NTS: The Future of TeX?

Philip Taylor
The Computer Centre, Royal Holloway and Bedford New College,
University of London, Egham Hill, Egham, Surrey, United Kingdom.
Internet: P.Taylor@Vax.Rhbnc.Ac.Uk

Abstract

Opinions on "the future of TeX" cover the entire spectrum, ranging from the definitive statement by Knuth — "My work on developing TeX ... has come to an end" — to proposals that TeX be completely re-written. In this paper, an intermediate position is taken, based on the fundamental premise that any successor to TeX must be 100% backward-compatible, both in terms of its behaviour when presented with a non-extended TeX document, and in terms of its implementation through the medium of WEB. A mechanism is proposed whereby extensions to TeX may be selectively enabled, and a further mechanism proposed which would enable conforming documents to determine which extensions, if any, are supported by a particular implementation. Finally, a proposal is made for an initial extension to TeX which would have implementation-specific dependencies, and mechanisms are discussed whereby access to such extensions could take place in a controlled manner through the use of implementation-independent and implementation-specific components of a macro library.

Introduction

Discussions on "The Future of TeX", both published and via the medium of e-mail/news-based lists, shew an enormous diversity of opinion: some would argue that Knuth's definitive statement that (paraphrased) "TeX is complete" leaves nothing further to be said, whilst others have advocated that TeX be entirely re-written, either *as* a procedural language or *in* a list-based language; in an earlier paper, I have myself suggested that one possible future derivative of TeX might be entirely window-based, allowing both input and output in textual and graphical formats. But events have occurred within the last eighteen months which have considerably influenced my point of view, and in this paper I present a far more modest proposal: that an extended TeX-based system (hereinafter referred to as *extended-TeX*, or *e-TeX* for short) be developed in a strictly controlled way, retaining not only the present look-and-feel of TeX but *guaranteeing* 100% backward compatibility with whatever truncation of the decimal expansion of π represents the most recent canonical version of TeX.

The reason for this change of heart dates from the 1992 AGM of DANTE (the German-speaking TeX Users' Group), to which I had the honour to be invited. There, Joachim Lammarsch, President of DANTE, announced the formation of a working group to investigate future developments based on TeX: the group was to be called the NTS group (for 'New Typesetting System'), to avoid any suggestion that it was TeX itself whose future was being considered, such activity being the sole remit of TeX's author and creator, Professor Donald E. Knuth. The group was to be chaired by Dr Rainer Schöpf, and included representatives of both DANTE and UK-TUG; Joachim emphasised that the group, although created under the ægis of DANTE, was to be a truly international body. An electronic mailing list, NTS-L, was announced, and participation was invited from any- and everyone throughout the world who wished to contribute to the discussion.

NTS-L proved a mixed success: it certainly attracted considerable interest, and in the early days discussion was almost non-stop; but it proved extraordinarily difficult to *focus* the discussion, and (like so many e-mail lists) the discussions frequently went off at a tangent... But then, after the initial burst of enthusiasm, discussions started to tail off; and as the time of the 1993 DANTE AGM came near, the only questions being asked on the list were "Is NTS dead?".

At about the same time, I was approached by Rainer, acting on behalf of Joachim who was indisposed, to ask if I would be interested in chairing the NTS group; Rainer felt (quite reasonably) that he had more than enough on his plate with his central involvement in the LaTeX-3 project (not to mention his real, paid, work!), and that he simply hadn't the time available to make NTS the success which it deserved to be. Needless to say, I viewed this offer with mixed feelings: it was a very great honour to be asked to chair such a group, but at the same time the group had already been in existence for nearly a year, and had apparently achieved nothing (in fact, it had never even met); would I be able not only to breath life back into the by now moribund project, but also go further and actually oversee the production of a realisation of NTS?

The more I thought about the problems, the more I became convinced that the key to success lay through simplicity: if NTS was ever to be more than a pipe-dream, a wish-fulfillment fantasy for frustrated TeXxies, then it had to be achievable with finite resources and in finite time; and if the results were to be acceptable to the vast number of TeX users throughout the world (a number which has been estimated to be at least 100 000), then it had to be completely backwards-compatible with TeX. Once I was convinced that I knew what *had* to be achieved, I also began to believe that it might be possible to accomplish it. And so, with some trepidation, I indicated to Joachim and Rainer that I would be honoured to accept their trust and confidence; I would agree to take over the NTS project.

