

Chapter 1

The SubML markup language

Copyright © 2001-2006, Tony R. Kuphaldt

1.1 Introduction

SubML stands for **S**ubstitutionary **M**arkup **L**anguage. Similar in structure to an SGML-based language, SubML is intended for simple text formatting with very few frills, but providing the capability of standard font emphasis modes, itemized lists, and image inclusion.

SubML is designed so that it may be translated into practically any markup language with nothing more than some search-and-replace commands (hence the term *substitutionary*), executed in the `sed` stream editor. Rather than rely on complex translational algorithms (i.e. a Perl or Python script), the philosophy here is to design ease of conversion into the structure of the original markup so that any fool can write a `sed` script to convert to any new markup. So far, the following conversions are provided in a set of `sed` scripts supplied with this tutorial:

- SubML to T_EX
- SubML to L^AT_EX
- SubML to HTML
- SubML to plain text (ASCII)

More conversion routines are planned. As far as I can see, none of them should present any unordinary difficulties in conversion. I simply haven't got around to writing and testing all the scripts yet. These include:

- SubML to nroff/troff/groff
- SubML to Texinfo
- SubML to DocBook (SGML and/or XML)
- SubML to Lout
- SubML to Qwertz

Also, it should be fairly easy to write an XML DTD for SubML, making it directly readable by XML-compatible browsers and other software.

Platform compatibility is limited only to the availability of a `sed` binary to perform the conversion. And since `sed` is such a widely used and robust utility (free, too, thanks to the Free Software Foundation!), this should not be a problem. I've successfully "compiled" SubML documents on both Linux and Microsoft Windows 95 with equal ease.

Characters usually interpreted as *escape characters* in other markup languages like `\`, `&`, `$`, `%`, `|`, `~`, `^`, and `_` are handled without special tagging as well (100% of the time, too – this makes SubML worth \$1,000,000 & that's not all!). The only characters SubML requires you to specially code (not type verbatim in your source document) are the `<` and `>` symbols, simply because SubML itself uses them as escape characters to mark the beginning and end of tags.

1.2 Levels of sections under each chapter

This is text contained in the first true *section* of this tutorial.

1.2.1 This is the first subsection (titlebar)

This is text contained in the first *subsection* of this tutorial.

1.2.2 This is the second subsection (titlebar)

This is text contained in the second *subsection* of this tutorial.

This is the first subsubsection (titlebar)

This is text contained in the first *subsubsection* of this tutorial, which is within the second *subsection*.

1.3 Gallery of inline text formatting tricks

In this section, we will explore the various inline (embedded within a sentence) formatting commands provided by SubML.

Note that this may not be the fanciest array of formatting commands, but it should suffice for most common formatting requirements.

If the standard SubML philosophy is followed, additional formatting capabilities may be included at a later date. The only real restriction is that whatever formatting capability is added must be translatable to the desired output type (T_EX, HTML, DocBook, etc.) using nothing more than *simple* search-and-replace algorithms.

1.3.1 Sub- and super-scripting

This is a test of the sub^{scripting} and super^{scripting} capabilities of SubML. This is useful to create simple mathematical ($-2^{-3} = -0.125$) and chemical (H₂O, ⁹²U²³⁵) expressions.

While the following displays in html, it does not display properly in ps or pdf due to tex/latex errors when using the normal <subscript>, <superscript>, as above. Instead, we use <subscript2>, <superscript2>.

$$10^{\log_{10}(V_I)}$$

$$10^{\log_{10}(V_I/V_O)}$$

Un-comment line here to create error.

Note the $$ around the whole subscript and superscript line in the tutorial.sml source above.(You need to be looking at tutorial.sml) Only use this if you have tex/Latex errors, no ps or pdf. Complex mixtures of both superscripts and subscripts are a reason.

1.3.2 Boolean overline negation

Boolean negation (not) is supported in LaTeX by the `\overline{ }` command, available in the math environment. HTML provides no such support for overline. However, it is customary in some texts to indicate negation with a single quote (') post-fixed to the negated variable. Thus, we support Boolean negation in SML with the <ob> and </ob> tags (overbar) enclosing the negated variable. The sed processed Latex output will show (dvi, ps, pdf) overline negated variables, the html has the post-fixed single quote form of negation. Equations with any negated variables must be surrounded by and tags to activate the "math" environment for latex.

