TEX Enslaved

ALAN E. WITTBECKER

Digital Equipment Corporation
110 Spit Brook Road (ZKO 1-2/C14)
Nashua, NH 03062–2698

ABSTRACT

Several documentation systems incorporate TEX as a text formatter, but use a generic markup language as a front-end. This means TEX is hidden from the users, and usually has been modified or extended. This paper discusses the advantages and disadvantages of such systems, as regards international standards, macro packages, and TEX itself.

1. Introduction

TEX, the mathematical typesetting program, has a number of good qualities and unique features. It is extremely precise in its measurements, generous in its use of fonts, sophisticated in its hyphenation, ultra-sophisticated in its display of mathematical characters, and generally device-independent in its output. And, it is now fixed and completed; it is the same for everyone.

But, TEX has limitations. TEX processes text in a straight line. It cannot rotate characters or text, and it has difficulty drawing "freehand" lines. Although it can insert graphics at arbitrary locations, it cannot produce them.

TEX is a general-purpose composition program. It has a relatively large number of commands that make it hard to learn. Many of its concepts, such as zero-width boxes, are not intuitive. TEX is often inconsistent in the way it presents its primitives and arguments.

Programmatically, most of its formatting macros are low-end, aimed at producing basic structures, such as paragraphs and boxes. For higher-level objects, custom macros are needed.

2. A Front-End

Many, if not most, of the perceived shortcomings of TEX could be solved by placing TEX behind a front-end system. The front-end would have its own simple, consistent, standardized commands for formatting documents. These commands would then be translated into TEX.

2.1 Advantages

Numerous advantages would accrue. Specifically:

1. The front-end could provide consistency and conformance with external standards, such as the Standard Generalized Markup Language (SGML).[1]

   The syntax of an SGML-encoded document is independent of processing. With SGML, a document is ordered as a hierarchy of structural elements. The standard provides conventions for beginning and ending structural elements; every element is delimited by a pair of start and end tags. The standard also provides ways of defining document types and limiting the structural elements permitted within each.

   TEX has been compared unfavorably with SGML standards; it has been labelled as a procedural, but not descriptive, text processor. Yet, some proposed SGML markup, 'lq' for instance (left

[1] As defined in International Standard (ISO) 8879 (10/86). "Markup" is used here to mean any non-text added to increase understanding of the text.

quad? long quote? loose quality?), is as cryptic as anything in TEX.[2] It is possible for SGML or TEX (in the form of macros) to be completely descriptive, depending on the interpretation.

2. The front-end could permit an object-oriented approach to documentation, which would seem more natural for users.

   (a) A front-end language would be considered object-oriented if it supported four specific object properties (Cox 1987):

      i. abstraction — a concise representation for a more complex idea, where the details are not essential to understanding the function. Each object is an instance of a class, its behavior limited by the properties of that class.

      ii. encapsulation — an object is the unit of encapsulation of an abstraction — is a process by which an individual component is defined. Encapsulation ensures that an individual object can have private properties, not shared with other objects in the class.

      iii. inheritance — classes that are sub-classes of others can inherit the properties of the super-classes. For example, in a document, a paragraph within a table would inherit the left margin of the table.

      iv. polymorphism — inherited messages can be re-defined by sub-class; the same message in different classes could be polymorphically defined.

   (b) The front-end could provide a set of object-oriented constructs which would then be used to form high-level, hierarchical document structures.

3. The front-end could, in fact, provide extensions, or perhaps "pretensions", to TEX. This would simplify the operation of the macros. For instance, when an argument in TEX is used for several purposes, such as a chapter title, running head, contents entry, and index entry, each case is handled differently; the text is a box, mark, or write, depending on the macro. Expansion occurs at different times. The front-end could copy the text, so it does not need to be done in the macros.

4. The front-end could be simpler to use and easier to learn. Complex parts of the code would be located in lower levels, in TEX, so that higher levels, the front-end, would be less complex. The markup would be descriptive and consistent.

   It could offer easier error correction. An SGML parser could check the context of elements (for example, emphasis, double columns, or indents) and issue error messages. Otherwise, this would have to be built into the TEX macros. Thus, TEX errors would be minimized. The front-end would detect errors and relate them to the elements used by the author, not to the arcane operations in TEX's gut.

5. The front-end could be extraordinarily flexible. It could provide another level of optimization. It could, in fact, use other text processors under certain circumstances. It could offer multiple styles of document preparation; the same dataset could be available as a manual, print-file, or on-line book. The layout would be determined by a style definition.

