SGML versus/and TEX

Robert W. McGaffey
Oak Ridge National Laboratory, Building 2506 MS 6302, P.O. Box 2008, Oak Ridge, TN 37831-6302 USA
615-574-0618; FAX: 615-574-1001
Internet: mcgaffeyrw@ornl.gov

Abstract

Everyone who handles computer documentation faces the problem of proliferating application-specific versions of a source file and the added difficulty of merging changes back into the source. SGML is a resource for building a generalized solution. TEX and SGML offer a particularly harmonious synergism for documentation applications.

Consider This Problem:

You have a database with important information. You need to publish some of the information and wrap it inside appropriate text. Furthermore, you need to create an abstract for printing inside a professional journal in preparation for a presentation of your paper at an international meeting. Meanwhile, a colleague calls and requests a copy of some of the database for his/her research *if* you furnish it. Finally, your latest experiment dictates that you must change some of the data you have already placed in half a dozen places. Wouldn't it be grand if you could just keep all of that information in one place and only have to modify one copy and be sure that all of your data was up to date? Of course it would. But you can't. Well, suppose you could keep one "official" copy and automatically generate all of the others whenever the "official" copy changed? Would you be interested? If so, welcome to the world of SGML.

What Is SGML?

SGML (Standard Generalized Markup Language) is an international standard which purports to standardize the way information is marked up in a storage medium. But in practical terms, SGML implies a system of programs which helps us to create both the "official" computer file and the automatic copies we have to generate. Here's how it works:

First we create a file, called a Document Type Definition (DTD), which describes completely how the information is organized in the "official" file. The DTD is intended to support hundreds or even thousands of documents organized in the same way. For example, consider a "theme" which will contain one title followed by one or more paragraphs.

Next, we can build what is called an "instance" file, which is the official name for our "official" file. A simple DTD file could lead us to the following instance file:

```
<document>
<title>The title</title>
<p>The first paragraph.</p>
<p>The second paragraph.</p>
</document>
```

Note that a SGML "element" is most often represented by something like <tag>⟨*the SGML element named* "tag" *goes here*⟩</tag>.

But our sample is not what I call *real* SGML because it is so attached to the formatting of the theme. We should be happier to see something like:

```
<theme>
<title>The title</title>
<idea>The first paragraph.</idea>
<idea>The second paragraph.</idea>
</theme>
```

Note that we are now tagging information rather than the formatting of the information.

Smart editors already exist which will help us with the two steps above. They will help us create a legal DTD file and then make sure that the instance file we create matches the DTD we created.

Now, we have our "official" file. How do we get the automatic copies?

Other programs exist which parse the instance file and translate it into another format. That means that I can translate my "official" file into my favorite typesetting language (TEX) by writing a program for my parser-translator and when I do so I will get something like:

```
\starttheme
\starttitle The title\finishtitle
\startidea The first paragraph.\finishidea
\startidea The second paragraph.\finishidea
\finishtheme
```

which, with a suitable set of TEX macros, will generate my paper copy.

Why Is SGML Better Than TEX?

Because it is more than a typesetter. Consider the way in which TEX typesets a superscript. How does TEX indicate a footnote marker?, an atomic weight?, the degree symbol?, and sometimes trademarks etc.? Answer: often all by the same mechanism; i.e., $\hat{}\langle whatever\rangle$. *Real* SGML forces you to treat these differently because they have different meanings. Thus, an SGML document has more inherent intelligence than a TEX document.

Since SGML is independent of the programs used to typeset, store in a database, extract an abstract, etc., your documents become "official", i.e., the one and only storage medium needed to hold all of the information. Therefore, any typesetter, database, etc., may be accommodated by changing only the program which drives your parser-translator. Thus a change in one file makes the new output automatic. So if I decide to switch typesetters from TEX to NTT without modifying my instance files at all.

The first time I heard that, my objection was that instead of having TEX files you then have SGML files and, "What happens if SGML changes and I want to convert my files to NISGML?" what is the difference? The answer is, if SGML is ever modified (and I sincerely hope it is!), all of your SGML files could be run through the parser mentioned earlier and converted in one huge batch file. Try doing that with any other system and you will get an appreciation for SGML and the available parsers.

