One-Document Scientific Publishing for Print and Web/CD

Peter Signell
Physics and Astronomy Department
Michigan State University
East Lansing, MI, 48824
signell@physnet.pa.msu.edu

One-content document, several auxilliaries

When a document must be published in more than one version and must also undergo periodic revision, the use of a different stored manuscript for each version may easily result in the versions getting out of step with each other, so it is considered good practice to make arrangements to have just one stored master version. This is especially important if the different versions vary greatly in the order in which the content elements are presented, as they are likely to do when one version is for the print medium and another for the Web or for CD-ROM. With all versions sharing the same master content document, each version must have its own auxiliary non-content documents specifying the version's unique architecture and formatting. Then any content revisions are made only to the master content document and any format revisions for a particular version are made only to the document which specifies the formatting of that particular version. This process is quite familiar to LaTeX document managers who often separate formatting from content in order to maintain uniform formatting throughout a document, throughout a product line, or across revisions. In LaTeX, the formatting information is usually placed in one or more auxiliary "style files", some of which are shared among different printed versions of the material or between different documents, a kind of "inheritance." In this paper we discuss some experiences with extending that publication model, of one master content document and auxiliary architecture and content documents, to the case of simultaneous print and Web publication. Along the way, we discuss the differences between the screen and print media and the implications for the auxiliary files, new linking opportunities in the Web version, and ways to move the content document toward compatibility with the new Extensible Markup Language, XML.[1]

[1] Commented links to documents on XML can be found at www.oasis-open.org/cover/xml.html and there are answers to frequently asked questions at www.ucc.ie/xml.

Overview of media differences

When a document is to be published both in print and on the Web, the formatted print and Web versions are likely to take rather different forms. This is because of the differing characteristics of the two media. For our present purposes, there are four main ways in which print presentation differs from computer-screen presentation: (1) the print medium has much higher resolution then the computer screen so scientific text can be packed much more densely in a print version; (2) the effective dimensions of the computer user's browser window can vary over a wide range, even at the whim of the user, in contrast to the totally controlled dimensions of book paper; (3) the computer is able to instantly present hidden material whenever the user asks to see it; and (4) the computer can have virtually unlimited amounts of material available for instant presentation.

Resolution-related differences

The limited resolution of the computer screen requires an increase in font size, particularly for the display of equations and math symbols, and this makes screen real estate particularly valuable. For example, one does not want to take up valuable space with task bars, sticky pads, and icons. In the computer-screen version, these functions can be provided through menus of choices that pop up when the right mouse button is pressed on a PC or when the shift key accompanies the mouse click on a Mac.

The fact that a very limited amount of material can be on the screen at any one time means that a figure should not float to the top or bottom of the page, as in LaTeX, but should instead be displayed next to the first reference to it in the text. The figure must also be available to be displayed at any point where it is referenced, since the small amount of material on the screen means that the figure is unlikely to still be on the screen at the time the user reads the reference. Also, because of the limited resolution, the user must be able to click on any figure to see an enlarged view that shows details with clarity sufficient to satisfy the user. A similar

kind of availability is necessary for equations and definitions: they must be actually on display at the first reference to each, and they must be available for display by the user at all further references.

Dimensional differences

We know quite well the size of paper on which a textbook will be printed, but we do not know the size of a computer user's browser window. Even if we know the dimensions of a particular user's screen, the user may shrink or expand the browser window at will in one or both dimensions. The user may increase the font size because of poor eyesight or limited screen capabilities. Any change in width or font size will produce a change in the number of characters allowed per line and so will require that the material on the screen be instantly and transparently reformatted. Another effect of a user narrowing the effective width of the browser window is that it will cause a figure caption to the right of a fixed-width narrow figure to be partially "off the screen to the right" unless the caption alone is instantly and transparently reformatted into lines of a narrower width alongside the figure. If the window is made too narrow, the caption must be seamlessly moved to a position underneath the figure and reformatted for that position. For equations, a good line-breaking algorithm must be used to allow equation formatting and reformatting to make the equation fit the screen size of the moment.[2]

