Static Site Maintenance

Timothy Vismor

May 2009

Abstract

Procedures for maintaining the portions of this website (vismor.com) that are served as static markup which is embedded in dynamically generated XHTML pages. The document is primarily for in-house use, but it contains some tidbits concerning the integration of LaTeXML, MultiMarkdown, and Scrivener that may prove useful to a wider audience.

Note

This document is retained for archival purposes. It does not describe current practice.

Contents

List of Tables

1 Introduction

This article describes manual operations that are required to publish static pages on this web site. We use MultiMarkdown documents maintained in Scrivener for content management. Currently, two types of static content are maintained in this environment:

1. Normal pages intended for web-only publication, and
2. Web documents suitable for both web and print publication.

They differ in that web documents are published as both XHTML pages and as PDF files. The publication workflow also differs between these cases.

This document does not discuss development and maintenance of the PHP infrastructure that drives this site. Suffice it to say, the PHP code is hand crafted and maintained on OS X using Coda as the text editor and IDE.

All site content (both PHP code and static material) is under version control with Subversion. Although comfortable with the Subversion command line, we prefer to use Versions for most repository maintenance tasks.

2 Web Publication

Following creation or update of the static content of a web page in Scrivener, the procedure for publishing the page to the web is as follows:

1. Compile a draft of the page selecting the MultiMarkdown -> HTML option. Make sure that the page of interest (and only the page of interest) is selected for export.
2. Copy the generated XHTML to the appropriate directory in the development web site.
3. Edit the HTML file with Coda. Delete everything preceding the first `<h1>` tag.
4. Delete the `</body>` and `</html>` tags at the end of the document.
5. Test the page locally.
6. If everything is good, publish the updated page to the production site. Otherwise, go back to Scrivener and correct the document, then repeat this procedure.

If we were a little more adept at declarative languages (i.e. XSL), we suspect that manual editing of the XHTML file (steps 3 and 4) would be unnecessary.

2.1 PHP Infrastructure Note

All pages on the web site are served as PHP. The typical scenario for static page publication is one or more static pages are housed in each directory. The directory also contains an "index.php" file that

1. Performs all necessary preliminary operations (e.g. session control, content negotiation, authentication, loading appropriate stylesheets and javascript.).
2. Displays the static xhtml content.
3. Performs all necessary clean up operations (e.g. displays the page footer, closes the page syntactically).

The PHP index file is created once, customized for the content page, and maintained with Coda. It rarely changes. As indicated above, modification of content is through Scrivener. The PHP infrastructure only changes when new pages, features, and facilities are added to the web site.

3 Document Publication

The creation or update of "web documents" in Scrivener has a slightly different workflow:

1. Enter the content into Scrivener as MultiMarkdown.
2. Compile a draft of the monograph as a LaTeX document.
3. Convert the LaTeX document to XHTML using LaTeXML.

LaTeXML is used in the workflow since many of the documents have extensive mathematical content, which is handled quite nicely by LaTeXML. LaTeXML also has the ability to break longer documents and books into an interlinked series of files (e.g. chapters or sections) with a global table of contents, links, references, etc. This feature is used frequently — in this document for example — and allows (in combination with Scrivener) an almost complete separation of form and content in complex technical documents, which is no small feat.

LaTeXML was developed by Bruce Miller at NIST (the U. S. National Institute of Standards and Technology) for publishing the NIST Digital Library of Mathematical Functions (DLMF). It is distributed as public domain software by NIST.

