omtext: Semantic Markup for Mathematical Text Fragments in LaTeX*

Michael Kohlhase
Jacobs University, Bremen
http://kwarc.info/kohlhase

November 17, 2015

Abstract

The `omtext` package is part of the sTeX collection, a version of TeX/LaTeX that allows to markup TeX/LaTeX documents semantically without leaving the document format, essentially turning TeX/LaTeX into a document format for mathematical knowledge management (MKM).

This package supplies an infrastructure for writing OMDoc text fragments in LaTeX.

---

*Version v1.1 (last revised 2015/11/04)

Contents

1 Introduction

The `omtext` package supplies macros and environment that allow to mark up mathematical texts in SₜₑX, a version of TₑX/LᴬTₑX that allows to markup TₑX/LᴬTₑX documents semantically without leaving the document format, essentially turning TₑX/LᴬTₑX into a document format for mathematical knowledge management (MKM). The package supports direct translation to the OMDoc format [**Kohlhase:OMDoc1.2**]

2 The User Interface

2.1 Package Options

showmeta — The `omtext` package takes a single option: `showmeta`. If this is set, then the metadata keys are shown (see [**Kohlhase:metakeys:ctan**] for details and customization options).

2.2 Mathematical Text

omtext — The `omtext` environment is used for any text fragment that has a contribution to a text that needs to be marked up. It can have a title, which can be specified via the

title= — `title` key. Often it is also helpful to annotate the `type` key. The standard relations

type= — from rhetorical structure theory `abstract`, `introduction`, `conclusion`, `thesis`, `comment`, `antithesis`, `elaboration`, `motivation`, `evidence`, `transition`, `note`, `annote` are recommended as values. Note that some of them are unary relations

for= — like `introduction`, which calls for a target. In this case, a target using the `for` key should be specified. The `transition` relation is special in that it is binary (a "transition between two statements"), so additionally, a source should be specified

from= — using the `from` key.

Note that the values of the `title` and `type` keys are often displayed in the text.

display= — This can be turned off by setting the `display` key to the value `flow`. Sometimes we want to specify that a text is a continuation of another, this can be done by

continues= — giving the identifier of this in the `continues` key.

Finally, there is a set of keys that pertain to the mathematical formulae in

functions= — the text. The `functions` key allows to specify a list of identifiers that are to be

theory= — interpreted as functions in the generate content markup. The `theory` specifies a

EdN:1 — module (see [**KohAmb:smmssl:svn**]) that is to be pre-loaded in this one[1] Finally,

verbalizes= — `verbalizes` specifies a (more) formal statement (see [**Kohlhase:smms:svn**]) that

EdN:2 — this text verbalizes or paraphrases.[2]

2.3 Phrase-Level Markup

\phrase — The `phrase` macro allows to mark up phrases with semantic information. It takes

verbalizes= — an optional `KeyVal` argument with the keys `verbalizes` and `type` as above and

type=

---

[1]EᴅNᴏᴛᴇ: this is not implemented yet.
[2]EᴅNᴏᴛᴇ: MK:specify the form of the reference.

style	style, class, index that are disregarded in the LaTeX, but copied into the gen-
class	erated content markup.
index	We use the \nlex{⟨phrase⟩} for marking up phrases that serve as natural
\nlex	language examples and \nlcex{⟨phrase⟩} for counter-examples (utterances that
\nlcex	are not acceptable for some reason). In natural language examples, we sometimes
	use "co-reference markers" to specify the resolution of anaphora and the like. We
\coreft	use the \coreft{⟨phrase⟩}{⟨mark⟩} to mark up the "target" of a co-reference and
\corefs	analogously \corefs for coreference source – e.g. for an anaphoric reference. The
	usage is the following:

```
\nlex{If \coreft{a farmer}1 owns \coreft{a donkey}2,
       \corefs{he}2 beats \corefs{it}2.}
```

is formatted to

$\qquad$ *If a farmer[1] owns a donkey[2], he$_2$ beats it$_2$.*

<table>
<tr><td>\sinlinequote</td><td>The sinlinequote macro allows to mark up quotes inline and attribute them.</td></tr>
</table>

The sinlinequote macro allows to mark up quotes inline and attribute them. The quote itself is given as the argument, possibly preceded by the a specification of the source in a an optional argument. For instance, we would quote Hamlet with

```
\sinlinequote[Hamlet, \cite{Shak:1603:Hamlet}]{To be or not to be}
```

	which would appear as "*To be or not to be*" Hamlet, (Shakespeare 1603) in the
	text. The style in which inline quotations appear in the text can be adapted
\@sinlinequote	by specializing the macros \@sinlinequote — for quotations without source and
\@@sinlinequote	\@@sinlinequote — for quotations with source.

