

Opinion and Analysis \TeX 's Interface Challenges

Arthur Ogawa
 \TeX Consultants

If you work with \TeX as frequently as I do, you no doubt have a great appreciation for its many strengths as a vehicle for the production of beautifully formatted documents. Yet you must also admit that \TeX 's market share (its *usership*) is, to put it mildly, minor: as \TeX users, we are islands (some would say, of excellence) in a vast sea of users of word processor and page layout applications.

Why has \TeX failed to attract larger usership? In my analysis, it is because \TeX fails to provide ease of use in certain crucial areas, all of them interfaces.

1 A quarter-century of computer typesetting

Looking back to 1978, one can discern several enduring lines of software development. One was \TeX 78, soon rewritten and now known as \TeX . Another is word processing, epitomized by **WordStar** and its successors. A third is visual page layout software, also known as desktop publishing, originating in **Ready, Set, Go** for Macintosh. And a fourth is the graphical user interface itself, tracing back to Xerox PARC's **SmallTalk** and carried forward by Apple's Mac OS and others.

Since those early days, the typical user's interface to the computer has changed completely, from a terminal-based dialog to a non-modal, point-and-click interaction, and the system manager's interface has been likewise simplified.

At the same time, word processors like **Microsoft Word** and **Corel WordPerfect** have achieved dominance in the field of office documents, and WYSIWYG page

layout applications like [QuarkXPress](#) and [Adobe InDesign](#) (née Aldus PageMaker) have done likewise in the area of highly formatted documents.

By contrast, for most people using \TeX , its interface is a throwback to times long past: marking up a document with codes, specifying format with a computer programming language, lacking immediate feedback from changes to document or format, and installing software components that require understanding one's file system in intimate detail.

\TeX offers excellent functionality as both a word processor and page layout application. It is possible and even practical for a business to use \TeX for all its office documents that are to be exchanged or eventually reduced to paper (or to static electronic form like PDF). And don't forget \TeX 's peerless approach to math typesetting, which its author Donald Knuth characterized as an expert system.

However, its widespread and enduring usage notwithstanding, \TeX has been marginalized. The reason for this development is, I believe, entirely a matter of interface. It is certainly not one of integrity of the software (freedom from bugs) nor is it a matter of power of formatting functionality. Even in the area of document interchange (this is a further interface), \TeX 's utility stands out because documents are coded in plain text. Yet by any sort of measure, \TeX is today a minor part of the picture, and I believe, based on contact with many in the commercial sector, that the barriers to adoption are the aforementioned interfaces: document maintenance, document processing, document design, software installation and system interface.

Knuth had a [concept](#) that there would exist computer programs serving as front ends to \TeX . In a sense, \TeX 's lack of appeal to the general computer user is due not to its own shortcomings, but to the failure to create compellingly useable interfaces for \TeX . Let's look at these interfaces in more detail.

2 \TeX 's shortcomings at the interface

2.1 Document creation and maintenance

The norm today for document creation and maintenance is for the user to view the formatted document and manipulate it directly. This paradigm is popularly called [WYSIWYG](#) (What You See Is What You Get), but a more precise description for the user interface is "direct manipulation".

T_EX installations, by contrast, require the user to maintain the source with all its markup exposed. This limits its adoption to people who are comfortable with this intermixture of code and content, who are able to work at a more abstract level, and who are willing to take the trouble to type in codes accurately.

Two exceptions are [Mackichan Software's Scientific Word](#) and the [Open Source Lyx Document Processor](#). Scientific Word for Windows is a visual (point-and-click) means of creating and maintaining a L^AT_EX document. The user works with a document that bears approximate visual formatting indicating the L^AT_EX document structure (this has been lightheartedly dubbed a "QuasiWYG" view). Lyx, running on UNIX (including Mac OS X) and Windows/Cygwin, works similarly. Both are limited to L^AT_EX documents, and both use an underlying T_EX installation.

These sorts of systems make maintaining T_EX documents easier because they facilitate selecting and inserting markup and because they show the structure of the document simply and visually, suppressing the codes themselves.