3.1 Simple Documents

Getting back to the topic at hand, the specific procedure for publishing a *simple* "*web document*" (i.e. a document is published as a single XHTML file) follows:

1. Compile a draft of the document selecting the MultiMarkdown -> LaTeX option. Typically, each document is maintained as a separate Scrivener project.
2. Edit the generated `tex` file with TexShop.
3. Move the abstract of the document to its proper location in the `tex` file, i.e. between the `\maketitle` and `\thispagestyle{empty}` LaTeX commands.
4. Save the `tex` file.
5. Typeset the document twice to generate a PDF file with a complete table of contents, list of figures, etc. The second typeset operation is necessary to resolve complex cross references.
6. Run the bash shell script that executes LaTeXML for the current document. This generates a PHP file that contains a XHTML version of the LaTeX document.
7. Copy the generated XHTML to the appropriate directory of the development web site. The markup is in one file with a `php` suffix and one or more `png` files (`x1`, `x2`, ... `xn`) — if the document has figures.
8. Edit the PHP file with Coda. Delete the first line of the file (the xml prolog). We provide this through the PHP infrastructure when appropriate.
9. Delete the lines containing the `<html>` tag and the `<head>` tag (lines 11 and 12 after step 8) of the XHTML file.
10. Move the copyright paragraph to the end of the abstract `<div>`.
11. Test the page locally.
12. If everything is good, publish the updated page to the production site. Otherwise, go back to Scrivener and correct the document, then repeat this procedure.
13. Copy the PDF file generated in step 5 to the appropriate location in the site's public download area.

One detail of this process was glossed over in step 6 of the document publication procedure. To avoid unnecessary typing for hunt-and-peck typists, a short bash shell script is maintained for each document. It calls two LaTexML command line operations (`latexml` and `latexmlpost`) with the appropriate parameters for the document in question.

3.2 Complex Documents

The procedure for publishing a *complex* "*web document*" (i.e. a document that is published as multiple interconnected files with a table of contents) is slightly more complicated in that each generated file is touched manually. Follow steps 1 - 6 from the simple document procedure, then continue as follows:

1. Copy the generated XHTML to the appropriate directory of the development web site. The markup is in several files with `php` suffixes (`doc_name`, `S1`, `S2`, ... , `Sn`) and one or more `png` files (`x1`, `x2`, ... `xn`) — if the document has figures.

2. Edit each PHP file with Coda.

    1. Delete the first line of the file (the xml prolog).
    2. Delete the lines containing the `<html>` tag and the `<head>` tag (lines 11 and 12 after step 2a) of the markup file.

3. While editing the abstract/table of contents file (`doc_name.php`), move the copyright paragraph to the end of the abstract `<div>`.

Complete the task by following steps 11 - 13 from the simple document procedure.

After you've made modifications to document in Scrivener, it takes 5-10 minutes to generate a complex document with 10 sections, publish it on the web, and publish it as as a typeset, cross referenced, PDF document with a table of contents, bibliography, list of figures, etc. While not exactly automatic, it's not too shabby.

4 Tool Customization

The standard Scrivener/MultiMarkdown installation was modified to suit the needs of this web site. Several changes were driven by the fact that some of the LaTeX styles and classes that MultiMarkdown uses by default are not supported by LaTeXML. Other changes were related to missing features (from our perspective) in MultiMarkdown. Additionally, minor incompatibilities in MultiMarkdown's support libraries were encountered (and repaired). Brief descriptions of the modifications follow.

4.1 MultiMarkdown to Latex Transformation

The first category of revisions involves changes to the default *MultiMarkdown to LaTeX transformation*. MultiMarkdown implements much of this transformation through XSL. The following items were implemented through changes to MultiMarkdown's XSL transforms.

1. MultiMarkdown bases its document format on the LaTeX `memoir` class. LaTeXML does not support this class. It supports the older `article` and `book` classes. Rather than write `memoir` support for LaTeXML, MultiMarkdown's XSLT files were modified to use the LaTeXML compatible classes.