2.4 Block-Level Markup

sblockquote	The sblockquote environment is the big brother of the \sinlinequote macro.
	It also takes an optional argument to specify the source. Here the four internal
\begin@sblockquote	macros \begin@sblockquote to \end@@sblockquote are used for styling and can
\end@@sblockquote	be adapted by package integrators. Here a quote of Hamlet would marked up as

```
\begin{sblockquote}[Hamlet, \cite{Shak:1603:Hamlet}]\obeylines
  To be, or not to be: that is the question:
  Whether 'tis nobler in the mind to suffer
\end{sblockquote}
```

and would render as

$\qquad$ *To be, or not to be: that is the question:*

$\qquad$ *Whether 'tis nobler in the mind to suffer*

$\hfill$ Hamlet, (Shakespeare 1603)

| `\lec` | The `\lec` macro takes one argument and sets it as a comment at the end of the line, making sure that if the content is too long it is pushed into a new line. We use it internally for placing the of source of the `sblockquote` environment above. The actual appearance of the line end comment is determined by the `\@@lec` macro, which can be customized in the document class. |

2.5 Index Markup

The `omtext` package provides some extensions for the well-known indexing macros of LaTeX. The main reason for introducing these macros is that index markup in OMDoc wraps the indexed terms rather than just marking the spot for cross-referencing. Furthermore the index commands only indexes words unless the `noindex` option is set in the `\usepackage`. The `omtext` package and class make the usual `\index` macro undefined[3].

The `\indextoo` macro renders a word and marks it for the index. Sometimes, we want to index a slightly different form of the word, e.g. for nonstandard plurals: while `\indextoo{word}s` works fine, we cannot use this for the word "datum", which has the plural "data". For this we have the macro `\indexalt`, which takes another argument for the displayed text, allowing us to use `\indexalt{data}{datum}`, which prints "data" but puts "datum" into the index.

The second set of macros adds an infrastructure for two-word compounds. Take for instance the compound "OMDoc document", which we usually want to add into the index under "OMDoc" and "document". `\twintoo{OMDoc}{document}` is a variant of `\indextoo` that will do just this. Again, we have a version that prints a variant: This is useful for situations like this the one in Figure 1:

```
We call group \twinalt{Abelian}{Abelian}{group}, iff \ldots
```

will result in the following

 We call group Abelian, iff . . .

and put "Abelian Group" into the index.

Example 1: Index markup

The third set of macros does the same for two-word compounds with adjectives, e.g. "wonderful OMDoc document". `\atwin{wonderful}{OMdoc}{document}` will make the necessary index entries under "wonderful" and "document". Again, we have a variant `\atwinalt` whose first argument is the alternative text.

All index macros take an optional first argument that is used for ordering the respective entries in the index.

---

[3]EdNote: implement this and issue the respective error message

3   Limitations

In this section we document known limitations. If you want to help alleviate them, please feel free to contact the package author. Some of them are currently discussed in the sTeX GitHub repository [**sTeX:github:on**].

1. none reported yet

4 Implementation

The omtext package generates two files: the LaTeX package (all the code between ⟨*package⟩ and ⟨/package⟩) and the LaTeXML bindings (between ⟨*ltxml⟩ and ⟨/ltxml⟩). We keep the corresponding code fragments together, since the documentation applies to both of them and to prevent them from getting out of sync.

The initial setup for LaTeXML:

```
 1 ⟨*ltxml⟩
 2 package LaTeXML::Package::Pool;
 3 use strict;
 4 use LaTeXML::Package;
 5 use LaTeXML::Util::Pathname;
 6 ⟨/ltxml⟩
```

4.1 Package Options

We declare some switches which will modify the behavior according to the package options. Generally, an option xxx will just set the appropriate switches to true (otherwise they stay false).[4]

```
 7 ⟨*package⟩
 8 \newif\if@omtext@mh@\@omtext@mh@false
 9 \DeclareOption{mh}{\@omtext@mh@true}
10 \newif\ifindex\indextrue
11 \DeclareOption{noindex}{\indexfalse}
12 \DeclareOption*{\PassOptionsToPackage{\CurrentOption}{modules}}
13 \ProcessOptions
14 \ifindex\makeindex\fi
15 ⟨/package⟩
16 ⟨*ltxml⟩
17 DeclareOption('mh', sub {RequirePackage('omtext-mh'); });
18 DeclareOption('noindex','');
19 DeclareOption(undef,sub {PassOptions('modules','sty',ToString(Digest(T_CS('\CurrentOption'))))); 
20 ProcessOptions();
21 ⟨/ltxml⟩

22 ⟨*package⟩
23 \if@omtext@mh@\RequirePackage{omtext-mh}\fi
24 \RequirePackage{xspace}
25 \RequirePackage{modules}
26 \RequirePackage{comment}
27 \RequirePackage{mdframed}
28 \RequirePackage{latexsym}
29 ⟨/package⟩
30 ⟨*ltxml⟩
31 RequirePackage('xspace');
32 RequirePackage('modules');
33 RequirePackage('lxRDFa');
```