As is well known, any user interface for maintaining a descriptively marked up document can be used in conjunction with T_EX as a back-end formatter. So where are such interfaces?

The deservedly popular XML descriptive markup format finds limited support in [Adobe Framemaker](#). And [BlastRadius XMetaL](#) and [Arbortext](#) also have a number of offerings including XML editors, but these companies market chiefly to enterprise customers, and their products, priced accordingly, are outside the reach of the mass of users.

Even Microsoft has entered the picture with its [Word 2003 support](#) for WordML and other XML schemas (see, for example, the [PC Magazine article](#) of 2004-12-28 by Richard V. Dragan). Likewise, Corel WordPerfect now comes with a built-in XML editor.

Meanwhile XML-based editors for mathematics notation have by no means flooded the market, although [Design Science's MathType](#) (a stripped-down version, Equation Editor, is bundled with Microsoft Word) can export to T_EX, L^AT_EX, and the XML scheme MathML.

2.2 Determining the formatting of the document

Word processors and page layout applications provide a control-panel based interface for determining the formatting of a paragraph or other block of text. This

approach is much more convenient than anything provided for $\text{T}_{\text{E}}\text{X}$, with the exception of Scientific Word, which has a Style Editor interface.

In the XML arena, a visual editor for an **XSL Formatting Object** would provide the desired user interface, but I do not know of one.

2.3 Prompt preview of formatted document

Promptness of visual feedback is a familiar user interface issue. If a direct manipulation interface to the document is not in the picture, the best one could hope for would be a very prompt preview of the typeset page.

Blue Sky Research's Textures, a commercial $\text{T}_{\text{E}}\text{X}$ implementation for Mac OS Classic, provided this function in a feature dubbed “Flash Mode”—along with a way of transitioning easily between the coded $\text{T}_{\text{E}}\text{X}$ view of the document and the corresponding Page Preview—“Synchronicity”. The utility of these features was astounding, yet few other versions of $\text{T}_{\text{E}}\text{X}$, commercial or otherwise, come at all close.

Because of the nature of pdf $\text{T}_{\text{E}}\text{X}$, prompt page preview is a virtual impossibility for systems using this engine, and the pdfsync mechanism does not come even close to the precision of Synchronicity. Yet Textures, having so far failed to make the transition to the Mac OS X native (Cocoa) world, is regrettably in danger of becoming irrelevant.

Jonathan Fine's **$\text{T}_{\text{E}}\text{X}$ Daemon** does provide a basis for achieving the equivalent of Flash Mode and Synchronicity under any Web2C-based $\text{T}_{\text{E}}\text{X}$ engine (in effect supporting these features on all platforms), and it deserves further development and support.

2.4 Installing and updating $\text{T}_{\text{E}}\text{X}$'s executables and runtime files

The computer operating systems of today are more reliable, powerful, and far more complex than those at the dawn of personal computing, and applications are similarly more complex. Yet the average user is effectively guided through the installation of system software and productivity applications with ease. And when updates to the system or applications are required, this too is carried out with point-and-click simplicity. Indeed, it is remarkable that Apple Computer has put the installation and maintenance of a powerful UNIX operating system,

Mac OS X Server, within the capacity of a reasonably conscientious but technically untrained person.

Current $\text{T}_{\text{E}}\text{X}$ packagings, by contrast, still require some degree of sophistication, and yet more is required if one is maintaining an installation for a number of users or one with local extensions, packages, or fonts.

Commercial packagings of $\text{T}_{\text{E}}\text{X}$ tend to be easier to install than the free versions, but in no case do we have the ease of Apple's Software Update, Microsoft's Windows Update, or Red Hat's rpms, although [i-Installer](#) and [MikTeX](#) offer a simple networked updater.

But what is further needed is a way to easily augment a $\text{T}_{\text{E}}\text{X}$ installation with a needed new resource, say a $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ document class or a font. The package in question is likely available on [CTAN](#) (the Comprehensive $\text{T}_{\text{E}}\text{X}$ Archive Network), but a truly user-friendly system would detect the dependencies of a document and automatically download and install the required new components.

