negative value — which nobody in their right mind would ever do, but if they do — apparently TEX goes crazy. So I have to fix that. Ugh.

What I love is when excellent new publications come out that I know wouldn't have been done at all without TEX, and also when I see people — as I said, Frank and Yannis — spending a considerable part of their lives doing work that has very high quality. They are excited just by the chance of improving the quality of publication. Those are the things that make me happy.

RG: Thank you very much.

Dreamboat

Moving a Fixed Point

Richard Palais

Abstract

In the past few years there has been increasing discussion of the question "Has the time has come to make basic changes to the inner workings of TEX?". In late May of 1992, Rainer Schoepf set up a mailing list on the Internet, called "NTS-L", to discuss the matter. I started out being completely opposed to the idea of even the slightest changes to the TEX code, feeling that whatever failings TEX might have, they are best approached by pre and post processing ("front and back ends"), and anyway are negligible compared to the danger of losing the remarkable coherence and interchangeability of TEX software, everywhere and on all platforms that is enforced by the discipline of having a single, universally accepted underlying piece of software (INITEX). However, after following the discussion carefully for nearly two months, I was convinced by evidence that, for certain purposes, TEX was no longer fulfilling its promise of providing typesetting of uncompromising high quality, and probably only careful and limited changes and additions to TEX primitives could correct this. What follows is a long message I posted to NTS-L, outlining a minimalist approach to changing TEX, and also a suggested method for implementing changes to TEX code that would insure documents written for standard TEX could still run under the new system. A number of replies to my message were posted to NTS-L and others were addressed to me personally via email. Rather than incorporate these comments by making appropriate changes to the version I posted, I have decided to append a short addendum, mentioning a few of the more important points made in these replies.

– – * – –

This is going to be more a "position paper" than a simple message. I have been following the NTS-L mail list discussion with considerable interest and finally felt that there were so many issues that I wanted to address and remarks that I wanted either to agree with or to dispute, that only a fairly extensive reply would do. Here is a table of contents:

Introduction

First a short personal introduction. The oldtimers of the TEX world will perhaps remember me — I was the founding chairman of TUG, worked closely with Don Knuth during the early years of TEX, and I wrote a column on mathematical typesetting in the *Notices of the AMS* for three years, with the goal of easing the transition in the mathematical community from the typewriter, along WYSIWYG road, and into the bright new Promised Land of TEX. But my name may well be unfamiliar to more recent arrivals in the TEX world, for lately I have been only a "lurker" on comp.text.tex, and while I read *TUGboat* and use TEX daily for writing my letters, papers, and books, and in connection with my duties as an editor of the *Bulletin of the AMS*, I have not recently been contributing either to the development or to the public discussion of TEX.

Next a disclaimer. While I know my way around in *The TEXbook* and have been writing my own macros and formats since 1978, I consider myself an amateur, not at all in the same league with TEXperts like Barbara Beeton, Michael Downes, Victor Eijkhout, Karl Berry, Larry Siebenmann, Tim Murphy, and others who have been contributing to this discussion. So I will happily defer to them on technical matters and hope that they will correct any of my misstatements. What I would like to do is take the point of view of a devoted TEX

user; one not so enamoured of TEX as to be unable to see its warts, but one who appreciates what a unique software miracle TEX is, and is willing to try to fix things only if assured that it will not subvert that miracle. One more fact about me bears emphasizing; as a mathematician I do have a somewhat biased view of TEX. For me TEX is not just *a* typesetting system, it is *the* mathematical and "TEXnical" typesetting system.

I would like to begin with a quotation from Don Knuth's "Remarks to Celebrate the Publication of *Computers & Typesetting*" at the Computer Museum, Boston, Massachusetts, May 21, 1986, as printed in *TUGboat*, vol. 7 (1986) no. 2, pp. 95–98:

> ...Ever since these beginnings in 1977, the TEX research project that I embarked on was driven by two major goals. The first goal was *quality*: we wanted to produce documents that were not just nice, but actually the best... My goal was to take the last step and go all the way to the finest quality that had ever been achieved in printed documents...
>
> The second major design goal was to be *archival*: to create systems that would be independent of changes in printing technology as much as possible. When the next generation of printing devices came along, I wanted to be able to retain the same quality already achieved, instead of having to solve all the problems anew. I wanted to design something that would still be usable in 100 years. In other words, my goal was to arrange things so that, if book specifications are saved now, our descendants should be able to produce an equivalent book in the year 2086. Although I expect that there will be a continual development of "front ends" to TEX and METAFONT, as well as a continual development of "back ends" or device drivers that operate on the output of the systems, I designed TEX and METAFONT themselves so they will not have to change at all: They should be *fixed points* in the middle, solid enough to build and rely on.

