Simultaneous Electronic and Paper Publication

John Lavagnino
Department of English and American Literature, Brandeis University, Waltham, MA 02254 USA
617-736-2080
Internet: lav@binah.cc.brandeis.edu

Abstract

Many current applications in computerized text processing involve the creation of "multiform texts". Such a text is designed for use in several forms: in both print and electronic form, for example. This is a valuable goal for many kinds of text; one example that may perhaps seem unlikely is the edition in progress of Thomas Middleton's complete works. The central question in creating a multiform text is the choice of a language for the basic text files; SGML seems to be the best choice. It has worked well on the Middleton project so far, and has worked well together with TEX in solving some of the problems that have arisen specific to this text.

Multiform Text

An underlying thread connects a number of different projects in the computer processing of texts: the idea of a "multiform text", a work that is meant to be read and used in several different forms — most characteristically, both electronic and printed forms.

One of the most familiar instances of such a text is the computer manual. If you're writing a manual for a computer you're likely to be using a computer to make it; and then why not use the computer to access it as well? That access doesn't necessarily require giving any special thought to making the electronic edition useful: many of us have long depended on having a copy on disk of the TEX source for *The TEXbook*. A text editor is all you need to work with it.

But one does not read TEX source very happily: this system is fine if you want to look at a macro definition, but it's unsatisfactory if you're interested in what an example produces on the printed page. More interesting are those systems that attempt to provide both print and electronic versions that are equally usable. On-line help systems for computers often work from a textual base that's adapted from, or also issued as, print manuals: both usually contain much the same information, and the attraction of writing the documentation once, not twice, is obvious. The UNIX `man` command is one familiar example: it draws on text encoded in the `troff` typesetting language, and formats that text for display either on the user's terminal or on a printer, so that when you ask for help on a command, you get the same text that's presented in the printed manual.

This system is possible because the documentation is encoded in a way that doesn't make it impossible to print on a typewriter-like device instead of on a real typesetter. The on-line access, however, gives you nothing more than page images; these provide as much information as the printed manual, but they also provide *no more* than that. In contrast, the programs distributed by the Free Software Foundation use a more sophisticated documentation system that takes better advantage of the computer's powers for structuring text in ways not available in print. The Texinfo system not only uses a descriptive markup (based on TEX) that encodes the structure of the text and lets macros decide how to present the information; it also includes encoding for cross-references that allow a user who's got the GNU Emacs editor to more effectively find information in the manual and move to related topics in it (see Stallman and Chassell).

Such a system combines the advantages of print and electronic editions. The print user can still read in bed, write on the copy, suffer less eyestrain, and use the document when the computer is down; the electronic user can search for a topic or phrase much faster, follow connections that may not be represented in the sequential text of the printed manual, and get assistance with a program from within that program. The best use of the system seems to come not from using one or the other form exclusively, but from switching back and forth,

using the form which is best for addressing each momentary need.

That sort of combined publication needn't be limited to computer manuals. The advantages of multiform text are there for all sorts of works.

The Oxford Middleton

My own interest in the multiform text comes out of my work as one editor of an edition of the complete works of Thomas Middleton (1580–1627), the English Renaissance playwright. This edition is being prepared by an international team of editors for publication by Oxford University Press in 1994. It will include the texts of all Middleton's works — above all, his twenty-seven extant plays, but also numerous masques, entertainments, poems, and prose works; and it will provide introductions and very detailed notes to all these works. This is the first complete edition of Middleton's works in over a century, and we hope that it will not only collect all the accumulated scholarship on Middleton, but also establish his importance as a writer.

Such a complex work is usually quite expensive to set in type, so that the advantages to us of using TEX to do the typesetting ourselves are clear. The advantages of creating a multiform text (instead of concerning ourselves solely with entering the right TEX codes to print the work) may be less immediately apparent. Yet a multiform text is of value both for us, during our preparation of the work, and for other readers and scholars after its publication. Editors and scholars have long depended on concordances to help them in understanding the characteristics of an author's style and thematic concerns; the electronic text gives us, in effect, such a concordance to the text as we prepare it, rather than long after it's published. Providing an electronic text also makes possible a later conversion of the work into a hypertext that can allow readers quick access to the various sorts of notes to the work.