But the road to damnation is paved with good intentions; and no sooner had I returned from the 1993 DANTE AGM, having once again had the honour to be invited to participate, than the spectre of TUG'93 began to loom large on the horizon; and the more work I put into its organisation, the more work it seemed to take. I was not alone — I willingly acknowledge the incredible amount of hard work put in by the entire TUG'93 committee, and in particular by Sebastian Rahtz — but the organisation of a multi-national conference, scheduled to take place at a University some 130 miles from one's own, is a mammoth undertaking, and one that leaves little time for anything, apart from one's normal, regular, duties. And, in particular, it left almost no time for the NTS project, to my considerable mortification and regret. But, by the time this paper appears in print, TUG'93 will be a reality, and, I hope, life will have sufficiently returned to normal that I will be able to devote the amount of time to NTS that the project so richly deserves.

But enough of the background: what matters today, and to this conference, is not how I as an individual partition my time; but rather what specific proposals I have for "The Future of TeX". I propose to discuss these under three main headings: *compatibility*, *extensions*, and *specifics*; under *compatibility* will be discussed compatibility both at the source (WEB) level and at the user (TeX) level; under *extensions* will be discussed a possible mechanism whereby extensions can be selectively enabled under user control, and a mechanism whereby an *e-TeX* conformant program can interrogate its environment in order to determine which extensions, if any, have been enabled; and under *specifics* will be discussed one possible extension to TeX which has been widely discussed and which will, in my opinion, provide the key to many other apparent extensions whilst in practice requiring only the minimum of additional *e-TeX* primitives. I must emphasise at this point that what follows are purely *personal* suggestions: they do not purport to reflect NTS policy or philosophy, and must be subjected to the same rigorous evaluation as any other formal proposal(s) for the NTS project.

Compatibility

What is compatibility? Ask a TeX user, and he or she will reply something like "unvarying behaviour: given a TeX document which I wrote in 1986, a compatible system will be one that continues to process that document, *without change*, and to produce results *identical* to those which I got in 1986". Ask a TeX implementor, on the other hand, and he or she will reply "transparency at the WEAVE and TANGLE levels; if *e-TeX* is truly compatible with TeX, then I should be able to use *exactly* the same changefile as I use with canonical TeX, and get a working, reliable, *e-TeX* as a result". Two overlapping sets of people; two totally different answers. And yet, if *e-TeX* is to be generally acceptable, and even more important, generally accepted, we have to satisfy both sets: the users, because without them the project will be still-born, and the implementors, because without them, parturition won't even occur! How, then, can we satisfy both sets? The answer, I believe, lies in the question itself: *e-TeX* must *be* TeX; it must use, as its primary source, the latest version of TEX.WEB, and it must make changes to TEX.WEB in a strictly controlled way, through the standard medium of a changefile; that is, *e-TeX* must be representable as a series of finite changes to standard TEX.WEB.

But if *e-TeX* is to be a changefile, how is the implementor to apply his or her own changefile as well? Fortunately there are several ways of accomplishing this: the KNIT system, developed by

Wolfgang Appelt and Karin Horn; the `TIE` system, developed by Dr Klaus Guntermann and Wolfgang Rülling; and the `PATCH-WEB` system, developed by Peter Breitenlohner. Each of these will, in varying ways, allow two or more change files to be applied to a single WEB source; thus the (system independent) changes which convert `TeX.Web` into `e-TeX.Web` can be implemented in one changefile, and the (system dependent) changes which implement *e-TEX* for a particular combination of hardware and operating system can be kept quite separate.

But this does not quite accomplish our original aim: to allow the implementor to use *exactly* the same change file for *e-TEX* as for TEX; in order to accomplish this, the changes effected by `e-TeX.ch` must be orthogonal to (i.e., independent of) the changes effected by `<implementation>.ch`; without a knowledge of the exact changes effected by each implementor's version of `<implementation>.ch`, such orthogonality cannot be guaranteed. None the less, provided that the changes effected by `e-TeX.ch` affect only the system-independent parts of `TeX.Web`, such orthogonality is probable, if not guaranteed; unfortunately, as we shall see, some proposals for *e-TEX* are guaranteed to conflict with this requirement.

So much for compatibility as far as implementors are concerned: what about compatibility from the point of view of the user? Here, at least, we are on safer ground: the users' requirements for compatibility are (let us remind ourselves) "unvarying behaviour: given a TEX document which was written in (say) 1986, a compatible system will be one that continues to process that document, *without change*, and to produce results *identical* to those which were achieved in (say) 1986". Thus (and here I intentionally stress an entire sentence) *the default behaviour of e-TEX must be **identical** to that of TEX, given a TEX-compatible document to process.* What does this imply, for *e-TEX*? I suggest two things:

- Every primitive defined by TEX shall have exactly the same syntax and semantics in *e-TEX*, and
- There shall be no new primitives (because existing TEX programs may depend on `\ifx \foo \undefined` yielding `-true-` for all currently undefined TEX primitives).