Perhaps it is because I was in the audience when Don made those remarks that they seem particularly important to me, but in any case, as my contribution to the NTS discussion, let me attempt to analyse the TEX system and some of its purported shortcomings in the light of Knuth's quotation. More specifically, I would like to address the following:

QUESTIONS.

1) *Are Knuth's two goals consistent, or has the continual quest for ultimate quality in typesetting exposed problems with TEX so intractible that they cannot be addressed simply by creating new and better front and back ends for the TEX system?*
2) *If so, can these "intractible" problems be solved by changes to TEX that will leave it compatible with the current version (and in particular able to pass Knuth's "trip-test").*

The Many Faces of TEX

TEX is a complex system that can appear as many things to different people (or even to one person at different times). In fact it is a little like the proverbial elephant that the blind men perceived in so many ways depending on how they "interfaced" with it.

I think that this many-faceted nature of TEX may account, at least in part, for some of the unfocused and chaotic discourse that has been taking place on this mailing list. Someone will comment either critically or in praise of one aspect of the TEX system and someone else will contradict that comment, but really in reference to some other aspect of the system. As anyone scanning comp.text.tex realizes, LATEX users face a whole different set of problems than plain TEX users, and likewise $\mathcal{AMS}$-TEX and LA$\mathcal{MS}$-TEX provide still other environments, with differing attendant strengths, problems, and difficulties. The complaint, repeated several times in the recent discussions, that TEX is incompetent to do commutative diagrams, may seem obvious to a frustrated user of plain TEX, but it would perplex a user of LA$\mathcal{MS}$-TEX who will tell you that it is an absolute snap using "TEX" to make beautiful commutative diagrams, even very complicated ones with arrows set at almost arbitrary slopes and with all kinds of decorations on them. Likewise, it is well-known that designing tables can be a painful chore with (plain) TEX. But there are a number of excellent macro packages around that automate this problem away. Even that most serious problem of integrating graphics into TEX can be considered solved in the right TEX environment. In the hands of a competent artist, a Macintosh equipped with Textures, Adobe Illustrator, and a PostScript printer can create strikingly professional integrated graphics and text. Yes, I know that this solution gives up the portability of TEX documents — bad things can sometimes even happen between the proofing device and the high resolution camera copy typesetter — but the point is that *many apparent problems with TEX can be solved by coupling TEX to suitable front and back ends, with no reprogramming at all of TEX itself.* Someone suggested that TEX

needs Bézier curves as a new primitive. I will argue that the Bézier curves belong in an Illustrator-like program, *not* in TEX. Solving the problem of portability is trivial in comparison with the nightmarish difficulties that I foresee as virtually certain to follow from trying to add anything so foreign as Bézier curves to TEX's data structures!