The creation of a multiform text is not an experimental approach, but instead one that keeps the labor for everyone to a minimum and creates the most valuable print and electronic editions. In the following discussion of the salient issues in this case, I'll mention these work requirements as they come up.

The Choice of a Language

Most of the important questions about how to create a multiform text are related to the choice of a "markup language" — the language in which the text and its structure are specified. The basic idea is to choose one form for the text from which all other forms, electronic and printed, will be derived. The markup language for this basic form should make the derivation of other forms work easily and well.

The UNIX `man` command, and the GNU Texinfo system, both use typesetting languages with macro capabilities — TEX itself, in the latter case. And that choice might seem to make sense in the general case: after all, one thing we want to make is a printed text created by TEX, and so using TEX as our markup language seems to automatically solve the problem of creating one of our final forms. But it isn't a helpful choice when it comes to the electronic side: TEX is not especially easy to translate into other markup languages. The nature of its macro definition facilities means that a program needs to know rather a lot of what TEX knows if it's to be able to make the conversion. Consider the rules in TEX for determining when a macro name has ended: according to Knuth [page 47], these require that we know the difference between letters and other categories of characters — a distinction that can be changed by a TEX input file. Argument delimitation is still more complicated [Knuth, pages 203–204]. (I am assuming here that the most desirable approach is to transform the basic form directly into other electronic forms. Carr and Partl have discussed separately approaches based on taking `dvi` output and converting it to other electronic forms, approaches that make things still more difficult.)

For the Middleton edition, we have chosen SGML, and use TEX only for the typesetting, not for our text representation. (See Laan for an introduction to SGML.) SGML is, first of all, rather easy to convert to other forms: the names of "tags" and "entities" in SGML, two different sorts of commands that are similar to different aspects of TEX macros, are in the normal usage terminated in an unvarying way — by '>' for tags, and ';' for entities. Converting our text from SGML to TEX seems to require nothing more complicated than global substitutions, and a few simple TEX macros to deal with the product.

A more important reason for choosing SGML lies in another facility it offers. It is intended not only to handle the electronic representation of a document's structure, but to allow the specification of rules governing that structure, and verification of a document's conformance to that structure. TEX checks only that you aren't transgressing the rules

of its input syntax; it has no facility for ensuring conformance to any specifications narrower than those in the *The TEXbook*. One can build such specifications into any macro set, to some extent: LATEX provides an example, in its checks on the proper nesting of `\begin` and `\end` commands (among other things). LATEX still doesn't check everything, and its specifications are those of the whole macro set, not a user's subset.

This question of verification matters for any kind of text, but it's of particular importance with the multiform text. It's necessary with such a text to keep close tabs on what commands are used: you want to ensure that you don't wind up with something that can be represented in one medium but not the other. We're familiar with struggles to get a page printed just right; but there's another level to the problem in this perspective, that of getting it "just right" in more than one medium. SGML helps prevent surprises in this realm.

The use of SGML's facilities does require some extra work to formulate the specifications for the text's structure, but some consideration of those specifications has usually been necessary with multiform texts; the advantage of SGML is that it can help to enforce those specifications.

SGML is also more securely oriented than any typesetting language towards encoding the structure and meaning of text elements, and not the details of how they're to be printed. The importance of an encoding that is focused on structure and meaning has already been argued at length (see Coombs et al. for a theoretical presentation, and Lafrenz for a publisher's agreement with it on practical grounds). Greater abstraction will also help us with uses we've never anticipated (but which may suddenly be of importance when our publication date in 1994 rolls around): our ability to generate new forms of our text will be enhanced if we have precise specifications of what's going on in our text.

