

# **omtext**: Semantic Markup for Mathematical Text Fragments in L<sup>A</sup>T<sub>E</sub>X\*

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

November 5, 2015

## **Abstract**

The **omtext** package is part of the sT<sub>E</sub>X collection, a version of T<sub>E</sub>X/L<sup>A</sup>T<sub>E</sub>X that allows to markup T<sub>E</sub>X/L<sup>A</sup>T<sub>E</sub>X documents semantically without leaving the document format, essentially turning T<sub>E</sub>X/L<sup>A</sup>T<sub>E</sub>X into a document format for mathematical knowledge management (MKM).

This package supplies an infrastructure for writing OMDoc text fragments in L<sup>A</sup>T<sub>E</sub>X.

---

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

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>The User Interface</b>	<b>3</b>
2.1	Package Options . . . . .	3
2.2	Mathematical Text . . . . .	3
2.3	Phrase-Level Markup . . . . .	3
2.4	Block-Level Markup . . . . .	4
2.5	Index Markup . . . . .	5
<b>3</b>	<b>Limitations</b>	<b>6</b>
<b>4</b>	<b>Implementation</b>	<b>7</b>
4.1	Package Options . . . . .	7
4.2	Metadata . . . . .	8
4.3	Mathematical Text . . . . .	8
4.4	Phrase-level Markup . . . . .	10
4.5	Block-Level Markup . . . . .	11
4.6	Index Markup . . . . .	13
4.7	L <sup>A</sup> T <sub>E</sub> X Commands we interpret differently . . . . .	15
4.8	Providing IDs for OMDoc Elements . . . . .	16
4.9	Miscellaneous . . . . .	18
4.10	Finale . . . . .	18

# 1 Introduction

The `omtext` package supplies macros and environment that allow to mark up mathematical texts in  $\text{\LaTeX}$ , a version of  $\text{\TeX}$ / $\text{\LaTeX}$  that allows to markup  $\text{\TeX}$ / $\text{\LaTeX}$  documents semantically without leaving the document format, essentially turning  $\text{\TeX}$ / $\text{\LaTeX}$  into a document format for mathematical knowledge management (MKM). The package supports direct translation to the OMDoc format [Koh06]

## 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 [Koh15a] 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=` key. Often it is also helpful to annotate the `type` key. The standard relations from rhetorical structure theory `abstract`, `introduction`, `conclusion`, `thesis`, `comment`, `antithesis`, `elaboration`, `motivation`, `evidence`, `transition`, `note`, `annotate` are recommended as values. Note that some of them are unary relations 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 using the `from=` key.

Note that the values of the `title` and `type` keys are often displayed in the text. 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 giving the identifier of this in the `continues=` key.

Finally, there is a set of keys that pertain to the mathematical formulae in the text. The `functions` key allows to specify a list of identifiers that are to be interpreted as functions in the generate content markup. The `theory` specifies a module (see [KGA15a]) that is to be pre-loaded in this one<sup>1</sup> Finally, `verbalizes=` specifies a (more) formal statement (see [Koh15b]) that this text verbalizes or paraphrases.<sup>2</sup>

### 2.3 Phrase-Level Markup

`\phrase` The `phrase` macro allows to mark up phrases with semantic information. It takes an optional `KeyVal` argument with the keys `verbalizes` and `type` as above and `style`, `class`, `index` that are disregarded in the  $\text{\LaTeX}$ , but copied into the gen-

---

<sup>1</sup>EdNOTE: this is not implemented yet.  
<sup>2</sup>EdNOTE: MK:specify the form of the reference.

erated content markup.

`\nlex` We use the `\nlex{<phrase>}` for marking up phrases that serve as natural language examples and `\nlcex{<phrase>}` for counter-examples (utterances that 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 use the `\coreft{<phrase>}{<mark>}` to mark up the “target” of a co-reference and 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

*If a farmer<sup>1</sup> owns a donkey<sup>2</sup>, he<sub>2</sub> beats it<sub>2</sub>.*

`\sinlinequote` 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

by specializing the macros `\@sinlinequote` — for quotations without source and `\@@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 macros `\begin@sblockquote` to `\end@@sblockquote` are used for styling and can 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

*To be, or not to be: that is the question:  
Whether 'tis nobler in the mind to suffer*

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 L<sup>A</sup>T<sub>E</sub>X. 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<sup>3</sup>.

`\indextoo` 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 non-standard 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.

`\twintoo` 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

`\atwintoo` 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.

<sup>3</sup>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].

