-unification of Germany will presumably lead to the integration of electronic mail within that country. Rumour has it that shortly the Soviet Union (or as much of it as is left) will also be connected by e-mail. (The spooks in Washington will have many happy hours decoding hexed TeX-files!)

As the accompanying reports from DANTE, GUTenberg, NTG and the Nordic group have made plain, there is a great deal of European activity. The meetings of DANTE and GUTenberg in particular are well-attended and encompass a wide range of TeX subject matter and expertise. The GUTenberg at Toulouse provided a most useful forum for represen-

tatives of the European groups to discuss common problems, and to meet the TUG President, Nelson Beebe. This was a particularly constructive meeting.

In addition to the conferences and meetings organised by the "local" groups, EuroTeX89 was held at Karlsruhe in September. This was a well-attended conference [6] which provided another useful addition to the accumulation of TeX knowledge and expertise. Apparently painlessly run, and with some fine social events — the organisers, Anne Brüggeman-Klein and Rainer Rupprecht deserve much praise. I keenly believe that the European TeX conferences are essential for the long-term viability of TeX, TUG and TeX-in-Europe. Without these meetings, and without participation from the whole European catchment we will all be a great deal poorer.

Of course, there are other activities which are relevant to us: the "Raster Imaging and Digital Typography" Conference in Lausanne [1] covered subjects close to the hearts of many of the TeX-literate. One outcome of this was the DIDOT project, a project designed to establish training in numerical typography. If this becomes established it could help to increase cooperation between those involved with traditional and digital typography.

Several more books have appeared in Europe. Norbert Schwarz' *Einführung in TEX* was translated into *Introduction to TEX* [8] by himself and his colleague Jost Krieger. There has also been a *LATEX Cookbook* by Theo de Klerk [7]. And of course, eventually the *TEX88 Conference Proceedings* have appeared ([4]). Not before time! The *Cahiers GUTenberg* continue their high standard (even including colour printing now), and have this year been joined with DANTE's *Die TEXnische Komödie*. The last year also saw another edition of *TeXline* — number 10.

Lest complacency set in as we review our past achievements, let's consider what has not happened. There seem to be no organised user groups in either Italy or Spain, despite sporadic activity in the past. It may be that the formation of user groups is not the best way forward for these countries. With a few exceptions, TeX makes slow progress against the yuppie tide of WYSIWYG. We still have not managed to get our act together to explain why markup is a *good* thing. And this when PostScript, the write-only typographic markup system, continues its dominance of the page description language arena. My WYSIWYG chums still tell me that PostScript is wonderful, but that TeX is too difficult! Equally, the faceless SGML, rescued from the dol-

drums by the US Department of Defense and the CALS initiative, is hailed as a great leap forward. A markup *meta*-language, it can describe the structure of a document, but requires some friendly help to let that structure be realised on a sheet of paper (or a screen). And where does that friendly help often come from? — $\text{T}_{\text{E}}\text{X}$ or $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$. But those who embrace SGML shy away from $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$. What is the difference? — simple: standards. The way of the future is through the standards world ([3, 5]). We join in or become a backwater. C'mon in, the water's lovely!

References

- [1] Jacques André and Roger D. Hersch. *Raster Imaging and Digital Typography*. Cambridge University Press, 1989.
- [2] Janusz Bien. On standards for computer modern font extensions. *TUGboat*, 11(2):175–182, 1990.
- [3] Malcolm Clark. Standards. *TEXline*, 9:1, 1989.
- [4] Malcolm Clark. *TEX: Applications, Methods, Uses*. Ellis Horwood Publishers, 1990.
- [5] Malcolm Clark. Tripping over our own feet. *TEXline*, 10:1, 1990.
- [6] Malcolm Clark. TUG10 and $\text{T}_{\text{E}}\text{X}90$: edited highlights. *TEXline*, 10:13–15, 1990.
- [7] Theo de Klerk. *L^AT_EX Cookbook*. Addison-Wesley, to appear.
- [8] Norbert Schwarz and Jost Krieger. *Introduction to TEX*. Addison-Wesley, 1990.

◊ Malcolm Clark
European Coordinator

In addition to final testing of $\text{T}_{\text{E}}\text{X}$ and $\text{M}_{\text{E}}\text{T}_{\text{A}}\text{-FONT}$, some of the auxiliary sources, particularly $\text{M}_{\text{E}}\text{T}_{\text{A}}\text{FONTware}$, need to be brought up to date. In the past, I have had difficulties getting the latest versions of some software, being at a site without FTP access. Dean Guenther has kindly sent me updates by mail from time to time, and Barbara Beeton has been mailing all the site coordinators *diff* files for updates to the main programs, which I've used to piece together the current test versions. Nevertheless, I've always felt hampered by the lack of direct access to the Stanford sources. Fortunately, this situation should improve soon, as I am assured that the University of Manitoba will in the next month or so be connected to Internet with FTP access. Then it should be much easier to keep all the sources up-to-date.

I have also experimented briefly with a large-memory version of $\text{T}_{\text{E}}\text{X}$, based on values in the UNIX distribution (`mem_max = 262140`). There appears to be a slight tradeoff in speed (about 9% slower), but the “big” $\text{T}_{\text{E}}\text{X}$ seems to run with no problems (it passes the trip test).

◊ Craig Platt
Dept. of Math & Astronomy
University of Manitoba
Winnipeg, Manitoba
R3T 2N2 Canada
Bitnet: `platt@ccm.umanitoba.ca`

Site Reports

MVS Site Report

Craig Platt

I am happy to report that MVS $\text{T}_{\text{E}}\text{X}$ 3.0 and $\text{M}_{\text{E}}\text{T}_{\text{A}}\text{-FONT}$ 2.0 have passed the `trip/trap` tests. Further testing remains, but I hope they will soon be ready for distribution. The current distribution, available from Maria Code, remains at $\text{T}_{\text{E}}\text{X}$ 2.9 and $\text{M}_{\text{E}}\text{T}_{\text{A}}\text{-FONT}$ 1.3, so if you want to order the new version, be sure to specify “ $\text{T}_{\text{E}}\text{X}$ 3.0” on your order.

VMS Site Report

David Kellerman

First, let me mention some news from the TUG90 conference. A quick informal survey during the site coordinators' reports showed that about two-thirds of those present used $\text{T}_{\text{E}}\text{X}$ on a VMS system. A large majority were $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ users. Most were not sure where they got $\text{T}_{\text{E}}\text{X}$. Maria Code, DECUS, Internet FTP, and commercial distributions each got only a few responses. $\text{T}_{\text{E}}\text{X}$ travels in mysterious ways! PostScript printers are by now the most common output device, with LN03s sliding into a distant second place. A handful of users have VAXstations.

The software developments are too numerous to mention in detail, but let me hit some of the high points. The DECUS distribution, assembled by Ted Nieland, remains the most accessible and broad-

ranging source of public-domain VMS \TeX ware in the United States—it's available on a variety of media from the DECUS library (219 Boston Post Road, BP02; Marlboro, MA 01752-4605; 508-480-3635), and from your local librarian, if there is a DECUS Local User Group in your area. In Europe, the archive at Aston University, Birmingham, provides the largest and best-organized collection of VMS \TeX ware, and they are currently testing an encoding scheme that should improve their ability to distribute files across gateways without corruption (Peter Abbott; Computing Service; Aston University; Aston Triangle; Birmingham B4 7ET). For those with Internet access, Don Hosek has been collecting and organizing a large VMS archive on the YMIR node at Harvey Mudd College, Claremont, that includes his latest VMS adaptations of \TeX ware and \LaTeX ware (dhosek@ymir.claremont.edu).

All the sources listed above should include versions of \TeX 3.0 by the time you read this, from work done by Brian Hamilton Kelly and Don Hosek. Also of note, good DECwindows previewers are starting to appear. XDVI appears to currently be the best public-domain implementation (DECUS distribution, YMIR archive), and both ArborText and Northlake Software now offer commercial products.

Finally, I want to discuss a more general issue: with \TeX development work now literally spread around the world, most information related to \TeX is exchanged electronically across computer networks via mail, newsgroups, and file transfers. If, like many VMS sites, you have no decent network connections, your sources of information are limited, and often out of date.

Northlake Software was faced with this problem about a year ago. Briefly, we solved it by getting a copy of the DECUS UUCP software (another public-domain DECUS library offering), and setting up a telephone connection to the UUCP network. The DECUS UUCP documentation is exceptional for public-domain software; installation and net management require care, but not deep sophistication. We use the UUNET node for our access point. UUNET Communication Services is a non-profit corporation that provides mail transfer and an Internet gateway (3110 Fairview Park Drive, Suite 570; Falls Church, VA 22042-4239; 703-876-5050). For simple mail and newsgroup access, you can expect to pay about \$50–60 per month for their services and telephone charges. The lack of FTP file transfers is the one real limitation to this solution, but the price and level of complexity are about right for us.

There are variations to this approach you might want to consider. We chose UUNET for its reliability and good Internet connections; if you have a friend at a local university with a UNIX system, you may be able to wrangle a connection for free. Low-speed modems are adequate for light use; we added a Telebit T2500 when our traffic started increasing. If you get started, yours probably will, too — welcome to the net.

◇ David Kellerman
Northlake Software
812 SW Washington Street,
Suite 1100
Portland, Oregon 97205 USA
Usenet: nls@davek

Data General Site Report

Bart Childs

\TeX 3.0 and METAFONT 2.0 have been installed without problem. The wisdom of Don's design is shown by the ease of installation. Few modules required change. Most of them were due to the strictness with which VS/Pascal interpreted Pascal's rules. The file type `text` must be used with `readln` and `writeln`. It does not allow the use of eight-bit ASCII with `text` files.

The current version includes drivers for PostScript, LaserJet, QMS, Imagen, and Date General printers. We are working on installing Tom Rorkicki's PostScript driver.

We have created a new driver for the vanilla Canon engine. It is written in C-WEB and is currently being tested. It seems to be quite good. As soon as we make a change file for the HP-LaserJet, we intend to distribute it too.

The European distribution of our software is being assisted by Dr. Wolfgang Slaby at Eichstätt. This help is greatly appreciated.

◇ Bart Childs
Dept. of Computer Science
Texas A & M University
College Station, Texas 77843-3112
Internet: bart@cssun.tamu.edu

Prime 50 Series Site Report

John M. Crawford

Prime sites running the PRIMOS operating system can contact my office to obtain a \TeX tape. The \TeX tape we send out generally reflects the latest software revisions available from Stanford and contributing users. It's a changing beast! Currently we have available \TeX version 3.0 and \METAFONT version 2.0. With this new version of \TeX we've also added some new system features such as support for input file search lists. The \TeX program comes in two sizes, one of which has greatly expanded storage capabilities.

The \TeX tape, as it currently ships, contains a substantial collection of work. All the standard \TeX -related software, as available from Stanford University, can be referenced in source form. \L\A\T\E\X and \A\A\MS-\TeX are easy to install. The programs in support of \WEB , \TeX and \METAFONT (commonly called \TeX ware and \METAFONT ware) are available to execute. We ship the standard fonts (in \tfm and \pk form) of the Computer Modern family, and include, naturally, those fonts that support \L\A\T\E\X and \A\A\MS-\TeX .