---

[4]EDNOTE: need an implementation for LaTeXML

```
34 RequirePackage('latexsym');
35 ⟨/ltxml⟩
```

4.2 Metadata

All the OMDoc elements allow to specify metadata in them, which is modeled by the `omdoc:metadata` element. Since the content of this element is precisely controlled by the Schema, we can afford to auto-open and auto-close it. Thus metadata elements from various sources will just be included into one `omdoc:metadata` element, even if they are supplied by different sTEX bindings. Also we add numbering and location facilities.

```
36 ⟨∗ltxml⟩
37 Tag('omdoc:metadata',afterOpen=>\&numberIt,afterClose=>\&locateIt,autoClose=>1,autoOpen=>1);
38 ⟨/ltxml⟩
```

the `itemize`, `description`, and `enumerate` environments generate `omdoc:li`, `omdoc:di` with `autoclose` inside a CMP. This behavior will be overwritten later, so we remember that we are in a CMP by assigning `_LastSeenCMP`.

```
39 ⟨∗ltxml⟩
40 Tag('omdoc:CMP', afterOpen => sub {AssignValue('_LastSeenCMP', $_[1], 'global');return;});#$
41 ⟨/ltxml⟩
```

the `itemize`, `description`, and `enumerate` environments originally introduced in the `omtext` package do double duty in OMDoc, outside a CMP they are transformed into a `<omgroup layout='itemize\|description\|enumerate'>`, where the text after the macros `\item` come to be the children. If that is only text, then it is enclosed in an `<omtext><CMP>`, otherwise it is left as it is. The optional argument of the `\item` is transformed into the `<metadata><dc:title>` of the generated `\item` element.

```
42 ⟨∗ltxml⟩
43 DefParameterType('IfBeginFollows', sub {
44    my ($gullet) = @_;
45    $gullet->skipSpaces;
46               my $next = $gullet->readToken;
47               $gullet->unread($next);
48               $next = ToString($next);
49               #Hm, falling back to regexp handling, the $gullet->ifNext approach didn't wo
50               return 1 unless ($next=~/^\\begin/);
51               return;
52             },
53  reversion=>'', optional=>1);
54 ⟨/ltxml⟩
```

4.3 Mathematical Text

We define the actions that are undertaken, when the keys are encountered. The first set just records metadata; this is very simple via the `\addmetakey` infrastructure [**Kohlhase:metakeys:ctan**]. Note that we allow math in the `title` field,

so we do not declare it to be `Semiverbatim` (indeed not at all, which allows it by default).

```
55 ⟨*package⟩
56 \srefaddidkey{omtext}
57 \addmetakey[]{omtext}{functions}
58 \addmetakey*{omtext}{display}
59 \addmetakey{omtext}{for}
60 \addmetakey{omtext}{from}
61 \addmetakey{omtext}{type}
62 \addmetakey*{omtext}{title}
63 \addmetakey*{omtext}{start}
64 \addmetakey{omtext}{theory}
65 \addmetakey{omtext}{continues}
66 \addmetakey{omtext}{verbalizes}
67 \addmetakey{omtext}{subject}
68 ⟨/package⟩
69 ⟨*ltxml⟩
70 DefKeyVal('omtext','functions','CommaList');
71 DefKeyVal('omtext','display','Semiverbatim');
72 DefKeyVal('omtext','for','Semiverbatim');
73 DefKeyVal('omtext','from','Semiverbatim');
74 DefKeyVal('omtext','type','Semiverbatim');
75 DefKeyVal('omtext','title','Plain'); #Math mode in titles.
76 DefKeyVal('omtext','start','Plain'); #Math mode in start phrases
77 DefKeyVal('omtext','theory','Semiverbatim');
78 DefKeyVal('omtext','continues','Semiverbatim');
79 DefKeyVal('omtext','verbalizes','Semiverbatim');
80 ⟨/ltxml⟩
```

The next keys handle module loading (see [**KohAmb:smmssl:ctan**]).

```
81 % \ednote{MK: need to implement these in LaTeXML, I wonder whether there is a general
82 % mechanism like numberit.}\ednote{MK: this needs to be rethought in the light of
83 % |\usemodule|. It is probably obsolete. Is this used? Is this documented?}
84 ⟨*package⟩
85 \define@key{omtext}{require}{\requiremodules{#1}{sms}}
86 \define@key{omtext}{module}{\message{module: #1}\importmodule{#1}\def\omtext@theory{#1}}
```

`\st@flow` We define this macro, so that we can test whether the `display` key has the value `flow`

```
87 \def\st@flow{flow}
```

We define a switch that allows us to see whether we are inside an `omtext` environment or a statement. It will be used to give better error messages for inline statements.

```
88 \newif\if@in@omtext\@in@omtextfalse
```

`omtext` The `omtext` environment is different, it does not have a keyword that marks it. Instead, it can have a title, which is used in a similar way. We redefine the `\lec` macro so the trailing `\par` does not get into the way.