1. none reported yet

## 4 Implementation

The `omtext` package generates two files: the  $\text{\LaTeX}$  package (all the code between `\package` and `\endpackage`) and the  $\text{\LaTeX}$ XML bindings (between `\ltxml` and `\endltxml`). 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  $\text{\LaTeX}$ XML:

```
1 \ltxml
2 package LaTeXXML::Package::Pool;
3 use strict;
4 use LaTeXXML::Package;
5 use LaTeXXML::Util::Pathname;
6 \endltxml
```

### 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).<sup>4</sup>

```
7 \package
8 \newif\ifindex\indextrue
9 \DeclareOption{noindex}{\indexfalse}
10 \DeclareOption*{\PassOptionsToPackage{\CurrentOption}{modules}}
11 \ProcessOptions
12 \ifindex\makeindex\fi
13 \endpackage
14 \ltxml
15 DeclareOption('noindex','');
16 DeclareOption(undef,sub {PassOptions('modules','sty',ToString(Digest(T_CS('\CurrentOption'))));});
17 ProcessOptions();
18 \endltxml

19 \package
20 \RequirePackage{xspace}
21 \RequirePackage{modules}
22 \RequirePackage{comment}
23 \RequirePackage{mdframed}
24 \RequirePackage{latexsym}
25 \endpackage
26 \ltxml
27 RequirePackage('xspace');
28 RequirePackage('modules');
29 RequirePackage('lxRDFa');
30 RequirePackage('latexsym');
31 \endltxml
```

---

<sup>4</sup>EdNOTE: need an implementation for  $\text{\LaTeX}$ XML

## 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  $\text{\TeX}$  bindings. Also we add numbering and location facilities.

```
32 <*ltxml>
33 Tag('omdoc:metadata',afterOpen=>\&numberIt,afterClose=>\&locateIt,autoClose=>1,autoOpen=>1);
34 </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`.

```
35 <*ltxml>
36 Tag('omdoc:CMP', afterOpen => sub {AssignValue('_LastSeenCMP', $_[1], 'global');return;});#$
37 </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='itemizedescriptionenumerate'>`, 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.

```
38 <*ltxml>
39 DefParameterType('IfBeginFollows', sub {
40     my ($gullet) = @_;
41     $gullet->skipSpaces;
42     my $next = $gullet->readToken;
43     $gullet->unread($next);
44     $next = ToString($next);
45     #Hm, falling back to regexp handling, the $gullet->ifNext approach didn't work
46     return 1 unless ($next=~/\begin/);
47     return;
48 },
49 reversion=>'', optional=>1);
50 </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 [Koh15a]. 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).

```
51 <*package>
52 \srefaddidkey{omtext}
```



```

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

```

The next keys handle module loading (see [KGA15b]).

```

77 % \ednote{MK: need to implement these in LaTeXML, I wonder whether there is a general
78 % mechanism like numberit.}\ednote{MK: this needs to be rethought in the light of
79 % |usemodule|. It is probably obsolete. Is this used? Is this documented?}
80 \end{package}
81 \define@key{omtext}{require}{\requiremodules{#1}{sms}}
82 \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`

```

83 \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.

```

84 \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.

```

85 \def\omtext@pre@skip{\smallskip}
86 \def\omtext@post@skip{}
87 \providecommand{\stDMemph}[1]{\textbf{#1}}
88 \newenvironment{omtext}[1] [] {\@in@omtexttrue%

```

```

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

```

## 4.4 Phrase-level Markup

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

```

113 \<*package>
114 \srefaddidkey{phrase}
115 \addmetakey{phrase}{style}
116 \addmetakey{phrase}{class}
117 \addmetakey{phrase}{index}
118 \addmetakey{phrase}{verbalizes}
119 \addmetakey{phrase}{type}
120 \addmetakey{phrase}{only}
121 \newcommand\phrase[2][\metasetkeys{phrase}{#1}%
122 \ifx\phrase@only\@empty\only<\phrase@only>{#2}\else #2\fi]
123 \</package>
124 \<*ltxml>
125 DefKeyVal('phrase','id','Semiverbatim');
126 DefKeyVal('phrase','style','Semiverbatim');
127 DefKeyVal('phrase','class','Semiverbatim');
128 DefKeyVal('phrase','index','Semiverbatim');
129 DefKeyVal('phrase','verbalizes','Semiverbatim');
130 DefKeyVal('phrase','type','Semiverbatim');
131 DefKeyVal('phrase','only','Semiverbatim');
132 DefConstructor('\phrase OptionalKeyVals:phrase {}',
133   "<ltx:text %&GetKeyVals(#1) ?&GetKeyVal(#1,'only')(rel='beamer:only' content='&GetKeyVal

