TEXtensions

Editing .dvi Files, or Visual TEX

Jonathan Fine

Abstract

This note outlines the specification of a TEX format, that will allow the resulting .dvi file to be edited via a suitable previewer and a .dvi file editor. Such close linking of editing and typesetting appears to be within the present capabilities of TEX.

Value Added Typesetting

Typesetting can be thought of as a process which adds value to the document being processed. This may not be true for works typeset from the author's original manuscript and corrected proofs, for such physical documents reveal change of mind, history of composition and other details which are lost in the printed version of the document. But here we consider the typesetting of, say, a suitably tagged ASCII file.

Throughout this document we will use the language and conventions of TEX, but most of the issues involved are of a more general nature, and apply to any computer typesetting system.

Suppose throughout that myfile.tex is typeset to produce myfile.dvi. If the latter file is the former, together with some added value, then it should be possible to recover the former from the latter. Oddly enough, a recent posting to an electronic discussion list raised precisely this problem. An author had in error deleted the original .tex file, and wished to recover its content, as best as was possible, from the .dvi file. This then is the definition of *value added typesetting*— from the typeset file it must be possible to extract the source file.

Poppelier (1991) also contains a discussion of the process by which typesetting adds value to the document, but from a different point of view.

Specials

TEX has a process by which special instructions can be transmitted to printing devices (*The TEXbook*, page 226), and that is the \special mechanism. Each \special that makes it to the .dvi file will produce in it a string of characters, attached to some specific location on the page. These characters are not usually intended to be printed on the page, rather their purpose is to control the printing process in some way.

This mechanism can be used to embed some text into the .dvi file, but one should (*The TEXbook*, page 228) "be careful not to make the list [of characters, i.e., the text] too long, or you might overflow TEX's string memory." The author does not know if this will be a danger for the constructions about to be described. Using emTEX he has created a .dvi file with 500,000 different specials, whose content is the numbers from 1 to 500,000, as digit strings.

One solution to the added value problem is to place the entire text of the input file myfile.tex as special in the document. Although this satisfies the formal requirement, it is a little coarse. To edit the file myfile.dvi consists of editing the copy of myfile.tex which is embedded as a special in myfile.dvi. It would not be difficult to adapt a text editing program, so that it operated on this embedded special, rather than a self-contained file. TEX can then be run, without an error arising one hopes, to refresh myfile.dvi.

Although coarse, this illustrates the essence of the method by which .dvi files may be edited via the previewer. What is required is that the process be refined.

So far as I know, products such as Lightning Textures continually refresh the previewed .dvi file as the the user changes the source .tex file, but do not associate the individual characters, words or markup in the underlying .tex file to the content of the displayed .dvi file. Thus, the user cannot edit the .tex file solely by interacting with the .dvi file. This is possible with Scientific Word, which should be thought of as a WSYIWYG or more exactly visual editor, whose underlying file format is LATEX. I believe that it is precisely because TEX as usually used does not allow the solution of the problems described here, that Scientific Word does not use tex to format files for the editor to display.

Smaller Specials

The text of a document, say as an ASCII file, is naturally broken down into paragraphs, words, characters, and spaces. It seems natural to break a document down into words. They are the smallest units of meaning. This is reflected in the very name of the tool used by authors to prepare documents, the word processor. Programmers are more accustomed to using the file editor.

For the moment, we shall assume that the document is very plain, with no changes of font or other control sequences. Suppose that we have a TEX format that will, besides typesetting the document, place before each word in the document a \special, whose content is the following word, as represented in the source file. Because of ligatures and hyphenation, this may not be the same as the characters which follow the \special in the .dvi file.

Suppose that myfile.dvi is created from myfile.tex by using this format. It will not be difficult, by extracting the text of the specials, to recreate myfile.tex from myfile.dvi. (This is not strictly correct. Assuming the usual category codes, additional spaces between words, additional lines between paragraphs, and the location of line breaks within paragraphs, will all be lost when passing from myfile.tex to myfile.dvi. This is probably no great loss. Contrarywise, typeset paragraph line breaks have been introduced. It may even be an advantage to have a source file whose line breaks agree with those of the .dvi file.)