Several device drivers are running under PRIMOS, and many friendly Prime sites (primarily academic institutions) have returned their work to us for inclusion on the distribution tape. A \BIB\TeX port to PRIMOS and several site documents have also been added to the tape by contributing users. We've tossed in some other goodies and have ended up with a \TeX distribution tape for PRIMOS that requires "extra length" tape media. Many sites have reported the installation goes well. It's always nice to know we have happy \TeX users in the world of PRIMOS.

◊ John M. Crawford
 Computing Services Center
 College of Business
 Ohio State University
 Columbus, Ohio 43210
 Internet:
`craw4d@prime.cob.ohio-state.edu`

UNIX \TeX Site Report

Pierre A. Mackay

The full suite of \TeX and \METAFONT programs has been updated to match the most recent sources on `labrea.stanford.edu`. We are now delivering \TeX version 3.0, \METAFONT version 2.0 and the versions of all the support software have been changed to match. The choice of a proper official version of \L\A\T\E\X is a bit problematic, but we have chosen to adopt Dominik Wujastyk's upgrade of `lplain.tex` in order to get the full advantage of \TeX 3.0. In addition, `splain.tex` has been brought into conformity with `lplain.tex`. It had drifted pretty badly over the past three or four years. I am deeply intrigued by the notion of multilingual slides all set in magnifications of `cmssq8`, with perfect hyphenation.

The new \WEB2C owes a great deal to Karl Berry, who not only did the major part of the work for the upgrade to version 3.0, but also provided the much needed change files for `gftodvi` and `mft`. At the time of writing [1 July 1990], the current version of \WEB2C is 5.0d, and it has achieved a very satisfactory success rate on the ever widening range of UNIX systems available. Tor Lillqvist (`tml@tik.vtt.fi`) has offered an alternative translation system based on `PtC`, which is more absolutely tied to POSIX (good) and ANSI C (not nearly so good) standards. We have not had the time to evaluate it properly yet, but we are very interested in moving towards POSIX-conformance throughout the distribution, and we shall be giving this our close attention.

For his own Pascal-based change files, Don Knuth introduced "omphaloscopy" into \TeX , a process by which any instance of `initex` or `virtex` looks at its own navel (`$0`) to find out what name it has been called by, and to select an appropriate format file for that name if one exists. This practice has been imported into the \WEB2C change files for both \TeX and \METAFONT , and effectively supercedes the use of `undump`, which is no longer supported. There are too many versions of `undump` needed across the broad spectrum of UNIX systems, `undumped` versions of \BIG\TeX take too much room both in disk storage and in RAM and swap partitions, and "omphaloscopy" makes private formats for specific projects much easier.

We have yielded to many requests and consolidated all standard \tfm files into a single directory called `./TeX3.0/TeXfonts`. Font raster files are still distributed by "families" for what seem to us to be fairly compelling reasons. Some minor alterations were noticed in the `cm*.mf` sources at Stanford, so

all the fonts have been recompiled for 300, 300w, 240 and 120 dpi devices. Note that the peculiar and regrettable 118 dpi resolution is no more. Even on the old BitGraph, for which 118 dpi was originally adopted, it would be possible to adjust the software to use 120 dpi fonts. As usual, only the fonts essential for testing your basic compilation are supplied. I have worked out a very thorough UNIX manual page (in troff source) which seems to reduce the irrational fear of METAFONT which we have all too often encountered. The potential font library is far too large to form part of a distribution, and users of T_EX owe it to themselves to discover the possibilities of METAFONT. The Makefiles associated with each family of fonts have been redesigned to allow the use of the simple command

```
make magsteps DPI=nnn MAGFACTOR=N
```

so that you don't have to go through the tedium of calculating DPI-related suffixes for a full set of fonts at a `magstep` magnification such as might be needed for L^AT_EX 11pt style.

The collection of old METAFONT79 fonts in `./amsfonts` is about to vanish from the distribution. In its place will be the METAFONT sources officially released by the American Mathematical Society. Sites which may wish to recreate the exact appearance of documents which used the M-series fonts (`msxm`, `msym`, `mcyr`, `mcyb`), should archive these fonts, together with the `tfm` files and the `amstex.tex` input files associated with them.

Proper response to the flood of interesting contributions to UNIX T_EX is way behind schedule, owing in part to the fact that the passage to T_EX 3.0 had to take priority. We apologize to all who have been so generous with their offerings, and can promise that we have not forgotten you.

This year even more than last, I have depended on the initiative and imagination of Elizabeth Tachikawa, who now does a great deal of the technical organization of the distribution as well as all the administration. During my year's virtual absence from the university, she has taken on an ever larger part of the responsibility for the entire operation, and whatever improvements there may have been in the general organization and documentation of UNIX T_EX are almost entirely hers.

- ◊ Pierre A. Mackay
Northwest Computer Support Group
University of Washington
Mail Stop DW-10
Seattle, Washington 98195
Internet: `MacKay@June.CS.Washington.Edu`

VM/CMS Site Report

Joachim Lammarsch

After I got the baton from Dean Guenther (thanks very much, Dean, for work in the last years), I have had to learn the job of a site-coordinator. Five things are very important:

1. Answering questions via e-mail. That is not very difficult. In most cases the answer may be:
The VM/CMS distribution tape is available from:
Maria Code
Data Processing Services
1371 Sydney Drive
Sunnyvale
CA 94087
2. The preparation of the new VM/CMS distribution tape. Peter Breitenlohner/München has done a lot of work on this. He has written the new change files for T_EX 3.0 and one for T_EX 3.0 with 128K memory words.

He has also made the change files for METAFONT 2.0, VFTOVP, VPTOVF, the new TANGLE, WEAVE and the rest of T_EXware and METAFONTware.

From Dean Guenther I have received the T_EXT1 macro package and the IPA Fonts, which were developed at Washington State University. Dean has also sent me a driver for the Apple LaserWriter which was ported from Nelson Beebe's driver family to VM/CMS by Shashi Sathaye.

From Ferdinand Hommes I got a collection of drivers for IBM laser printers (e.g., the IBM 3820) and IBM graphic display stations (e.g., the IBM 3179, 3192, ...). These drivers are only available as `text` files and you need an IBM PASCAL VS compiler and GDDM to install them.

3. The next point is not very pretty: under another typesetting system, someone wrote a text file containing a virus. I ran a test to see if it works under VM/CMS with T_EX 2.991 in the same way. It did! The method is to use the command `\write15` to send VM/CMS commands to the operating system. To avoid this, the new T_EX 3.0 is changed so that there is a new variable which controls the use of `\write15`. The variable has to be used to allow `\write15` in the first 5 lines of an input source and before the first input statement.

4. Distribution via e-mail. I will put the VM/CMS change files at the different servers. The change files will be available using FTP from labrea (36.8.0.47) and from LISTSERV@WSUVM1 (Washington State University). In Europe you can order the files from LISTSERV@DHDURZ1 (Uni. Heidelberg). The new DANTE FTP server at the University of Stuttgart/Germany (129.69.1.12) will also contain a copy of the complete distribution tape.
5. Last but not least I will open the distribution list TEX-IBM hosted at LISTSERV@DHDURZ1 for all T_EX users at IBM mainframes. I hope this will help make information better and above all faster to access.

◊ Joachim Lammarsch
 Research Center
 Universität Heidelberg
 Im Neuenheimer Feld 293
 D-6900 Heidelberg
 Federal Republic of Germany
 Bitnet: X92@DHDURZ1

Q & A

Report from the Question and Answer Session

Barbara Beeton

There was a new twist to the question and answer session at the T_EXas A&M TUG meeting. Tom Reid, a T_EXnician at A&M, set up a mail drop for advance questions, and arranged for receipts to be distributed to a small group of volunteer screeners. The address was publicized in the "usual" places — T_EXhax, UKT_EX, T_EX-Euro, and a few others. Questions didn't exactly come streaming in, but enough arrived to give us a good start.

The questions received at the mail drop were augmented by a few more gleaned from T_EXhax, UKT_EX and GUT. Answers were drafted by Michael Doob (one of the volunteers), Ron Whitney, and myself, and the questions and concise versions of the answers turned into slides. This summary is an edited composite of pre-meeting answers and discussion from the session itself.

I think this format was moderately successful, and hope that a similar arrangement can be made for next year; a repeat should be even more successful, as people will be more familiar with the procedure, and will realize when seeing this summary that questions posed to that forum haven't simply dropped into a black hole.

Thanks particularly to Tom and Michael, and to the folks who sent in the questions.

Herewith the questions, their sources, and answers.

Q 1. Christina Thiele

In making up a sort of letterhead, the idea is to have some text to the left, to the right, and in the middle, all of different font styles and sizes. Here is what I tried.

```
\line{left\hfil middle\hfil right}
\line{more\hfil and more\hfil even more}
Even where I fudged, things don't centre properly.
left           middle           right
more           and more         even more
```

How can this be fixed?

Answer. You have to remember that the stuff in the middle should be centered with respect to the outside boundaries, not just given equal space on either side to separate it from whatever is on the left and right.

```
left           middle           right
more           and more         even more
```

This can't be done using just one box (`\line`), but there are two easy ways to get the effect you want with several nested boxes.

The first approach uses two full-width text boxes, superimposed on one another with an `\rlap`.

```
\line{\rlap{\line{\hfil middle\hfil}}%
left \hfil right}
\line{\rlap{\line{\hfil and more\hfil}}%
more \hfil even more}
```

You get the same result using one full-width box and two lapped boxes at the ends.

```
\line{\rlap{left}\hfil middle\hfil
\llap{right}}
\line{\rlap{more}\hfil and more\hfil
\llap{even more}}
```

Note that the "main" full-width box is always outside the others. It is possible to have the boxes side-by-side rather than nested, but to do so, one must remember that in vertical mode, an `\hbox` will not switch into horizontal mode, so adjacent `\hboxes`

would be set one above another rather than in the same line, unless an explicit mode-changing command (`\noindent`, `\leavevmode`) were also used. It's easier to avoid those complications by nesting.

Q 2. Reinhard Wonneberger

Who is going to rewrite `MakeIndex` in `WEB`?

Answer. `MakeIndex` was created in C, and has not been rewritten in any other language. To implement it in `WEB` would indeed make it more widely usable than it is now. We have not found a volunteer yet, but are looking for one.

Q 3. Jim Diamond

What is the purpose of the `\kern-\dimen@` in the `\ifraggedbottom` section of `\pagecontents` on page 364 of the *The T_EXbook*?

Answer. Let us start with the cited definition of `\pagecontents`:

```
\def\pagecontents{%
  \ifvoid\topins\else\unvbox\topins\fi
  \dimen@=\dp255 \unvbox255
  \ifvoid\footins
  \else % footnote info present
    \vskip\skip\footins \footnoterule
    \unvbox\footins
  \fi
  \ifraggedbottom \kern-\dimen@ \vfil
\fi}
```

`\dimen@` is set to the depth of `\box255` before it is unboxed. This will be positive (if the last box within it—presumably the last line of text—has positive depth) or zero.

First, note that `\pagecontents` is called only in `\pagebody`:

```
\def\pagebody{\vbox to\vszie
  {\boxmaxdepth\maxdepth \pagecontents}}
```

If `\raggedbottom` is false, the last baseline will be vertically justified by adding (vertical) glue within the page so that the last baseline is at the bottom of the page, that is, at `\vszie`. (If the page is sufficiently full, this additional glue may be less than the depth of the last line, `\dimen@`.) But if `\raggedbottom` is true, glue is added following the page box and the footnote to fill the space between the last baseline and the bottom of the page. The depth of the last line, `\dimen@`, is now included in the height of the material being set to `\vszie`. Without the negative kern the page could actually be bigger than `\vszie` if `\dimen@` is positive.