Why Is TEX Better Than SGML?

When you consider the example of a mathematical expression, taking full advantage of all the capabilities that SGML offers, you cannot read the equation. The corresponding TEX equation is rather easy to read. The SGML document may also contain cumbersome structures necessary to distinguish between the various uses of say, periods. In such cases, the ⟨*filename*⟩`.tex` document is much easier to type and to read than is the instance file. But these are excuses — if the information needs to be available, then the tagging needs to be done no matter what the result is in the instance file.

Naturally, TEX is better at typesetting because SGML is not a typesetting system. There are those who try to make it so by misusing SGML's attributes and forcing SGML files to contain formatting information. But it is not. It should be used to mark the information without regard to its format. There are even those who would sacrifice printing quality for the sake of the instance file. But when you can have your SGML file and TEX it too, why not?

Then We Say, "SGML and TEX for Publishing"

There are at least four main reasons for inserting SGML in front of TEX:

1. Due to the writing of smart editors, it is much easier to create a properly structured document with SGML. But what we did not mention before is that the smart editors *force* typists to enter all of the data and in the correct order. (It is true that you can still make a mistake but you have to make it on purpose.)
2. SGML allows us to have the luxury of the "official" document. I don't think that I will appreciate all of the problems that this solves until we have had a working system in use for some period of time.
3. SGML can be used (with the proper parser-translator) to generate input to a database system as well as other typesetting systems or anything else where information is stored. Thus, the input information can be confined to one file (the instance file) and yet many output mediums are possible.
4. Because of extra intelligence available from SGML, the TEX macros needed to typeset the document are easier to write. Consider, for example, that we do not need to concern ourselves with whether or not our document has (or does not have) a title because we will know that it does. Thus, our TEX macros need not check for this.

There are at least four reasons why TEX should follow SGML:

1. There are SGML aficionados who feel that typesetters like TEX should be executed quietly behind the translator and that users need never know that it is being used at all. TEX is one

of the most programmable typesetters around and thus more capable of this than other typesetters. For example, consider typesetting a table from an instance file. SGML is aware of the structure of the information but has no idea of the lengths of the various elements. So, typesetting a table from an instance file means adopting some standard format which will hopefully satisfy some high percentage of tables. Some of the table information will be wide and some narrow. And it is not likely that the lengths of the table headings will correspond to the lengths of the data they head. As a result, the simple table of Figure 1. becomes "strung out" and ugly because its headings are so long.

	No. of samples	Exposure rate			Standard Error
		Max	Min	Ave	
Input Summary	80	110	6.6	18	3.5
Output Summary	254	11	5.8	7.1	0.043

Figure 1: Table generated with eyes closed.

	No. of samples	Exposure rate			Standard Error
		Max	Min	Ave	
Input Summary	80	110	6.6	18	3.5
Output Summary	254	11	5.8	7.1	0.043

Figure 2: Table generated after seeing disaster above.

Most of us would probably agree that the rendering shown in Figure 2 is better. TEX is one of few typesetters capable of making the decision to change the pattern of the table on the fly. That is, TEX is smart enough to decide that the second alternative is better, all by itself. The number of places where such decisions could be made is probably only a limit of our own imaginations. Thus, TEX can hide behind SGML better than most typesetters. But there are cases that cannot be handled by TEX automatically.

2. There are things that you can do with TEX that make the published-on-paper copy so clean. For example, TEX can assure us that all of the columns of any output page will never end in a hyphenated word and yet will be the same length. The author thinks the best way to do this is to look at the final document and then use `\hbox{`⟨*hyphenated word*⟩`}` whenever it is needed. Sometimes the use of `\looseness=`⟨*number*⟩ is also required to prevent widows and orphans in such columns.
3. Editors will want to make formatting changes. Editors *always* want to make changes.
4. Yet the main reason to use TEX as a backend to SGML is the reason we all started using it in the first place. It is still true that nothing else sets math like TEX (or paragraphs either) so we should continue to use TEX simply because it does such a beautiful job.