

# **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>

February 18, 2014

## **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 v1.0 (last revised 2012/11/06)

# 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 . . . . .	4
2.6	Support for <b>MathHub</b> . . . . .	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 . . . . .	12
4.7	L <sup>A</sup> T <sub>E</sub> X Commands we interpret differently . . . . .	15
4.8	Providing IDs for OMDoc Elements . . . . .	15
4.9	Support for <b>MathHub</b> . . . . .	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 [Koh13a] 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 [KGA13a]) that is to be pre-loaded in this one<sup>1</sup> Finally, `verbalizes=` specifies a (more) formal statement (see [Koh13b]) that this text verbalizes or paraphrases.<sup>2</sup>

### 2.3 Phrase-Level Markup

`phrase` The `phrase` environment 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

<sup>1</sup>EDNOTE: this is not implemented yet.

<sup>2</sup>EDNOTE: MK:specify the form of the reference.

generated content markup.

`\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.

`\@sinlinequote`  
`\@@sinlinequote`

## 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`  
`\end@@sblockquote`

```
\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  
`\@@lec` 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  $\text{\LaTeX}$ . 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<sup>3</sup>.

`noindex`

`\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.

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.

## 2.6 Support for MathHub

Much of the  $\text{\LaTeX}$  content is hosted on MathHub (<http://MathHub.info>), a portal and archive for flexiformal mathematics. MathHub offers GIT repositories (public and private escrow) for mathematical documentation projects, online and offline authoring and document development infrastructure, and a rich, interactive reading interface. The `modules` package supports repository-sensitive operations on MathHub.

Note that MathHub has two-level repository names of the form  $\langle group \rangle / \langle repo \rangle$ , where  $\langle group \rangle$  is a MathHub-unique repository group and  $\langle repo \rangle$  a repository name that is  $\langle group \rangle$ -unique. The file and directory structure of a repository is arbitrary – except that it starts with the directory `source` because they are Math Archives in the sense of [Hor+11]. But this structure can be hidden from the  $\text{\LaTeX}$  author with MathHub-enabled versions of the `modules` macros.

`\mhcgraphics` The `\mhcgraphics` macro is a variant of `\mycgraphics` with repository sup-

---

<sup>3</sup>EDNOTE: implement this and issue the respective error message

port. Instead of writing

```
\defpath{MathHub}{/user/foo/lmh/MathHub}  
\mycgraphics{\MathHub{fooMH/bar/source/baz/foobar}}
```

we can simply write (assuming that `\MathHub` is defined as above)

```
\mhcgraphics[fooMH/bar]{baz/foobar}
```

Note that the `\mhcgraphics` form is more semantic, which allows more advanced document management features in `MathHub`.

If `baz/foobar` is the “current module”, i.e. if we are on the `MathHub` path `...MathHub/fooMH/bar...`, then stating the repository in the first optional argument is redundant, so we can just use

```
\mhcgraphics{baz/foobar}
```

Of course, neither `LATEX` nor `LATEXML` know about the repositories when they are called from a file system, so we can use the `\mhcurrentrepos` macro from the `modules` package to tell them. But this is only needed to initialize the infrastructure in the driver file. In particular, we do not need to set it in each module, since the `\importmhmodule` macro sets the current repository automatically.

**Caveat** if you want to use the `MathHub` support macros (let’s call them mh-variants), then every time a module is imported or a document fragment is included from another repos, the mh-variant `\importmhmodule` must be used, so that the “current repository” is set accordingly. To be exact, we only need to use mh-variants, if the imported module or included document fragment use mh-variants.

### 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 `gTEX` TRAC [sTeX].

1. none reported yet

## 4 Implementation

The `omtext` package generates two files: the  $\text{\LaTeX}$  package (all the code between `<*package>` and `</package>`) and the  $\text{\LaTeX}$ XML 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.

### 4.1 Package Options

The initial setup for  $\text{\LaTeX}$ XML:

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

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 \DeclareOption{showmeta}{\PassOptionsToPackage{\CurrentOption}{metakeys}}
9 \newif\ifindex\indextrue
10 \DeclareOption{noindex}{\indexfalse}
11 \ProcessOptions
12 \ifindex\makeindex\fi
13 </package>
14 <*ltxml>
15 DeclareOption('noindex','');
16 </ltxml>
```