Finally, for our particular project, there is the advantage that the international Text Encoding Initiative is currently developing guidelines based on SGML for tagging electronic texts, with particular attention to the needs of scholars (see Sperberg-McQueen and Burnard for a draft of these guidelines). We want our text to get used and studied, and adhering to standards is a good way of doing that.

Most people get the impression that SGML is not good to use for data entry, because its markup appears to be very bulky. This is only true when no use is made of the extensive provisions SGML makes for minimizing the markup; with proper use of these features, SGML requires no more typing than TEX does. But we began our project without any SGML tools, and so we handle the data entry in another way. Our approach has been to devise a very terse markup that's used solely for data entry, adapted very narrowly to the kinds of texts we're encoding; we convert this immediately to SGML, and perform all further processing on the SGML files. The creation of the programs to do this conversion has been one of the principal tasks involved in setting up this mode of working — though it has hardly been an onerous one. If we had obtained appropriate SGML tools at the beginning of our work, even this task would have been unnecessary.

Referring to the Printed Text

The careful choice of a markup language should make it possible to contain the problems that come from our need to do a great deal of computer processing of our text; it should make the necessary transformations easy, and ensure that we aren't entering textual elements that can't be processed within both realms.

But there is another layer of problems that can arise. What would happen if we needed to include information in an electronic text about the details of how the printed text looked? That would mean that the printed text would not just be a spinoff of the electronic text, but that we'd need to extract information from our printed text — or from the `dvi` file — and fold it back into the electronic version; it could be a difficult task.

The conventional index is a good example of this: the text of an index is an analysis of the book in which it appears, and it's dependent in a very sensitive way upon how the page makeup came out. Of course, we know how to handle index-making with TEX. Its `\write` command is designed to facilitate capturing information for an index or table of contents that needs to know about page numbers. In other multiform texts it's been common to use references not to page numbers but to important structural divisions, which don't depend on the page makeup: the UNIX documentation that's used by the `man` command is broken up into small chapters, rarely more than a few pages long, one chapter for each command.

The particular traditions of publishing in literary studies pose a problem for us with Middleton. One demand that scholars make and are not going to give up is for a very precise system for referring to particular lines in the text, a system traditionally

implemented, in fact, with line numbering. Middleton's plays are typically written in a mixture of prose and verse, often changing within a speech. Prose is traditionally numbered using physical line numbers: that is, each actual line of type is counted as a line. Verse is numbered using logical or structural line numbers: a line of verse may take more than one line of type to print, but it's still counted as only one line in this numbering scheme. On top of this, stage directions are handled in a different way: whenever a stage direction appears that's not on the same line as spoken text, it's given a physical line number in a decimal numbering annexed to the previous speech's line number: 18.1, 18.2, etc. The stage directions that appear at the opening of a scene are numbered 0.1, 0.2, etc.

The force of tradition makes it impossible to use a different system (and it seems difficult to come up with another that would be as precise and as easy to use for readers of the printed text). We can print such line numbers readily enough, using the EDMAC macros (see Lavagnino and Wujastyk). EDMAC can also create footnotes and endnotes that use such line numbers in references, but we also need to get them back into our SGML text: that is, to mark in the SGML text the point at which each line begins. The reason is that users of the electronic version, as well as users of the paper version, need to be able to look things up using these line references, and to find the line address for a passage so that they can tell others where they're looking.

This is a problem because the line numbers that we ultimately want to fold back into our base text are all generated in the course of typesetting, and actually it's not an easy matter to find out what they are and get them back into our SGML text. Consequently, there's a need for software that can take information out of our typesetter file — out of a file that is usually deliberately made to focus on niggling presentational details and tell us nothing about structure — and interpret it for incorporation into the SGML. It's a striking instance of how the printed page is not merely an end product that leads no further, at least not within the electronic world.