2. MultiMarkdown table support is problematic, for our purposes, in two ways:

   - MultiMarkdown bases its table support on the LaTeX `tabulary` class. LaTeXML does not support this class. It supports the older `tabularx` and `tabular` classes. Rather than write `tabulary` support for LaTeXML, MultiMarkdown's XSLT files were modified to use LaTeXML compatible classes.
   - MultiMarkdown tables do not support table notes (i.e. footnotes associated with a specific table) well. To overcome this limitation, we introduce the LaTeX `threeparttable` class into the XSL transform. To take advantage of this feature, tables must be coded as LaTeX and passed on directly to the LaTeX processor. Raw LaTeX is included in MultiMarkdown source by enclosing it in HTML (SGML) comment delimiters. In other words, the contents of HTML comments are not processed by MultiMarkdown.

3. MultiMarkdown disables the table of contents, the list of figures, etc. in its `article` class. MultiMarkdown's XSLT files were modified to produce documents containing the required entities.

4. We modified MultiMarkdown's `hyperref` processing instructions to suit our tastes.

5. MultiMarkdown does not support the LaTeX `listings` style, references to listings, or generating a "list of listings" in its `article` class. Among other things, we use the `listings` style to implement algorithmic displays. MultiMarkdown's XSLT files were modified to enable this support.

6. Our documents are generated using the LaTeX derivative XeLaTeX (XeLaTeX provides unicode support and native font support for LaTeX). MultiMarkdown is not, by default, set up to support XeLaTeX. Its most obvious omissions on this front are as follows.

    - The default MultiMarkdown transformation does not invoke some of the base packages recommended for use with XeLaTeX (e.g. `xunicode` and `xltxtra`).
    - The default MultiMarkdown transformation does not initialize the XeLaTeX font system. To this end, we use the `mathspec` and `fontspec` packages.

7. MultiMarkdown does not, by default, include support for the `amsmath` package which provides enhanced support for typesetting mathematics. MultiMarkdown's XSLT files were modified to provide `amsmath` support..

We have posted an example that illustrates these modifications to the default MultiMarkdown `article` configuration. You can find the modified XSLT files in the MMD revisions folder of our public download area. It consists of two files: `article_aft.xslt` and `article_aft_xhtml2latex.xslt`. These files transform a MultiMarkdown document into a LaTeX "article" (one-sided) that contains a table of contents, a list of figures, a list of tables, and a list of algorithms. If you are interested in the details of the changes, browse the files and search for the string *vismor*.

4.2 LaTeXML Bindings

MultiMarkdown (and our permutations of it) use several additional LaTeX styles that are not directly supported by LaTeXML. We were, however, unable to discern any significant impact of their omission on the quality of LaTeXML output. Often their functionality was provided to the XHTML output through css (e.g. `xcolor.sty`). We created stub LaTeXML bindings (`ltxml` files) for these styles — to quiet down the LaTeXML processing errors. We have posted these files to the LaTeXML bindings folder of our public download area. Only one of these files, `booktabs.sty.ltxml`, contains any functional code. It attempts to implement some of the more critical horizontal rules of the LaTeX `booktabs` style.

4.3 MultiMarkdown Libraries

We encountered a couple of problems in the MultiMarkdown to Latex transformation that we attributed to MultiMarkdown's support libraries:

1. MultiMarkdown uses the ASCIIMathML Perl module to generate MathML from MMD text. Its handling of bold math fonts is slightly incompatible with the XSLT MathML library. Changes were made to the ASCIIMathML library to work around this difficulty and accommodate our needs. We have posted these modifications to the MMD revisions folder of our public download area. If you are interested in the details of the changes, browse the file and search for the string *vismor*.

2. MultiMarkdown uses the XSLT MathML library to generate LaTeX math code from MathML. Problems were encountered using mathvariant font operations with XSLT MathML (specifically, bold math fonts). Changes were made to the XSLT MathML library to work around this bug? and accommodate our needs. We have also posted these modifications to the MMD revisions folder of our public download area. If you are interested in the nature of the changes, browse the file and search for the string *vismor*.

4.4 Other Observations

Revisions were also made to the native LaTeXML XSLT code to support inserting PHP processing instructions required by this web site into generated code. These changes are so esoteric and site specific that they are not publicly posted.