Q 4. Aidan Delaney (from T_EXhax)

How can a L^AT_EX `\caption` be modified so that, instead of

Figure 3.1: Structural trends in the Celtic Sea area
(after Gardiner & Sheridan, 1981)

it will produce

Figure 3.1: Structural trends in the Celtic Sea area
(after Gardiner & Sheridan, 1981)

Answer. `\@makecaption` (in the document style files) is the macro that does the work. Here is the modified definition from `report.sty`.

```
\long\def\@makecaption#1#2{
  \vskip 10pt
  \setbox\@tempboxa\hbox{#1: #2}
  \ifdim \wd\@tempboxa >\hszie
    \setbox\@tempboxa\hbox{#1: }% new
    \leavevmode % new
    \hangindent\wd\@tempboxa % new
    \copy\@tempboxa #2\par % change
  \else \hbox to\hszie
    {\hfil\box\@tempboxa\hfil}
  \fi}
```

Of course, this works only on the first paragraph, but it's usually considered bad form to have really long, multi-paragraph captions.

Q 5. Larry Denenberg

We wish to provide a `\ruledinsert` macro similar in function to `\topinsert` with the difference that the insertions are set off from one another by rules. In addition, the final such insertion is to be followed by a fairly hefty skip (say 24pt). Here is a "sample page" showing the desired output.

```
<headline>
horizontal rule
<first insert>
horizontal rule
<second insert>
horizontal rule
\vskip 24pt
<rest of page>
```

This arrangement is complicated by the fact that normal `\topinserts` may also occur in the document; when this occurs, all normal `\topinserts` come first, followed by a rule, then by the `\ruledinserts`. The document may also contain footnotes.

Answer. `\topinserts` are already a distinct class in plain T_EX, tested in the output routine (*The T_EXbook*, p. 364), as are footnotes. A new class should be established for `\ruledinsert`. Multiple

The right column will be split when the equation

$$a^n = b^n + c^n + d^n + e^n + f^n + \dots + z^n$$

appears in the left column. Splitting must occur with horizontal rules to separate a split column from the dis-material that spans the entire page.

played equation that appears within it. These rules are affected if the

split appears on the top or bottom of a page. The columns must be balanced and may include single column

Figure 1. Q 6: Two-column IEEE format

insertion classes, when properly defined, will not conflict with one another. Rules are not doubled between adjacent `\ruledinserts`; instead a structural analysis can consider the rule above each to be part of the insert, and the final rule to be a function of the output routine. Then definitions can be modeled on the plain code for `\topinsert`:

```
\newinsert\ruledins
\skip\ruledins=24pt
\def\ruledinsert{\@ins
  (horizontal rule) (vertical skip)}
\def\endruledinsert{\egroup
  \insert\ruledins{\box0
  \nobreak (vertical skip)}\endgroup }
```

Note that a `\ruledinsert` has its own terminator, here called `\endruledinsert`. The `\endinsert` provided by plain already covers so many cases that it isn't a good idea to add anything more.

The insertion is placed on the page in an extension of plain's `\pagecontents`:

```
\def\pagecontents{
  \ifvoid\topins\else\unvbox\topins\fi
  \ifvoid\ruledins
  \else\unvbox\ruledins
  \hrule width\hsize
  \vskip\skip\ruledins
  \fi
  ...}
```

Additional parameters should also be provided, as they are for `\topinsert`; see *The T_EXbook*, p. 363.

Q 6. Steven Smith

I am looking for a sophisticated T_EX or L^AT_EX double-column macro capable of the following:

- i. Double/single column capability on the same page. (L^AT_EX starts a new page every time its format is switched from `\doublecolumn` to `\singlecolumn`.)

- ii. The ability to split the column opposite a large displayed equation, as illustrated in Figure 1.

This format mimics that of many IEEE journals.

Is there an IEEE.sty L^AT_EX style file anywhere? Have any T_EX hackers attacked this problem?

Answer. A non-L^AT_EX solution to this problem was described in some detail in a paper presented at last year's annual meeting. See the proceedings: *Inserts in a multiple-column format*, by Gary Benson, Debi Erpenbeck, and Janet Holmes, *TUGboat* 10, no. 4, pp. 727-742.

We haven't found any L^AT_EX style file.

Q 7. Guy Metcalfe

It is fairly common in experimental science to prepare large tables of data. When the width of a table is greater than 1 page then the typesetter usually rotates the table 90 degrees so that it fits along the page to be read from top to bottom [*sic*] rather than left to right. I find this impossible to do in T_EX/L^AT_EX, especially if the table needs then to be split over more than 1 page.

How can I handle tables both wide and long?

Answer. An unextended implementation of T_EX can handle only horizontal setting from left-to-right. However, some output device drivers do have the ability to "paste in" segments of material prepared separately and accessed via the `\special` command. Although this usually means "graphics", a .dvi file should also be a suitable candidate, as long as the rotation is a multiple of 90°.

The characteristics of your device driver should be checked to see if it can handle rotated inserts. This suggestion will be conveyed to the device driver standards committee for possible inclusion in a future extension of the standards beyond level 0.

If one wishes to keep such a table in the same file as the rest of the paper, it would be possible to write macros that would write out the table code to

another file and leave blank pages or advance the page counter in the appropriate place.

Q 8. Mark Moline

Is it possible to access the width of a given field within an `\halign` construction. For example, can one determine the width of column 3 and use that dimension within the `\halign`?

Answer. `TEX`'s alignment structures don't provide a direct way of determining the width of a particular column or cell. An ad hoc technique for obtaining a useful value is to determine the widest entry in the column, save it in a named box before beginning the alignment, and use the width of that box.

Q 9. Chris Hand (from GUT list)

While preparing a recent `compte-rendu` GUT with `LATEX`, the line

```
\subsection{Journ\'{e}e
  \<<{europ\'{e}enne}>>
  \ du lundi 14 mai}
```

caused `TEX` to write a line of 509 characters into the `.aux` file. The macros for the guillemets were responsible for much of the length. (When these macros were replaced by others that used the

guillemets in the font `mcyr10`, the problem went away.)

`TEX` reported

```
[...]
[3] [4] [5] [6] (#tz020859.aux
Unable to read an entire line---bufsize=500
and then terminated.
```

Why does `TEX` write lines so long that it can't read them?

Is the only solution to recompile `TEX` with an enlarged `bufsize`?

Answer. The preferred technique of avoiding long lines in an `.aux` file is to expand only those control sequences whose expansions will change before the `.aux` file is read in; this includes, of course, section headings, page numbers, and the like. However, control sequences within section heading text should usually *not* be expanded.

The next generation of `LATEX` is expected to suppress this level of expansion automatically. For the present, if you are using `LATEX`, `\fragile` can be inserted before any control sequence that shouldn't be expanded in a string being written out to an `.aux` file.

`\noexpand` and `\string` function in relatively comparable ways in non-`LATEX` environments.

Participants

at the
11th Annual TUG Meeting
June 17-20, 1990
College Station, Texas

Exhibitors are indicated by *

Cynthia S. Actis

Boeing Computer Services
Seattle, Washington

Robert A. Adams

University of British Columbia
Vancouver, British Columbia, Canada

Clifford Alper

`TEX` Users Group
Providence, Rhode Island

Abass Andulem

Houston, Texas

Bernadette Archuleta

Los Alamos National Laboratory
Los Alamos, New Mexico

William W. Babcock

Northern Michigan University
Marquette, Michigan

*Michael Ballantyne

`TEX`plorators Corporation
Houston, Texas

Elizabeth M. Barnhart

TV Guide
Radnor, Pennsylvania

Stephan v. Bechtolsheim

Integrated Computer Software, Inc.
West Lafayette, Indiana

Micah Beck

Cornell University
Ithaca, New York

Nelson H. F. Beebe

University of Utah
Salt Lake City, Utah

Barbara Beeton

American Mathematical Society
Providence, Rhode Island

Nancy Blachman

Wolfram Research
Champaign, Illinois

Jennifer L. Bohac

Texas A&M University
College Station, Texas

Jerry T. Borges

Lawrence Berkeley Laboratory
Berkeley, California

David M. Bowen

Cray Research Incorporated
Carrollton, Texas

Len Boyle

SUNY at Stony Brook
Stony Brook, New York

Judi Briesmeister

Los Alamos National Laboratory
Los Alamos, New Mexico

*Terry Bryll

Northlake Software
Portland, Oregon

Mimi Burbank

Florida State University
Tallahassee, Florida

Neil A. Bursleson
Texas A&M University
College Station, Texas

Karen Butler
T_EX Users Group
Providence, Rhode Island

William Butler
T_EX Users Group
Providence, Rhode Island

Helen M. Byers
Los Alamos National Laboratory
Los Alamos, New Mexico

Pierce Cantrell
Texas A&M University
College Station, Texas

***Lance Carnes**
Personal T_EX Incorporated
Mill Valley, California

Christopher Carruthers
University of Ottawa
Ottawa, Ontario, Canada

Evelyn E. Chaney
Sandia National Laboratories
Livermore, California

Ken Chang
Texas A&M University
College Station, Texas

Sheryl D. Chapman
Cogni Seis Development
Houston, Texas

S. Bart Childs
Texas A&M University
College Station, Texas

Malcolm W. Clark
Imperial Cancer Research Fund
Laboratories
London, England

David M. Cobb
SAIC
Oak Ridge, Tennessee

Kay Coen
Los Alamos National Laboratory
Los Alamos, New Mexico

Arvin C. Conrad
Menil Foundation
Houston, Texas

John M. Crawford
Ohio State University
Columbus, Ohio

***Betsy J. Dale**
ArborText Incorporated
Ann Arbor, Michigan

Jackie A. Damrau
Superconducting Super Collider
Laboratory
Dallas, Texas

Walter Daugherty
Texas A&M University
College Station, Texas

***Beth Dehlinger**
Northlake Software
Portland, Oregon

Luzia Dietsche
Universität Heidelberg
Heidelberg
Federal Republic of Germany

Michael Doob
University of Manitoba
Winnipeg, Manitoba, Canada

Ken Dreyhaupt
Springer-Verlag
New York, New York

Lincoln K. Durst
T_EX Users Group
Providence, Rhode Island

Allen R. Dyer
Computer Law Laboratory
Ellicott City, Maryland

David J. Fenwick
Quasar Knowledge Systems
Washington, D.C.