(gurus will appreciate that this is a considerable simplification of the truth, but I hope they will allow me this in the interests of clarity; clearly other constraints must obtain as well, for example identical semantics for category codes, and no additions/deletions to the list of context-dependent keywords).

Extensions

But given this as a definition of *e-TEX*, have we not backed ourselves into a black hole, from which there is no escape? How, if there are no new primitives, and all existing primitives are to retain their identical syntax/semantics, are we to access any of the *e-TEX*-specific extensions? I propose that we implement one, and only one, change between the behaviour of TEX and the behaviour of *e-TEX*: *if, on the command-line which invokes e-TeX, two consecutive ampersands occur, then the string following the second ampersand shall be interpreted as an **extension (file) specification***, in a manner directly analogous to TEX's treatment of a single ampersand at such a point, which is defined to introduce a ***format (file) specification***. Thus there is one infinitesimally small difference between the behaviour of *e-TEX* and TEX: if TEX were to be invoked as "`TeX &&foo myfile`", it would attempt to load a format called `&foo`; *e-TEX*, on the other hand, would attempt to load an *extensions-file* called `foo` — I suggest that the chances of this causing a genuine conflict are vanishingly small.

OK, so we have a possible way out of the black hole: we have a means of specifying an *extensions-file*, but what should go therein, and with what semantics? This is, I suggest, a valid area for further research, but I would propose the following as a possible starting point:

- if `&&<anything>` appears on the command line, then *e-TEX* shall enable one additional primitive, `\enable`;
- *extensions-file* shall commence with a record of the form `\enable {options-list}`;
- *options-list* shall consist of a series of (?comma-delimited?) primitives and brace-delimited token-lists;
- if a given primitive occurs in the options-list to `\enable`, and if a meaning to that primitive is given by (or modified by) *e-TEX*, then henceforth that primitive shall have its *e-TEX*-defined meaning; (and if no such meaning exists, a non-fatal error shall occur);
- if a given token-list occurs in the options-list to `\enable`, and if that token-list has an intrinsic meaning to *e-TEX*, then the effect of that meaning shall be carried out; (by which we allow modifications to the semantics of *e-TEX* without requiring the creation of new, or the modification of existing, primitives; thus `{re-consider partial paragraphs}`, for example, might change *e-TEX*'s behaviour at top-of-page w.r.t. the partial paragraph which remains after

page-breaking; no new primitive is involved, nor are the semantics of any existing primitive changed). If the token-list has no intrinsic meaning to *e-TeX*, a non-fatal error shall occur.

Thus, by modifying only that area of the initialisation code which inspects the command line for a format-specifier, we allow for arbitrary extensions to the syntax and semantics of *e-TeX*. What we next need is a mechanism whereby *e-TeX*-conformant (as opposed to TeX-conformant) programs can determine which extensions, if any, have been enabled; thus a document could ascertain whether it is running in a TeX environment or an *e-TeX* environment, and modify its behaviour to take advantage of facilities which are available in the latter but not in the former.

In order for a program to be able to carry out this check in a manner which will be both TeX and *e-TeX* compatible, it must use TeX-compatible methods to check whether further *e-TeX*-compatible compatibility checks are supported: if we assume that the proposals above are implemented, then there is one reliable way of determining whether we are running (1) under TeX, or under *e-TeX* with no extensions enabled, or (2) under *e-TeX* with (some, as yet undetermined) extensions enabled:

```
\ifx \enable \undefined
    ... pure TeX, or e-TeX with
        no extensions
\else
    ... extended e-TeX
\fi
```

This relies, as does much existing TeX code, on `\undefined` being undefined; perhaps one extension implemented by *e-TeX* might be to render `\undefined` undefinable, just to ensure the integrity of such checks!

Once we are sure we are running under *e-TeX* with extensions enabled, we are in a position to make further environmental enquiries; but to do so will require an *a priori* knowledge of whether the environmental enquiries extensions have been enabled: a chicken-and-egg situation! Thus we need to proceed in a slightly convoluted manner, in order to ensure that we don't trip over our own bootstraps. Let us posit that, in order to enable environmental enquiries, we use something like the following in our extensions-file:

```
\enable {{environmental-enquiries}}
```

Then, in our *e-TeX*-compatible source (having ensured that we are running under *e-TeX* with extensions enabled), we need to be able to write something like:

```
\ifenabled {{environmental-enquiries}}
```