Some enhancements to the `core.css` file generated by LaTeXML were also required to prevent the algorithms displayed by the `listings` package from wrapping inappropriately — at seemingly random locations. Obviously, css modifications were also required so the markup generated by LaTeXML conforms to the conventions and visual style of this site. These adjustments were generally made in site specific stylesheets (`document.css` and `docprint.css`) that are loaded after (hence supersede) the native LaTeXML stylesheet.

5 Latex Code Customization

MultiMarkdown is quite good at generating LaTeX documents from MultiMarkdown formatted documents. However, for our purposes, it has a few shortcomings that require modification of LaTeX documents generated by Scrivener's `Compile Draft` operation. The required changes fall into eight categories:

1. *Manual markup of table notes.* MultiMarkdown's table parsing facilities do not take into account that some tables have footnotes associated with them.

2. *Manual markup of multiline equations.* MultiMarkdown's math parsing facilities do not take into account that some equations do not fit (or are not typically displayed) on a single line.

3. *Manual markup of "case" equations.* MultiMarkdown's math parsing facilities do not take into account that some equations are piecewise — do not have the same form for all elements in their domain. These are often referred to as case equations (i.e. the equations take on different forms for different conditions or cases).

4. *Manual markup of "split" equations.* MultiMarkdown's math parsing facilities do not take into account that, in some instances, multiple equations should be aligned and considered as a single equation for numbering purposes (e.g. systems of equations in linear algebra). Among other constructs, LaTeX provides a `split` operation to handle this situation.

5. *Manual markup of square brackets in math expressions.* MultiMarkdown's math parsing facilities use square brackets in its link/reference syntax. It has problems disambiguating their dual uses in equations — even if they are escaped as literals.

6. *Manual markup of limits in math expressions.* MultiMarkdown's math parsing facilities had some problems interpreting limit expressions in nontraditional locations. Occasionally, manual intervention was required.

7. *Manual markup of algorithms.* MultiMarkdown's intrinsic `verbatim` processing does not come close to being compatible with the more sophisticated `listings` or `algorithms` styles available in LaTeX for documenting algorithms.

8. *Manual markup of references to equations in lists.* This is just a bug. If you create a reference to an equation in an item in an ordered (or unordered) list, it prints as plain text, not as a reference. If you replace the MultiMarkdown reference with the appropriate LaTeX `\autoref` command, it resolves correctly.

These situations are encountered in various locations vismor.com. Specific examples of are documented in Table 1.

Table 1: Manual Markup Requirements

Document	Manual Operation	Location
Graph Algorithms	Algorithms	All algorithms
	Case equation	Adjacency Matrix - Eq 1
Matrix Algorithms	Algorithms	All algorithms
	Split equation	Linear system - Eq 1
	Case equation	Multiplicative identity
	Square brackets	Symmetric data structures
	Limits	Sparse pivoting
	Equation reference	Solving linear equations
		Factor update
Transformers	Table notes	Ybus maintenance - Tbl 4
	Case equation	Tnew - Eq 68
Transmission Lines	Table notes	Constant k - Tbl 1
		Z constants - Tbl 2
	Equation splitting	P expansion - Eq 13
		Q expansion - Eq 14
	Case equation	Impedance Matrix - Eq 19
		Potential Matrix - Eq 23

5.1 Table Notes

As mentioned above, we ultimately enter tables with footnotes into MultiMarkdown as raw LaTeX. However, we initially create these tables in MultiMarkdown using its native syntax. We then use MultiMarkdown to generate a LaTeX skeleton of the table which we modify appropriately. Our convention for entering table notes via MMD is to create one row at the bottom of the table for each note. Each of these "note rows" contains a table note in its first column.

After the `tex` file is generated, it is edited with TeXShop and the note rows are moved out of the `tabular` block to their new position following the `\medskip` LaTeX commands. This will cause the table notes to be centered beneath that table and included on the same page as the table (which is good enough for now).