Information-hiding differences

The computer has the unique ability to pop up information at the user's discretion, and this affects the placement of material in the flow of the document. For example, in printed textbooks the answer to a homework problem is never printed at the end of the problem because it would then be too easily seen by a user working the problem. Instead, in textbooks the printed answers to problems and exercises are almost always collected at the ends of the books. Other "optional" materials are collected away from the points at which they will be needed by some users but not needed by others. In the computer version, each of these elements can be made to pop up at the relevant point if the user so desires. In our case, these optional pop-up elements consist of specifically targeted help sequences and additional skill-based instructional elements and practice problems as well as the usual problem and exercise answers. Thus the computer-screen and print versions are very different in the flow and user-activated flow of document elements. In addition, there are proposals for "information that knows about me (my needs and preferences)" and this would require a multitude of possible paths through the kinds of information that may be available to construct a custom document. Finally, we note that a print version is limited in the amount of material that can be included because more information results in a higher price and a heavier weight, and sufficient amounts of different kinds of optional material can make the user navigate what seems to be a gigantic maze. No such problem occurs in the Web version.

Next year's solution: XML

Both print and Web versions of books have recently been produced from content-only documents, plus version-specific non-content documents, using the World Wide Web Consortium's "Extensible Markup Language," universally called XML.[3] However, very few of XML's eventual capabilities have been used because parts of the XML specification suite are still under development by working groups of the World Wide Web Consortium (hereafter "the W3C"). The basic specification for XML was "recommended" by the W3C in February and full approval is expected in the fall. The math markup language is in the "recommended" stage and may also be approved this fall by the members of the W3C. The specification for the XML formatting ("style") language, XSL, may emerge from the XSL working group this summer. As for XML browsers, Microsoft's *Internet Explorer* 4.0 already includes some XML tools and Netscape Navigator is scheduled for significant XML compliance in version 5.0. IBM has produced XML tools and Sun has put its extensive Solaris documentation into XML.

The power and relative simplicity of XML have led to its endorsement by IBM, Netscape, Microsoft, Sun, Adobe, and a host of other institutions and individuals prominent in the information industry. Developers are creating XML tools and XML workshops are being held around the country. It is expected that XML will be used instead of HTML for many Web pages and will be used for many printed

[2] See Michael Downes, *Breaking Equations*, *TUGboat* 18, 3, September 1997, pages 182-194. A new release of the software is expected in early August, 1998 (private communication from M. Downes). We hope that this work, so important for the Web, can eventually be made available for use in XML.

[3] A publishing house use of XML to produce both HTML and RTF versions, the former for a Web version and the latter for the commercial printed-book version, can be seen in some detail at `www.mcp.com/info/1-57521/1-57521-334-6`. That example is also interesting because it includes use of TEI, the Text Encoding Initiative, and because it treats XML as a special case of SGML, the Standard Generalized Markup Language.

publications. LaTeX may turn out to be an application of choice for printing XML documents, especially those involving math. The feeling of some XML working groups and developers seems to be that true XML Web browser and print applications, including math, formatting, linking, data, pointer, and document architecture, will gradually become usable starting next spring.

Math in XML

The XML math markup language, called MathML, has already been incorporated into several tools.[4] Although MathML makes sense in terms of the ambitious goals of the MathML working group, it is rather laborious to write and difficult to proof-read. In an example from IBM,[5] markup for the quadratic root formula, LaTeX takes one line while the Presentation form of MathML takes 35 lines:

```
LaTeX: $$x=-b\pm\sqrt(b^2-4ac)/(2a)$$
MathML:
       <mrow>
         <mi>x</mi>
         <mo>=</mo>
         <mfrac>
           <mrow>
             <mrow>
               <mo>-</mo>
               <mo>b</mo>
             </mrow>
             <mo>&PlusMinus;</mo>
             <msqrt>
         ...(22 lines)...
         </frac>
       </mrow>
```

As a result of this complexity, it has been proposed that LaTeX or another math markup language might be used in XML documents with "helper applications" converting it "on the fly" to MathML for processing by the user's XML browser. It is assumed that LaTeX is exactly equivalent to Presentation MathML.[6]