But we can't do this without first checking that `\ifenabled` is defined... Clearly this is becoming very messy (rather like one's first attempt at writing handshaking code for networking; how many times do you have to exchange *are-you-there/yes-i'm-here; are-you-there*s before it's safe to proceed with real data?). Fortunately, in this case at least, the algorithm converges after one further iteration: our TeX-compatible/*e-TeX*-compatible/totally-safe-environment-checking code becomes:

```
\ifx \enable \undefined
    ... pure TeX, or e-TeX with
        no extensions
\else
    \ifx \ifenabled \undefined
        e-TeX without the benefit
        of environmental enquiries
    \else
        ... e-TeX with environmental
            enquiry support
    \fi
\fi
```

(A similar approach could be used if environmental enquiries were implemented through the medium of `\enable {\ifenabled}` rather than `\enable {{environmental-enquiries}}`; it is a philosophical question as to which is the 'cleaner' approach).

One interesting issue, raised by the anonymous reviewer, remains to be resolved: if an *e-TeX* user decides to (a) enable some specific extension(s), whilst leaving others disabled, and (b) to dump a format file, what happens if that format file is loaded with a different set of extensions enabled? I have to confess that the answer to that question is unclear to me, and that an initial investigation suggests that extensions should *only* be permitted during the creation of the format file, not during its use; but that could have implications in the `\disable` functionality elsewhere referred to, and for the moment at least I prefer to leave this as a valid area for further research. Perhaps the whole extension/format area requires unification, and the enabling/disabling of extensions should simply become a part of the rôle of *Ini-e-TeX*.

Specifics

So far, I have concentrated on a generic approach to the question of *e-TeX*, and quite intentionally proposed only an absolute minimum of differences between TeX and *e-TeX*; but once the framework is in place, we are in a position to consider what features are genuinely lacking in TeX. This is a *very* contentious area, and one in which it necessary to tread warily, very contentious area, and one in which it is necessary to tread warily, particularly in view of Professor Knuth's willingness to regard TEX as complete: after all, if the creator and author of TEX sees no need for further enhancements, who are we, as mere users, to question his decision? Fortunately there is both precedent and guidelines; at the end of `TeX.Bug`, one finds the following text:

> "My last will and testament for TEX is that no further changes be made under any circumstances. Improved systems should not be called simply 'TEX'; that name, unqualified, should refer only to the program for which I have taken personal responsibility. — Don Knuth
>
> **Possibly nice ideas that will not be implemented.**
>
> - classes of marks analogous to classes of insertions;
> - `\showcontext` to show the current location without stopping for error;
> - `\show` commands to be less like errors;
> - `\everyeof` to insert tokens before an `\input` file ends (strange example: `\everyeof {\noexpand}` will allow things like `\xdef \a {\input foo}`!)
> - generalize `\leftskip` and `\rightskip` to token lists (problems with displayed math then);
> - generalize `\widowline` and `\clubline` to go further into a paragraph;
> - `\lastbox` to remove and box a charnode if one is there;
> - `\posttolerance` for third pass of line breaking.
>
> **Bad ideas that will not be implemented.**
>
> - several people want to be able to remove arbitrary elements of lists, but that must never be done because some of those elements (e.g., kerns for accents) depend on floating point arithmetic;
> - if anybody wants letter spacing desperately they should put it in their own private version (e.g., generalize the hpack routine) and NOT call it TEX."

Thus we have clear evidence that there are some possible extensions to TEX which Professor Knuth does not completely deprecate; he may not wish them to be incorporated in TEX, but I think we may safely assume that he would have no violent objection to their being considered for *e-TEX*.

But there is another source of information, too, in which he makes it plain that there is an area of TEX in which an extension would be deemed legitimate, and here (very surprisingly, in my opinion), he has suggested that the semantics of an existing TEX primitive could legitimately be modified *as part of the system-dependent changes to TEX itself*, without violating his rules for the (non-)modification of TEX. This arose during discussions between himself and others including (I believe) Karl Berry and Frank Mittelbach concerning the implementation of an interface to the operating system; Don suggested that it would be legitimate to extend the semantics of `\write` such that if the stream number were out of range (perhaps a specific instance of 'out-of-range', for safety, e.g., `\write 18 {...}`), then the parameter to that `\write` could be passed to the operating system for interpretation, and the results made available to TEX in a manner still to be defined.

When I first learned of this, I was horrified (and I still am...); not only is this a proposal to abuse `\write` for a purpose for which it was never intended (and in a manner which could wreak havoc on any program extant which uses `\write 18 {...}` to send a message to the console, which it is perfectly entitled to do (cf. *The TEXbook*, pp. 226 & 280)), it is a proposal to extend `\write` in a system-dependent manner. I found (and find) it hard to believe that Don could have acceded to these suggestions.