Michael J. Ferguson
Université du Québec
Verdun, Québec, Canada

Peter Flynn
University College of Cork
Cork, Republic of Ireland

Jim Fox
University of Washington
Seattle, Washington

Harumi Fujiura
ASCII Corporation
Kawasaki, Japan

Stephen A. Fulling
Texas A&M University
College Station, Texas

Richard Furuta
University of Maryland
College Park, Maryland

Edward A. Garay
University of Illinois at Chicago
Chicago, Illinois

Mary Louise Garcia
Los Alamos National Laboratory
Los Alamos, New Mexico

Diane M. Gehan
I.M.S.L. Incorporated
Houston, Texas

Helen M. Gibson
Wellcome Institute for the History
of Medicine
London, England

Regina Girouard
American Mathematical Society
Providence, Rhode Island

Raymond Goucher
T_EX Users Group
Providence, Rhode Island

John M. Gourlay
ArborText Incorporated
Ann Arbor, Michigan

Sharon L. Gray
Los Alamos National Laboratory
Los Alamos, New Mexico

Andrew Marc Greene
Massachusetts Institute of Technology
Cambridge, Massachusetts

Gayla Groom
Blue Sky Research
Portland, Oregon

Dean R. Guenther
Washington State University
Pullman, Washington

Hisato Hamano
ASCII Corporation
Minato-ku Tokyo, Japan

Hope Hamilton
National Center for Atmospheric
Research
Boulder, Colorado

Chris Hamlin
American Institute of Physics
Woodbury, New York

Nancy K. Hannigan
MIT Lincoln Laboratory
Lexington, Massachusetts

Marvin V. Harlow
Los Alamos National Laboratory
Los Alamos, New Mexico

Niki Harris
Texas A&M University
College Station, Texas

Robert L. Harris
Micro Programs Incorporated
Syosset, New York

Doug Henderson
Blue Sky Research
Portland, Oregon

Amy Hendrickson
T_EXnology Incorporated
Brookline, Massachusetts

Mildred H. Hoak
Los Alamos National Laboratory
Los Alamos, New Mexico

John D. Hobby
AT&T Bell Laboratories
Murray Hill, New Jersey

Alan Hoenig

John Jay College (CUNY)
New York, New York

Anita Z. Hoover

University of Delaware
Newark, Delaware

Don Hosek

Harvey Mudd College
Claremont, California

Portia Hsiung

American Institute of Physics
Woodbury, New York

Calvin W. Jackson

California Institute of Technology
Pasadena, California

***Claire Kahan**

Personal T_EX Incorporated
Mill Valley, California

William S. Kaster

Hewlett-Packard Company
Palo Alto, California

***David Kellerman**

Northlake Software
Portland, Oregon

John T. Kesich

New York University
New York, New York

***Richard J. Kinch**

Kinch Computer Company
Ithaca, New York

Peter F. Klammer

University of Colorado
Denver, Colorado

Howard Kong

Mitre Corporation
Bedford, Massachusetts

David H. Kratzer

Los Alamos National Laboratory
Los Alamos, New Mexico

Ryoichi Kurasawa

ASCII Corporation
Kawasaki, Japan

Mimi Lafrenz

ETP Services Co.
Portland, Oregon

Joachim Lammarsch

Universität Heidelberg
Heidelberg
Federal Republic of Germany

Charlotte V. Laurendeau

T_EX Users Group
Providence, Rhode Island

Luby Liao

University of San Diego
San Diego, California

Nelson A. Logan

Austin, Texas

Pierre MacKay

University of Washington
Seattle, Washington

Kenneth Marshall

Rice University
Houston, Texas

Charles R. Martin

Duke University
Durham, North Carolina

John McClain

Texas A&M University
College Station, Texas

Robert W. McGaffey

Martin Marietta Energy
Systems Incorporated
Oak Ridge, Tennessee

Wendy McKay

University of Montreal
Montreal, Quebec, Canada

Lothar Meyer-Lerbs

Universität Bremen
Bremen, Federal Republic of Germany

Terry W. Miller

Sundstrand Avionics Support Services
Redmond, Washington

Lia M. Mitchell

Los Alamos National Laboratory
Los Alamos, New Mexico

Frank Mittelbach

Universität Mainz
Mainz, Federal Republic of Germany

Yoshiyuki Miyabe

Matsushita Electric Ind Company Ltd
Osaka, Japan

Mark Moline

Publication Services
Champaign, Illinois

Patricia Monohon

University of Washington
Seattle, Washington

Mary Jean Moore

University of California
Oakland, California

Mark Motl

Texas A&M University
College Station, Texas

Norman Naugle

Texas A&M University
College Station, Texas

David Ness

University of Pennsylvania
Philadelphia, Pennsylvania

Robert A. Nilsson

Texas A&M University
College Station, Texas

Toshiharu Ohno

ASCII Corporation
Minato-ku Tokyo, Japan

José Luis Olivares

Sociedad Mexicana de Física
Coyoacan, Mexico

Daniel D. Olson

ETP Services Co.
Portland, Oregon

Peter W. Owings

University of Rochester
Rochester, New York

Yeongbok Park

RIST
Pohang, Korea

Noel C. Peterson

Library of Congress
Washington, D.C.

Laurie Petrycki

Addison-Wesley Publishing Company
Reading, Massachusetts

Mark M. Pickrell

Wynne-Manley Software, Inc.
Los Alamos, New Mexico

Craig R. Platt

University of Manitoba
Winnipeg, Manitoba, Canada

Julie A. Pomroy

American Mathematical Society
Providence, Rhode Island

***Mike Pond**

Wolfram Research
Champaign, Illinois

Ekman T.W. Poon

United Bible Societies
Kowloon, Hong Kong

Robert B. Pratt

Zycor Incorporated
Austin, Texas

Jon Radel

Reston, Virginia

Thomas Reid

Texas A&M University
College Station, Texas

Norman Richert

University of Houston
Houston, Texas

Stewart Robinson

Texas A&M University
College Station, Texas

Cynthia Rodriguez
University of Illinois at Chicago
Chicago, Illinois

Tomas G. Rokicki
Radical Eye Software
Stanford, California

Jan Michael Rynning
Royal Institute of Technology
Stockholm, Sweden

David Salomon
California State University,
Northridge
Northridge, California

Arturo Sánchez y Gándara
Sociedad Mexicana de Física
Coyoacan, Mexico

Aaron Sawdey
Publication Services
Champaign, Illinois

Glenn Scholebo
Wolfram Research
Champaign, Illinois

Sean C. Simmons
Quasar Knowledge Systems
Washington, D.C.

***Barry Smith**
Blue Sky Research
Portland, Oregon

Lowell Smith
Hercules Aerospace
Salt Lake City, Utah

Michael Sofka
Publication Services
Champaign, Illinois

Friedhelm Sowa
Heinrich Heine University
Düsseldorf
Federal Republic of Germany

Susan J. Spach
Los Alamos National Laboratory
Los Alamos, New Mexico

Robert J. Sparks
Texas A&M University
College Station, Texas

***Michael D. Spivak**
TEXplorators Corporation
Houston, Texas

David K. Steiner
Rutgers University
Piscataway, New Jersey

Alan Stolleis
Texas A&M University
College Station, Texas

Carol Sullivan
United States Geological Survey
Menlo Park, California

***Leb Tannenbaum**
Blue Sky Research
Portland, Oregon

***Lin Tay**
Micro Press, Inc.
Forest Hills, New York

Christina Thiele
Carleton University
Ottawa, Ontario, Canada

David Thompson
Lawrence Livermore National
Laboratory
Livermore, California

***D. R. Tsai**
Micro Press, Inc.
Forest Hills, New York

Lisa Ungar
IBM T. J. Watson Research Center
Yorktown Heights, New York

Russel B. Vanderhoof
Cape Canaveral, Florida

Barbara J. Velarde-Maes
Los Alamos National Laboratory
Los Alamos, New Mexico

***Michael Vulis**
Micro Press, Inc.
Forest Hills, New York

Leslie C. Watson
SFA, Inc.
Ft. Washington, Maryland

Alan Wetmore
White Sands Missile Range
New Mexico

Ron Whitney
TEX Users Group
Providence, Rhode Island

Pat Wilcox
Cool Spring Banjo Works
Delaware, Ohio

Linda Williams
University of Tennessee Space
Institute
Tullahoma, Tennessee

Cheryl W. Winstead
NASA Langley Research Center
Hampton, Virginia

Janene Winter
American Mathematical Society
Providence, Rhode Island

Cheryl Woods
TEX Users Group
Providence, Rhode Island

Jerry Woods
San Diego, California

William B. Woolf
American Mathematical Society
Providence, Rhode Island

Ralph E. Youngen
American Mathematical Society
Providence, Rhode Island

Salih Yurttas
Texas A&M University
College Station, Texas

There were 167 participants at the Meeting.

Calendar

1990

T_EX90 Conference
University College
Cork, Ireland

- Sep 3-7 Intensive Beginning/Intermed. T_EX
 Sep 3-5 Intensive L^AT_EX
 Sep 3-7 Intensive METAFONT
 Sep 5-7 SGML/T_EX
 Sep 7-8 Advanced T_EX
 Sep 10-13 **TUG's 1st Conference in Europe**
 Sep 14-15 Macro Writing
 Sep 14-15 L^AT_EX Style Files
 Sep 14-15 Graphics in T_EX
-

- Sep 11 **TUGboat Volume 11,**
3rd regular issue:
 Deadline for receipt of manuscripts
 (tentative).
- Sep 18-20 EP'90
 National Institute of Standards
 and Technology, Gaithersburg,
 Maryland. For information,
 contact Richard Furuta
 (furuta@brillig.umd.edu).
- Oct 3-5 Seybold Computer Publishing
 Conference, San Jose Convention
 Center, San Jose, California.
 For information, contact Seybold
 Publications, West Coast Office
 (213-457-5850).
- Oct 10-12 9th annual meeting, "Deutsch-
 sprachige T_EX-Interessenten";
 DANTE e.V.: 3rd meeting, GWD,
 Göttingen. For information, contact
 Dr. Peter Scherber (Bitnet:
 PSCHERB@DGOGWDG1) or DANTE e.V.
 (Bitnet: DANTE@DHDURZ1)

- Dec 6-8 European Publishing Conference,
 Netherlands Congress Centre,
 The Hague, Holland.
 For information, contact Seybold
 Publications, U. K. Office
 ({44} 323 410561).
-

1991

- Jan 15 **TUGboat Volume 12,**
1st regular issue:
 Deadline for receipt of manuscripts
 (tentative).
- Feb 20-22 10th annual meeting, "Deutsch-
 sprachige T_EX-Interessenten";
 DANTE e.V.: 4th meeting,
 Technical University of Vienna.
 For information, contact
 Dr. Hubert Partl (Bitnet:
 Z3000PA@AWITUW01) or DANTE e.V.
 (Bitnet: DANTE@DHDURZ1)
- Apr 9 **TUGboat Volume 12,**
2nd regular issue:
 Deadline for receipt of manuscripts
 (tentative).
- May 28-30 Congres GUTenberg'91
 Paris, France. For information,
 contact Olivier Nicole (Bitnet:
 Nicole@FRINRA72.Bitnet, or
 +33 1 34 65 22 32)
- Sep 10 **TUGboat Volume 12,**
3rd regular issue:
 Deadline for receipt of manuscripts
 (tentative).

For additional information on the events listed
 above, contact the TUG office (401-751-7760) unless
 otherwise noted.