A Special Format

It is not so difficult to create a format that will read the input file word by word, and place the words as it reads them into \specials. The code below, which is intended to be read in an environment where white space is ignored, and ~ is a space character, shows the basic features of such an environment.

The macro \sentinel is used simply to indicate the end of a paragraph, or the end of the file.

```
\def \sentinel { \noexpand \sentinel }
```

The idea now is to define \dopar so that

```
\dopar
The first paragraph is not very
long at all.

The second paragraph is even
shorter.

\sentinel
```

will result in appropiate typesetting and specials.

Here is a simple (too simple) implementation. The macro \dopar will read text paragraph by paragraph, until the \sentinel follows a blank line or explicit \par.

```
\def\dopar #1 \par #2
{
  \doword #1 ~\sentinel \par
  \if #2 \sentinel
    \let \next \relax
  \else
    \let \next \dopar
  \fi
  \next
}
```

The macro \doword similarly goes through the paragraph word by word until \sentinel is reached.

```
\def \doword #1~#2
{
  \special { #1 }  #1~
  \ifx #2 \sentinel
    \let \next \gobble
  \else
    \let \next \doword
  \fi
  \next #2
}
```

This sample code is not intended to be the basis for a practical implementation of a format that will create value-added .dvi files. Rather, its purpose is to show that such a format is possible, and to draw attention to some of the difficulties which may be encountered when creating such an object.

Editing via a Previewer

Suppose now that myfile.dvi has been created by a format file as above. The viewer notices a misspelt wrod. Within a special in the .dvi file, it is easy to change the letters wrod into word. It will be harder to add or delete letters within the special, because there would not be room for the addition at the correct point in the .dvi file, or a hole would be left in it. But this is the sort of problem which editing programs are accustomed to dealing with.

Now that the copy of myfile.tex which is within myfile.dvi has been changed, one would like the rest of myfile.dvi to be brought up to date. For simplicity, we shall assume that myfile.dvi is simply one page, a galley that is long enough to accommodate all that is placed on it. Changing wrod to word will change the paragraph in which it is placed. The change will in general be more complicated than replacing wrod by word. Even this simple change may change the line breaks in the paragraph. Hyphenation may change, as may ligatures and kerning. Correcting a simple letter transposition error will require resetting the whole paragraph.

In most TEX formats, the size and content of one paragraph does not influence the setting of the others. All then that needs to be reset is the paragraph in which the change occurred. If the document were set into pages rather than just a galley, then page breaks would also need to be reconsidered.

This discussion has focussed on changing the letters in a single word in a paragraph. Adding or deleting whole words will go the same way, as will addition or deletion of paragraphs, provided the format does not do something like numbering paragraphs. In any case, TEX will be required to process some text when a change is made to the `.tex` file embedded in the `.dvi` file, to bring the `.dvi` file up to date.

Calling TEX from the Previewer

When the user has finished making changes to the paragraph, or earlier if wished, the previewer must call upon TEX to reprocess the changed paragraph. As before, we assume that the difficulty faced by all editing programs, of deleting or adding material in the middle of a file, has been solved.

TEX turns a text file into a `.dvi` file. Ordinarily, this `.dvi` file is not accessible until TEX has come to an `\end`. By writing a suitable device or virtual file, which will depend on the operating system, it should be possible to use the `.dvi` output of TEX before the `\end`. Thus, a command such as

```
\shipout\vbox{\input tempfile}
```

will cause TEX to produce a page which contains the revised and reset paragraph, which the editing functions attached to the previewer can now paste into place, replacing the old version. Incidentally, if the output of TEX is a virtual `.dvi` file, then there should be no reason why the input `tempfile.tex` should not also be a virtual device.