A "Standards" Approach to Solving TEX Portability Problems

As just suggested above, I believe that at least some of the major defects currently perceived in the TEX system are not so much problems with TEX itself, but rather arise from the vital requirement that *TEX documents should be completely portable between various hardware platforms.* As long as we are dealing with TEX itself, this portability is assured by the minimal requirement that all true TEX systems will produce the same DVI file from a given source file. But of course a DVI file is only part of the way to a printed page, so TEX without some sort of back end is virtually useless. We sometimes forget that even the software combinations formed by a set of font glyphs (either bitmaps or outlines) and a screen previewer or printer driver is already a back end to TEX. If we are willing to stick with the Computer Modern family of fonts in the bitmapped format provided by METAFONT, then virtually all screen previewers and printer drivers will work faultlessly and provide "identical" output to a tolerance limited only by resolution. The reason of course is that these fonts are a carefully specified standard, on which the writer of a device driver can completely rely. But of course Knuth never intended TEX to be limited to the CM family of fonts, or even to METAFONT designed fonts. Currently, Adobe's PostScript Type 1 fonts are the world's favorite, and it has become increasingly the case that a typesetting system, if it is to remain acceptable, **must** be able to deal at the very least with the basic thirty-five fonts built into PostScript printers. Of course TEX was easily up to the challenge. All that is necessary is to build a TFM file for each Type 1 font (or better yet an AFM to TFM conversion program), and add the basic code to the device driver to handle a Type 1 font. On any given system this is an easy task, since again the Type 1 format is a completely specified standard. I know this was done several years ago on the Macintosh, and I believe it has also been done for most of the other major hardware platforms. There are now even a number of well hinted Type 1 versions of the basic Computer Modern fonts available. However even this quite simple new back end leads to portability problems between systems. I have never tried it, but I suspect strongly that if I sent a colleague with an IBM clone one of my Textures source files that used Times Roman, it would not work under PCTEX or emTEX without modification. The problems here are quite trivial, involving little more than differences in font naming conventions. All that would be necessary to regain complete cross-platform portability when using PostScript fonts is some standardized naming conventions. I have made a point of this not because it is a difficult problem that has worried people much; rather because it is a simple problem with an easy solution — but one that I think can be generalized to solve many other TEX problems without in any way tampering with TEX itself.

For a hard example, let's consider a problem that has been the subject of a great deal of discussion in the TEX community and in *TUGboat*, namely specifying graphics within a TEX source file. Of course one possibility that has been mentioned would be to add a number of graphics primitives to TEX: lines, circles, Bézier curves, colors, fills, bitmaps, etc. To my mind this would be absolute madness, and I find it hard to believe any one would seriously consider it. The obvious reason to reject this approach is that it would lead to a program infinitely more complex than TEX that could never be made bug free or portable. Moreover in a few years, when Bézier curves are perhaps out of fashion, and some new graphics goodies are all the rage, there will be a call for yet another "upgrade" of TEX. But a better reason to reject it is that one should not attempt to brush one's teeth with a paintbrush or try to paint a picture with a toothbrush — use the correct tool for each job. And while Swiss Army knives may make fine souvenirs and conversation pieces, they are not high quality tools.

The simple and straightforward solution is to consider a graphic as just another box (a "bounding box"), just like any other TEX box, and let some appropriate back end worry about what is inside the box and render it appropriately on a screen or sheet of paper. Then one can always create graphics with the very best front end graphics tools currently available on a given platform, save it in an appropriate ASCII-based file format, such as encapsulated PostScript, tell TEX about its bounding box and its format, and let the back end take over from there. "But wait a minute," you say, "isn't that exactly the old "`\special`" approach?" Of course it is, and I claim that

the \special mechanism has worked very well *except* for the problems with portability that it has introduced. Now experience has taught that the correct approach to portability problems is *not* to create complex do-it-all programs and then struggle to make them work on dozens of different platforms. Rather, one should have single purpose modules with simple data structures and well-defined interfaces, and use these to build up more complex systems. So, I maintain that what is required to solve the portability difficulties caused by graphic elements in TEX is to make a serious effort to set up cross-platform TEX standards for various officially recognized graphics formats and a standard syntax for \specials to go along with them. It would have to be understood that as technology advances, older formats will probably die out and be replaced by newer ones, so there should probably be a standing committee, perhaps of TUG, to oversee the promulgation and maintenance of these graphics standards. In the same way there could be another standing committee for setting TEX standards for font formats and naming conventions for fonts.

By the way, while we are on the matter of fonts and standards, let me complain about what I feel is a serious failing of the TEX community. The Grand Wizard, as a sort of parting gift, gave us a potentially very valuable tool to handle all sorts of font problems. This was in the form of a well-defined standard — I'm referring of course to virtual fonts (VF). I'm a little over my head here technically, but I believe that as well as solving the more obvious problems for which they were introduced, virtual fonts could be used to handle some more esoteric tricks like adding color and other attributes to fonts. But my feeling is that we have dropped the ball. Not enough TEX systems have implemented VF to make it a dependable way to solve cross-platform TEX problems — even Blue Sky Research, which prides itself in providing a state of the art TEX environment for their Textures system on the Macintosh, has yet to implement it.