Then we need to set up the packages by requiring the `sref` package to be loaded.

```
17 <*package>
18 \RequirePackage{sref}
19 \RequirePackage{xspace}
20 \RequirePackage{modules}
21 \RequirePackage{comment}
22 </package>
23 <*ltxml>
24 RequirePackage('sref');
25 RequirePackage('xspace');
26 RequirePackage('modules');
27 RequirePackage('lxRDFa');
28 </ltxml>
```

---

<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.

```
29 <*ltxml>
30 Tag('omdoc:metadata',afterOpen=>\&numberIt,afterClose=>\&locateIt,autoClose=>1,autoOpen=>1);
31 </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`.

```
32 <*ltxml>
33 Tag('omdoc:CMP', afterOpen => sub {AssignValue('_LastSeenCMP', $_[1], 'global');return;});#$
34 </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.

```
35 <*ltxml>
36 DefParameterType('IfBeginFollows', sub {
37   my ($gullet) = @_ ;
38   $gullet->skipSpaces;
39   my $next = $gullet->readToken;
40   $gullet->unread($next);
41   $next = ToString($next);
42   #Hm, falling back to regexp handling, the $gullet->ifNext approach didn't work
43   return 1 unless ($next=~/\begin/);
44   return;
45 },
46 reversion=>'', optional=>1);
47 </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 [Koh13a]. 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).

```
48 <*package>
49 \srefaddidkey{omtext}
```



```

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

```

The next keys handle module loading (see [KGA13b]).

```

74 % \ednote{need to implement these in LaTeXML, I wonder whether there is a general
75 % mechanism like numberit.}
76 \endpackage
77 \define@key {omtext} {require} {\requiremodules{#1}}
78 \define@key {omtext} {module} {\message{module: #1}\importmodule{#1}\def\omtext@theory{#1}}
79 \endpackage
80 \letxml
81 \letxml

```

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

```

82 \endpackage
83 \def\st@flow{flow}
84 \endpackage

```

**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 \endpackage
86 \def\omtext@pre@skip{\smallskip}
87 \def\omtext@post@skip{}
88 \providecommand{\stDMemph}[1]{\textbf{#1}}
89 \newenvironment{omtext}[1] [] {\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}
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\prhase@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>

```

`nlex` For the moment, we do disregard the most of the keys

```

135 <*package>
136 \def\nlex#1{\green{\sl{#1}}}
137 \def\nlcex#1{* \green{\sl{#1}}}
138 </package>
139 <*txml>
140 DefConstructor('\nlex{}',
141   "<ltx:text class='nlex'>#1</ltx:text>");
142 DefConstructor('\nlcex{}',
143   "<ltx:text class='nlcex'>#1</ltx:text>");
144 </txml>

```

`sinlinequote`

```

145 <*package>
146 \def\@sinlinequote#1{'\sl{#1}}'
147 \def\@sinlinequote#1#2{\@sinlinequote{#2}~#1}
148 \newcommand\sinlinequote[2]{}
149 {\def\@opt{#1}\ifx\@opt\empty\@sinlinequote{#2}\else\@sinlinequote\@opt{#2}\fi}
150 </package>
151 <*txml>
152 DefConstructor('\sinlinequote [] {}',
153   "<ltx:quote type='inlinequote'>"
154     . "?#1(<dc:source>#1</dc:source>\n)()"
155     . "#2"
156     . "</ltx:quote>");
157 </txml>