Any extensive use of Boolean equations should be xcircuit images so that real overlines will be available in html as well as LaTeX. The methods here are meant to support simple in-sentence Boolean expressions, not free-standing equations.

<math>Y = (<ob>A</ob> + <ob>B</ob>)</math> This markup gives the result below:

$$Y = (\overline{A} + \overline{B}) \quad \text{This result.}$$

$\langle\mathbf{Y} = \langle\mathbf{A} + \mathbf{B}\rangle\rangle$ This markup gives:

$Y = (\overline{A + B})$ This result with long overline is due to outer tags. The span of the overline is analogous to the span of a pair of bold tags. While the parenthesis are not necessary in the LaTeX rendition, they are mandatory in the "single quoted" html version to indicate the extent of the negation.

Some other examples follow:

$$Y = \overline{(\overline{A + B})} = \overline{(\overline{A B})}$$

$Y = (\overline{A \overline{BCED}})$ Incorrect in LaTeX, we wanted broken bar BC like AB.

$Y = (\overline{ABCED})$ This is incorrect in LaTeX, OK on html. We wanted broken bar between ABC.

$Y = (\overline{A \overline{B CED}})$ Like this by putting spaces between ABC. See tutorial.sml

$Y = \overline{\overline{((A (\overline{B \overline{C}}))})}}$ This is better as an xccircuit image; html is difficult to follow.

1.3.3 Emphasis fonts

Italicized, **boldface**, and underlined type are also available in SubML.

1.3.4 Special dashes

The regular dash, such as that used for hyphenation, looks-like-this. A dash specifically used for subtraction is typeset using a special SubML tag, so that 5–3 (math dash) looks distinct from 5-3 (ordinary dash). Some people don't care too much about this, so use this tag at your discretion.

Sometimes it is useful to show a pair of dashes – not the "em-dash" used in setting off a section of text like this – but a real *pair* of dashes. In this case, another special SubML tag has been created to do this — and you just read over it! I use it to denote series-connected electronic components in symbolic form. For example, a pair of resistors (R_1 and R_2) are connected in parallel with each other, but together they're in series with R_3 . Symbolically, I represent such a configuration like this: $(R_1//R_2)---R_3$.

1.4 Block formatting

An important feature I’ve found in document processing is the ability to typeset a literal segment of text. That is, a section of print in a monospaced font with all normal paragraph formatting features of the target markup language turned off.

One common usage of this feature is for the typesetting of computer programming code. An example follows:

File listing: hello.c

```
. . . . .
.
.  #include <stdio.h>
.
.  int main(void)
.  {
.    printf("\nHello, world!\n");
.    return (0);
.  }
.
. . . . .
```

The dots are inserted manually within the SubML document to “set off” the literal block of text from the rest of the document. Also, the leading dots (at very left of each line) help overcome a problem I’m having with T_EX formatting where leading spaces get discarded and everything ends up smashed against the left margin.

Without the dots, it looks like this:

```
#include <stdio.h>
int main(void)
{
printf("\nHello, world!\n");
return (0);
}
```

The “set off” leading dot may be replaced by the <sp> tag to avoid the dot in your literal block.

```
#include <stdio.h>
int main(void)
{
printf("\nHello, world!\n");
return (0);
}
```

Another kind of block formatting is the inclusion of offset quotations. Note the following example:

”Vague and insignificant forms of speech, and abuse of language, have so long passed for mysteries of science; and hard or misapplied words with little or no meaning have, by prescription, such a right to be mistaken for deep learning or height of speculation, that it will not be easy to persuade either those who speak or those who hear them, that they are but the covers of ignorance and hindrance of true knowledge.” - John Locke

Italics may also be added to “set off” a quotation from the rest of the text, especially in HTML. Combining the `italic` and `bold` tag sets inside of the `quotation` tag set accomplishes this goal nicely:

*”Vague and insignificant forms of speech, and abuse of language, have so long passed for mysteries of science; and hard or misapplied words with little or no meaning have, by prescription, such a right to be mistaken for deep learning or height of speculation, that it will not be easy to persuade either those who speak or those who hear them, that they are but the covers of ignorance and hindrance of true knowledge.” - **John Locke***

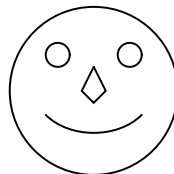
While perhaps not a true block-formatting feature, itemized lists can be created using SubML. Take the following example:

- This is the first item
- This is the second item
- This is the third item
- Isn't this fun?