2.5 Host system's fonts and character representation

Today's computers come fitted out with high-quality scalable fonts that all applications can use.

Yet $\text{T}_{\text{E}}\text{X}$'s free fonts are not generally available in a form native to the host operating system (say, Computer Modern in OpenType format). And native fonts are not generally available for use in $\text{T}_{\text{E}}\text{X}$ typesetting. Also, with Unicode as the standard interface to the user and keyboard, its support in $\text{T}_{\text{E}}\text{X}$ ought to be the standard today.

The picture here may be changing in the near future: the 2004 distribution of [\$\text{T}_{\text{E}}\text{X}\$ Live](#) incorporates Petr Olsak's [\$\text{encTeX}\$](#) extension to $\text{T}_{\text{E}}\text{X}$, which allows UTF-8 support. However, a full system must provide for all of the languages the operating system supports (Mac OS X has over a dozen), and multibyte characters (needed for Chinese and Japanese, for instance) raise the bar further.

Also, Jonathan Kew's [\$\text{XeTeX}\$](#) (on Mac OS X) provides support for system (OpenType) fonts, and `pdftex` may follow suit shortly. May we see support for TrueType fonts, common on Microsoft Windows systems, in the near future?

3 Moving toward a solution

One reason for the difficulty in resolving the user interface issues is the cost of software development. Creating and maintaining software with a sensible and capable user interface is a much bigger effort than writing [T_EX the program](#) (itself not easy), and involves far more executable code. But the problem goes further: one would have to do the job for at least three software platforms—Microsoft Windows, Mac OS X, and i86 Linux—in order to achieve success. And there is no development environment I know of that would allow one to do the job in all three “at once”.

At the same time, open-source software development, exemplified by Linux, Lyx, T_EXLive, and by T_EX itself, points one possible way to the future.

And there are certain things that can be done right away to prepare for the development of better interfaces.

3.1 Input parser

On the document side, we need a T_EX macro package that parses a purely descriptively marked up document (such as XML or the descriptive subset of L^AT_EX) for the benefit of a plug-in formatter. In no respect, though, should this macro package make any assumptions about what the markup schema is, what documents are to be considered valid, or how the document is to be formatted. A 2000 [article by Michel Goossens](#) discusses this side of the picture.

3.2 Typespec parser

On the formatting side, we need a T_EX macro package that interfaces to a descriptive representation of the typographic specification of an editorial element (font, size, leading, measure, justification, etc.). Such representation must be purely descriptively coded—no procedural code involved. A collection of such components would be the complete expression of a typographic specification for an entire document class.

The first of these macro packages would endow T_EX with the ability to input any descriptively coded document for further processing, with the expectation that some sort of user interface would be the vehicle for the creation and mainte-

nance of such documents. The second would allow T_EX to carry out the formatting of such documents according to a given typographic specification, with the expectation that the collection of typographical representations would be maintained by some appropriate user interface.

A complete typesetting system for a particular document class would entail a statement of the input markup syntax, a collection of typespec descriptions for each editorial element, and a key connecting the former with the latter. A further task would be to create a number of such systems to serve as examples; perhaps reimplementing L^AT_EX's core document classes (article, book, and report) would be a useful starting point.

3.3 A standard for package installation

On the installation side, each functional extension of T_EX (extending its ability to parse or format), each T_EX package, should have with it an installer script that would allow it to be installed over the Internet employing, say, a Web interface, that itself would ensure the integrity and completeness of the T_EX installation in question. The creation of such scripts would have an immediate benefit to those preparing T_EX packagings, such as T_EXLive. It is high time that CTAN promulgate standards for package submission that provide for automatic installation and updating.

3.4 Prompt page output

The popular pdftex engine should return to outputting each page as it is shipped out of T_EX's memory. Whether this is even compatible with the production of a PDF output stream, I do not know. But the alternative, that T_EX will never again provide prompt interaction with the user, is simply not acceptable.

3.5 Character-level source synchronization

The T_EX engine should support a scheme for synchronizing between source and formatted output, improving upon the current pdfsync mechanism. Doing so will have a cost, but for some users the benefit is well worth it. Synchronization is the basis for any future direct-manipulation interface to the T_EX document.