Let me end this part of the discussion with a mention of one thing that I feel should neither be a part of NTS nor even a standardized front end for it, and that is the user interface. I would not have brought this up except that there has been discussion on this list giving favorable mention to creating a standardized graphical user interface as part of NTS. But the hardest part of programming these days, and the most system dependent, is building a GUI. Even on a single platform, like the Macintosh, these can break when a new system update comes out. In general, even with systems as close in spirit as the Mac OS, Windows, and NeXT, it is extremely difficult to write a uniform GUI for a program meant to run on several platforms, and porting a GUI from one of these to say X-Windows on UNIX would be even harder. Moreover, each platform has certain User Interface Guidelines for its own GUI, and users get quite upset when a program deviates from them. Since these guidelines differ from one platform to the next, some users, and most likely all, would be upset by any uniform choice. Finally, what is the point? All this would do is stifle creativity and progress. Let the implementors of NTS on each platform design and construct the user interface most suitable for that platform.

The Matter of Compatibility

There has been a lot of discussion on NTS-L concerning the question of whether NTS should necessarily be compatible with the current version of TEX. Until this point I have tried to be calmly analytical, but this is a crucial issue, and one I feel very strongly about, so I am going to drop into a more polemical mode at this point (though I will try to keep my arguments rational). In a word I feel that *backwards compatibility is an absolute sine qua non for any system that aspires to be accepted as a "successor" to TEX.*

Of course, if a group wants to break off to design a completely new typesetting system from scratch that is fine with me — just as long as they don't use TEX in the name or pretend it is some sort of "successor" to TEX. As for me, I would like to see NTS be an improved version of TEX, and for this, it should either be 100% compatible with TEX, or if not it should at least default to a "compatibility mode" which is 100% compatible. I will suggest later a method by which major internal changes could be made to TEX and still satisfy this essential requirement, but now let me be precise about what I mean by compatibility and say why I feel that this a no-compromise issue.

INITEX is the core TEX program, the basic compiled version of the TEX code that knows only TEX's primitives. In a certain sense INITEX *is* TEX. It is the implementation of INITEX that determines whether a "TEX" system is authentic, i.e., passes Knuth's trip-test, and I think there is little doubt that INITEX is one of the "fixed points" that Don was referring to in the above quotation. Let me argue as strongly as I can that *whatever NTS is, its core typesetting function should be based on* INITEX — a version that will pass the trip-test. The

reason has nothing to do with "keeping the faith". Rather it is purely practical. If the new system is compatible with TEX, it will find ready acceptance. But if it is not, then the immense installed base of TEX users will almost certainly shun it, and it will consequently be stillborn.

Let me provide some details about the part of this "user base" that I know something about, the mathematical community, since I have seen comments on the mailing list that indicate a serious lack of comprehension of how sizable this group is (relative to the TEX community) and how dependent it has become on TEX. This in turn may have led to what I consider a very unfair comment, namely that TEX is a "toy for mathematicians". By the way, while my firsthand knowledge is restricted to mathematics, I know by hearsay that much of the following holds true for theoretical physics and also in many other scientific and technical disciplines in which mathematical text makes up a substantial part of papers written in that discipline.

First, virtually all mathematics graduate students now write their dissertations in TEX, and from then on write all their papers in TEX. Secondly, nearly all mathematicians below age forty have learned TEX, and an increasing number of the older generation are either switching to TEX, if they write their own papers, or else are having their secretaries and technical typists learn TEX and write their papers in it. A couple of years ago many mathematicians were still using WYSIWYG mathematical word processors, but now one sees very few preprints prepared in any format except TEX. There are of course lots of reasons for this rapid, wholesale switching to TEX, and probably different reasons have been important for different people. Here are a few:

- Mathematics set by TEX looks much more professional.
- Setting mathematics with TEX is faster and easier (after a painful, but short, learning curve).
- Mathematical text in TEX format can be sent over the Internet and works on all machines. This makes TEX an ideal medium for joint authors to use in their collaboration. WYSIWYG formats are machine dependent and need special coding and decoding when sent over the net.
- As a result of the above, the TEX mathematical input language is becoming a *lingua franca* for the linearization of mathematical text in email and other ASCII documents, even if they are not meant for typesetting.
- The two largest mathematical publishers, the American Mathematical Society and Springer-Verlag (and many others besides), now accept papers in TEX format, either on disc or over the Internet. Papers submitted this way often get published more rapidly and of course final proofreading is minimal.