In the spirit of simplicity, I haven't created the option of enumerated lists, indented lists, or anything fancy like that within the language of SubML.

1.5 Including graphic images in a document

Graphic image inclusion is perhaps the best feature of SubML. Note the following example:



Have a nice day!

You must be sure to specify an HTML-compatible image in the markup code. This means an image file specified with a filename ending in `.png`, `.jpg`, `.bmp`, or `.gif` (three-character extensions only: `.jpg`, not `.jpeg`!). For `TeX` or `LaTeX` output, there must be an Encapsulated Postscript image file `.eps` in the same directory, but not specified in the markup code.

For example, the markup code necessary to place the "happy face" image shown above is as follows:

```
<image>test.png</image>
```

Two versions of the image exist: `test.png` for inclusion into the HTML output, and `test.eps` for inclusion into the `TeX` or `LaTeX` output, but only the HTML-compatible file need be specified in the SubML source code.



Have a nice day!

Figure 1.1: *This is a fine caption.*

Below is the markup code necessary to place the "happy face" image with a caption shown in figure 1.1. A "Figure x.x" string precedes the caption in `LaTeX`. It also generates `LaTeX` code for a `//label test.eps`, which is used to reference the figure. The caption is included in the html without the "Figure x.x" designation.

```
<image>test.png<caption>This is a fine caption.</caption></image>
```


Note that in the previous paragraph, we reference "figure 1.1" or "figure above" in tutorial.ps and tutorial.html, respectively. The markup below, between the ref tags, is for referencing the above image as a figure. The image name, test.png, is a symbolic reference, replaced by 1.1, 1.2, etc., during "latex tutorial.latex" processing. Put the image name between the tags.

See figure<ref>test.png</ref> for a "happy face".

If you read about Latex figures, labels, and references, you will find that the label is completely arbitrary. The only requirement is that the //ref command must call out the label associated with the figure. In our case the sml2latx.sed file contains substitutions which fill in the image number, eg: test.png, 02041.png, for the label. Thus, we do not have to manually fill that in for each of our images, which we may or may not reference. If we do wish to reference a figure we reference the image number. It may be necessary to run "latex tutorial.latex" twice to resolve the references.

As an option for html, a word may follow the image name as below. Eg., "test.eps above" will put "above" into the tutorial.html. We have no way to generate numbered figures in the html. So, figure above, figure below, may be usefull. View tutorial.html vs tutorial.ps for "figure 1.1" vs "figure above", respectively. Here we reference figure 1.1 again, but only in tex/latex, no html as in the above markup. The markup below shows the optional html word.

See figure<ref>test.png above</ref> for a "happy face".

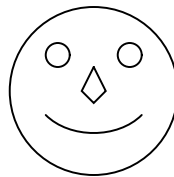
In the case of html, we do not have the referencing facilities provided by L^AT_EX. The best we can do is refer to the figure above or below as shown in the above markup.

Unrelated, take a look at tutorial.html to see how we have indented the above markup code without a leading dot. Compare to previous unindented markups.

See caution in next section: only one reference per line (pair of <ref> tags). Else, split line with (return).

1.5.1 Labeling a figure

Do not confuse the "Labeling" with the caption on a figure. In most all cases you can skip this section and let the sed processing automatically generate the label which the "figure" requires so that it may be referenced. The automatic label is the same as the image file name (eg 02221.eps). The previous section covers this. The only reason to read this section is in the rare event that a second instance of a figure is being used. In which case, it needs a new, unique, not automatically generated label, not the (automatic) label for the first instance of the image. You may also skip this, if there is no caption for the figure. We will give the second instance of the image a unique label so that it will not be confused with the first instance when we reference it. See Figure 1.2



Have a nice day!