Institutional Members

- The Aerospace Corporation,
El Segundo, California
- Air Force Institute of Technology,
Wright-Patterson AFB, Ohio
- American Mathematical Society,
Providence, Rhode Island
- ArborText, Inc.,
Ann Arbor, Michigan
- ASCII Corporation,
Tokyo, Japan
- Aston University,
Birmingham, England
- Belgrade University,
Faculty of Mathematics,
Belgrade, Yugoslavia
- Brookhaven National Laboratory,
Upton, New York
- Brown University,
Providence, Rhode Island
- California Institute of Technology,
Pasadena, California
- Calvin College,
Grand Rapids, Michigan
- Carleton University,
Ottawa, Ontario, Canada
- Carnegie Mellon University,
Pittsburgh, Pennsylvania
- Centre Inter-Régional de
Calcul Électronique, CNRS,
Orsay, France
- College of William & Mary,
Department of Computer Science,
Williamsburg, Virginia
- DECUS, L&T Special Interest
Group, *Marlboro, Massachusetts*
- Department of National Defence,
Ottawa, Ontario, Canada
- Digital Equipment Corporation,
Nashua, New Hampshire
- Edinboro University
of Pennsylvania,
Edinboro, Pennsylvania
- Emerson Electric Company,
St. Louis, Missouri
- Environmental Research
Institute of Michigan,
Ann Arbor, Michigan
- European Southern Observatory,
*Garching bei München,
Federal Republic of Germany*
- Fermi National Accelerator
Laboratory, *Batavia, Illinois*
- Fordham University,
Bronx, New York
- General Motors
Research Laboratories,
Warren, Michigan
- Geophysical Company
of Norway A/S,
Stavanger, Norway
- GKSS, Forschungszentrum
Geesthacht GmbH,
*Geesthacht, Federal Republic of
Germany*
- Grinnell College,
Computer Services,
Grinnell, Iowa
- Harvard University,
Computer Services,
Cambridge, Massachusetts
- Hatfield Polytechnic,
Computer Centre,
Herts, England
- Hewlett-Packard Co.,
Boise, Idaho
- Hughes Aircraft Company,
Space Communications Division,
Los Angeles, California
- IBM Corporation,
Scientific Center,
Palo Alto, California
- Institute for Advanced Study,
Princeton, New Jersey
- Institute for Defense Analyses,
Communications Research
Division, *Princeton, New Jersey*
- Intevop S. A., *Caracas, Venezuela*
- Iowa State University,
Ames, Iowa
- The Library of Congress,
Washington D.C.
- Los Alamos National Laboratory,
University of California,
Los Alamos, New Mexico
- Louisiana State University,
Baton Rouge, Louisiana
- Marquette University,
Department of Mathematics,
Statistics and Computer Science,
Milwaukee, Wisconsin
- Massachusetts Institute
of Technology,
Artificial Intelligence Laboratory,
Cambridge, Massachusetts
- Mathematical Reviews,
American Mathematical Society,
Ann Arbor, Michigan
- Max Planck Institut
für Mathematik,
Bonn, Federal Republic of Germany
- Max Planck Institute Stuttgart,
*Stuttgart, Federal Republic of
Germany*
- McGill University,
Montréal, Québec, Canada
- Michigan State University,
Mathematics Department,
East Lansing, Michigan
- National Cancer Institute,
Frederick, Maryland
- National Research Council
Canada, Computation Centre,
Ottawa, Ontario, Canada
- Naval Postgraduate School,
Monterey, California
- New Jersey Institute of
Technology, *Newark, New Jersey*
- New York University,
Academic Computing Facility,
New York, New York
- Nippon Telegraph &
Telephone Corporation,
Software Laboratories,
Tokyo, Japan
- Northeastern University,
Academic Computing Services,
Boston, Massachusetts
- Norwegian Pulp & Paper
Research Institute,
Oslo, Norway

- Pennsylvania State University,
Computation Center,
University Park, Pennsylvania
- Personal T_EX, Incorporated,
Mill Valley, California
- Princeton University,
Princeton, New Jersey
- Promis Systems Corporation,
Toronto, Ontario, Canada
- Peter Isaacson Publications,
Victoria, Australia
- Purdue University,
West Lafayette, Indiana
- Queens College,
Flushing, New York
- RE/SPEC, Inc.,
Rapid City, South Dakota
- Rice University,
Department of Computer Science,
Houston, Texas
- Rogaland University,
Stavanger, Norway
- Ruhr Universität Bochum,
Rechenzentrum,
*Bochum, Federal Republic of
Germany*
- Rutgers University, Hill Center,
Piscataway, New Jersey
- St. Albans School,
*Mount St. Alban, Washington,
D.C.*
- Sandia National Laboratories,
Albuquerque, New Mexico
- SAS Institute,
Cary, North Carolina
- I. P. Sharp Associates,
Palo Alto, California
- Smithsonian Astrophysical
Observatory, Computation Facility,
Cambridge, Massachusetts
- Software Research Associates,
Tokyo, Japan
- Sony Corporation,
Atsugi, Japan
- Space Telescope Science Institute,
Baltimore, Maryland
- Springer-Verlag,
*Heidelberg, Federal Republic of
Germany*
- Stanford Linear
Accelerator Center (SLAC),
Stanford, California
- Stanford University,
Computer Science Department,
Stanford, California
- Stefan Ram, Programming and
Trade, *Berlin, Federal Republic of
Germany*
- Syracuse University,
Syracuse, New York
- Talaris Systems, Inc.,
San Diego, California
- TECOGRAF Software,
Milan, Italy
- Texas A & M University,
Department of Computer Science,
College Station, Texas
- Texcel, *Oslo, Norway*
- TRW, Inc., *Redondo Beach,
California*
- Tufts University,
Medford, Massachusetts
- TV Guide, *Radnor, Pennsylvania*
- TYX Corporation,
Reston, Virginia
- UNI-C, *Aarhus, Denmark*
- Universidad Sevilla,
Sevilla, Spain
- Universidade de Coimbra,
Coimbra, Portugal
- Università degli Studi Milano,
Istituto di Cibernetica,
Milan, Italy
- University College,
Cork, Ireland
- University of Alabama,
Tuscaloosa, Alabama
- University of British Columbia,
Computing Centre,
*Vancouver, British Columbia,
Canada*
- University of British Columbia,
Mathematics Department,
*Vancouver, British Columbia,
Canada*
- University of Calgary,
Calgary, Alberta, Canada
- University of California,
Division of Library Automation,
Oakland, California
- University of California, Berkeley,
Computer Science Division,
Berkeley, California
- University of California, Berkeley,
Space Astrophysics Group,
Berkeley, California
- University of California, Irvine,
Department of Mathematics,
Irvine, California
- University of California, Irvine,
Information & Computer Science,
Irvine, California
- University of California,
Los Angeles, Computer
Science Department Archives,
Los Angeles, California
- University of California,
San Diego, *La Jolla, California*
- University of Canterbury,
Christchurch, New Zealand
- University of Chicago,
Computing Organizations,
Chicago, Illinois
- University of Chicago,
Chicago, Illinois
- University of Crete,
Institute of Computer Science,
Heraklio, Crete, Greece
- University of Delaware,
Newark, Delaware
- University of Exeter,
Computer Unit,
Exeter, Devon, England
- University of Glasgow,
Department of Computing Science,
Glasgow, Scotland
- University of Groningen,
Groningen, The Netherlands
- University of Illinois at Chicago,
Computer Center,
Chicago, Illinois
- University of Kansas,
Academic Computing Services,
Lawrence, Kansas
- University of Maryland,
Department of Computer Science,
College Park, Maryland

University of Maryland
at College Park,
Computer Science Center,
College Park, Maryland

University of Massachusetts,
Amherst, Massachusetts

Université de Montréal,
Montréal, Québec, Canada

University of Oslo,
Institute of Informatics,
Blindern, Oslo, Norway

University of Oslo,
Institute of Mathematics,
Blindern, Oslo, Norway

University of Ottawa,
Ottawa, Ontario, Canada

University of Salford,
Salford, England

University of Southern California,
Information Sciences Institute,
Marina del Rey, California

University of Stockholm,
Department of Mathematics,
Stockholm, Sweden

University of Texas at Austin,
Austin, Texas

University of Vermont,
Burlington, Vermont

University of Washington,
Department of Computer Science,
Seattle, Washington

University of Western Australia,
Regional Computing Centre,
Nedlands, Australia

University of Wisconsin,
Academic Computing Center,
Madison, Wisconsin

Uppsala University,
Uppsala, Sweden

USDA Forest Service,
Washington, D.C.

Vereinigte Aluminium-Werke AG,
Bonn, Federal Republic of Germany

Villanova University,
Villanova, Pennsylvania

Vrije Universiteit,
Amsterdam, The Netherlands

Washington State University,
Pullman, Washington

Widener University,
Computing Services,
Chester, Pennsylvania

John Wiley & Sons, Incorporated,
New York, New York

Worcester Polytechnic Institute,
Worcester, Massachusetts

Yale University, Computer Center,
New Haven, Connecticut

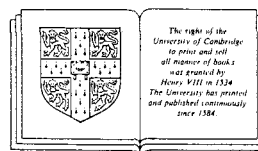
Yale University,
Department of Computer Science,
New Haven, Connecticut

A new and unique service from the Printing Division of the Oldest Press in the World

TEX-to-TYPE

The CAMBRIDGE service that lets you and your publisher decide how your
mathematical or scientific text will appear.

Monotype output in Times and Helvetica as well as a complete range of
Computer Modern faces from your T_EX keystrokes



For details contact

TECHNICAL APPLICATIONS GROUP CAMBRIDGE UNIVERSITY PRESS
UNIVERSITY PRINTING HOUSE SHAFTESBURY ROAD CAMBRIDGE CB2 2BS ENGLAND

TELEPHONE (0223) 325070

Request for Information

The TeX Users Group maintains a database and publishes a membership list containing information about the equipment on which TeX is (or will be) installed and about the applications for which TeX is used. This list is updated periodically and distributed to members with TUGboat, to permit them to identify others with similar interests. Thus, it is important that the information be complete and up-to-date.

Please answer the questions below, in particular those regarding the status of TeX and the hardware on which it runs. (Operating system information is particularly important in the case of IBM mainframes and VAX.) This hardware information is used to group members in the listings by computer and output device.

If accurate information has already been provided by another TUG member at your site, indicate that member's name and the same information will be repeated automatically under your name. If your current listing is correct, you need not answer these questions again. Your cooperation is appreciated.

- *Send completed form with remittance* (checks, money orders, UNESCO coupons) to:
TeX Users Group
P. O. Box 594
Providence, Rhode Island 02901, U.S.A.
- *For foreign bank transfers* direct payment to the TeX Users Group, account #002-031375, at:
Rhode Island Hospital Trust National Bank
One Hospital Trust Plaza
Providence, Rhode Island 02903-2449, U.S.A.
- *General correspondence* about TUG should be addressed to:
TeX Users Group
P. O. Box 9506
Providence, Rhode Island 02940-9506, U.S.A.

Name: _____
Home <input type="checkbox"/> _____
Bus. <input type="checkbox"/> Address: _____

Qty	1990 Membership/TUGboat Subscription (Jan.-Dec.)	Amount
	New (first-time): <input type="checkbox"/> \$35.00 each; students <input type="checkbox"/> \$25.00 each Renewal: <input type="checkbox"/> \$45.00; <input type="checkbox"/> \$35.00 - reduced rate if renewed before February 1, 1990 Mailing charges per subscription : Canada/Mexico - \$5; Europe - \$10; Other Countries - \$15	
	TUGboat back volumes 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 Circle volume(s) desired: v. 1 v. 2 v. 3 v. 4 v. 5 v. 6 v. 7 v. 8 v. 9 v. 10 \$18 \$50 \$35 \$35 \$35 \$50 \$50 \$50 \$50 \$75	

Issues of TUGboat will be shipped via air service outside North America.
Quantity discounts available on request.