In any case, the mathematical community now has become so dependent on TEX and has such a substantial investment in software, personal macro files, and source files for the current version of TEX, that I believe *it is virtually certain to reject any purported successor system that does not protect that investment.*

Since I seem to be at odds with Mike Dowling on this matter, let me quote some of his remarks and point out an important issue he seems to have overlooked:

> (1) Upwards compatibility is a very minor issue for the user. Theses are written only once; there is little or no need to recompile under the successor to TEX after the thesis has been submitted. The same comment goes for publications. It is easy to dream up exceptions to this, but I contend that they are just that, exceptions. (A good counter example is a script accompanying a course. This script will be modified and recompiled every time the course is offered.)

Well, let me dream up another minor exception for you! If you take a look in your local science library you will find several feet of shelf space occupied by the issues of Mathematical Reviews (MR) from just the past year. In fact, every year the American Mathematical Society not only publishes many tens of thousands of pages of books and primary mathematical journals in TEX, it also publishes more tens of thousands of pages of MR. The cost of producing just one year of MR is well in excess of five million dollars, and all of MR going back to 1959 (about one million records) is stored online *in TEX format* in the MathSci database. People all over the world download bibliographic data and reviews from MathSci and use TEX software to preview or print it. Many others spend hundreds of dollars per year to lease two CD-ROMs with the last ten years of MathSci. Obviously the AMS is unlikely to agree with the above assessment of the importance of compatibility. In fact they are certain to protect their investment in MathSci by making sure that the retrieval system they have invested in so heavily does not break. And they have a

powerful means to protect that investment — with Knuth's blessing, they own the trademark on the TEX name and logo, and will not let it be used for a system that does not pass the trip-test.

TEX as a Front End

Early in the NTS-L discussion there was some discussion concerning extending TEX so it could flow text around pictures, and have other sophisticated facilities of page layout programs such as PageMaker or QuarkXPress. This quickly died out, I think because most people on the list had thought enough about such matters to realize that typesetting and page layout are almost orthogonal activities. The ability of TEX to break text into lines, paragraphs, and pages is aimed at producing printed pages consisting mainly of text for books and journals. Of course, such pages frequently do need diagrams, pictures, and other graphic elements. But these usually fit neatly inside captioned boxes, with no need to have text flow around them, and we have already discussed making such extensions to TEX. The page layout programs, on the other hand, are designed with the quite different purpose of producing illustrated magazines, newsletters, and newspapers. These are documents in which the graphics often outweighs the text, and in which each page can have a complex, and different pattern of text and pictures. Building such pages is an interactive process best handled with a WYSIWYG interface. The good page layout programs often have only quite limited word-processing facilities built in, because the proper way to use them is *not* for creating either text or graphics, but rather to organize into pages text and graphics imported from other programs.

But this brings up an interesting point. To what extent would it be possible to import text typeset by TEX into a page layout program? Certainly this would not be easy! The way TEX freezes the shape of a paragraph, once it has created it, is quite different from the way a normal word processor works, so one would probably have to create a special page layout program, one that understood TEX's data structures and could have an interactive dialog with TEX during the layout process. This would be a tough but worthy undertaking.