9

```
89 \def\omtext@pre@skip{\smallskip}
90 \def\omtext@post@skip{}
91 \providecommand{\stDMemph}[1]{\textbf{#1}}
92 \newenvironment{omtext}[1][]{\@in@omtexttrue%
93 \bgroup\metasetkeys{omtext}{#1}\sref@label@id{this paragraph}%
94 \def\lec##1{\@lec{##1}}%
95 \ifx\omtext@display\st@flow\else\omtext@pre@skip\par\noindent%
96 \ifx\omtext@title\@empty%
97 \ifx\omtext@start\@empty\else\stDMemph{\omtext@start}\xspace\fi%
98 \else\stDMemph{\omtext@title}:\xspace%
99 \ifx\omtext@start\@empty\else\omtext@start\xspace\fi%
100 \fi% omtext@title empty
101 \fi% omtext@display=flow
102 \ignorespaces}
103 {\egroup\omtext@post@skip\@in@omtextfalse}
104 ⟨/package⟩
105 ⟨∗ltxml⟩
106 DefEnvironment('{omtext} OptionalKeyVals:omtext',
107   "<omdoc:omtext "
108     . "?&GetKeyVal(#1,'id')(xml:id='&GetKeyVal(#1,'id')')() "
109     . "?&GetKeyVal(#1,'type')(type='&GetKeyVal(#1,'type')')() "
110     . "?&GetKeyVal(#1,'for')(for='&GetKeyVal(#1,'for')')() "
111         . "?&GetKeyVal(#1,'from')(from='&GetKeyVal(#1,'from')')()>"
112   . "?&GetKeyVal(#1,'title')(<dc:title>&GetKeyVal(#1,'title')</dc:title>)()"
113     .    "?&GetKeyVal(#1,'start')(<ltx:text class='startemph'>&GetKeyVal(#1,'start')</ltx:text>)
114                  .     "#body"
115                .  "</omdoc:omtext>");
116 ⟨/ltxml⟩
```

4.4   Phrase-level Markup

\phrase   For the moment, we do disregard the most of the keys

```
117 ⟨∗package⟩
118 \srefaddidkey{phrase}
119 \addmetakey{phrase}{style}
120 \addmetakey{phrase}{class}
121 \addmetakey{phrase}{index}
122 \addmetakey{phrase}{verbalizes}
123 \addmetakey{phrase}{type}
124 \addmetakey{phrase}{only}
125 \newcommand\phrase[2][]{\metasetkeys{phrase}{#1}%
126 \ifx\prhase@only\@empty\only<\phrase@only>{#2}\else #2\fi}
127 ⟨/package⟩
128 ⟨∗ltxml⟩
129 DefKeyVal('phrase','id','Semiverbatim');
130 DefKeyVal('phrase','style','Semiverbatim');
131 DefKeyVal('phrase','class','Semiverbatim');
132 DefKeyVal('phrase','index','Semiverbatim');
133 DefKeyVal('phrase','verbalizes','Semiverbatim');
```

```
134 DefKeyVal('phrase','type','Semiverbatim');
135 DefKeyVal('phrase','only','Semiverbatim');
136 DefConstructor('\phrase OptionalKeyVals:phrase {}',
137       "<ltx:text %&GetKeyVals(#1) ?&GetKeyVal(#1,'only')(rel='beamer:only' content='&GetKeyVal
138 ⟨/ltxml⟩
```

\coref*

```
139 ⟨∗package⟩
140 \providecommand\textsubscript[1]{\ensuremath{_{#1}}}
141 \newcommand\corefs[2]{#1\textsubscript{#2}}
142 \newcommand\coreft[2]{#1\textsuperscript{#2}}
143 ⟨/package⟩
144 ⟨∗ltxml⟩
145 DefConstructor('\corefs{}{}',
146   "<ltx:text class='coref-source' stex:index='#2'>#1</ltx:text>");
147 DefConstructor('\coreft{}{}',
148   "<ltx:text class='coref-target' stex:index='#2'>#1</ltx:text>");
149 ⟨/ltxml⟩
```

\n*lex

```
150 ⟨∗package⟩
151 \newcommand\nlex[1]{\green{\sl{#1}}}
152 \newcommand\nlcex[1]{∗\green{\sl{#1}}}
153 ⟨/package⟩
154 ⟨∗ltxml⟩
155 DefConstructor('\nlex{}',"<ltx:text class='nlex'>#1</ltx:text>");
156 DefConstructor('\nlcex{}',"<ltx:text class='nlcex'>#1</ltx:text>");
157 ⟨/ltxml⟩
```

sinlinequote

```
158 ⟨∗package⟩
159 \def\@sinlinequote#1{``{\sl{#1}}''}
160 \def\@@sinlinequote#1#2{\@sinlinequote{#2}~#1}
161 \newcommand\sinlinequote[2][]
162 {\def\@opt{#1}\ifx\@opt\@empty\@sinlinequote{#2}\else\@@sinlinequote\@opt{#2}\fi}
163 ⟨/package⟩
164 ⟨∗ltxml⟩
165 DefConstructor('\sinlinequote [] {}',
166                 "<ltx:quote type='inlinequote'>"
167                 . "?#1(<dc:source>#1</dc:source>\n)()"
168                 . "#2"
169               . "</ltx:quote>");
170 ⟨/ltxml⟩
```