For example, the LaTeX code for the final rows of a "table with notes" that was entered into MMD using these conventions appears as:

```
CDU - Conductor diameter unit \\
CRU - Conductor radius unit \\
CSU - Conductor separation unit \\
LLU - Line length unit \\
        \bottomrule
    \end{tabular}
\\\medskip\medskip
\end{minipage}
\end{table}
```

The manual fix describe in this section results in the following LaTeX markup:

```
        \bottomrule
    \end{tabular}
\\\medskip\medskip
CDU - Conductor diameter unit \\
CRU - Conductor radius unit \\
CSU - Conductor separation unit \\
LLU - Line length unit \\
\end{minipage}
\end{table}
```

5.2 Split Equations

When an equation is too long to fit on a single line, the math processing intrinsic to the MultiMarkdown bundle ignores the problem and continues on its merry way. To correct this oversight we augment the `equation` environment generated by MMD with the AMS `split` environment.

More specifically, edit the generated `tex` file with TeXShop and look for the equation of interest.

```
\begin{equation}
    ...
\end{equation}
```

Add a `split` clause around the miscreant formula.

```
\begin{equation}
\begin{split}
    ...
\end{split}
\end{equation}
```

Replace the equal sign in the equation with the text

```
= &
```

This defines the character immediately following the equal sign as an alignment point. Finally, add the the markup

```
\\
&
```

at each point that you want to break the equation. It splits the equation and aligns the remaining part of the formula to the right of the equal sign. For a practical example consider the following LaTeX equation markup generated by MultiMarkdown math support.

```
\begin{equation}
P=\frac{\pi }{8}-k\frac{cos\theta }{3\sqrt{2}}+{k}^{2}\frac{cos
\left(2\theta \right)(0.6728+ln\left(\frac{2}{k}\right))}{16}+{k}^{2}
\frac{\theta sin\left(2\theta \right)}{16}+{k}^{3}\frac{cos\left(3\theta
\right)}{45\sqrt{2}}-{k}^{4}\frac{\pi cos\left(4\theta \right)}{1536}
\end{equation}
```

To split this equation at the third order term, edit the LaTeX markup as follows. The markup changes in the equation are highlighted by enclosing them in —> <– indicators. They would not appear in the actual document.

```
\begin{equation}
\begin{split}
P= -->&<-- \frac{\pi }{8}-k\frac{cos\theta }{3\sqrt{2}}+{k}^{2}\frac{cos
\left(2\theta \right)(0.6728+ln\left(\frac{2}{k}\right))}{16}+{k}^{2}
\frac{\theta sin\left(2\theta \right)}{16} \\
-->&<-- +{k}^{3}\frac{cos\left(3\theta \right)}{45\sqrt{2}}-{k}^{4}\frac{\pi
cos\left(4\theta\right)}{1536}
\end{split}
\end{equation}
```

This split operation is provided by the amsmath LaTeX package, which is automatically included in all documents generated for this site.

5.3 Case Equations

When an equation is piecewise — does not have the same form for all elements in its domain, it is referred to as a "case equation". That is, an equation that has different forms for different conditions or cases. MultiMarkdown does not provide direct conventions for properly formatting case equations. To correct this oversight, we must insert native LaTeX markup for the case equation directly into the the MultiMarkdown text.

We usually generate properly formatted output by running MMD generated LaTeX through TeXShop and cleaning up the result. Our convention for entering case equations into MMD is

1. Separate the condition definition of a case from its mathematical formula with an ampersand (`\&`).
2. Separate each case with double backslashes (`\\`).

To correct the case equation, edit the generated `tex` file with TeXShop and look for the equation of interest. Next, locate the equal sign.

```
\begin{equation}
   ... = ...
\end{equation}
```

Add a `begin{cases}` clause immediately after the equal sign and an `end{cases}` before the end of the equation

```
\begin{equation}
  ...   =\begin{cases}
  ...
\end{cases}
\end{equation}
```

Now remove the `\` from each `\&` and replace each `\\` with `\\` followed by a newline.