TEX as a Programming Language

Many contributors to NTS-L have complained that the TEX programming language is terrible. In its favor one should point out that it is Turing effective — and so just as powerful as say C or Pascal — and it is the programmability provided by this macro language that gives TEX its remarkable flexibility and survivability. However, there is no denying that, while TEX macros may indeed always behave exactly the way (a careful reading of) the TEXbook says they will, it often takes a lot of study for a non-wizard to find the features responsible for a macro behaving the crazy way it does, rather than the way that was intended. Still, most TEX users do learn easily enough to write simple substitution macros or even special purpose macros with parameters. The real problems arise when one tries to write a complex package of general purpose macros for others to use in an unknown environment. One can take the attitude that this activity is simply intrinsically difficult, and should be left to the experts, but it seems to me that those complaining have a good point. Someone who has learned to program in a standard programming language should not have to learn another whole new system of programming; they should be able to use the familiar syntactic and semantic features that they are used to for programming TEX. Since changing the TEX macro language would introduce the worst kind of compatibility problems, some other solution is called for. One that comes to mind is to write a "compiler" whose source language would be some sort of high-level, ALGOL-like language, with all the usual features such as strongly typed variables and scoping rules, and whose target language would be the TEX macro language. Creating such a compiler would not be an easy task, but it would constitute another important application of Knuth's principle of keeping TEX itself a fixed point while making "changes" to the TEX system by creating new front ends.

Changing the Fixed Point

I would be a lot happier if I could stop at this point and conclude that there is no need for any changes to the TEX code itself — that all of TEX's perceived problems can be solved by creating the appropriate front and back ends. For the overwhelming majority of TEX users this is in fact the case. If one is willing to put up with occasionally having TEX fall just short of perfection, or if one doesn't mind making up for these lapses on TEX's part by doing some careful manual tuning (my own approach), then the current TEX is all one will ever need. But for those who take seriously Knuth's goal of not compromising on quality, and moreover insist on a system that permits them to automate excellence, a

very good case has been made that TEX has several serious deficiencies hard-wired into it.

Frank Mittelbach made this point very cogently and convincingly in his presentation "E-TEX: Guidelines for future TEX" at the 1990 TUG meeting (published in *TUGboat* vol. 11 no. 3, September 1990). And Michael Downes amplified and extended Mittelbach's comments in a message he sent to the tex-euro mail list, February 20, 1992, in response to an announcement by Joachim Lammarsch of the intent of Dante e.V. to set up a working group on "Future Developments of TEX". Downes posted a copy of that message to NTS-L on June 2, 1992, and I see no need to repeat either of their remarks here.

Instead I would like to suggest a mechanism to permit necessary changes to be made to TEX code and still maintain compatibility in the sense described above. The idea is both simple and obvious. When NTS starts up it will be ordinary TEX. However if the first string of characters in the source is, let us say, "`\VERSION=NTS`" then the TEX code will be rolled out of RAM and replaced with NTS code.

But how are we going to get from TEX to NTS? My own preference would be to take a gradual approach, analyzing the problems that have been pointed out in TEX into families of related problems, each reasonably independent of the others, and then tackling these families one by one in stages, from easiest to hardest, starting from the original TEX sources and gradually perturbing them. In this way NTS could evolve in a controlled way from the current version of TEX through a sequence of versions, each compatible with standard TEX, each new version curing one more of the difficulties that Mittelbach, Downes and others have pointed out, and each being carefully tested before going on to the next stage. I know this may seem like a dull and pedestrian way to go about things, particularly to those wishing to strike out boldly in new directions. But I think it has the a very good chance of success. It will not demand many resources to get started so it stands a reasonable chance of getting off the ground. And once the first step is taken, well as the saying goes, nothing succeeds like success.

Summary

Let me now summarize my major points and suggestions:

- Many of the problems and "missing features" in the TEX system that have been discussed in NTS-L are not really deficiencies of TEX, but rather features omitted as a consequence of Knuth's decision to limit the functionality of TEX, in order to make it stable and transportable. Many of these problems have been solved in a quite satisfactory manner on one or more platforms by coupling TEX with the appropriate front or back end. What remains is to solve these problems in a manner that preserves transportability of TEX sources, and the way to do this is to specify standard file formats and other data strucures, and a standard `\special` syntax for instructing TEX to interact with them.
- To carry out the above, TUG should appoint a "Committee on TEX Standards". This committee should have the overall responsibility for deciding what types of standards are important to insure that important front and back ends for TEX can be built in a way that is platform independent, and it should appoint committees of experts to promulgate and maintain these various standards.
- Nevertheless, an excellent case has been made that certain specific features of TEX's primitives and coding make it nearly impossible to automate certain functions required to attain one of Knuth's goals for TEX, production of "the finest quality that had ever been achieved in printed documents". While most users may never feel the need for the subtle touches that make the difference between typesetting that is merely excellent, and typesetting that is "the finest quality", for those that do a follow-on to TEX, NTS, should be developed.
- NTS should be backward compatible with source files from the current version of TEX. This means that it should default to a "compatibility mode" that would pass the trip-test, and that any new features that might introduce incompatibilities should have to be "turned on" by the user.
- NTS should be developed in a sequence of versions, starting with TEX and curing its problems one at a time.