The foregoing discussion is intended to demonstrate that by combining TEX as it is, together with a suitable format, a suitable previewer, editors for text and `.dvi` files, and a bit of operating system virtual file magic, it is possible to produce a WSYIWYG variant of TEX. Users of this composite program will be able to edit their documents via the previewer. The format as described is limited to a single page, a single font, and no mathematics, but it does have multiple paragraphs.

Breaking Pages

Suppose now that the document is broken into pages, and a paragraph is added. All subsequent pages, and perhaps the previous page, will have to be reconsidered. Assume TEX is being used in a normal manner, so that parameters such as `\linepenalty` and `\brokenpenalty` do not change from page to page. Were TEX to reprocess the whole of the changed `myfile.tex`, all paragraphs from the previous version would be broken exactly as before, and so there would be no need for them to be reset.

The previewer and editor combination can ask TEX to break the new document by passing it a suitably coded sequence of boxes, penalties, skips and kerns to break, for this is all the page breaking mechanism (*The TEXbook*, Chapter 15) operates on. Alternatively, some other program could be asked to do the breaking. This is the approach taken by Type & Set (Asher, 1992).

Control Words

The aspect which presents the most difficulties is now to be discussed, and briefly at that. Most documents contain control words, such as `\TeX` and `\section` and `\eqalign`. These are part of the source `myfile.tex` and so they must go into `myfile.dvi` as specials. When it comes to the editing process, even though the addition or deletion of a control word such as `\TeX` is fairly innocuous, to add or remove an `\equalign` can have a drastic effect.

The braces `{}`, and the mathematics shift character `$`, will also have a drastic effect when added or removed from `myfile.tex`. The same is true when the delimiter required by some macro is omitted. For this approach to have all the typesetting power and programmability which TEX provides, access to local change of font etc., parameter delimiters, and mathematics shift must be provided. This has to be done in a manner which is consistent with the editing requirements imposed by the embedded `\special` approach.

Unbalanced braces, missing or extra mathematics shift characters, and missing delimiters all provide difficulties for users of TEX. A format which detected such input errors early, before they gave rise to an error message from the stomach of TEX, could make TEX easier for many users. A format which allows the user to edit the copy of `myfile.tex` embedded within `myfile.dvi` would similarly have to detect input errors before they reach TEX's stomach.

Performance

It is quite possible that such a format, which reproduces the input text as specials, and detects all errors before TEX does, will run at perhaps a tenth of the speed of a regular format. However, the usual approach requires the document to be typeset as a whole, and so an unchanged paragraph may be typeset a dozen times or more during the revisions. A format which sets whole documents slower, but which is able to reset the document paragraph by paragraph may very well consume fewer machine cycles over the life of a document.

Moreover, the paragraphs can be set or reset as completed, rather than file by file. If the computer is sufficiently rapid, and many are today, this machine work can be done as required. This will result in the user being locked out for a short period at the end of each paragraph, while processing takes place, just as an editing program may deny access to the user while a file is being written.

Thinking alike

Since writing this article, I found the following statement put forward to motivate the CONCUR feature provided by SGML.

> It is sometimes useful to maintain information about a source and a result document simultaneously in the same document, as in "what you see is what you get" (WSYIWYG) word processors. There, the user appears to interact with the formatted output, but the editorial changes are actually made in the source, which is then reformatted for display.

This quotation comes from Annex C.3.1 of ISO 8879 (the SGML standard) and is also reproduced (as is the whole of ISO 8879) in Goldfarb (page 88).

Conclusions

More can be done with TEX as it is, than is commonly realised. Some of its limitations exist in the imagination of the critics rather than in the program itself. I hope that all those who say that TEX cannot do such-and-such think carefully as to why it is that TEX cannot do what they wish.