TOTAL ENCLOSED: _____
(Prepayment in U.S. dollars required)

Membership List Information

Institution (if not part of address): _____

Date: _____

Title: _____

Status of TeX: Under consideration

Phone: _____

Being installed

Network address: _____

Up and running since: _____

- Arpanet BITnet
- CSnet uucp
- JANET other _____

Approximate number of users: _____

Specific applications or reason for interest in TeX: _____

Version of TeX:

Pascal

C

other (describe)

From whom obtained: _____

My installation can offer the following software or technical support to TUG: _____

Hardware on which TeX is used:

Please list high-level TeX users at your site who would not mind being contacted for information; give name, address, and telephone.

Computer(s)	Operating system(s)	Output device(s)
_____	_____	_____
_____	_____	_____
_____	_____	_____

Publishing Companion translates

WordPerfect

to

TEX

It doesn't take a TEXpert to use TEX.

With **Publishing Companion**, you can publish documents using TEX with **little or no TEX knowledge**. Your WordPerfect files are translated into TEX files, so anyone using this simple word processor can immediately begin typesetting their own documents!

And now, **Publishing Companion** translates **WordPerfect 5.0 and 5.1** files into TEX.

Retail Price \$249.00

Academic Discount Price \$199.00

For the power of TEX with the ease of a word processor, **Publishing Companion** is your "best friend" for desktop publishing.

For more information to place an order, call or write:

K-TALK[®]
COMMUNICATIONS

30 West First Ave., Suite 100
Columbus, Ohio 43201
(614) 294-3535
FAX (614) 294-3704

DESKTOP PUBLISHING HAS NEVER BEEN SIMPLER
AND WILL NEVER BE THE SAME

TEX

T
H
E

Virtual Fonts



A
R
B
O
R
T
E
X
T

ü Ü ö Ö ä Ä

Another first from ArborText
for the TEX Community...

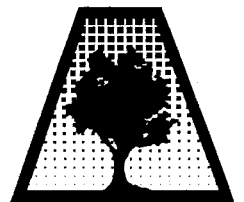
...built in support for Virtual Fonts
with Preview and DVILASER for TEX 3.0.

W
A
Y

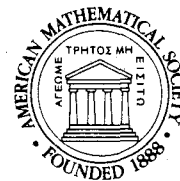
See Us in Cork

at the

First TUG Meeting in Europe



Updated T_EX Products from the American Mathematical Society



A_MS-T_EX Version 2.0

A_MS-T_EX, the T_EX macro package that simplifies the typesetting of complex mathematics, has been updated to version 2.0. A_MS-T_EX is intended to be used in conjunction with AMSFonts 2.0 (see below). However, A_MS-T_EX can also be used without AMSFonts. A_MS-T_EX is available on IBM or Macintosh diskettes—either format may be uploaded to many mainframe computers. **Prices:** \$32 list, \$29 AMS member.

AMSFonts Version 2.0

AMSFonts 2.0 are designed for use with either A_MS-T_EX 2.0 or Plain T_EX. AMSFonts 2.0 **cannot** be used with previous versions of A_MS-T_EX. Two distributions of fonts are available: one for use on PCs and mainframes (with any implementation of T_EX), the other for use on a Macintosh with *Textures*. The fonts included on these distributions are:

Font Name	Description	Point Sizes	Font Name	Description	Point Sizes
CMEX	CM Math Extension	7-9*	EUSM	Euler Script Medium	5-10
CMCSC	CM Caps and Small Caps	8-9*	EUEX	Euler Compatible Extension	7-10
CMMIB	CM Math Italic Boldface	5-9*	MSAM	Symbols	5-10
CMBSY	CM Bold Symbols	5-9*	MSBM	Symbols (w/Blackboard Bold)	5-10
EURB	Euler Cursive Boldface	5-10	WNCYR	Cyrillic Upright	5-10**
EURM	Euler Cursive Medium	5-10	WNCYI	Cyrillic Italic	5-10**
EUFB	Euler Fraktur Boldface	5-10	WNCYB	Cyrillic Boldface	5-10**
EUFM	Euler Fraktur Medium	5-10	WNCYSC	Cyrillic Caps and Small Caps	10**
EUSB	Euler Script Boldface	5-10	WNCYSS	Cyrillic Sans Serif	8-10**

* 10 point is included in the standard T_EX distribution.

** Developed by the University of Washington

AMSFonts for use on a PC or mainframe

- Font Resolution: 118, 180, 240, 300, 400 dpi (one resolution per order).
- Magnification: All the standard T_EX magnifications are included. The standard magnifications are: 100, 109.5, 120, 144, 172.8, 207.4, and 248.8%.
- Format: high-density 5.25" diskettes.
- Prices: \$48 list, \$43 AMS member.

AMSFonts for use on a Macintosh with *Textures*

- Font Resolution: 72, 144, and 300 dpi (all resolutions included in each order.)
- Magnification: The standard distribution includes fonts at 100% and 120%. An extended distribution, containing all the standard T_EX magsteps, is also available.
- Format: double-sided double-density 3.5" diskettes.
- Prices: *Standard* (magsteps 0-1): \$32 list, \$29 AMS member. *Extended* (magsteps 0-5): \$48 list, \$43 AMS member.

SHIPPING AND HANDLING CHARGE: \$8 per order in the US and Canada, \$15 elsewhere.

HOW TO ORDER: Prepayment is required. Send orders to: American Mathematical Society, P. O. Box 1571, Annex Station, Providence, RI 02901 or call the AMS at (401) 455-4000, or (800) 321-4AMS in the continental U.S. and Canada, or write to: T_EX Library, American Mathematical Society, P.O. Box 6248, Providence, RI 02940. Fax: (401) 455-4004 Telex: 797192. When ordering AMSFonts for the PC, specify desired resolution.

Public Domain T_EX

The public domain versions of T_EX software are available from *Maria Code - Data Processing Services* by special arrangement with Stanford University and other contributing universities. The standard distribution tape contains the source of T_EX and METAFONT, the macro libraries for A_MS-T_EX, L_AT_EX, SliT_EX and HP T_EX, sample device drivers for a Versetec and LN03 printers, documentation files, and many useful tools.

Since these are in the public domain, they may be used and copied without royalty concerns. A portion of your tape cost is used to support development at Stanford University.

Compiled versions of T_EX are available for DEC VAX/VMS, IBM CMS, IBM MVS and DEC TOPS systems. Systems using a standard format must compile T_EX with a Pascal compiler.

T_EX Order Form

T_EX Distribution tapes:

- ___ Standard ASCII format
- ___ Standard EBCDIC format
- ___ Special VAX/VMS format Backup
- ___ Special DEC 20/TOPS 20 Dumper format
- ___ Special IBM VM/CMS format
- ___ Special IBM MVS format

Font Library Tapes (GF files)

- ___ 300 dpi VAX/VMS format
- ___ 300 dpi generic format
- ___ IBM 3820/3812 MVS format
- ___ IBM 3800 CMS format
- ___ IBM 4250 CMS format
- ___ IBM 3820/3812 CMS format

Tape prices: \$92.00 for first tape, \$72.00 for each additional tape. Postage: allow 2 lbs. for each tape.

Documents:

	Price \$	Weight	Quantity
T _E Xbook (vol. A) softcover	30.00	2	___
T _E X: The Program (vol. B) hardcover	44.00	4	___
METAFONT book (vol. C) softcover	22.00	2	___
METAFONT: The Program (vol. D) hardcover ...	44.00	4	___
Computer Modern Typefaces (vol. E) hardcover	44.00	4	___
L _A T _E X document preparation system	30.00	2	___
WEB language *	12.00	1	___
T _E Xware *	10.00	1	___
BibT _E X *	10.00	1	___
Torture Test for T _E X *	8.00	1	___
Torture Test for METAFONT *	8.00	1	___
METAFONTware *	15.00	1	___
Metamarks *	15.00	1	___

* published by Stanford University

Orders from within California must add sales tax for your location.

Shipping charges: domestic book rate-no charge, domestic priority mail-\$1.50/lb, air mail to Canada and Mexico-\$2.00/lb, export surface mail (all countries)-\$1.50/lb, air mail to Europe, South America-\$5.00/lb, air mail to Far East, Africa, Israel-\$7.00/lb.

Purchase orders accepted. Payment by check must be drawn on a U.S. bank.

Send your order to: Maria Code, DP Services, 1371 Sydney Drive, Sunnyvale, CA 94087
 FAX: 415-948-9388 Tel.: 408-735-8006.

TEX EDITION of MathEdit

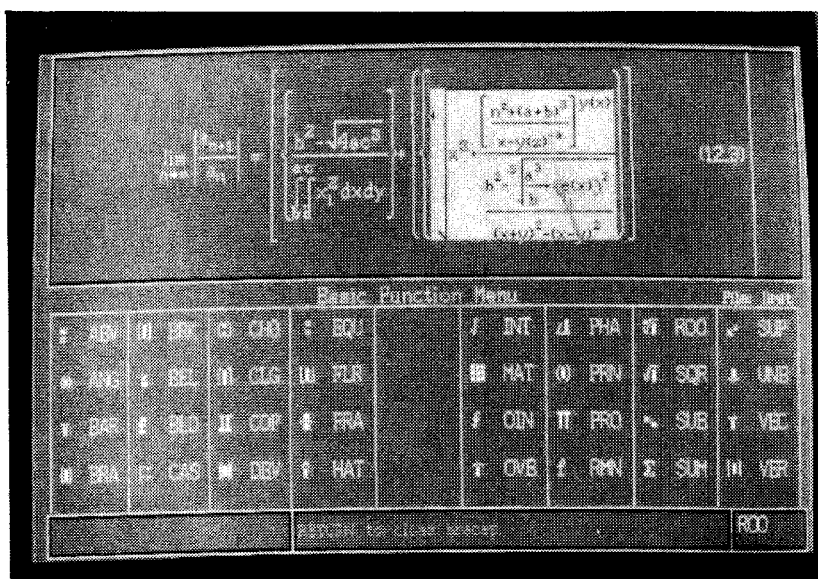
This powerful new equation editor can
be used to create equations for TEX.
For IBM PC

MathEdit

- * Easy to Use Version 2.0
- * Menu driven so no codes need to be learned
- * High-quality TEX printing

WYSIWYG ->

View your equation as you create it. Then insert into your TEX document with one command.



TEX Edition ONLY \$129.00

Professional Edition \$199.00

Shipping: \$4 (U.S.A.), \$25 (Canada), \$35 (Overseas)

VISA, Mastercard and University and Government P.O.'s accepted.

K-TALK
COMMUNICATIONS

30 West First Avenue
Columbus, Ohio 43201
(614) 294-3535
FAX (614) 294-3704

TYPESETTING: JUST
\$2.50
PER PAGE!

Send us your T_EX DVI files and we will typeset your material at 2000 dpi on quality photographic paper — \$2.50 per page!

Choose from these available fonts: Computer Modern, Bitstream Fontware™, and any METAFONT fonts. (For each METAFONT font used other than Computer Modern, \$15 setup is charged. This ad was composed with PCT_EX® and Bitstream Dutch (Times Roman) fonts, and printed on RC paper at 2000 dpi with the Chelgraph IBX typesetter.)

And the good news is: just \$2.50 per page, \$2.25 each for 100+ pages, \$2.00 each for 500+ pages! Laser proofs \$.50 per page. (\$25 minimum on all jobs.)

Call or write today for complete information, sample prints, and our order form. **TYPE 2000, 16 Madrona Avenue, Mill Valley, CA 94941. Phone 415/388-8873.**

TYPE
2000

Do more and do it better with new PCT_EX.