```

```

134 </ltxml>

\coref*
135 <*package>
136 \providecommand\textsubscript[1]{\ensuremath{_{\{#1\}}}}
137 \newcommand\corefs[2]{#1\textsubscript{#2}}
138 \newcommand\coreft[2]{#1\textsuperscript{#2}}
139 </package>
140 <*ltxml>
141 DefConstructor('\corefs{}{}',
142   "<ltx:text class='coref-source' stex:index='#2'>#1</ltx:text>");
143 DefConstructor('\coreft{}{}',
144   "<ltx:text class='coref-target' stex:index='#2'>#1</ltx:text>");
145 </ltxml>

\n*lex
146 <*package>
147 \newcommand\nlex[1]{\green{\sl{#1}}}
148 \newcommand\nlcex[1]{*\green{\sl{#1}}}
149 </package>
150 <*ltxml>
151 DefConstructor('\nlex{}', "<ltx:text class='nlex'>#1</ltx:text>");
152 DefConstructor('\nlcex{}', "<ltx:text class='nlcex'>#1</ltx:text>");
153 </ltxml>

sinlinequote
154 <*package>
155 \def\@sinlinequote#1{'\sl{#1}}'
156 \def\@@sinlinequote#1#2{\@sinlinequote{#2}~#1}
157 \newcommand\sinlinequote[2]{}
158 {\def\@opt{#1}\ifx\@opt\empty\@sinlinequote{#2}\else\@@sinlinequote\@opt{#2}\fi}
159 </package>
160 <*ltxml>
161 DefConstructor('\sinlinequote [] {}',
162   "<ltx:quote type='inlinequote'>"
163     . "?#1(<dc:source>#1</dc:source>\n)()"
164     . "#2"
165     . "</ltx:quote>");
166 </ltxml>

```

## 4.5 Block-Level Markup

```

sblockquote
167 <*package>
168 \def\begin@sblockquote{\begin{quote}\sl}
169 \def\end@sblockquote{\end{quote}}
170 \def\begin@@sblockquote#1{\begin@sblockquote}
171 \def\end@@sblockquote#1{\def\@lec##1{\rm #1}}\@lec{#1}\end@sblockquote}

```

```

172 \newenvironment{sblockquote}[1] []
173   {\def\@opt{#1}\ifx\@opt\@empty\begin@sblockquote\else\begin@@sblockquote\@opt\fi}
174   {\ifx\@opt\@empty\end@sblockquote\else\end@@sblockquote\@opt\fi}
175 \}
```

**sboxquote**

```

180 \package
181 \newenvironment{sboxquote}[1] []
182   {\def\@src{#1}\begin{mdframed}[leftmargin=.5cm,rightmargin=.5cm]}
183   {\@lec{\rm\@src}\end{mdframed}}
184 \}
```

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.

```

189 \package
190 \providecommand{\@lec}[1]{(#1)}
191 \def\@lec#1{\strut\hfil\strut\null\nobreak\hfill\@lec{#1}}
192 \def\lec#1{\@lec{#1}\par}
193 \}
```

**\mygraphics** 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.

```

198 \ltxmlRawTeX(
199 \ltxml | package)
200 \newcommand\mygraphics[2] [] {\includegraphics[#1]{#2}}
201 \newcommand\mycgraphics[2] [] {\begin{center}\mygraphics[#1]{#2}\end{center}}
202 \newcommand\mybgraphics[2] [] {\fbox{\mygraphics[#1]{#2}}}
203 \newcommand\mycbgraphics[2] [] {\begin{center}\fbox{\mygraphics[#1]{#2}}\end{center}}
204 \ltxml | package)
205 \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 at the end of the document. If the `at` key is given, then we use that for sorting in the index.