But these proposals received a wide airing, and were met by quite a degree of enthusiasm; not because people wanted to abuse `\write`, but because they were desperate for an interface to the operating system. Such an interface grants TEX incredible flexibility: one can sort indices, check for `tfm` files, in fact do anything of which the host operating system is capable, all from within TEX, and in such a way that the results of the operation become available to TEX, either for further calculation or for typesetting. Of course, there were also (very sound) arguments against: "what if the program performs a `$ delete [*...]*.*;* /nolog /noconfirm`?" was asked over and over again. (The command deletes all files to which the user has delete access, regardless of directory or owner, and recurses over the whole file system under VAX/VMS; there are equally powerful and unpleasant commands for most other operating systems.) What indeed? But if this feature were implemented through the abuse of `\write`, there would not necessarily be any provision for disabling it; and users would become legitimately paranoid, scanning each and every imported TEX document for the slightest trace of a system call, in the fear that computer viruses had migrated into their (previously safe) world of TEX.

Of course, TEX has never been truly safe; perhaps we are fortunate that the challenge of writing computer viruses appears not to be of any interest to those who are also capable of writing TEX (or perhaps those who have the intelligence to prefer, and to write in, TEX, have by definition the intelligence to see that writing viruses is a distinctly anti-social activity, and to refrain therefrom). I will not elaborate on this point, just in case it falls into the wrong hands...

And so, I propose that one of the first extensions to *e-TEX* which the NTS project should consider is the implementation, in a clean and controlled way, of a genuine `\system` primitive; implemented through the medium of `\enable`, it would be up to each individual user whether or not to allow of its use, sacrificing security for sophistication or preferring power and performance over paranoia. We might posit, too, a `\disable` primitive, so that even if the system manager had installed *e-TEX* with `\system` enabled, an individual user could choose to disable it once again (there are complications involved in this which I do not propose further to discuss here).

And once we have a `\system` primitive, we can then implement, through its medium, a whole raft of further extensions which have from time to time been requested by the TEX community (the following are taken almost verbatim from a submission by Mike Piff):

- Delete a file;
- Rename a file;
- Copy a file;
- Create a directory;
- Remove a directory;
- Change directory;
- Spawn a sub-process.

But these tasks are, by their very nature, incredibly operating-system specific; whilst I might type `$ delete foo.bar;`, another might write `%rm foo.bar` (I *hope* I have the latter syntax correct...); and surely one of the most important reasons for the use of TEX is its machine-independence: documents behave *identically* when typeset on my IBM PS/2 and on the College's VAX/VMS 6430. But if *e-TEX* users were to start hard-coding `\system {$ delete foo.bar;}` into their *e-TEX* files, machine-independence would fly out of the window; and *e-TEX* would have sown the seeds of its own destruction...

And so, I propose that for each *e-TEX* implementation, there shall exist a macro library which will be composed of two parts: a generic component, created by the NTS team, which implements in a system-independent manner each interaction with the operating system which is deemed 'appropriate' (whatever that means) for use by *e-TEX*; and a specific component, created by each implementor of *e-TEX*, which maps the generic command to the system-specific syntax and semantics of the `\system` primitive. The macro library is by definition easily extensible: if the *e-TEX* community decides that it needs a `\sys^delete_file` macro, and no such macro exists, it will be very straight-forward to implement: no re-compilation of *e-TEX* will be required.

Clearly there is an enormous amount of further work to be done: how, for example, is the `\system` primitive to return its status and results? What is to happen if `\system` spawns one or more asynchronous activities? Which of Don's "Possibly nice ideas" should be integrated into *e-TEX* at an early stage? How about the 'Skyline' question, or `\reconsiderparagraphs`? Should *e-TEX* be based, *ab initio*, on TeX--XeT? How are the NTS team to liaise with the TWG-MLC group, and with other interested parties? How are we to ensure that practising typographers, designers, and compositors are able to contribute their invaluable ideas and skills to the development of *e-TEX*? Some of these questions will, I hope, be debated openly and fully on NTS-L; others must be answered by the NTS team themselves (and here I have to confess that because of the pressures of this conference, the membership of that team is still in a state of flux). What matters most, at least to me, is that the philosophy and paradigms which characterise TEX are perpetuated and preserved for future generations: we have, in TEX, something very precious — the creation of a single person, Professor Knuth, which has had a profound effect on the professional lives of thousands, if not tens of thousands, of people; if we are to seek to extend that creation, then we must do so in a way which is entirely faithful to the ideals and intentions of its creator. I truly hope that we are up to that task.