```

## 4.5 Block-Level Markup

`sblockquote`

```

158 <*package>
159 \def\begin@sblockquote{\begin{quote}\sl}
160 \def\end@sblockquote{\end{quote}}
161 \def\begin@sblockquote#1{\begin@sblockquote}
162 \def\end@sblockquote#1{\def\@lec##1{\rm ##1}\@lec{#1}\end@sblockquote}
163 \newenvironment{sblockquote}[1]{}
164 {\def\@opt{#1}\ifx\@opt\empty\begin@sblockquote\else\begin@sblockquote\@opt\fi}
165 {\ifx\@opt\empty\end@sblockquote\else\end@sblockquote\@opt\fi}
166 </package>
167 <*txml>
168 DefEnvironment('{sblockquote} []',
169   "<ltx:quote>?#1(<ltx:note role='source'>#1</ltx:note>())#body</ltx:quote>");
170 </txml>

```

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.

```
171 <*package>
172 \providecommand{\@@lec}[1]{( #1 )}
173 \def\@lec#1{\strut\hfil\strut\null\nobreak\hfill\hbox{\@@lec{#1}}}
174 \def\lec#1{\@lec{#1}\par}
175 </package>
176 <*txml>
177 DefConstructor(' \lec{ } ',
178   "\n<omdoc:note type='line-end-comment'>#1</omdoc:note>");
179 </txml>
```

`\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.

```
180 <txml>RawTeX('
181 <*txml | package>
182 \newcommand\mygraphics[2] [] {\includegraphics[#1]{#2}}
183 \newcommand\mycgraphics[2] [] {\begin{center}\mygraphics[#1]{#2}\end{center}}
184 \newcommand\mybgraphics[2] [] {\fbox{\mygraphics[#1]{#2}}}
185 </txml | package>
186 <txml>');
```

## 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 we are in  
a module `<mod>`, then we do that by `\importmodule{<mod>}`, else we import all  
the imported modules.

```
187 <*package>
188 \newcommand\omdoc@index[2] [] {\ifindex\def\@test{#1}%
189 \ifx\@test\@empty\def\@@idx{#2}\else\def\@@idx{#1}\fi%
190 \@bsphack\begingroup\@sanitize%
191 \@ifundefined{mod@id}% if we are not in a module
192 {\protected@write\@indexfile{}\string\indexentry%
193 {\@@idx @\string\importmodules{\imported@modules}#2}{\thepage}}}%
194 {\protected@write\@indexfile{}\string\indexentry%
195 {\@@idx @\string\importmodules{mod@id}#2}{\thepage}}}%
196 \endgroup\@esphack}
```

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

`\indexalt`

```
197 \newcommand\indexalt[3] [] {{#2}\omdoc@index[#1]{#3}} % word in text and index
198 </package>
199 <*txml>
```

```

200 DefConstructor('\indextoo[]{}',
201     "<omdoc:idx>"
202     . " <omdoc:idt>#2</omdoc:idt>"
203     . " <omdoc:ide ?#1(sort-by='#1')(>"
204     . " <omdoc:idp>#2</omdoc:idp>"
205     . " </omdoc:ide>"
206     . "</omdoc:idx>");
207 </ltxml>

\indextoo
208 <*package>
209 \newcommand\indextoo[2] []{{#2}\omdoc@index[#1]{#2}} % word in text and index
210 </package>
211 <*ltxml>
212 DefConstructor('\indexalt[]{}{}',
213     "<omdoc:idx>"
214     . " <omdoc:idt>#2</omdoc:idt>"
215     . " <omdoc:ide ?#1(sort-by='#1')(>"
216     . " <omdoc:idp>#3</omdoc:idp>"
217     . " </omdoc:ide>"
218     . "</omdoc:idx>");
219 </ltxml>

\@twin this puts two-compound words into the index in various permutations
220 <*package>
221 \newcommand\@twin[3] []{\ifindex\def\@test{#1}%%
222 \ifx\@test\@empty\def\@idx{#2}\else\def\@idx{#1}\fi%
223 \@ifundefined{mod@id}%
224 {\index{\@idx @#2!#3}%
225 \ifx\@test\@empty\def\@idx{#3}\else\def\@idx{#1}\fi%
226 \index{\@idx @#2!#3}%
227 {\index{\@idx @{\importmodule{\mod@id} #2}!\importmodule{\mod@id} #3}}%
228 \ifx\@test\@empty\def\@idx{#3}\else\def\@idx{#1}\fi%
229 \index{\@idx @{\importmodule{\mod@id} #3}!\importmodule{\mod@id} #2}}\fi}}