Consider the following example. A simple equation with two cases (adhering to our case markup conventions) generates the following LaTeX code when processed by MultiMarkdown

```
\begin{equation}
\mathbf{{z}_{ij}}={R}_{ii-g}+j{X}_{ii-g}\&i=j\\{R}_{ij-g}+
     j{X}_{ij-g}\&i\ne j
\end{equation}
```

We modify the generated LaTeX code as follows to create a pair of equations.

```
\begin{equation}
\mathbf{{z}_{ij}}=\begin{cases}
{R}_{ii-g}+j{X}_{ii-g} & i=j\\
{R}_{ij-g}+j{X}_{ij-g} & i\ne j
\end{cases}
\end{equation}
```

This `cases` environment is provided by the amsmath LaTeX package, which is automatically included in all documents generated for this site.

5.4 Algorithms

Since we required captions, labels, and references to algorithms as well as keyword processing in algorithmic text, we were unable to use the intrinsic verbatim processing supported by MultiMarkdown. After looking into the issue, we concluded that both the Latex `algorithms` and `listings` packages would suit our needs. Since native LaTeXML only supports the `listings` package, our decision tree was simplified. It was decided that we would markup all algorithms using raw LaTeX taking advantage of the `listings` style.

This was possible because MultiMarkdown allows you to pass raw LaTeX to the TeX engine by enclosing the text in html style comments, i.e.

```
  <!---  Raw LaTeX text stream --->
```

The string "Raw LaTeX text stream" gets passed to the LaTeX engine without MMD intervention. A typical example of algorithmic text entry in Scrivener follows.

```
 <!---
\lstset{emph={first,insert,next}, emphstyle=\itshape}
\begin{lstlisting}[float,mathescape,caption={Adjacency List},label={adjlist}]

i = first( V )
```

```
while k $\ne$ eol
    insert( A,v,header )
    k = next( k )
k = first( E )
while k $\ne$ eol
    insert( A,e.v,e.w )
    insert( A,e.w,e.v )
    k = next( k )

\end{lstlisting}
--->
```

Since MultiMarkdown does not manage the labels of algorithms coded in this manner. You must enter all references to these entities as native LaTeX. For example, you would reference the preceding algorithm in a MultiMarkdown document as follows:

```
  in Algorithm <!---\ref{adjlist}---> which updates
```

5.5 Equation References in Lists

The workaround for MultiMarkdown's failure to resolve equation references in lists (ordered or unordered), is similar to the technique for referencing algorithms described in the previous section. For example, the following snippet illustrates references to equations labeled `forward_sub` and `back_sub` in an ordered MultiMarkdown list.

```
1. Compute an LU decomposition of bA.
2. Solve <!---\autoref{forward_sub}---> for c by
   forward substitution.
3. Solve <!---\autoref{back_sub}---> for x by back
   substitution.
```

6 LaTeX for the LaTeX Challenged

At the current time, `<!--\autoref{tbl_manualmarkuprequirements}-->` is reduced to pedantic relic. It is no longer maintained and no longer guides "real life" document publication.

After due consideration (and ample consternation), we decided not to fight the system. We now use MultiMarkdown's LaTeX pass thru facilitates to insert raw Latex into the MultiMarkdown documents for all "special case" situations. It was not our preferred solution, but ultimately this decision made publishing document revisions much more convenient.

Not being (nor wanting to become) LaTeX markup experts, choosing the "native LaTeX" route led to an obvious question. How do the "LaTeX challenged" create good LaTeX code for situations that are too sophisticated for MultiMarkdown?

The answer is actually quite simple. Let MultiMarkdown do the grunt work while you add the finishing touches.