I said that one reason behind our use of SGML was to stress the representation of meaning rather than structure in our text. But the reference-system problem leads to a curious inversion of this situation: if we want a print-based reference system, we must process the output from our text formatter — output which consists of text that's been converted to a format that tells us as little as possible about *meaning*, and far too much about *appearance*.

For ordinary prose this isn't really a huge problem. In dramatic texts, line numbering is complicated, being partly logical and partly physical. It's quite difficult for a program to determine the numbers by just looking at the type on the final page, unless every single line is numbered. It can be done, but at the expense of writing a program that's highly dependent on the details of how your pages are laid out, since a lot of the clues that we as readers depend on to figure out whether something is prose or verse or whatever have to do with indentations, details of spacing, and font selection.

Our approach to this problem puts all the burden of assigning line numbers to blocks of text on TEX itself. Rather than try and write software that guesses the line numbers, we have TEX itself issue `\special` commands at the start and end of every line of text: to specify the line number, and to mark that text as a part that is numbered (since every page includes headings, marginal line numbers, and other text that is part of the presentation of the text, not the text itself). This much is a simple extension of the EDMAC macros that generate the line numbers: those macros already add each line of text to the output page separately, so inserting the `\special` commands that enclose each line of text is straightforward.

The bigger task is interpreting the resulting `dvi` file: we need to convert it into something that's close enough to our original SGML file that we can match up the texts and see where to put the line numbers. This is by far the most substantial programming task that the production of the Middleton edition has required so far, and I don't expect that anything to come will prove more difficult. The problem, however, would be more difficult with any typesetter other than TEX. Not only does it have the unusual, but very useful, `\special` mechanism; it also comes with its binary-file formats documented in a very thorough manner, and with ancillary programs that already demonstrate how to read things like `dvi` files. Indeed, the `dvitype` program already does a great deal of the task for us: our preliminary version of this software simply starts from `dvitype` output, not from the `dvi` file itself.

Although this is a thorny problem, it is one whose solution is made much simpler by certain well-known (but perhaps insufficiently-appreciated) merits of TEX: its extensibility, its excellent documentation on its internal workings and file formats, and its wealth of supporting programs, all available in source code.

Conclusion

Many people who prepare texts on computers are already finding themselves drawn to the creation of multiform texts. In this account I've tried to identify the general questions that should be considered in doing this; but it's probable that others will also run into problems specific to the kind of text they're working with, as we have with reference systems in our work. The use of flexible tools also makes the successful resolution of these problems easier.

Bibliography

Bell Telephone Laboratories. *Unix Time-Sharing System: Unix Programmer's Manual.* Seventh edition. Murray Hill, New Jersey, January, 1979.

Carr, L., S. Rahtz, and W. Hall. "Experiments in TEX and HyperActivity." *TUGboat* 12(1), pages 13–20, 1991.

Coombs, James H., Allen H. Renear, and Steven J. DeRose. "Markup Systems and the Future of Scholarly Text Processing." *Communications of the ACM* 30(11), pages 933–947, 1987.

Knuth, Donald E. *The TEXbook.* Reading, Mass.: Addison-Wesley, 1984.

Laan, C. G. van der. "SGML (, TEX and ...)." *TUGboat* 12(1), pages 90–104, 1991.

Lafrenz, Mimi L. "Textbook Publishing — 1990 and Beyond." *TUGboat* 11(3), pages 413–416, 1990.

Lavagnino, John, and Dominik Wujastyk. "An Overview of EDMAC: A plain TEX Format for Critical Editions." *TUGboat* 11(4), pages 623–643, 1990.

Partl, Hubert. "Producing On-Line Information Files with LATEX." *TUGboat* 10(2), pages 241–244, 1989.

Sperberg-McQueen, C. M., and Lou Burnard, eds. *Text Encoding Initiative: Guidelines For the Encoding and Interchange of Machine-Readable Texts.* July 1990.

Stallman, Richard M., and Robert J. Chassell. *Texinfo: The GNU Documentation Format.* Cambridge, Mass.: Free Software Foundation, 1988.