```

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

```

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

`\indexalt`

```

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

```

`\indextoo`

```

233 <*package>
234 \newcommand\indextoo[2][]{\ifindex\omdoc@index{#1}{#2}} % word in text and index
235 </package>
236 <*ltxml>
237 DefConstructor('\indextoo[]{}',
238     "<omdoc:idx>"
239     . " <omdoc:idt>#2</omdoc:idt>"
240     . " <omdoc:ide ?#1(sort-by='#1')()>"
241     . " <omdoc:idp>#2</omdoc:idp>"

```

```

242 . "</omdoc:ide>"
243 . "</omdoc:idx>");
244 </ltxml>

```

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

```

245 <*package>
246 \newcommand\@twin[3] [] {\omdoc@index[#1]{#2!#3}\omdoc@index[#1]{#3!#2}}

```

And again we have two interface macros building on this

`\twinalt`

```

247 \newcommand\twinalt[4] [] {#2\@twin[#1]{#3}{#4}}
248 </package>
249 <*ltxml>
250 DefConstructor('twinalt[]{}{}',
251     "<omdoc:idx>"
252     . "<omdoc:idt>#2</omdoc:idt>"
253     . "<omdoc:ide ?#1(sort-by='#1')()>"
254     . "<omdoc:idp>#2</omdoc:idp>"
255     . "<omdoc:idp>#3</omdoc:idp>"
256     . "</omdoc:ide>"
257     . "</omdoc:idx>");
258 </ltxml>

```

`\twintalt`

```

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

```

EdN:5

`\@atwin` this puts adjectivized two-compound words into the index in various permutations<sup>5</sup>

```

272 <*package>
273 \newcommand\@atwin[4] [] {\omdoc@index[#1]{#2!#3!#4}\omdoc@index[#1]{#3!#2 (#4)}}

```

and the two interface macros for this case:

`\@atwinalt`

```

274 \newcommand\@atwinalt[5] [] {#2\@atwin[#1]{#3}{#4}{#4}}
275 </package>

```

---

<sup>5</sup>EdNOTE: what to do with the optional argument here and below?

```

276 (*\txml)
277 DefConstructor('\atwinalt[]{}{}{}{}',
278     "<omdoc:idx>"
279     . "<omdoc:idt>#2</omdoc:idt>"
280     . "<omdoc:ide ?#1(sort-by='#1')(>"
281     . "<omdoc:idp>#2</omdoc:idp>"
282     . "<omdoc:idp>#3</omdoc:idp>"
283     . "<omdoc:idp>#4</omdoc:idp>"
284     . "</omdoc:ide>"
285     . "</omdoc:idx>");
286 /\txml\

\atwintoo

287 (*package)
288 \newcommand\atwintoo[4][]{{#2 #3 #4}\@atwin[#1]{#2}{#3}{#4}} % and use it too
289 /\package)
290 (*\txml)
291 DefConstructor('\atwintoo[]{}{}{}{}',
292     "<omdoc:idx>"
293     . "<omdoc:idt>#2 #3</omdoc:idt>"
294     . "<omdoc:ide ?#1(sort-by='#1')(>"
295     . "<omdoc:idp>#2</omdoc:idp>"
296     . "<omdoc:idp>#3</omdoc:idp>"
297     . "<omdoc:idp>#4</omdoc:idp>"
298     . "</omdoc:ide>"
299     . "</omdoc:idx>");
300 /\txml\

```

## 4.7 L<sup>A</sup>T<sub>E</sub>X 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 `CMPs`. 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.

```
301 <!*xml>
302 Tag('omdoc:CMP', autoClose=>1, autoOpen=>1);
303 Tag('omdoc:omtext', autoClose=>1, autoOpen=>1);
304 Tag('ltx:p', autoClose=>1, autoOpen=>1);
305 </!xml>
```

the rest of the reinterpretations is quite simple, we either disregard presentational markup or we re-interpret it in terms of OMDoc.<sup>6</sup>

```

306 \package
307 \def\omspace#1{\hspace*{#1}}

```

---

<sup>6</sup>EDNOTE: MK: we should probably let LaTeXML deal with these and allow more text in the omdoc+|txml.xsl