4.5 Block-Level Markup

sblockquote

```
171 ⟨∗package⟩
172 \def\begin@sblockquote{\begin{quote}\sl}
```

```
173 \def\end@sblockquote{\end{quote}}
174 \def\begin@@sblockquote#1{\begin@sblockquote}
175 \def\end@@sblockquote#1{\def\@@lec##1{{\rm ##1}}\@lec{#1}\end@sblockquote}
176 \newenvironment{sblockquote}[1][]
177   {\def\@opt{#1}\ifx\@opt\@empty\begin@sblockquote\else\begin@@sblockquote\@opt\fi}
178   {\ifx\@opt\@empty\end@sblockquote\else\end@@sblockquote\@opt\fi}
179 ⟨/package⟩
180 ⟨∗ltxml⟩
181 DefEnvironment('{sblockquote} []',
182   "<ltx:quote>?#1(<ltx:note role='source'>#1</ltx:note>)()#body</ltx:quote>");
183 ⟨/ltxml⟩
```

sboxquote

```
184 ⟨∗package⟩
185 \newenvironment{sboxquote}[1][]
186 {\def\@@src{#1}\begin{mdframed}[leftmargin=.5cm,rightmargin=.5cm]}
187 {\@lec{\rm\@@src}\end{mdframed}}
188 ⟨/package⟩
189 ⟨∗ltxml⟩
190 DefEnvironment('{sboxquote} []',
191   "<ltx:quote class='boxed'>?#1(<ltx:note role='source'>#1</ltx:note>)()#body</ltx:quote>");
192 ⟨/ltxml⟩
```

The line end comment macro makes sure that it will not be forced on the next line unless necessary.

\lec   The actual appearance of the line end comment is determined by the `\@@lec` macro, which can be customized in the document class. The basic one here is provided so that it is not missing.

```
193 ⟨∗package⟩
194 \providecommand{\@@lec}[1]{(#1)}
195 \def\@lec#1{\strut\hfil\strut\null\nobreak\hfill\@@lec{#1}}
196 \def\lec#1{\@lec{#1}\par}
197 ⟨/package⟩
198 ⟨∗ltxml⟩
199 DefConstructor('\lec{}',
200    "\n<omdoc:note type='line-end-comment'>#1</omdoc:note>");
201 ⟨/ltxml⟩
```

\my*graphics   We set up a special treatment for including graphics to respect the intended OM-Doc document structure. The main work is done in the transformation stylesheet though.

```
202 ⟨ltxml⟩RawTeX('
203 ⟨∗ltxml | package⟩
204 \newcommand\mygraphics[2][]{\includegraphics[#1]{#2}}
205 \newcommand\mycgraphics[2][]{\begin{center}\mygraphics[#1]{#2}\end{center}}
206 \newcommand\mybgraphics[2][]{\fbox{\mygraphics[#1]{#2}}}
207 \newcommand\mycbgraphics[2][]{\begin{center}\fbox{\mygraphics[#1]{#2}}\end{center}}
208 ⟨/ltxml | package⟩
209 ⟨ltxml⟩');
```

4.6 Index Markup

`\omdoc@index` this is the main internal indexing command. It makes sure that the modules necessary for interpreting the math in the index entries are loaded. If the `loadmodules` key is given, we import the module we are in otherwise all the currently imported modules. We do not have to require the module files, since the index is a the end of the document. If the `at` key is given, then we use that for sorting in the index.

```
210 ⟨*package⟩
211 \addmetakey{omdoc@index}{at}
212 \addmetakey[false]{omdoc@index}{loadmodules}[true]
213 \newcommand\omdoc@index[2][]{\ifindex%
214 \metasetkeys{omdoc@index}{#1}%
215 \@bsphack\begingroup\@sanitize%
216 \ifx\omdoc@index@loadmodules\@true%
217 \protected@write\@indexfile{}{\string\indexentry%
218 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
219 {\string\importmodules{\@ifundefined{mod@id}\imported@modules\mod@id}%
220 #2}}{\thepage}}%
221 \else%
222 \protected@write\@indexfile{}{\string\indexentry%
223 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi#2}{\thepage}}%
224 \fi% loadmodules
225 \endgroup\@esphack\fi}%ifindex
```

Now, we make two interface macros that make use of this:

`\indexalt`

```
226 \newcommand\indexalt[3][]{{#2}\omdoc@index[#1]{#3}}            % word in text and index
227 ⟨/package⟩
228 ⟨*ltxml⟩
229 DefConstructor('\indexalt[]{}{}',
230         "<omdoc:idx>"
231     .   "<omdoc:idt>#2</omdoc:idt>"
232     .   "<omdoc:ide ?#1(sort-by='#1')()>"
233       .   "<omdoc:idp>#3</omdoc:idp>"
234     .   "</omdoc:ide>"
235     ."</omdoc:idx>");
236 ⟨/ltxml⟩
```

`\indextoo`

```
237 ⟨*package⟩
238 \newcommand\indextoo[2][]{{#2}\omdoc@index[#1]{#2}}            % word in text and index
239 ⟨/package⟩
240 ⟨*ltxml⟩
241 DefConstructor('\indextoo[]{}',
242         "<omdoc:idx>"
243     .   "<omdoc:idt>#2</omdoc:idt>"
244     .   "<omdoc:ide ?#1(sort-by='#1')()>"
245       .   "<omdoc:idp>#2</omdoc:idp>"
```

```
246      .  "</omdoc:ide>"
247       ."</omdoc:idx>");
248 ⟨/ltxml⟩
```

\@twin   this puts two-compound words into the index in various permutations

```
249 ⟨∗package⟩
250 \newcommand\@twin[3][]{\omdoc@index[#1]{#2!#3}\omdoc@index[#1]{#3!#2}}
```

And again we have two interface macros building on this

\twinalt

```
251 \newcommand\twinalt[4][]{#2\@twin[#1]{#3}{#4}}
252 ⟨/package⟩
253 ⟨∗ltxml⟩
254 DefConstructor('\twinalt[]{}{}{}',
255       "<omdoc:idx>"
256      .  "<omdoc:idt>#2</omdoc:idt>"
257      .  "<omdoc:ide ?#1(sort-by='#1')()>"
258      .    "<omdoc:idp>#2</omdoc:idp>"
259      .    "<omdoc:idp>#3</omdoc:idp>"
260      .  "</omdoc:ide>"
261       ."</omdoc:idx>");
262 ⟨/ltxml⟩
```

\twinalt

```
263 ⟨∗package⟩
264 \newcommand\twintoo[3][]{{#2 #3}\@twin[#1]{#2}{#3}} % and use the word compound too
265 ⟨/package⟩
266 ⟨∗ltxml⟩
267 DefConstructor('\twintoo[]{}{}',
268       "<omdoc:idx>"
269      .  "<omdoc:idt>#2 #3</omdoc:idt>"
270      .  "<omdoc:ide ?#1(sort-by='#1')()>"
271      .    "<omdoc:idp>#2</omdoc:idp>"
272      .    "<omdoc:idp>#3</omdoc:idp>"
273      .  "</omdoc:ide>"
274       ."</omdoc:idx>");
275 ⟨/ltxml⟩
```

EdN:5   **\@atwin**   this puts adjectivized two-compound words into the index in various permutations[5]

```
276 ⟨∗package⟩
277 \newcommand\@atwin[4][]{\omdoc@index[#1]{#2!#3!#4}\omdoc@index[#1]{#3!#2 (#4)}}
```

and the two interface macros for this case:

\@atwinalt

```
278 \newcommand\atwinalt[5][]{#2\@atwin[#1]{#3}{#4}{#4}}
279 ⟨/package⟩
```

---

[5]EDNOTE: what to do with the optional argument here and below?

14

```
280 ⟨∗ltxml⟩
281 DefConstructor('\atwinalt[]{}{}{}{}',
282        "<omdoc:idx>"
283     .  "<omdoc:idt>#2</omdoc:idt>"
284     .  "<omdoc:ide ?#1(sort-by='#1')()>"
285     .    "<omdoc:idp>#2</omdoc:idp>"
286     .    "<omdoc:idp>#3</omdoc:idp>"
287     .    "<omdoc:idp>#4</omdoc:idp>"
288     .  "</omdoc:ide>"
289    ."</omdoc:idx>");
290 ⟨/ltxml⟩
```

\atwintoo

```
291 ⟨∗package⟩
292 \newcommand\atwintoo[4][]{{#2 #3 #4}\@atwin[#1]{#2}{#3}{#4}}          % and use it too
293 ⟨/package⟩
294 ⟨∗ltxml⟩
295 DefConstructor('\atwintoo[]{}{}{}',
296        "<omdoc:idx>"
297     .  "<omdoc:idt>#2 #3</omdoc:idt>"
298     .  "<omdoc:ide ?#1(sort-by='#1')()>"
299     .    "<omdoc:idp>#2</omdoc:idp>"
300     .    "<omdoc:idp>#3</omdoc:idp>"
301     .    "<omdoc:idp>#4</omdoc:idp>"
302     .  "</omdoc:ide>"
303    ."</omdoc:idx>");
304 ⟨/ltxml⟩
```