Of course, I am not advocating changing $\text{T}_{\text{E}}\text{X}$, which we know is frozen. Instead, I am suggesting further extensions to $\text{T}_{\text{E}}\text{X}$, in the spirit of $\text{eT}_{\text{E}}\text{X}$ and $\text{pdfT}_{\text{E}}\text{X}$.

3.6 *metafont*-based fonts in system format

A program to turn *metafont*-based fonts into PostScript Type 1 fonts should be morphed into one to turn them into OpenType or TrueType fonts. All of the many fine fonts available as *metafont* source should be made available to all applications running on the host (word processor, page layout, illustration), not just by $\text{T}_{\text{E}}\text{X}$.

Current projects like the Latin Modern fonts and Newmath fonts should think seriously about creating the fonts in this way.

3.7 Full access to system features

Finally, as relates to the system interface, there is nothing standing in the way for the current widely distributed $\text{T}_{\text{E}}\text{X}$ engine, *pdfetex*, to access the full Unicode input capability of the host operating system (extending to multibyte character representation) and the ability to use system fonts.

4 Getting off on the right foot

4.1 Shedding the legacy

Certain efforts have already been made to develop the needed macro packages, but I would argue for a fresh start. In particular, I see little benefit in carrying $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ along, since the goal would be the elimination of procedural markup in the document instance and in the formatting specification, both enduring features of today's $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ and unlikely to be eliminated in future. Also, $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$'s core contains definitions for user-level commands, of little use in a setting where the the document markup might have been, say, XML. Finally, $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ and its required packages are tainted at the deepest level with restrictions about what is and is not proper typographic design. This limitation in flexibility restricts its effectiveness as a foundation for fully generalized typesetting.

4.2 Dividing the problem

The key to success will be maintaining a correct factoring of the system. The core of the formatting package should give access to the complete generality of type composition but make no assumptions about input syntax or design specification. As those in the field of commercial typography well know, a typographic specification will sometimes call for rather unusual formatting. Few typesetting systems are built to implement all of the formatting mandates of a commercial typespec, but $\text{T}_{\text{E}}\text{X}$'s native capabilities come as close as any to doing so. The formatting macro package should not restrict these capabilities.

4.3 Allowing extensions

It might seem quite daunting to consider expressing a complex type specification in a completely descriptive way (that is, exclusive of procedural coding), say something as complex as the title page of a typical book. However, I envision the creation of formatting extensions to the core formatter. An extension could well contain procedural code (even written with $\text{T}_{\text{E}}\text{X}$ primitives only), but it would be parameterized so as to allow its use to be controlled by the aforementioned style editor.

Extensibility of this kind would enable the system to eventually accommodate any formatting task, while still embodying a descriptive specification of the formatting. This aspect in turn enables a user-friendly interface for format specification.

5 Following up

I hope that we may engage in a dialog on how $\text{T}_{\text{E}}\text{X}$'s acceptance in the marketplace may be enhanced by addressing these interface issues and how to focus efforts on their resolution. Interested parties may contact the author or this publication.

In the meantime, let us $\text{T}_{\text{E}}\text{X}$ developers all think about how our efforts fit into the larger scheme to make $\text{T}_{\text{E}}\text{X}$ more usable. So far our remarkable body of work has helped extend $\text{T}_{\text{E}}\text{X}$ from the single-host, static, personal publishing system of one computer scientist to a multilingual, multiscrypt typesetting system for professional-quality publications of all kinds, one that is ported to virtually ev-

ery known computer. Now it is time to step up to the task involving the truly heavy lifting: putting all this power in the hands of the ordinary user.

Acknowledgements

I have benefited from comments by and information from Karl Berry, Lance Carnes, Peter Flynn, Hans Hagen, Steve Peter, Philip Taylor, Christina Thiele, and David Walden, and I have gratefully received the encouragement of Tim Null.

My opinions are based on over 20 years as a T_EX professional answering to the demands of the marketplace and in constant and broad contact with people in the commercial sphere, both those who use T_EX and those who do not. And as one who would like to see more of the latter join the former.