2.2 Disadvantages

Putting a front-end on TEX does have disadvantages, however:

1. A comprehensive front-end may be almost as difficult as TEX to learn, especially if it has to serve a variety of applications.

2. The interface between the programs becomes complex.

3. Maintenance and debugging also become more complex and lengthy.

[2] See Coombs *et al* 1987.

4. Running time may be increased.

5. The front-end may not be as portable as TEX.

3. An Example in Progress: Digital's VAX DOCUMENT

Front-ends have been advertised for a number of years. The Tyxset system (1985, Tyx Corporation, Reston, VA) was designed to hide TEX from users with no typesetting experience, while letting them typeset. Page One (1987, McCutcheon Graphics, Toronto, Canada) offers templates for automatic book production on the Macintosh, where the skill requirement was listed as the ability to use a mouse. MARKUP (1987, Hewlett-Packard, Palo Alto, CA) is an SGML parser and application generator that uses TEX to print internal documentation (Price 1987).

Digital Equipment Corporation's VAX DOCUMENT provides a front-end for a formatting engine that is currently "based on TEX". The engine has been modified and does not pass the `trip` test. Changes include making TEX into a callable function, instead of a standalone program, providing 8-bit support (to permit fonts with 256 characters, instead of 128), raising the memory limit, replacing error messages with more normal VMS messages, and increasing the number of marks, dimens, and fonts.

This means that a VAX DOCUMENT-produced file cannot run through public domain TEX, or similarly, an arbitrary TEX file won't run through VAX DOCUMENT's text formatter. In fact, end users are not allowed to load their own format files. VAX DOCUMENT consists of the following components:

1. A proprietary generic markup language, based on preliminary international standards.

2. A tag translator, which reads the source file and produces an intermediate file (with TEX syntax) that can be read by text formatting software.

3. The text formatter, which uses the input file from the translator and font definition files.

4. Device converters that produce output for different devices: monospaced, LN03 printers, PostScript printers.

The text formatter controls page layout, typography, and output dependencies. Text formatting macros are coded using a proprietary TEX macro package, dependent on TEX extensions (and inclusive of future bug fixes). Most of the macros (and language dependencies, font loads, device-dependent characters) are considered internal to the product and are shipped to customers in machine-readable form only.

Invoking VAX DOCUMENT begins a chain of processing. One command calls the components of the system and their associated files. Except for the top level `verb` component, which parses the command line and some files and creates an item list to facilitate communication, all the components are data processors that manipulate an input file to produce a readable file for the next processor. The tag translator reads an ASCII input file with embedded generic tags and replaces them with definitions from a saved tag table file, which supplies a mapping from the tags to strings of TEX macros. The tag translator output becomes input to the text formatter, which looks up the macros from a format file; the macros are parameterized by a design file.[3] At the end of the chain, a formatted file is displayed or printed.

The major components of VAX DOCUMENT are the tag translator and text formatter. Both components are macro processors, i.e., text substitution programs. Each processor defines a set of primitive macros that do "work". The primitive macros include a kind of programming language that allows an algorithm to be written. The tag translator macros are called *tags*, to conform with generic tagging and to avoid confusion with TEX macros.

VAX DOCUMENT is still evolving. Currently, it supports government standards (DOD 2167, 2167a). Other features are being studied: automatic language strings; support for languages that read from right-to-left and left-to-right, as well as the mixture of the two; a modular form for the text formatter macros; an on-line bookreader.

[3]Users are not told how to modify the saved tag table file and format files, only design files.

4. Summary

TEX is a sophisticated system for setting type. But TEX has a high learning curve, and TEX does not meet international standards.

TEX is a public domain program with all the attendant problems of public domain programs, such as maintenance and improvement — who will support and disburse TEX and for how long? By extending TEX, making it proprietary, companies such as Digital guarantee interest and investment in TEX. By using a front-end, they can complement the strengths of TEX with those of the front-end. Also, they are flexible in their choice of a formatter.

An object-oriented front-end that conformed to SGML standards could handle the logical structures of a variety of documents. That front-end could then harness TEX's impressive text-formatting capabilities.

Bibliography

Coombs, J.H., A.H. Renear, and J.S. DeRose. "Markup Systems and the Future of Scholarly Text Processing." *Communications of the ACM* 30:933–947, 1987.

Cox, Brad. *Object Oriented Programming: An Evolutionary Approach*. Reading, MA: Addison Wesley, 1987.

Price, Lynne. "SGML and TEX." *TUGboat* 8(2):221–225.