PCT_EX, PCT_EX/386 & Big PCT_EX/386, Versions 3.0

Feature/Specification		PC T _E X 2.93	PC T _E X 3.0	PC T _E X/386 3.0	Big PC T _E X/386 3.0
Page & Memory Capacity	mem_max	65534	131070	131070	262140
You won't see "T _E X Capacity Exceeded"!		(1.00)	(Double!)	(Double!)	(Quadruple!)
Hyphenation Table Size	trie_size	15000	30000	30000	60000
Space for hyphenation patterns		(1.00)	(Double!)	(Double!)	(Quadruple!)
Trie Op Size	trie_op_size	255	1024	1024	2048
Complexity of hyphenation patterns		(1.00)	(4.02)	(4.02)	(8.03)
Maximum Trie Ops Per Language		N/A	512	512	512
Especially important for Dutch and German hyphenation					
Buffer Size	buf_size	1024	1500	1500	3000
Maximum # of characters on input lines		(1.00)	(1.46)	(1.46)	(2.93)
Stack Size	stack_size	200	200	200	300
Maximum # of simultaneous input sources		(1.00)	(1.00)	(1.00)	(1.50)
Maximum # of Strings	max_strings	4500	5000	5000	10000
		(1.00)	(1.11)	(1.11)	(2.22)
String Pool	pool_size	50000	50000	60000	60000
Maximum # of characters in strings		(1.00)	(1.00)	(1.20)	(1.20)
Save Size	save_size	600	2000	2000	4000
Space for saving values outside current group		(1.00)	(3.33)	(3.33)	(6.67)
Maximum # of T_EX Commands	hash_size	3000	5000	5000	10000
		(1.00)	(1.66)	(1.66)	(3.33)
Minimum Free RAM Required		385K	385K	1.3M	1.3M
		(1.00)	(1.00)	(3.38)	(3.38)
Minimum Free RAM Recommended		550K	550K	1.3M	4.0M
Memory recommended for optimum performance		(1.00)	(1.00)	(2.36)	(7.27)
Font Memory	font_mem_size	51199	65534	65534	65534
For TFM data storage		(1.00)	(1.28)	(1.28)	(1.28)
Maximum Fonts Per Job	font_max	127	127	127	255
		(1.00)	(1.00)	(1.00)	(2.00)
List Price		\$249	\$249	\$295	\$349
Order yours today!		(1.00)	(1.00)	(1.18)	(1.40)
Upgrade Price		\$50	\$50	\$99	\$149
From PC T _E X 2.93 or earlier version		(1.00)	(1.00)	(1.98)	(2.98)

This all adds up to...

More power, greater performance, and increased memory capacity for the latest versions of popular macro packages like L^AT_EX and A_MS-T_EX. And all three new PC T_EX products feature the character sets and hyphenation tables to handle even the most complex European languages.

Order today. Call (415) 388-8853.

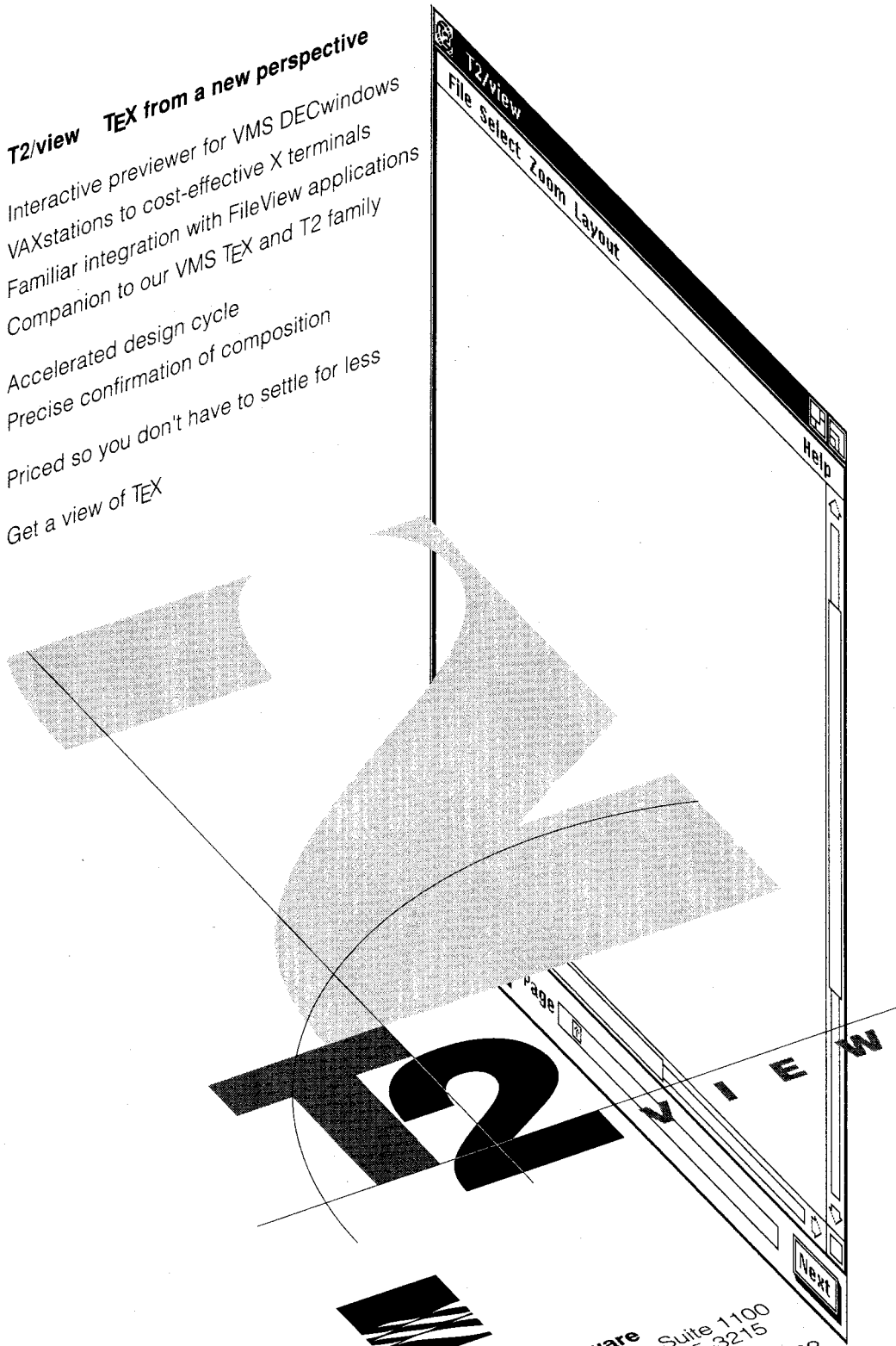
PERSONAL

INC

12 Madrona Avenue
Mill Valley, CA 94941

T2/view TEX from a new perspective

Interactive previewer for VMS DECwindows
VAXstations to cost-effective X terminals
Familiar integration with FileView applications
Companion to our VMS TeX and T2 family
Accelerated design cycle
Precise confirmation of composition
Priced so you don't have to settle for less
Get a view of TeX

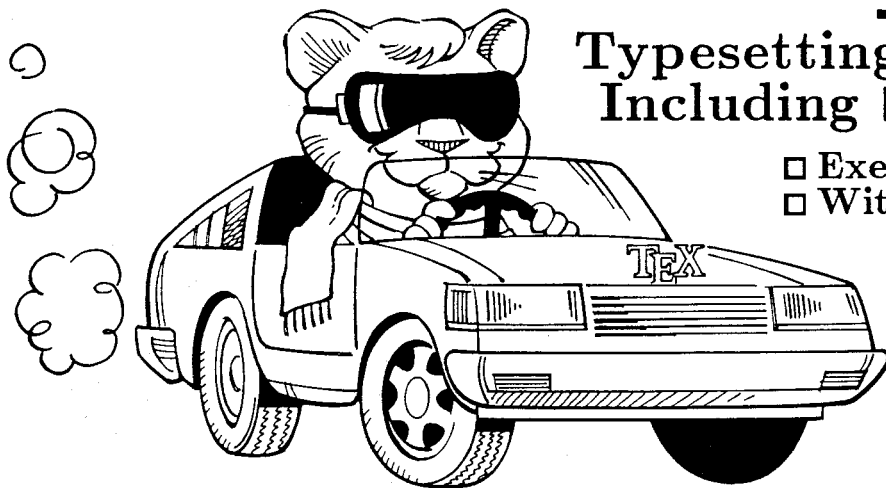


Northlake Software
812 SW Washington, Suite 1100
Portland, Oregon 97205-3215
USA
503-228-3383 Fax 503-228-5662

TurboTeX

Typesetting Software Including METAFONT

- Executables \$150
- With source \$300



TeX 3.0
Update!

TurboTeX Release 3.0 software brings you the latest TeX 3.0 and METAFONT 2.0 standards: preloaded plain TeX, L^ATeX, A^AS-TeX and A^AS-L^ATeX, and plain METAFONT interfaced to CGA/EGA/VGA/Hercules graphics; TRIP and TRAP certification; Computer Modern and L^ATeX fonts, and printer drivers for HP LaserJet Plus/II/IIP, HP DeskJet, PostScript, and Epson LQ and FX dot-matrix printers. This wealth of software runs on your IBM PC (MS-DOS or OS/2), UNIX, or VAX/VMS system.

■ **Best-selling Value:** TurboTeX sets the standard for power and value among TeX implementations: one price buys a complete, commercially-hardened typesetting system. *Computer* magazine recommended it as "the version of TeX to have," *IEEE Software* called it "industrial strength," and thousands of satisfied users worldwide agree.

TurboTeX gets you started quickly, installing itself automatically under MS-DOS, and compiling itself automatically under UNIX. The 90-page User's Guide includes generous examples and a full index, and leads you step-by-step through installing and using TeX and METAFONT.

■ **Power Features:** TurboTeX breaks the 640K memory barrier under MS-DOS on any IBM-compatible PC with our virtual memory sub-system. Even without expanded memory hardware, you'll

have the same sized TeX that runs on multi-megabyte mainframes, with plenty of memory for large documents, complicated formats, and demanding macro packages (like PICTeX and A^AS-L^ATeX 2.0) that break other TeX implementations. On larger computers, TurboTeX runs up to 3 times faster in less memory than the Stanford Pascal distribution.

■ **Source code:** Order the TurboTeX source in portable C, and you will receive more disks with over 85,000 lines of generously commented TeX, TurboTeX, METAFONT, and printer driver source code, including: our WEB system in C; PASCHAL, our proprietary Pascal-to-C translator; and preloading, virtual memory, and graphics code. TurboTeX meets C portability standards like ANSI and K&R, and is robustly portable to a growing family of operating systems.

■ **Availability & Requirements:** TurboTeX executables for IBM PC's include the User's Guide and require 640K and hard disk. Order source code (includes Programmer's Guide) for other machines. Source compiles with Microsoft C 5.0 or later on the PC; other systems need 1 MB memory and a C compiler supporting UNIX standard I/O. Media is 360K 5-1/4" PC floppy disks; other formats at extra cost.

■ **Upgrades:** If you have TurboTeX Release 2.0, you can upgrade the executables for only \$40. If you have the source distribution, upgrade

both executables and source for \$80. Or, get either applicable upgrade free when you buy the AP-TeX fonts (see facing page) for \$200!

■ **No-risk trial offer:** Examine the documentation and run the PC TurboTeX for 10 days. If you are not satisfied, return it for a 100% refund or credit. (Offer applies to PC executables only.)