Meanwhile: LaTeX and techexplorer

While waiting for XML to become usable for documents that contain math, we are using LaTeX and IBM's **techexplorer**[7] to produce Web and print versions of a physics textbook. The LaTeX compiler combines its own style files and the one master file to produce the `.dvi` file for the printed version. **techexplorer** is a plug-in for current browsers that combines its own "macro" style file with the one master file, on the user's machine and in real time, to produce the on-screen version. The LaTeX compiler and the **techexplorer** interpreter can work from the same master file because **techexplorer** uses LaTeX's command structure and also because it recognizes many LaTeX commands. Thus many formatting macros in the LaTeX style file can be taken over directly to **techexplorer**'s macros file. **techexplorer** simply ignores the LaTeX commands in the master file that are not in its repertoire. In addition to the many LaTeX commands that it understands, **techexplorer** has commands that are useful for Web browser display and which provide some of the capability expected in XML. While we are using **techexplorer** and LaTeX, we are also using a specific LaTeX markup scheme that is designed to capture the information needed for a future conversion to XML. It is fortunate that one of XML's strongest requirements is also a requirement of LaTeX; namely, that scopes be nested (which makes possible the description of elements as distinct objects).

techexplorer's new "user-embed" link

We make considerable use of **techexplorer**'s implementation of XML's new "user-embed" link.[8] The **techexplorer** command is "`\altLink`" and it allows us to specify two hot elements (elements that are visually identifiable as clickable links) which alternate as the user clicks on them. For example, the default hot element can be the word "help" and the alternate element can be a long sequence of help

[4] A list of tools incorporating MathML is available at `http://www.w3.org/Math/`.

[5] Download **techexplorer** from `www.software.ibm.com/techexplorer` Install it as a plug-in to Netscape Navigator, then display, in the Navigator:
`Netscape/Communicator/Program/Plugins/techexplorer/Examples/MathML/mml002.html`.

[6] Another set of MathML markup, without any formatting, is called Content MathML. In contrast to Presentation MathML, Content MathML is strictly generic (formatless) markup. To see an example, display the file `mml002.html` referenced in Footnote 5.

[7] See the **techexplorer** reference in Footnote 5.

[8] XML specifies a suite of six pre-defined links and allows for custom-designed links. The built-in types are the six combinations produced by combining the "show" attributes "auto" and "user" with the "actuate" attributes "replace," "new," and "embed." Here "auto" and "user" indicate who controls activation of a link, while "replace," "new," and "embed" indicate the action to be taken when a link is activated. Whereas "replace" and "new" switch to a different flow of information, one in the current browser window and the other in a new window, "embed" causes the link-targeted object to be seamlessly incorporated into the current flow of information at some designated spot just as though the targeted element had always been there. Another part of the XML specification says that the element to be embedded need only be an identifiable element, not a complete file.

that includes text, graphics, and interactive computer programs. When the user clicks on the hot word "help," that word is instantly replaced by the actual help sequence which is sometimes quite long. The insertion is downward from the point of the default element, with the elements above the point of insertion remaining fixed in position on the screen. The actual help sequence, no matter how long, is also visually identified as a hot element so the user can click on it and cause it to disappear and be replaced by the first alternative, the single hot word "help."

Our use of user-embed links

We use **techexplorer**'s version of the user-embed link to let the user bring in objects that in print would only be referred to, not displayed, after their first occurrence. For example, the first time Figure 6 is referred to in a print version, it is displayed. Thereafter, however, the print version will merely show the words "...Figure 6 ..." and it is up to the reader to turn back and find the appropriate page to see the figure. Using the user-embed link, however, the screen version has all references beyond the first as hot elements that can bring in the actual figure, complete with caption, and then take it out again, all at the user's discretion. Similarly, the displayed figure can be clicked on to be exchanged with an enlarged version for detailed examination. References to previously-encountered equations and definitions are also shown as user-embed links that will alternate the reference to the object, usually hot words, with the actual object. Finally, we use user-embed links for objects that are not displayed at all unless or until the user wants to see them: answers to problems, helpful hints at specific points in the discussion or in homework problems, additional problems to practice specific skills, tutorials that provide additional instruction for students that need it, answers to problems in the tutorials, and items in the chapter summaries.