```

And again we have two interface macros building on this

```

\twinalt
230 \newcommand\twinalt[4] []{{#2\@twin[#1]{#3}{#4}}
231 </package>
232 <*ltxml>
233 DefConstructor('\twintoo[]{}{}',
234     "<omdoc:idx>"
235     . " <omdoc:idt>#2 #3</omdoc:idt>"
236     . " <omdoc:ide ?#1(sort-by='#1')(>"
237     . " <omdoc:idp>#2</omdoc:idp>"
238     . " <omdoc:idp>#3</omdoc:idp>"
239     . " </omdoc:ide>"
240     . "</omdoc:idx>");
241 </ltxml>

```

\twinalt

```

242 <*package>
243 \newcommand\twintoo[3][]{\ifindex\def\twin[#1]{#2}{#3}} % and use the word compound too
244 </package>
245 <*ltxml>
246 DefConstructor('\twinalt[]{}{}{}',
247     "<omdoc:idx>"
248     . "<omdoc:idt>#2</omdoc:idt>"
249     . "<omdoc:ide ?#1(sort-by='#1')()>"
250     . "<omdoc:idp>#2</omdoc:idp>"
251     . "<omdoc:idp>#3</omdoc:idp>"
252     . "</omdoc:ide>"
253     . "</omdoc:idx>");
254 </ltxml>

```

EdN:5

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

```

255 <*package>
256 \newcommand\@atwin[4][]{\ifindex\def\@test{#1}%
257 \ifx\@test\empty\def\@idx{#2}\else\def\@idx{#1}\fi%
258 \@ifundefined{mod@id}%
259 {\index{\@idx @#2!#3!#4}%
260 \ifx\@test\empty\def\@idx{#3}\else\def\@idx{#1}\fi%
261 \index{\@idx @#3!#2 (#4)}}%
262 {\index{\@idx @{\importmodule{\mod@id} #2}%
263 !{\importmodule{\mod@id} #3}!{\importmodule{\mod@id} #4}}%
264 \ifx\@test\empty\def\@idx{#3}\else\def\@idx{#1}\fi%
265 \index{\@idx @{\importmodule{\mod@id} #3}%
266 !{\importmodule{\mod@id} #2} ({\importmodule{\mod@id} #4})}\fi}}

```

and the two interface macros for this case:

\@atwin

```

267 \newcommand\atwinalt[5][]{\ifindex\def\@atwin[#1]{#2}{#3}{#4}{#5}}
268 </package>
269 <*ltxml>
270 DefConstructor('\atwinalt[]{}{}{}{}',
271     "<omdoc:idx>"
272     . "<omdoc:idt>#2</omdoc:idt>"
273     . "<omdoc:ide ?#1(sort-by='#1')()>"
274     . "<omdoc:idp>#2</omdoc:idp>"
275     . "<omdoc:idp>#3</omdoc:idp>"
276     . "<omdoc:idp>#4</omdoc:idp>"
277     . "</omdoc:ide>"
278     . "</omdoc:idx>");
279 </ltxml>

```

\atwintoo

```

280 <*package>

```

---

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

```

281 \newcommand\atwintoo[4][{}]{#2 #3 #4}\@atwin[#1]{#2}{#3}{#4}} % and use it too
282 \end{package}
283 \end{ltxml}
284 DefConstructor('\atwintoo[]{}{}{}',
285     "<omdoc:idx>"
286     . " <omdoc:idt>#2 #3</omdoc:idt>"
287     . " <omdoc:ide ?#1(sort-by='#1')()>"
288     . " <omdoc:idp>#2</omdoc:idp>"
289     . " <omdoc:idp>#3</omdoc:idp>"
290     . " <omdoc:idp>#4</omdoc:idp>"
291     . "</omdoc:ide>"
292     . "</omdoc:idx>");
293 \end{ltxml}