The following procedure proved workable for generating LaTeX code for troublesome equations:

1. Write the equation in MultiMarkdown math mode.
2. Generate the LaTeX document.
3. Open the LaTeX document with TeXShop, typeset it.
4. Fiddle with the LaTeX until the equation looks good in the PDF document.
5. Copy the MultiMarkdown version of the equation into Scrivener's research folders (for possible future reference).
6. Copy the corrected LaTeX markup into the MultiMarkdown document (in lieu of the original MultiMarkdown equation).
7. Enclose the LaTeX equation in HTML comments.

A similar procedure was used to generate LaTeX code for tables with footnotes.

1. Enter the table using MultiMarkdown syntax.
2. Generate the LaTeX document.
3. Open the LaTeX document with TeXShop, typeset it.
4. Fiddle with the LaTeX until the table is properly typeset in the PDF document.
5. Copy the MultiMarkdown version of the table into Scrivener's research folders (for possible future reference).
6. Copy the corrected LaTeX markup into the MultiMarkdown document (in lieu of the original MultiMarkdown table).
7. Enclose the LaTeX table in HTML comments.

There is one minor complication to this approach. You can no longer use native MultiMarkdown references to these entities. The solution is rather obvious — and not too annoying once you get used to it — transform all MultiMarkdown references to the LaTeX objects into native LaTeX references.

7 LaTeXML Output Quirks

LaTeXML is an excellent tool that is a key element in the workflow used to publish documents on this site. However, we have discovered a few “peculiarities” with the XHTML output that it produces. They are documented in this section. For the record, we are using the most recently published version (0.7.0) of this software.

7.1 Table Notes

Tables with footnotes are somewhat problematic when LaTeXML is used to generate XHTML code from a LaTeX document. More specifically, the table notes are left aligned on the page, while the table is centered on the page.

The best solution devised to date — given our limited interest in LaTeXML development and debugging — is to manually edit the LaTeXML generated documents and correct the XHTML markup directly. The technique involves including all the table notes in a multiline, single celled table that is displayed immediately after the main table. This degenerate table is centered on the page and the cell text is left aligned. It’s a kluge, but the two tables end up looking like a single, footnoted table when they are displayed in a browser.

The basic template is

```
</table>
<table align="center">
<td align="left" >
...
footnote markup
...
</td>
</table>
```

Each document that needs these contortions has a *Manual Publication* section in Scrivener’s research folder detailing its manual edit requirements.

7.2 Split Equations

We have been unable to make LaTeXML observe `split` commands, even though it includes support for the `amsmath` package.

7.3 Listings

LaTeXML's support for the `listings` style (used to display algorithm's on this site) is incomplete. For example, creation of arrays of emphasis lists/styles failed when the XHTML markup was generated by LaTeXML. That being said, if you live within its limitations, LaTeXML's `listings` support is quite reliable.

7.4 Spacing Failures

LaTeXML code generated for equations containing text such as "where" and "and" clauses rarely observes the spacing that is present in the LaTeX generated PDF (e.g. «x_i = j," where " i < 10») is often displayed as «x_i = j,»where«i < 10» by LaTeXML. The only solution discovered to date is to manually edit the LaTeX documents and add extraneous spaces for use in XHTML generation by LaTeXML.

Each document that needs these contortions has a *Manual Publication* section in Scrivener's research folder detailing its manual edit requirements.

8 Mea Culpa

We would like to thank Fletcher Penny (MultiMarkdown), Bruce Miller (LaTeXML), Vasil Yaroshevich (XSLT MathML), and Mark Nodine (ASCIIMathML) for their excellent work. If these products weren't good, we wouldn't use them.

The preceding sections chronicling the modifications made to the default installations of these products are primarily compiled for in-house documentation. They are not intended to be critical, just informative. They could just as easily reflect misunderstandings on our part as implementation glitches. They could also mean that we are just trying to do things for which the software wasn't designed.

In any event, these products combine to make complex operations routine. Those of us old enough to remember earlier incarnations of generating technical documentation can but marvel.