Figure 1.2: *Caption for the second instance of our image.*

Note that our new figure is captioned as Figure 1.2. The caption is different than the caption for the previous Figure 1.1. We are able to assign a label to it:

```
<image>test.png<caption>This is a fine caption.</caption>
<label>newtest.png</label></image>
```

Note that the above markup must be on one line. It is too wide for our page. So, we wrapped it. It may wrap in the text editor. But there cannot be a (return) except at the end of the line. The sed script processes a line at a time for each command. We process all the tags in the line with one command for image, caption, label, and ref tags.

Once it has a label, we can distinguish it from the other figure by referencing it the same way we reference other figures (just a different label):

If we compare the above image caption for newtest.png to the previous caption for test.png, we find that both specify the same image, test.png. The latter has a different label "newtest.png" This is just a label. There is no image by that name.

See figure<ref>newtest.png above</ref> for a 2nd "happy face".

Caution, a limitation of the sed script for caption processing is that only one figure reference (eg.: <ref>newtest.png</ref>) may be processed properly per line. Typically, there is only one line, all the words up to the end-of-line between <para> tags. If we need more than one <ref></ref> in a paragraph, the paragraph may be split into two or more lines between the two paragraph tags. See tutorial.sml for an example of this in the paragraph "Note that our new figure is captioned. . . "

1.5.2 Scaling an image

Once in a long while, an image which is of satisfactory size in the html version of a document is too small in the LaTeX produced pdf document. The solution is to make the image the "right size" for the html document, then scale it to a suitable size in the LaTeX file. This is done by a sed (string substitution

program) command in `sml2latex.sed`. When the `sml` source is processed, a scale factor is added to the `.latex` file, but not the `.html` file.

The scale factor must be added to the `.sml` as a modification between the `<image>` tag and the file name of the image. This markup produces Figure 1.3).

```
<image>[scale=0.5]test.png<caption>This is scaled down in LaTeX.</caption><label>smalltest.png</label></image>
```

The image command must be on a single line, a CR only at the end, none in the middle. Though, we wrapped it above for appearance. And, don't put two on one line— split into two lines. This scale parameter, `[scale=0.5]`, only works if the `<caption>` tags are used, due to `sed` script limitations. The same is true of the `<label>` tags. The caption tags generate a figure number, even if there is nothing between the tag. There must be a unique label between the `<label>` tags, else LaTeX give an error. There must be no space between `[scale=0.5]` and `test.png`. LaTeX doesn't want a space in front of the image file name. It must be like this `[scale=0.5]test.png`.



Have a nice day!

Figure 1.3: *This is scaled down in LaTeX.*

1.6 Special characters

In addition to special logos like T_EX, SubML provides for certain often-used characters of the Greek alphabet.

The ratio of a circle's circumference to its diameter is symbolized by the Greek letter “pi,” which SubML represents like this: π . The area of a circle is given as $A=\pi r^2$. Not many people realize that the standard symbol π is actually the *lower-case* version of the Greek letter. The capital version looks like this: Π , and it does not represent the same thing in mathematics.

But there are other useful Greek characters for us to use in SubML as well. When SubML is converted to plain ASCII text, some of the Greek characters like μ and ρ will be represented by the closest-resembling Roman (English alphabet) character available. If there is no Roman character close enough, the Greek character's name will be spelled in parentheses. T_EX, on the other hand, is very Greek-literate and requires no “fudging” to obtain perfect representation. HTML output from SubML conversion renders these characters using Unicode. In order for a web browser to properly display them, it must be set up with Unicode character support. For your viewing pleasure, we have:

- Alpha (lower-case): α
- Beta (lower case): β
- Gamma (lower case): γ Gamma (capital): Γ
- Delta (lower case): δ Delta (capital): Δ
- Epsilon (lower case): ϵ
- Varepsilon (lower case): ε
- Zeta (lower case): ζ
- Eta (lower case): η
- Theta (lower case): θ Theta (capital): Θ
- Vartheta (lower case): ϑ
- Iota (lower case): ι
- Kappa (lower case): κ
- Lambda (lower case): λ Lambda (capital): Λ
- Mu (lower case): μ
- Nu (lower case): ν
- Xi (lower case): ξ Xi (capital): Ξ
- Pi (lower case): π Pi (capital): Π