```

## 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.

```

294 \end{ltxml}
295 Tag('omdoc:CMP', autoClose=>1, autoOpen=>1);
296 Tag('omdoc:omtext', autoClose=>1, autoOpen=>1);
297 Tag('ltx:p', autoClose=>1, autoOpen=>1);
298 \end{ltxml}

```

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>

```

299 \end{package}
300 \def\omspace#1{\hspace*{#1}}
301 \end{package}
302 \end{ltxml}
303 DefConstructor('\footnote[]{}',
304     "<omdoc:note type='foot' ?#1(mark='#1')>#2</omdoc:note>");
305 DefConstructor('\footnotemark[]{}', "");
306 DefConstructor('\footnotetext[]{}',
307     "<omdoc:note class='foot' ?#1(mark='#1')>#2</omdoc:note>");
308 \end{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.

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

```

309 <*ltxml>
310 Tag('omdoc:omtext',afterOpen=>\&numberIt,afterClose=>\&locateIt);
311 Tag('omdoc:omgroup',afterOpen=>\&numberIt,afterClose=>\&locateIt);
312 Tag('omdoc:CMP',afterOpen=>\&numberIt,afterClose=>\&locateIt);
313 Tag('omdoc:idx',afterOpen=>\&numberIt,afterClose=>\&locateIt);
314 Tag('omdoc:ide',afterOpen=>\&numberIt,afterClose=>\&locateIt);
315 Tag('omdoc:idt',afterOpen=>\&numberIt,afterClose=>\&locateIt);
316 Tag('omdoc:note',afterOpen=>\&numberIt,afterClose=>\&locateIt);
317 Tag('omdoc:metadata',afterOpen=>\&numberIt,afterClose=>\&locateIt);
318 Tag('omdoc:meta',afterOpen=>\&numberIt,afterClose=>\&locateIt);
319 Tag('omdoc:resource',afterOpen=>\&numberIt,afterClose=>\&locateIt);
320 Tag('omdoc:recurse',afterOpen=>\&numberIt,afterClose=>\&locateIt);
321 Tag('omdoc:imports',afterOpen=>\&numberIt,afterClose=>\&locateIt);
322 Tag('omdoc:theory',afterOpen=>\&numberIt,afterClose=>\&locateIt);
323 Tag('omdoc:ignore',afterOpen=>\&numberIt,afterClose=>\&locateIt);
324 Tag('omdoc:ref',afterOpen=>\&numberIt,afterClose=>\&locateIt);
325 </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.

```

326 <*ltxml>
327 Tag('ltx:p',afterOpen=>\&numberIt,afterClose=>\&locateIt);
328 Tag('ltx:tabular',afterOpen=>\&numberIt,afterClose=>\&locateIt);
329 Tag('ltx:thead',afterOpen=>\&numberIt,afterClose=>\&locateIt);
330 Tag('ltx:td',afterOpen=>\&numberIt,afterClose=>\&locateIt);
331 Tag('ltx:tr',afterOpen=>\&numberIt,afterClose=>\&locateIt);
332 Tag('ltx:caption',afterOpen=>\&numberIt,afterClose=>\&locateIt);
333 Tag('ltx:Math',afterOpen=>\&numberIt,afterClose=>\&locateIt);
334 </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.

```

335 <*ltxml>
336 sub numberIt {
337   my($document,$node,$whatsit)=@_;
338   my(@parents)=$document->findnodes('ancestor::*[@xml:id]', $node);
339   my $prefix= (@parents ? $parents[$#parents]->getAttribute('xml:id')."." : '');
340   my(@siblings)=$document->findnodes('preceding-sibling::*[@xml:id]', $node);
341   my $n = scalar(@siblings)+1;
342   my $id = ($node -> getAttribute('xml:id'));
343   my $localname = $node->localname;
344   $node->setAttribute('xml:id'=>$prefix."$localname$n") unless $id;
345   my $about = $node -> getAttribute('about');
346   $node->setAttribute('about'=>'#'.$node->getAttribute('xml:id')) unless $about;

```