Postscript

As indicated above, I believe it is possible for a group to design and implement *ab ovo* a completely new and state of the art typesetting system — a "TEX for the Twenty-first Century" to use Philip Taylor's words. As explained above, I also believe that such a system could be implemented in a way that would keep it functionally compatible with the current TEX system. But, before getting started on

such a massive project, ample consideration should first be given to some prior considerations:

- Don't forget what a monumental task the creation of TEX was, and remember that its author is a totally exceptional individual. He is not only a great computer scientist who happens to love and understand high quality typography, he is also, fortunately, an incredibly good programmer — and finally he has unmatched *Sitzfleisch*. Whole work groups of system analysts and programmers could easily have failed in the same task — and if they had succeeded they would probably have taken longer to create a buggy program that runs on a single platform. *And they certainly would not have put the code in the Public Domain!*
- Knuth is a tenured Full Professor at Stanford. While he was designing TEX and writing the code, he had NSF grant support that not only provided him with the time and equipment he needed, but also supported a team of devoted and brilliant graduate students who did an enormous amount of work helping design and write the large quantity of ancillary software needed to make the TEX system work.
- So, consider this question: Where will the resources come from for what will have to be at least an equally massive effort? And will the provider of those resources be willing, at the end of the project, to put the fruits of all this effort in the Public Domain? I consider this point particularly important. I think it is accepted that it is the combination of the quality and the PD status of the TEX code that have been the two principal factors responsible for its remarkable and unique universality. I doubt that any system that is not PD would have much chance of weaning away a sufficient number of TEX users to make all the effort worthwhile.
- Finally, don't repeat the sad history of ALGOL 68! The ALGOL 60 programming language was a gem. True, it had its flaws, but these were well-known and understood, and I think all of us ALGOL lovers assumed that the ALGOL 68 design committee was going to polish that gem for us and remove the flaws. Instead they decided to start over from scratch and came up with a language that nobody understood, loved, or used. And that spelled the doom of poor old ALGOL — who was going to maintain an ALGOL 60 compiler once ALGOL 68 was "on the way"? Needless to say, even a botched NTS isn't going to kill TEX, but it would be sad to waste all that time and effort — and a great opportunity.

Addendum

A number of people responded to my posting — some by email directly to me, and others by a posting of their own to NTS-L. I would like to thank all who took the trouble to reply, but for reasons of space I will mention here only a couple of replies that bear most directly on my previous remarks.

I would particularly like to thank Nelson Beebe for pointing out that several of the front and back ends I was wishing for either already exist or are in the works. First, and perhaps most important, Nelson himself has made a proposal for a standardized syntax for `\specials` that he has submitted to the TUG DVI Committee, and this will appear shortly in *TUGboat*. Second, Nelson reminded me of an article by Luigi Semenzato and Edward Wang in the November 1991 issue of *TUGboat*. This describes a LISP front end for TEX macro writing, of just the sort I was calling for in the section "TEX as a Programming Language". (But I'd still like to see one based on an ALGOL family syntax!) And finally he pointed out Graham Asher's article "Inside Type & Set" in the April 1992 issue of *TUGboat*, describing a program that does page makeup with lines and paragraphs typeset using TEX code.

Larry Siebenmann sent me a long list of interesting comments, however I will not mention them here since I hope and expect he will himself write something on these matters in these pages.

◇ Richard Palais
Department of Mathematics
Brandeis University
Waltham, Massachusetts 02254
palais@binah.cc.brandeis.edu