- Rho (lower case): ρ
- Varrho (lower case): ϱ
- Sigma (lower case): σ Sigma (capital): Σ
- Varsigma (lower case): ς
- Tau (lower case): τ
- Upsilon (lower case): υ Upsilon (capital) Υ
- Phi (lower case): ϕ Phi (capital): Φ
- Varphi (lower case): φ
- Chi (lower case): χ
- Psi (lower case): ψ Psi (capital): Ψ
- Omega (lower case): ω Omega (capital): Ω
- non-breaking space 1 1 1 1 2 2 2 3 3 3 4 4 4 4
- Tau (lower case): τ
- bigtriangledown: ∇
- oplus, exclusive or sign: \oplus
- almothequal: \approx
- approxequal, approximately equal: \cong
- neq, not equal: \neq
- plusminus, plus or minus: \pm
- cdot, centered dot, times dot: \cdot
- leq, less than or equal: \leq
- geq, greater than or equal: \geq
- times, times sign: \times
- registered, registration sign: \circledR

Another special symbol available in SubML is the \angle symbol (`<angle>`), used in mathematical statements to designate an angle. This is useful for expressing complex numbers in polar form. Take for example this impedance: $500\ \Omega\ \angle -34.61^\circ$. By the way, the way I typeset the "degree" symbol is with a superscript letter "o".

Other mathematical symbols included in SubML's vocabulary are the integration symbol (\int), partial derivative symbol (∂), and the infinity symbol (∞). Here are some examples of these symbols in use:

$$V = \int Q \, dt + C$$

$$\partial x / \partial t$$

∞ is bigger than BIG!

Note that you cannot show upper and lower integration limits for a definite integral using the " \int " markup tag. It is useful for crude in-line formatting of an integral equation only. If you want to show lower and upper integration limits in a SubML document, you must use a graphic image – sorry!

For special characters used in other languages (Spanish, French, German, etc.), the following are available in the SubML vocabulary:

- "a" with grave (back prime): à Â
- "a" with acute (forward prime): á Á
- "a" with circumflex (caret): â Â
- "a" with umlaut/dieresis/tremat: ä Ä
- "a" with tilde: ã Ã
- "a" with "ring" above: å. Å
- "c" with cedilla: ç Ç
- "e" with grave (back prime): è È
- "e" with acute (forward prime): é É
- "e" with circumflex (caret): ê Ê
- "e" with umlaut/dieresis/tremat: ë Ë
- "i" with grave (back prime): ì Ì
- "i" with acute (forward prime): í Í
- "i" with circumflex (caret): î Î
- "i" with umlaut/dieresis/tremat: ï Ï
- "n" with tilde: ñ Ñ
- "o" with grave (back prime): ò Ò
- "o" with acute (forward prime): ó Ó

- "o" with circumflex (caret): ô Ô
- "o" with umlaut/dieresis/tremat: ö Ö
- "o" with tilde: ã Õ
- "u" with grave (back prime): ù Û
- "u" with acute (forward prime): ú Ú
- "u" with circumflex (caret): û Û
- "u" with umlaut/dieresis/tremat: ü Ü
- Inverted question mark ¿
- Inverted exclamation mark ¡

So, now you may impress all your Español-speaking amigos with the following phrases in your documents:

"¿Dónde está el cuarto de baño?"

"¡Más cerveza, por favor!"

"¿Puede indicarme dónde está en el mapa?"

"Por favor, dígame tu amigo que voy a llegar cinco minutos tarde."

"Aquí tiene mi casa."

And when your friend asks you this . . .

"¿Qué procesador de textos usted utiliza?"

. . . you may respond with pride:

"No utilizo un procesador de textos. ¡En lugar, utilizo un lenguaje de marcas!"

1.7 Tex/Latex only, HTML only

Tags `<tex>`, `</tex>`, `<html>`, `</html>` are provided to include text from .sml selectively only in .latex, .tex or only in .html. The `<tex>` `</tex>` tags mark text that is only included in the .latex and .tex outputs of "sed -f sml2latx.sed" and "sed -f sml2tex.sed". Text that is only to be included in the .html is marked of by the `<html>`, `</html>` tags.