Separating form from content

The feedback we have received over the years from students and instructors, along with insights from research, have led to a never-ending stream of alterations of the contents of the book we have been converting for print and Web. These continuous alterations have led us to the removal of all formatting commands from the content files and the placing of them in a separate style file, a procedure long advocated by experts and which is advocated by virtually all XML developers. One justification for this separation becomes evident when even a small revision upsets the formatting for the entire remainder of an unseparated document. It is best to save time and frustration by letting the the LaTeX compiler handle the reformatting using a style file. To make the decisions involved, the LaTeX compiler must be informed of the type of each element in the content file. This is accomplished by making each element be the argument of a LaTeX command whose name labels the type of the element. Thus, for example, the title of a book could be the argument of a "`\BookTitle`" command in the content file and this might be converted to a "`\textit`" command in the style file. In general, the style file should give LaTeX all the information it needs to make an appropriate formatting decision for each type of element that occurs in the book and for each type of formatting situation in which LaTeX might have to format that type of element. The complete separation of the content from the format instructions has the added benefit of enforcing 100% conformity with the publisher's and author's desired format for each of the various types of elements in the book. This enables the user to immediately and reliably recognize the intent of an element just from its appearance. It also allows the author or publisher to easily change the format of all members of a particular class of element.

Problems in separating the content

It is well known that one cannot completely separate content from style within the confines of the current LaTeX compiler used for print versions of books, but our experience is that such separation can easily be made complete for the screen version within the confines of the current **techexplorer**. The reason for this difference is mainly that the screen version has no page ends (**techexplorer** ignores page-end commands) whereas a number of page-end formatting "tweaks" must be put into the content file for the print version. Even experts have this problem. In *The LaTeX Companion*, Goosens, Mittlebach, and Samarin remark that they inserted 237 commands in the book's content files to over-rule formatting decisions that were made by the LaTeX compiler as it followed the instructions the authors had placed in the book's style file.[9] We hope that the table of tweaks shown in that book can sometime be used by a LaTeX expert to give us some commands which will cover the situations the authors (and we) have encountered.

[9] See *The LaTeX Companion*, M. Goosens, F. Mittlebach, and A. Samarin, Addison-Wesley, Reading, MA, 1994, second page after page 528.

Moving the markup toward XML

Eventually we will be able to encode our content files in XML to produce both the print and screen versions, and we are moving toward that capability by capturing some of the necessary information in our master content documents. To move our files in that direction while retaining our LaTeX and **techexplorer** capabilities, we followed these procedures: (1) We removed *all* formatting instructions from the content (".tex") files. (2) We made each content element's type identifier be a "backslash" command with braces around its argument. Here are some examples using names that seemed reasonable to us:

$...$ ⇒ \m{...}
%... ⇒ \rem{...}
each paragraph ⇒ \p{...}.

(3) We put, near the head of the style file, each content type identifier in a single line with either a simple format definition or the name of a more complex formatting macro (the third case below):

```
\newcommand{\m}[1]{$#1$}
\newcommand{\BookTitle}[1]{\textit{#1}}
\newcommand{\Def}[2]{\DefF{#1}{#2}}.
```

(4) We put, near the head of the content file, definitions of elements that may be used more than once such as figures, definitions, and equations. Here is an example of a definition which appears in a box that is labeled "C-1" in the right margin of both the print and screen versions:

```
\newcommand{\DefWrdC1}{mass}
\newcommand{\DefDefC1}{Mass is...};
```

Each figure contains a graphic and a caption. The graphic part is an `eps` file for LaTeX and a `gif` file for **techexplorer**. These graphics files are called "external entities" in XML and they require special markup in both LaTeX and **techexplorer**.

Markup of figures, without \ifthenelse

At the present time, **techexplorer** does not have the `\ifthenelse` and `\equal` commands that come with the LaTeX `IfThen` package. This forces us to write out figure references in messy detail.

Here is a fragment of the list of figure captions and figure graphics files that we put at the head of the content document (with `\nc` indicating `\newcommand`):

```
...
\nc{\figEbGrap...
\nc{\figEcCapt}{Fig. E-3. This fig...}
\nc{\figEcGrap}{m407gr19}
\nc{\figEdCapt...
...
```

This shows data for parts or all of figures 2, 3, and 4 in the document's Section E. Numbers are not allowed in LaTeX command names so lower case letters have been used instead: `b` in place of `2`, etc.

Here is an XML equivalence for the figure graphics command:

```
<!ENTITY figEcGrap SYSTEM "m407gr19.gif"
                                NDATA GIF>
```

where the first pair is the object data (type and name), the second pair is entity-retrieval data (attribute and value), and the third pair is application data (type and application). Here "NDATA" indicates "notation data."