4.7   LaTeX Commands we interpret differently

The first think we have to take care of are the paragraphs, we want to generate
OMDoc that uses the `ltx:p` element for paragraphs inside `CMP`s. For that we have
modified the DTD only to allowed `ltx:p` elements in `omdoc:CMP` (in particular
no text). Then we instruct the `\par` macro to close a `ltx:p` element if possible.
The next `ltx:p` element is then opened automatically, since we make `ltx:p` and
`omdoc:CMP` autoclose and autoopen.

```
305 ⟨∗ltxml⟩
306 Tag('omdoc:CMP', autoClose=>1, autoOpen=>1);
307 Tag('omdoc:omtext', autoClose=>1, autoOpen=>1);
308 Tag('ltx:p', autoClose=>1, autoOpen=>1);
309 ⟨/ltxml⟩
```

the rest of the reinterpretations is quite simple, we either disregard presentational
markup or we re-interpret it in terms of OMDoc.[6]

EdN:6

```
310 ⟨∗package⟩
311 \def\omspace#1{\hspace*{#1}}
```

---

[6]EDNOTE: MK: we should probably let LaTeXML deal with these and allow more text in the
omdoc+ltxml.xsl

```
312 ⟨/package⟩
313 ⟨∗ltxml⟩
314 DefConstructor('\footnote[]{}',
315       "<omdoc:note type='foot' ?#1(mark='#1')>#2</omdoc:note>");
316 DefConstructor('\footnotemark[]',"");
317 DefConstructor('\footnotetext[]{}',
318       "<omdoc:note class='foot' ?#1(mark='#1')>#2</omdoc:note>");
319 ⟨/ltxml⟩
```

4.8   Providing IDs for OMDoc Elements

To provide default identifiers, we tag all OMDoc elements that allow `xml:id` attributes by executing the `numberIt` procedure below. Furthermore, we use the `locateIt` procedure to give source links.

```
320 ⟨∗ltxml⟩
321 Tag('omdoc:omtext',afterOpen=>\&numberIt,afterClose=>\&locateIt);
322 Tag('omdoc:omgroup',afterOpen=>\&numberIt,afterClose=>\&locateIt);
323 Tag('omdoc:CMP',afterOpen=>\&numberIt,afterClose=>\&locateIt);
324 Tag('omdoc:idx',afterOpen=>\&numberIt,afterClose=>\&locateIt);
325 Tag('omdoc:ide',afterOpen=>\&numberIt,afterClose=>\&locateIt);
326 Tag('omdoc:idt',afterOpen=>\&numberIt,afterClose=>\&locateIt);
327 Tag('omdoc:note',afterOpen=>\&numberIt,afterClose=>\&locateIt);
328 Tag('omdoc:metadata',afterOpen=>\&numberIt,afterClose=>\&locateIt);
329 Tag('omdoc:meta',afterOpen=>\&numberIt,afterClose=>\&locateIt);
330 Tag('omdoc:resource',afterOpen=>\&numberIt,afterClose=>\&locateIt);
331 Tag('omdoc:recurse',afterOpen=>\&numberIt,afterClose=>\&locateIt);
332 Tag('omdoc:imports',afterOpen=>\&numberIt,afterClose=>\&locateIt);
333 Tag('omdoc:theory',afterOpen=>\&numberIt,afterClose=>\&locateIt);
334 Tag('omdoc:ignore',afterOpen=>\&numberIt,afterClose=>\&locateIt);
335 Tag('omdoc:ref',afterOpen=>\&numberIt,afterClose=>\&locateIt);
336 ⟨/ltxml⟩
```

We also have to number some LaTeXML tags, so that we do not get into trouble with the OMDoctags inside them.

```
337 ⟨∗ltxml⟩
338 Tag('ltx:p',afterOpen=>\&numberIt,afterClose=>\&locateIt);
339 Tag('ltx:tabular',afterOpen=>\&numberIt,afterClose=>\&locateIt);
340 Tag('ltx:thead',afterOpen=>\&numberIt,afterClose=>\&locateIt);
341 Tag('ltx:td',afterOpen=>\&numberIt,afterClose=>\&locateIt);
342 Tag('ltx:tr',afterOpen=>\&numberIt,afterClose=>\&locateIt);
343 Tag('ltx:caption',afterOpen=>\&numberIt,afterClose=>\&locateIt);
344 Tag('ltx:Math',afterOpen=>\&numberIt,afterClose=>\&locateIt);
345 ⟨/ltxml⟩
```

The `numberIt` procedure gets the prefix from first parent with an `xml:id` attribute and then extends it with a label that reflects the number of preceding siblings, provided that there is not already an identifier. Additionally, it estimates an XPointer position in the original document of the command sequence which produced the tag. The `locateIt` subroutine is a sibling of `numberIt` as it is required as an