This following markup is to only show text in tutorial.latex and tutorial.tex. Following the markup, see text in tutorial.latex, tutorial.tex, but not in tutorial.html

```
<tex>This only shows in tutorial.latex and tutorial.tex</tex>
```

This only shows in tutorial.latex and tutorial.tex

This following markup is to only show text in tutorial.html. Following the markup we see the text in tutorial.html but not tutorial.latex, tutorial.tex.

```
<html>This only shows in tutorial.html</html>
```

Given both a portrait and landscape version on a same-size image, a practical application of the `<tex>`, `<html>` tags is to selectively direct those images to tutorial.latex or tutorial.html. We do not actually do this in tutorial.sml, but show the markup. For example, we wish to send the landscape version of a big image to the html version of our book so that readers do not have to rotate their monitors. This landscape is too big for our .latex, .tex, .ps, .pdf 6-inch wide book pages. We cannot reduce the size of the landscape, which would be unreadable. So, we rotate our big landscape to a portrait. It started out 4-inches tall and is now 4-inches wide. It fits side ways nicely on a book page. We have not reduced the size, just rotated it. A book reader can easily rotate the book to view the large image.

```
<html><image>landscape.png</image></html>
```

```
<tex><image>portrait.png</image></tex>
```


1.8 Hyperlinks and targets

[link at](#) end of this section.

sample target located here, jump here from a link (Click) near the bottom of this section

The `<url>`, `</url>` tags provide clickable links to URLs in both the html and pdf versions of a document. The pdf is derived from LaTeX. Internal links are provided by `<hyperlink>`, `</hyperlink>` tags, which link to targets defined by the `<hypertarget>`, `</hypertarget>` tags. The syntax for these tags takes the following form:

```
<url>url_link[text]</url>
<hyperlink>link[text]</hyperlink>
<hypertarget>link[optional text]</hypertarget>
```

The "link" for `<hyperlink>` must match the "link" at the `<hypertarget>` to actually jump there on clicking. The links for `<hypertarget>` in the case of multiple targets needs to be unique—no two targets the same. The "link" for `<hyperlink>` and `<hypertarget>` may not contain any underscores, eg., `invisible_link`. Though, it works in html, the pdf links will be dead. And, no errors are generated. The `<url>` and `<hyperlink>` text will appear colored in both html and pdf when viewed. The `<hypertarget>` text is not colored, and is optional.

The following markup provides an external link to a URL in both html and pdf documents:

Go to

```
<url>http://www.ibiblio.org/obp/electricCircuits/index.htm[Lessons in
Electric Circuits]</url>
```

to learn about electricity.

Go to [Lessons in Electric Circuits](#) to learn about electricity.

Why are there no quotes around the URL above? While the quotes are needed in html code, they are not used in LaTeX. Therefore, we do not include them here. They are added by the `sml2html.sed` script to the html document.

Click this link to [jump to invisible](#) target at end of section. At the top of this section click on "link at" to also jump to the end of the section.

The following markup provides the link below it to the top of this section:

```
<hyperlink>LINK[Click]</hyperlink> to go to target at top of section.
```

[Click](#) to go to target at top of section.

Here is the markup for an "invisible" target at the end of this section:

```
<hyperlink>invisibleTarget[]</hyperlink>
```

1.9 Bibliography and citations

The `<thebibliography>`, `</thebibliography>` tags mark a section of text to be treated as a list of bibliographic references. Contained therein are individual bibliographic entries delimited by `<bibitem>``</bibitem>` tags. These entries may be referenced from the body of the main text by `<cite>``</cite>` tags. The syntax of these tags is as follows:

```
<thebibliography>
  <bibitem[ref]text</bibitem>
  <bibitem[ref2]text2</bibitem>
</thebibliography>
```

The purpose of this paragraph is to reference the bibliography below. This paragraph is broken into several lines terminated by a return.^[2] You should skip to the bibliography and look at the first entry, here.^[1] The second entry in the bibliography is here.^[3] Note that the fourth bibitem contains a url to link to home of this project.^[4]