Here is the markup at the place the figure is first mentioned in the document, the place where the figure will naturally appear:

```
\Fg{\figEcCapt}{\figEcGrap}
```

Next we have the markup to be placed at succeeding references to the figure. During LaTeX processing for print, the third argument, the reference to the figure, will simply be printed. During **techexplorer** processing for the Web, reference to the figure will be a hot word whose selection will cause its replacement by the actual figure as a hot object (click on the figure and it instantly goes back to being the third-argument hot word):

```
\FgRef{\figEcCapt}{\figEcGrap}{Fig. E-3b.}
```

To finish the markup, here are the LaTeX style file definitions for the print version of the document, with `\nc` again indicating `\newcommand` and with a period on each side of the figure caption indicating code that is unrelated to the issues being discussed:

```
\nc{\Fg}[2]{.#1.\epsfig{file=#2.eps}}
\nc{\FgRef}[3]{#3}
```

Finally, here are the **techexplorer** style file definitions for the Web/CD version of the document, with more code being shown because it may be less familiar:

```
\nc{\Fg}[2]{
  \fcolorbox{black}{green}{
    \begin{tabular}{l p{0cm}}
    \fbox{\includegraphics{#2.gif}} & #1\\
    \end{tabular}
  }
}
\nc{\FgRef}[3]{\altLink{\Fg{#1}{#2}}{#3}}
```

Note the tabular attribute `p{0cm}` which tells **techexplorer** to format the figure caption using all of the remaining horizontal space in the browser window at the moment. Also note the `\altLink` command that displays the third `\FgRef` argument, the figure reference, as a hot word. Its selection by the user

will cause the reference to be replaced with the first and second `\FgRef` arguments, the actual figure, as a hot object. Subsequent selection of the figure will cause it to change back to being just the reference.

Markup of figures, with `\ifthenelse`

If and when `\ifthenelse` and `\equal` are implemented in **techexplorer**, the figure references can be made simpler in two ways: (1) we can use the usual LaTeX simulation of associative arrays to identify a figure by a simple ID; and (2) we can write the first- and consecutive-figure references as one command, branching inside the associated macro on whether the hot-word argument is empty or not. Here is a fragment of the set of figure data at the head of the document, simulating an associative array:

```
\nc{\fig}[2]{
  ...
  \ifthenelse...{E2}...
  \ifthenelse
    {\equal{#1}{E3}}
    {\Fg{Fig. E-3...}{m407gr19}{#2}}{}
  \ifthenelse...{E4}...
  ...
}
```

Here is the first text reference to the figure, where the empty second argument indicates that the figure is to appear here and there is to be no user choice:

```
\fig{E3}{}
```

Finally, here is the subsequent reference which will appear to the user as the hot word contained in the second argument and whose activation by the user will instantly replace the hot word with the actual figure as a hot object (click on the hot figure and it instantly goes back to being the second-argument hot word):

```
\figRef{E3}{Fig. E-3b.}
```

Dealing with our upgrade-process errors

During the rather lengthy upgrading toward XML, our LaTeXfiles were also undergoing continual content revision and had to be continuously available for the usual LaTeX printing. We found this to be workable providing: (1) we first made any markup change to one element and then checked that the change had occurred properly before applying it to all occurrences of the same type of element; (2) after each markup change to all elements of the same type, we checked the changes in somewhat random places through visual checking of appropriate `.dvi` files; (3) we kept a log of the markup changes made each day, recording them in a lab notebook; and (4) we had our office server make backup copies of all files in the middle of each night. The main use of the "markup changes log" was in handling cases where the markup changes we made were irreversible and turned out to be erroneous. When that happened, and it did happen, we could bring back the previous day's backup files and then repeat the good changes noted in the log (we saved the code used for each change). However, we did learn the hard way to check that the correct backup tape was in the DAT drive before we went home each night.

The software we used

For search and examination through the file system we used the programmer's editor called TextPad,[10] and for making changes to all items having a common pattern of characters we used Perl.[11] Our Perl script used macros that find elements delineated by braces that may themselves contain arbitrary numbers and levels of nested elements. We intend to use Perl to convert from LaTeX braces to XML angle brackets when the proper time arrives.

[10] See `www.textpad.com`.
[11] See `www.ActiveState.com`.