■ **Free Buyer's Guide:** Ask for the free, 70-page Buyer's Guide for more details on TurboTeX and dozens of TeX-related products: previewers, TeX-to-FAX and TeX-to-Ventura/Pagemaker translators, optional fonts, graphics editors, public domain TeX accessory software, books and reports.

Ordering TurboTeX

Ordering TurboTeX is easy and delivery is fast, by phone, FAX, or mail. Terms: Check with order (free media and ground shipping in US), VISA, Mastercard (free media, shipping extra); Net 30 to well-rated firms and public agencies (shipping and media extra). Discounts available for quantities or resale. International orders gladly expedited via Air or Express Mail.

The Kinch Computer Company
PUBLISHERS OF TURBOTeX
501 South Meadow Street
Ithaca, New York 14850 USA
Telephone (607) 273-0222
FAX (607) 273-0484

TEX Users

Take Note

Computer Composition Corporation offers the following services to those who are creating their technical files using TEX:

- Convert your DVI files to fully paginated typeset pages on our APS-5 phototypesetters at 1400 dpi resolution.
- Files can be submitted on magnetic tape or PC diskettes.
- Provide 300 dpi laser-printed page proofs which simulate the typeset page. (Optional service \$1.50 per page)
- Macro writing and keyboarding from traditionally prepared manuscripts **in several typeface families** via the TEX processing system. Send us your manuscript for our review and quotation.
- Full keylining and camera work services, including halftones, line art, screens and full-page negatives or positives for your printer.
- Quick turnaround (**usually less than 48 hours!**) on customer supplied DVI files of 500 typeset pages or less.
- From DVI files: first 100 typeset pages at \$4.75 per page; 100 pages and over at \$3.50 per page. **Lower prices for slower turnaround service.**

*For further information and / or a specific quotation,
call or write Frank Frye or Tim Buckler*



COMPUTER COMPOSITION CORPORATION

1401 West Girard Avenue • Madison Heights, MI 48071

(313) 545-4330 FAX (313) 544-1611

— Since 1970 —

AP-TEX Fonts

485 TEX fonts identical to
Adobe PostScript Fonts for \$200

Get ready for the quality of Adobe PostScript fonts for your TEX documents and non-PostScript printer! If you use any brand of TEX with an HP LaserJet or DeskJet printer, the AP-TEX fonts from Kinch add a wealth of attractive typefaces identical to the popular PostScript extended font families.

By de-crypting the Adobe coding, we are able to exactly translate the PostScript fonts into TEX font bit map and metric files. These translated fonts include the renowned Adobe "hints," which render the smaller point sizes of the fonts with remarkable clarity on laser and ink-jet printers. The fonts use the TEX character set encoding and font metrics, including full kerning and ligature programs.

The AP-TEX fonts, supplied on ten 360K 5-1/4" PC floppy disks, contain 35 typefaces in PK format (including TEX font metric (TFM) files) for 300 dots/inch laser and ink-jet printers. The fonts included are identical to the Adobe PostScript implementations of the trade names and samples shown at right. The point sizes for each typeface included are the TEX sizes 5, 6, 7, 8, 9, 10, 11, 12, 14.4, 17.3, 20.7, and 24.9 points. Headline styles (equal to Times Roman, Helvetica, and Palatino, all in bold) also are included at 29.9, 35.8, 43.0, 51.6, 61.9, and 74.3 points.

The Kinch Computer Company
PUBLISHERS OF TURBOTEX
501 South Meadow Street
Ithaca, New York 14850
Telephone (607) 273-0222
FAX (607) 273-0484

Helvetica, Palatino, Times, and New Century Schoolbook are trademarks of Allied Linotype Co. ITC Avant Garde, ITC Bookman, ITC Zapf Chancery, and ITC Zapf Dingbats are registered trademarks of International Typeface Corporation. LaserJet and DeskJet are trademarks of Hewlett-Packard Corporation. PostScript is a registered trademark of Adobe Systems Incorporated. TEX is a trademark of the American Math Society. TurboTEX and AP-TEX are trademarks of Kinch Computer Company. Prices and specifications subject to change without notice. Revised February 8, 1990.

Times Roman
Times Bold
Times Italic
Times Bold Italic
Helvetica
Helvetica Bold
Helvetica Oblique
Helvetica Bold Oblique
Courier
Courier Bold
Courier Oblique
Courier Bold Oblique
Avant Garde Book
Avant Garde Book Oblique
Avant Garde Demi
Avant Garde Demi Oblique
Bookman Demi
Bookman Demi Italic
Bookman Light
Bookman Light Italic
Helvetica Narrow
Helvetica Narrow Bold
Helvetica Narrow Bold Oblique
Helvetica Narrow Oblique
New Century Schoolbook Roman
New Century Schoolbook Bold
New Century Schoolbook Italic
New Century Schoolbook Bold Italic
Palatino Roman
Palatino Bold
Palatino Italic
Palatino Bold Italic
Zapf Chancery Medium Italic
Zapf Dingbats ☸☉*✻*✻*✻*✻*✻*✻*✻*✻*
Symbol ΔΦΓϑΛΠΘΣΥΩΞΨαβχδεφγ

Kinch

VECTOR TEX

T B
W U
C O
& H
E A
M

TEX FOR THE 90'S

Are you still
struggling with
PXL's, PK's or GF's?
Move on to scalable
fonts:

- Save megabytes of storage—entire VT_EX fits on one floppy.
- Instantly generate any font in any size and in any variation from 5 to 100 points.
- Standard font effects include compression, slant, smallcaps, outline, shading and shadow.
New: landscape.
- Discover the universe of MicroPress Font Library professional typefaces: not available from any other T_EX vender.

List price \$399 **Introductory offer \$299**

Includes the VT_EX typesetter (superset of T_EX), 10 scalable typefaces, VVIEW (arbitrary magnification on EGA, CGA, VGA, Hercules, AT&T), VLASER (HP LaserJet), VPOST (PostScript), VDOT (Epson, Panasonic, NEC, Toshiba, Proprinter, Star, DeskJet) and manuals.

Introductory offer expires on September 1, 1990. S/H add \$5. COD add \$5. WordPerfect Interface add \$100. Site licenses available. Dealers' inquiries welcome. Professional typefaces available for older implementations of T_EX.



MicroPress Inc.

67-30 Clyde Street, #2N, Forest Hills, NY 11375
Tel: (718) 575-1816 Fax: (718) 575-8038



TeX Publishing Services



From the Basic


The American Mathematical Society can offer you a basic TeX publishing service. You provide the DVI file and we will produce typeset pages using an Autologic APS Micro-5 phototypesetter. The low cost is basic too: only \$5 per page for the first 100 pages; \$2.50 per page for additional pages, with a \$30 minimum. Quick turnaround is important to you and us ... a manuscript up to 500 pages can be back in your hands in just one week or less.

To the Complex

As a full service TeX publisher, you can look to the American Mathematical Society as a single source for all your publishing needs.

Macro-Writing	TeX Problem Solving	Autologic Fonts	Keyboarding
Art and Pasteup	Camera Work	Printing	Binding

For more information or to schedule a job, please contact Regina Girouard, American Mathematical Society, P.O. Box 6248, Providence, RI 02940 or call 401-455-4060 or 800-321-4AMS in the continental U.S.

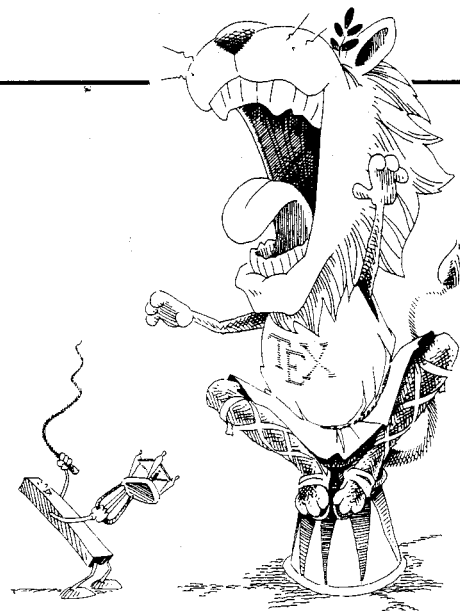
<ul style="list-style-type: none"> ▼ ● ☆ ➔ □ ⇩ ★ 	<h1>\$79</h1> <p>IS ALL IT WILL TAKE TO ADD A FEW OF THESE TO YOUR TeX DOCUMENTS</p>	<ul style="list-style-type: none"> ◇ ← ● ▲ ☆ ⇩ ○ 	<p>With TeXPIC Graphics language*, you will have the tools to make graphics for your TeX documents. TeXPIC is now available from Bob Harris at:</p> <div style="text-align: center;">  <p>MICRO PROGRAMS INC 251 Jackson Avenue, Syosset NY 11791 Telephone: (516) 921 1351.</p> <p><small>*TUG Boat Volume 10, No. 4, Page 627 1989 Stanford Conference Proceedings</small></p> </div>
---	--	---	---

Index of Advertisers

469, 479	American Mathematical Society	476, 477	Kinch Computer Company
468	ArborText	479	Micro Programs, Inc.
cover 3	Blue Sky Research	478	MicroPress, Inc.
465	Cambridge University Press	474	Northlake Software
475	Computer Composition	473	Personal TeX Inc.
470	DP Services	480	TeX Users Group
467, 471	K-Talk Communications	472	Type 2000

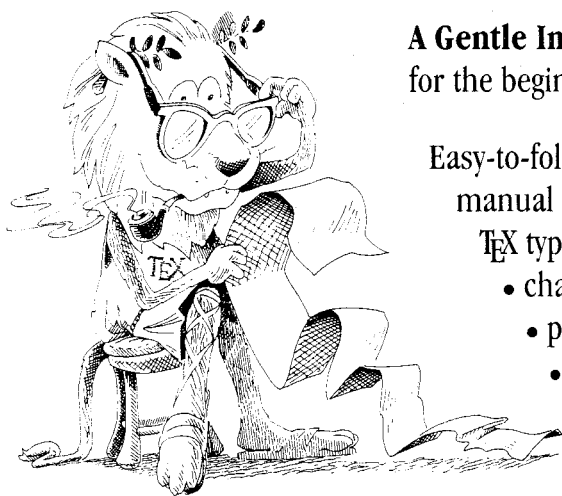
TAME THAT TEX LION!

with



A Gentle Introduction to TEX

A Manual for Self-study by Michael Doob



©Lions, *The TEXbook*, 1986; used by permission of Addison-Wesley Publishing Co.

A Gentle Introduction to TEX is perfect for the beginning TEX user.

Easy-to-follow and straightforward, this 89 page, softbound manual reveals the basics behind

TEX typesetting:

- characters and words
- paragraphs and pages
- mathematics
- simple tables
- simple macros

...with dozens of helpful examples and solutions in 11 friendly chapters.

Special offer for TUG members ... **\$10**
 10 or more copies (members) **\$ 8**
 Regular price **\$15**

Discounts good through December 31, 1990

Shipping: U.S. \$2 per copy
 Outside of U.S. \$3 per copy surface; \$4 air

Ask about TUG's other publications for beginning TEXers



Available now
from TUG

TEX Users Group
P. O. Box 9506

Providence, RI 02940 U. S. A.

Phone: (401) 751-7760

Fax: (401) 751-1071

Mastercard/Visa, checks and money orders accepted