The bracketed reference, [ref], in the bibitem needs to be matched by the corresponding citation reference `<cite>ref</cite>` in the body of the text. See above and below. In LaTeX, this is usually an easy to remember mnemonic. This is replaced by bracketed a number, eg. [2], in the processed LaTeX version of the document. However, the html version of the document will not have numbers unless the reference is a number, eg. `<cite>4</cite>`. The bibliography in html is a numbered list. However, these numbers do not necessarily correspond to the sml bibitem reference. Use numbers instead of mnemonics in the bibitem reference for numbers in the html.^[4]

A sample bibliography with four items follows:

Bibliography

- [1] Helmut Kopka and Patrick W. Daly, A Guide to LaTeX: Document Preparation for Beginners and Advanced Users (Addison-Wesley, Reading, MA, 1999), 3rd. ed.
- [2] The html sed processing only handles one citation per line. Though, LaTeX can handle more.
- [3] B. C. Freasier, C. E. Woodward, and R. J. Bearman, “Heat capacity extrema on isotherms in one-dimension: Two particles interacting with the truncated Lennard-Jones potential in the canonical ensemble,” J. Chem. Phys. 105, 3686–3690 (1996).
- [4] Kuphaldt, Tony R., [Lessons in Electric Circuits](#) in the open book project at [ibiblio.org](#)

Note that the last entry above contains a url. The whole bibterm must be on one line, only one return, at the end.

1.10 What SubML won't do

SubML is designed to be a *simple* markup language, and as such it lacks certain advanced features found in other, more capable languages like T_EX or DocBook. One of these missing features is tables. However, I have found that it often works well to create a table using a graphics editor and then insert it into the document as an image. One advantage to doing tables this way is consistency in appearance between different outputs (T_EX, HTML, etc.).

Another thing SubML makes no provision for is easy, verbatim display of its own markup code. In order to show verbatim SubML code, you must mark all < and > symbols with the appropriate <lt> and <gt> tags. The following paragraph shows the markup required for this paragraph. For a really wild experience, view the source code of this file to see how I mark up *that* paragraph:

```
<para>
Another thing SubML makes no provision for is easy, verbatim display
of its own markup code. In order to show verbatim SubML code, you
must mark all <lt> and <gt> symbols with the appropriate
<lt>lt<gt> and <lt>gt<gt> tags. The
following paragraph shows the markup required for this paragraph.
For a really wild experience, view the source code of this file to
see how I mark up <italic>that</italic> paragraph:
</para>
```

I could carry the recursion one step further, but that would be cruel and unusual punishment for both of us.

1.11 How to do the conversion

First, you need to have `sed` installed and operational on your computer. Next, be sure that all conversion scripts (`sml2tex.sed`, `sml2html.sed`, etc.) have been installed in the same directory as the SubML document that you wish to convert. If you wish to convert your SubML document to \TeX , groff, or some other markup language requiring further processing, you must of course have the necessary software installed on your computer to process the markup format(s) of choice.

For instance, if you converted your SubML document into a \TeX document using the `sml2tex.sed` script provided with this tutorial, but didn't have Donald Knuth's \TeX processing system installed on your computer, all the `sed` script will do is produce a \TeX source file: a new document marked up with \TeX commands and tags in place of SubML tags. In other words, these scripts simply convert SubML source code into source code for other markup languages. With the exceptions of HTML and plain ASCII text, none of the output formats generated by these `sed` scripts will be ready-to-use.

If you wish to convert your source document (entitled `foo.sml`) to HTML, here is what you would have to type at the command prompt:

```
sed -f sml2html.sed foo.sml > foo.htm
```

The `-f` option tells `sed` to look to file `sml2html.sed` for instructions rather than take direct search-and-replace commands from the command prompt when processing the input file `foo.sml`. The output file is named `foo.htm`.

The redirection command (`>`) is necessary, otherwise `sed` will simply send the converted text to standard output (the computer's command-line screen) and all of it will flash before your very eyes instead of being saved in a file. Of course, you can name the target file anything you wish, so long as the extension is appropriate to the type of converted document that it is (i.e. `.htm` or `.html` for HTML output, so that a browser will recognize the filename).