For a portable WSYIWYG variant of TEX to be produced, the various additional components, which are previewer, `.dvi` file editor, text file editor, and operating system magic, must also be portable or ported. An editor for `.dvi` files is probably the most important new program. For this to work most effectively, it may be necessary to extend the `.dvi` file specifications.

Also required is a format file, which satisfies requirements far more exacting than met at present. This also would be a major piece of work.

Acknowledgements

The author thanks Mimi Burbank for correcting many errors in a previous version of this article, and Alan Hoenig for some encouraging remarks.

Postscript

For various reasons the publication of this article has been long delayed. (It was widely circulated in preprint form at TUG'93 in Aston, England, and was submitted later in that year. Not all of the delay has been due to the author.) Since then, the author has solved all the fundamental problems involved in creating a TEX format file that will create a `.dvi` file that can be visually processed. Also since then the World Wide Web has 'taken off' and this provides an additional reason for producing `.dvi` files which support rich visual interaction. The article "TEX innovations at the Louis-Jean printing house", by Laugier and Haralambous, describes an interesting program that allows certain changes to be made interactively to a `.dvi` file. Also relevant is the author's own article, "Documents, compuscripts, programs and macros".

There is some overlap between this article and Rahtz, "Another look at LATEX to SGML conversion". In both cases, the placing of text from the document into the `.dvi` file as `\specials` is a crucial technique. In this article the purpose is to store the original text of the article. For Rahtz, the purpose is to create a transformed form of the article.

Also relevant is the article of Kawaguti and Kitajima, which describes a different approach. They have created a loosely coupled composite from adaptions of two existing tools, namely `emacs` and `xdvi`. Special tags are added to a traditional (La)TEX source file, which produce `\specials` containing file name and line numbers in the `.dvi` file. These are used to support new `emacs` and `xdvi` commands, which together allow the user to move the `emacs` cursor by clicking on the previewed `.dvi` file. This is, as the authors recognise, not the same as what this article calls visual typesetting.

Would those who are interested in following the path outlined in the present article, particularly on the viewing and editing side, please contact me. For such a system to provide an open architecture,

standards for the use of specials, going far beyond Rokicki's "A proposed standard for specials", will be required.

Bibliography

Asher, Graham. "Inside Type & Set", *TUGboat* **13** (1), pages 13–22, 1992.

Fine, Jonathan. "Documents, compuscripts, programs and macros", *TUGboat* **15** (3), pages 381–385, 1994.

Goldfarb, Charles F. *The* SGML *Handbook*, Oxford University Press, 1990

Kawaguti, Minato and Kitajima, Norio. "Concurrent use of an interactive TEX previewer with an Emacs-type editor", *TUGboat* **15** (3), pages 293–300, 1994.

Laugier, Maurice, and Yannis Haralambous, "TEX innovations at the Louis-Jean printing house", *TUGboat* **15** (4), pages 438–443, 1994.

Lavagnino, John. "Simultaneous electronic and paper publication", *TUGboat* **12** (3), pages 401–405, 1991.

Mittelbach, Frank. "E-TEX: Guidelines for future TEX", *TUGboat* **11** (3), pages 337–345, 1990.

Poppelier, N. A. F. M. "SGML and TEX in Scientific Publishing", *TUGboat* **12** (1), pages 105–109, 1991.

Rahtz, Sebastian. "Another look at LATEX to SGML conversion". *TUGboat* **16** (3), pages 315–324, 1995.

Rokicki, Tomas G. "A proposed standard for specials", *TUGboat* **16** (4), pages 395–401, 1995.

Taylor, Philip. "The future of TEX", *TUGboat* **13** (4), pages 433–442, 1992.

Siebenmann, Laurent. "The Lion and the Mouse", EuroTEX '92 Proceedings, edited by Jiří Zlatuška, pages 43–52, 1992.

◇ Jonathan Fine
203 Coldhams Lane
Cambridge, U.K.
CB1 3HY
J.Fine@pmms.cam.ac.uk