```

347 #Also, provide locators:
348 my $locator = $whatsit->getProperty('locator');
349 #Need to inherit locators if missing:
350 $locator = (@parents ? $parents[$#parents]->getAttribute('stex:srcref') : '') unless $locator
351 if ($locator) {
352     # There is a BUG with namespace declarations (or am I using the API wrongly??) which
353     # does not recognize the stex namespace. Hence, I need to redeclare it...
354     my $parent=$document->getNode;
355     if(! defined $parent->lookupNamespacePrefix("http://kwarc.info/ns/sTeX"))
356     { # namespace not already declared?
357         $document->getDocument->documentElement->setNamespace("http://kwarc.info/ns/sTeX","stex")
358     }
359     $node->setAttribute('stex:srcref'=>$locator);
360 }return;}
361
362 sub locateIt {
363     my($document,$node,$whatsit)=@_;
364     #Estimate trailer and locator:
365     my $locator = $node->getAttribute('stex:srcref');
366     return unless $locator; # Nothing to do here...
367     my $trailer = $whatsit->getProperty('trailer');
368     $trailer = $trailer->getLocator if $trailer;
369     $trailer = $locator unless $trailer; # bootstrap
370     # TODO: Both should be local, or both remote, any mixture or undefinedness will produce garba
371     my $file_path = LookupValue('SOURCEFILE');
372     my $baselocal = LookupValue('BASELOCAL');
373     # Hmm, we only care about relative paths, so let's just do a URL->pathname map
374     $file_path=~s/^\w+\.\\/// if $file_path;
375     $baselocal=~s/^\w+\.\\/// if $baselocal;
376     if ($file_path && $baselocal && ($locator =~ s/^(\[^\#\]+\)\#/#/)) {
377         my $relative_path = pathname_relative($file_path,$baselocal);
378         $locator = $relative_path.$locator;
379     }
380     if ($locator =~ /^(.+from=\d+;\d+)/) {
381         my $from = $1;
382         if ($trailer =~ /(,to=\d+;\d+.)$/ ) {
383             my $to = $1;
384             $locator = $from.$to;
385         } else { Error("stex","locator",undef, "Trailer is garbled, expect nonsense in stex:srcref")
386     } else { Error("stex","locator",undef, "Locator \"$locator\" is garbled, expect nonsense in s
387     my $parent = $document->getNode;
388     if(! defined $parent->lookupNamespacePrefix("http://kwarc.info/ns/sTeX"))
389     { # namespace not already declared?
390         $document->getDocument->documentElement->setNamespace("http://kwarc.info/ns/sTeX","stex",
391     }
392     $node->setAttribute('stex:srcref' => $locator);
393     return;
394 }
395 </txml>#$

```

## 4.9 Support for MathHub

`\mh*graphics` Use the current value of `\mh@currentrepos` or the value of the `mhrepos` key if it is given in `\my*graphics`.

```
396 <package>\addmetakey{Gin}{mhrepos}
397 <ltxml>DefKeyVal('Gin','mhrepos','Semiverbatim');
398 <ltxml>RawTeX('
399 <*ltxml | package>
400 \newcommand\mhgraphics[2] [] {\metasetkeys{Gin}-{#1}%
401 \ifx\Gin@mhrepos\empty\mygraphics[#1]{\MathHub{\mh@currentrepos/source/#2}}%
402 \else\mygraphics[#1]{\MathHub{\Gin@mhrepos/source/#2}}\fi}
403 \newcommand\mhcgraphics[2] [] {\begin{center}\mhgraphics[#1]{#2}\end{center}}
404 \newcommand\mhgraphics[2] [] {\fbox{\mhgraphics[#1]{#2}}}
405 </ltxml | package>
406 <ltxml>');
```

## 4.10 Finale

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

```
407 <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		Abelian	
group,	<i>5</i>	group,	<i>5</i>