The use of standard input and standard output in a `sed` script allows for great flexibility in the use of SubML. For instance, I have a book I'm writing (*Lessons In Electric Circuits*), in which I'm using Makefiles to direct compilation from SubML to \LaTeX and HTML. By using `stdin/stdout` redirection within the Makefile commands, I'm able to prepend and append files containing special \LaTeX and HTML code to the basic text (written in SubML format) to achieve markup capabilities beyond the basic scope of SubML. For instance, I may want to generate a coverpage for my book using a series of special \LaTeX commands. SubML doesn't specify detailed layout tags, and so I write the necessary \LaTeX code in a file that gets prepended to the `sed`-converted output of the main text body. Same for the generation of an index: a special file containing the necessary \LaTeX commands gets appended to the very end, after `sed` has converted the main body of the text. Same for navigation buttons at the beginning and end of each HTML file generated from SubML.

1.12 How mini TOC works

A mini Table of Contents (TOC) is automatically inserted after the chapter title for (1) html, (2) \LaTeX which provides dvi, ps, and pdf. There is no mini TOC support for other formats: txt, tex, or groff. This requires different packages for (1) html, (2) \LaTeX . Thus, the method of generation of the mini TOC is different for the two case. In both cases the automatic generation is initiated by the sed command file substitution for the `</chaptertitle>` tag. Other features in headers or makefiles cause the required software to generate and insert the mini TOC after the chapter title.

In the case of html, the `sml2htm.sed` file contains the `</chpatertitle>` tag substitution: `<!-->` which flags the html for inclusion of the mini TOC. We use a perl script, `htmltoc`, modified for our requirements to `htmltoc2` for placing a mini TOC at the `<!-->` tag. The original script placed the mini TOC before the chapter title. So, we modified it to place the mini TOC at our tag, which is after the title. The Makefile has a line calling with appropriate parameters:

```
./htmltoc2 -inline -noorg -toclabel " " -tocmap toc.map \
- "<!\-\\-\\-\\->" AC.1.html
```

See the documentation for details. We added the `-` parameter to the `htmltoc` perl script for our `htmltoc2` so that it looks for the quoted tag which follows it. In our case we want the mini TOC at the `<!-->` tag, so that tag with escaping backslashes follows.

The makefile for each book has a make target for each of the book chapters. The chapters for which we want a mini TOC require the above `htmltoc2` command in the make targets. We include it in chapter targets, `_1`, `_2`, etc., but not the appendix targets, `_A1`, `_A2`, `_A3`. Thus, all chapters but the appendices have a mini TOC after the chapter title. Eg., see AC/Makefile targets: `AC_14.html`, `AC_A1.latex` for chapter vs appendix.

In the case of the \LaTeX translation, `.latex`, the `</chaptertitle>` in `.sml` is replaced by `/`. See `sml2latx.sed`. This `/` tells \LaTeX where to place the mini TOC.

Also, the header, `hi.latex`, contains `\usepackage{}` and `\do` to load the package and "do" the minitable of contents respectively. The table will be placed where the `/` command is encountered in the chapter text.

Nothing unusual is required of the makefile to generate the mini TOC. However, if we do not want the mini TOC in the appendices, a sed script in each of the latex appendix targets, removes the `/` command from the `.latex`. Normal target processing, puts a chapter mini TOC in for all chapters but appendices. Eg., see AC/Makefile targets: `lines.latex`, `about.latex` for chapter vs appendix.

1.12.1 Table of contents - TOC

The LaTeX table of contents is due to commands in the `hi.latex` header file. The command `\setcounter{tocdepth}{1}` limits the depth of the TOC entries to one level below chapter. Thus, we get chapter and section entries. The file `hi_appendix`, inserted between the chapters and appendices by Makefile, sets the

counter to the chapter level with `\settocdepth{chapter}`. This leaves a single TOC entry for each appendix. The package `tocvsec2` is required to reset the counter. See `\usepackage{../bin/tocvsec2}` in `hi.latex`

The `hyperref` package (`hi.latex`) generates a list of bookmarks along the left side of the acrobat viewer. The depth of this bookmark TOC only extends to the chapter level if there is a "real" TOC. It is possible to generate expandable bookmarks to more levels, if the TOC is suppressed by removing `\tableofcontents`, `\setcounter{tocdepth}{1}`, `\settocdepth{chapter}`. At this time we opt for the printed TOC over the expanded bookmark version.