`afterClose` handle for tags produced by LATEX environments, as opposed to commands. `locateIt` estimates an XPointer end position of the LaTeX environment, allowing to meaningfully locate the entire environment at the source.

```
346 ⟨*ltxml⟩
347 sub numberIt {
348   my($document,$node,$whatsit)=@_;
349   my(@parents)=$document->findnodes('ancestor::*[@xml:id]',$node);
350   my $prefix= (@parents ? $parents[$#parents]->getAttribute('xml:id')."." : '');
351   my(@siblings)=$document->findnodes('preceding-sibling::*[@xml:id]',$node);
352   my $n = scalar(@siblings)+1;
353   my $id = ($node -> getAttribute('xml:id'));
354   my $localname = $node->localname;
355   $node->setAttribute('xml:id'=>$prefix."$localname$n") unless $id;
356   my $about = $node -> getAttribute('about');
357   $node->setAttribute('about'=>'#'.$node->getAttribute('xml:id')) unless $about;
358   #Also, provide locators:
359   my $locator = $whatsit && $whatsit->getProperty('locator');
360   #Need to inherit locators if missing:
361   $locator = (@parents ? $parents[$#parents]->getAttribute('stex:srcref') : '') unless $locator
362   if ($locator) {
363     # There is a BUG with namespace declarations (or am I using the API wrongly??) which
364     # does not recognize the stex namespace. Hence, I need to redeclare it...
365     my $parent=$document->getNode;
366     if(! defined $parent->lookupNamespacePrefix("http://kwarc.info/ns/sTeX"))
367       { # namespace not already declared?
368         $document->getDocument->documentElement->setNamespace("http://kwarc.info/ns/sTeX","stex
369       }
370     $node->setAttribute('stex:srcref'=>$locator);
371   }return;}
372
373 sub locateIt {
374   my($document,$node,$whatsit)=@_;
375   #Estimate trailer and locator:
376   my $locator = $node->getAttribute('stex:srcref');
377   return unless $locator; # Nothing to do here...
378   my $trailer = $whatsit && $whatsit->getProperty('trailer');
379   $trailer = $trailer->getLocator if $trailer;
380   $trailer = $locator unless $trailer; # bootstrap
381   # TODO: Both should be local, or both remote, any mixture or undefinedness will produce garba
382   my $file_path = LookupValue('SOURCEFILE');
383   my $baselocal = LookupValue('BASELOCAL');
384   # Hmm, we only care about relative paths, so let's just do a URL->pathname map
385   $file_path=~s/^\w+\:\/// if $file_path;
386   $baselocal=~s/^\w+\:\/// if $baselocal;
387   if ($file_path && $baselocal && ($locator =~ s/^([^\#]+)\#/\#/)) {
388     my $relative_path = pathname_relative($file_path,$baselocal);
389     $locator = $relative_path.$locator;
390   }
391   if ($locator =~ /^(.+from=\d+;\d+)/) {
```

```
392    my $from = $1;
393    if ($trailer =~ /(,to=\d+;\d+.+)$/) {
394      my $to = $1;
395      $locator = $from.$to;
396    } else { Error("stex","locator",undef, "Trailer is garbled, expect nonsense in stex:srcref
397  } else { Error("stex","locator",undef, "Locator \"$locator\" is garbled, expect nonsense in s
398  my $parent = $document->getNode;
399  if(! defined $parent->lookupNamespacePrefix("http://kwarc.info/ns/sTeX"))
400    { # namespace not already declared?
401      $document->getDocument->documentElement->setNamespace("http://kwarc.info/ns/sTeX","stex",
402    }
403  $node->setAttribute('stex:srcref' => $locator);
404  return;
405 }
406 ⟨/ltxml⟩#$
```

4.9 Miscellaneous

Some shortcuts that use math symbols but are not mathematical at all; in particular, they should not be translated by LaTeXML.

```
407 ⟨*package⟩
408 \newcommand\hateq{\ensuremath{\hat=}\xspace}
409 \newcommand\hatequiv{\ensuremath{\hat\equiv}\xspace}
410 \@ifundefined{ergo}%
411 {\newcommand\ergo{\ensuremath{\leadsto}\xspace}}%
412 {\renewcommand\ergo{\ensuremath{\leadsto}\xspace}}%
413 \newcommand{\reflect@squig}[2]{\reflectbox{$\m@th#1\rightsquigarrow$}}%
414 \newcommand\ogre{\ensuremath{\mathrel{\mathpalette\reflect@squig\relax}}\xspace}%
415 ⟨/package⟩
416 ⟨*ltxml⟩
417 DefMacro('\hateq','\@hateq\xspace');
418 DefConstructor('\@hateq',"\x{2259}");
419 DefMacro('\hatequiv','\@hatequiv\xspace');
420 DefConstructor('\@hatequiv',"\x{2A6F}");
421 DefMacro('\ergo','\@ergo\xspace');
422 DefConstructor('\@ergo',"\x{219D}");
423 DefMacro('\ogre','\@ogre\xspace');
424 DefConstructor('\@ogre',"\x{2B3F}");
425 ⟨/ltxml⟩
```

4.10 Finale

We need to terminate the file with a success mark for perl.

```
426 ⟨ltxml⟩1;
```