```

308 </package>
309 <*ltxml>
310 DefConstructor('\footnote[]{}',
311     "<omdoc:note type='foot' ?#1(mark='#1')>#2</omdoc:note>");
312 DefConstructor('\footnotemark[]', "");
313 DefConstructor('\footnotetext[]{}',
314     "<omdoc:note class='foot' ?#1(mark='#1')>#2</omdoc:note>");
315 </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.

```

316 <*ltxml>
317 Tag('omdoc:omtext', afterOpen=>\&numberIt, afterClose=>\&locateIt);
318 Tag('omdoc:omgroup', afterOpen=>\&numberIt, afterClose=>\&locateIt);
319 Tag('omdoc:CMP', afterOpen=>\&numberIt, afterClose=>\&locateIt);
320 Tag('omdoc:idx', afterOpen=>\&numberIt, afterClose=>\&locateIt);
321 Tag('omdoc:ide', afterOpen=>\&numberIt, afterClose=>\&locateIt);
322 Tag('omdoc:idt', afterOpen=>\&numberIt, afterClose=>\&locateIt);
323 Tag('omdoc:note', afterOpen=>\&numberIt, afterClose=>\&locateIt);
324 Tag('omdoc:metadata', afterOpen=>\&numberIt, afterClose=>\&locateIt);
325 Tag('omdoc:meta', afterOpen=>\&numberIt, afterClose=>\&locateIt);
326 Tag('omdoc:resource', afterOpen=>\&numberIt, afterClose=>\&locateIt);
327 Tag('omdoc:recurse', afterOpen=>\&numberIt, afterClose=>\&locateIt);
328 Tag('omdoc:imports', afterOpen=>\&numberIt, afterClose=>\&locateIt);
329 Tag('omdoc:theory', afterOpen=>\&numberIt, afterClose=>\&locateIt);
330 Tag('omdoc:ignore', afterOpen=>\&numberIt, afterClose=>\&locateIt);
331 Tag('omdoc:ref', afterOpen=>\&numberIt, afterClose=>\&locateIt);
332 </ltxml>

```

We also have to number some L<sup>A</sup>T<sub>E</sub>X XML tags, so that we do not get into trouble with the OMDoc tags inside them.

```

333 <*ltxml>
334 Tag('ltx:p', afterOpen=>\&numberIt, afterClose=>\&locateIt);
335 Tag('ltx:tabular', afterOpen=>\&numberIt, afterClose=>\&locateIt);
336 Tag('ltx:thead', afterOpen=>\&numberIt, afterClose=>\&locateIt);
337 Tag('ltx:td', afterOpen=>\&numberIt, afterClose=>\&locateIt);
338 Tag('ltx:tr', afterOpen=>\&numberIt, afterClose=>\&locateIt);
339 Tag('ltx:caption', afterOpen=>\&numberIt, afterClose=>\&locateIt);
340 Tag('ltx:Math', afterOpen=>\&numberIt, afterClose=>\&locateIt);
341 </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 L<sup>A</sup>T<sub>E</sub>X 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.

```

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

```

```

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

```

## 4.9 Miscellaneous

Some shortcuts that use math symbols but are not mathematical at all; in particular, they should not be translated by L<sup>A</sup>T<sub>E</sub>X<sub>M</sub>L.

```

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

```

## 4.10 Finale

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

```

422 <ltxml>1;

```

## Index

Numbers written in *italic* refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in *roman* refer to the code lines where the entry is used.

Abelian		group	
group,	<u>5</u>	Abelian,	5

## References

- [KGA15a] Michael Kohlhase, Deyan Ginev, and Rares Ambrus. *modules.sty: Semantic Macros and Module Scoping in sTeX*. Tech. rep. 2015. URL: <https://github.com/KWARC/sTeX/raw/master/sty/modules/modules.pdf>.
- [KGA15b] Michael Kohlhase, Deyan Ginev, and Rares Ambrus. *modules.sty: Semantic Macros and Module Scoping in sTeX*. Tech. rep. Comprehensive T<sub>E</sub>X Archive Network (CTAN), 2015. URL: <http://www.ctan.org/get/macros/latex/contrib/stex/modules/modules.pdf>.
- [KGA15c] Michael Kohlhase, Deyan Ginev, and Rares Ambrus. *modules.sty: Semantic Macros and Module Scoping in sTeX*. Self-documenting L<sup>A</sup>T<sub>E</sub>X package. 2015.
- [Koh06] Michael Kohlhase. OMDOC – *An open markup format for mathematical documents [Version 1.2]*. LNAI 4180. Springer Verlag, Aug. 2006. URL: <http://omdoc.org/pubs/omdoc1.2.pdf>.
- [Koh15a] Michael Kohlhase. *metakeys.sty: A generic framework for extensible Metadata in L<sup>A</sup>T<sub>E</sub>X*. Tech. rep. Comprehensive T<sub>E</sub>X Archive Network (CTAN), 2015. URL: <http://www.ctan.org/tex-archive/macros/latex/contrib/stex/metakeys/metakeys.pdf>.
- [Koh15b] Michael Kohlhase. *statements.sty: Structural Markup for Mathematical Statements*. Tech. rep. 2015. URL: <https://github.com/KWARC/sTeX/raw/master/sty/statements/statements.pdf>.
- [sTeX] *KWARC/sTeX*. URL: <https://svn.kwarc.info/repos/stex> (visited on 05/15/2015).