`omtext`: Semantic Markup for Mathematical Text Fragments in LaTeX*

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

January 28, 2012

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 v1.0 (last revised 2012/01/28)

1

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 [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 [Koh10a] 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
EdNote:1 · module (see [KGA10]) that is to be pre-loaded in this one[1] Finally, `verbalizes`
verbalizes= · specifies a (more) formal statement (see [Koh10b]) that this text verbalizes.

2.3 Phrase-Level Markup

phrase · The `phrase` environment allows to mark up phrases with semantic information. It
verbalizes= · takes an optional KeyVal argument with the keys `verbalizes` and `type` as above
type= · and `style`, `class`, `index` that are disregarded in the LATEX, but copied into the
style · generated content markup.
class ·   The `sinlinequote` macro allows to mark up quotes inline and attribute them.
index ·
\sinlinequote

---

[1]EDNOTE: this is not implemented yet.

3

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 actual appearance of the line end comment is determined by the \@@lec macro, which can be customized in the document class.

\@@lec

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 thenoindex option is set in the \usepackage. The omtext package and class make the usual \index macro undefined[2].

noindex

EdNote:2

\indextoo

The \indextoo macro renders a word and marks it for the index. Some-

---

[2]EDNOTE: implement this and issue the respective error message

4

times, 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.

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 TRAC [Ste].

    1. none reported yet

5

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.

4.1 Package Options

The initial setup for LATEXML:

```
1 ⟨*ltxml⟩
2 package LaTeXML::Package::Pool;
3 use strict;
4 use LaTeXML::Package;
5 use Cwd qw(cwd abs_path);
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).[3]

```
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{comment}
20 ⟨/package⟩
21 ⟨*ltxml⟩
22 RequirePackage('sref');
23 ⟨/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

---

[3]EDNOTE: need an implementation for LATEXML

element, even if they are supplied by different ꟷTEX bindings. Also we add numbering and location facilities.

```
24 ⟨∗ltxml⟩
25 Tag('omdoc:metadata',afterOpen=>\&numberIt,afterClose=>\&locateIt,autoClose=>1,autoOpen=>1);
26 ⟨/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`.

```
27 ⟨∗ltxml⟩
28 sub declareFunctions{
29   my ($stomach,$whatsit) = @_;
30   my $keyval = $whatsit->getArg(1);
31   my $funval = KeyVal($keyval,'functions') if KeyVal($keyval,'functions');
32   my @funsymbs = ParseKeyValList($funval);
33   #Unread the function declarations at the Gullet
34   foreach (@funsymbs) {
35     $stomach->getGullet->unread(Tokenize('\lxDeclare[role=FUNCTION]{$'.$_.'$}')->unlist);
36   }
37   return;
38 }
39 Tag('omdoc:CMP', afterOpen => sub {AssignValue('_LastSeenCMP', $_[1], 'global');return;});#$
40 ⟨/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.

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

4.3   Mathematical Text

We define the actions that are undertaken, when the keys are encountered. Here this is very simple, we just define an internal macro with the value, so that we can

7

use it later. 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).

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

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

```
80 ⟨∗package⟩
81 \def\st@flow{flow}
82 ⟨/package⟩
```

`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.

```
83 ⟨∗package⟩
84 \def\omtext@pre@skip{\smallskip}
85 \def\omtext@post@skip{}
86 \providecommand{\stDMemph}[1]{\textbf{#1}}
87 \newenvironment{omtext}[1][]{\bgroup\metasetkeys{omtext}{#1}\sref@label@id{this paragraph}%
88 \def\lec##1{\@lec{##1}}
89 \ifx\omtext@display\st@flow\else\omtext@pre@skip\par\noindent%
90 \ifx\omtext@title\@empty\else\stDMemph{\omtext@title}: \fi\fi\ignorespaces}
91 {\egroup\omtext@post@skip}
92 ⟨/package⟩
93 ⟨∗ltxml⟩
```

8

```
 94 DefEnvironment('{omtext} OptionalKeyVals:omtext',
 95   "<omdoc:omtext "
 96     . "?&KeyVal(#1,'id')(xml:id='&KeyVal(#1,'id')')() "
 97     . "?&KeyVal(#1,'type')(type='&KeyVal(#1,'type')')() "
 98     . "?&KeyVal(#1,'for')(for='&KeyVal(#1,'for')')() "
 99         . "?&KeyVal(#1,'from')(from='&KeyVal(#1,'from')')()>"
100   . "?&KeyVal(#1,'title')(<dc:title>&KeyVal(#1,'title')</dc:title>)()"
101   . "<omdoc:CMP>"
102       "?&KeyVal(#1,'start')(<ltx:text class='startemph'>&KeyVal(#1,'start')</ltx:text>)()"
103             . "#body"
104                 ."</omdoc:omtext>");
105 ⟨/ltxml⟩
```

We also make our life easier If defining an environment that is turned into something that contains `<CMP>`⟨*body*⟩`</CMP>`, use this method instead

```
106 ⟨*ltxml⟩
107 sub DefCMPEnvironment {
108   my ($proto, $replacement, %options) = @_;
109   my @before = $options{beforeDigest} ? ($options{beforeDigest}) : ();
110   push(@before, \&useCMPItemizations);
111   $options{beforeDigest} = \@before;
112   my @after = $options{afterDigestBegin} ? ($options{afterDigestBegin}) : ();
113   push(@after, \&declareFunctions);
114   $options{afterDigestBegin} = \@after;
115   DefEnvironment($proto, $replacement, %options);
116 }
117 sub DefCMPConstructor {
118   my ($proto, $replacement, %options) = @_;
119   my @before = $options{beforeDigest} ? ($options{beforeDigest}) : ();
120   push(@before, \&useCMPItemizations);
121   $options{beforeDigest} = \@before;
122   DefConstructor($proto, $replacement, %options);
123 }#$
124 ⟨/ltxml⟩
```

4.4 Phrase-level Markup

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

```
125 ⟨*package⟩
126 \srefaddidkey{phrase}
127 \addmetakey{phrase}{style}
128 \addmetakey{phrase}{class}
129 \addmetakey{phrase}{index}
130 \addmetakey{phrase}{verbalizes}
131 \addmetakey{phrase}{type}
132 \newenvironment{phrase}[1][]{\metasetkeys{phrase}{#1}}{}
133 ⟨/package⟩
134 ⟨*ltxml⟩
135 DefKeyVal('phrase','id','Semiverbatim');
```

9

```
136 DefKeyVal('phrase','style','Semiverbatim');
137 DefKeyVal('phrase','class','Semiverbatim');
138 DefKeyVal('phrase','index','Semiverbatim');
139 DefKeyVal('phrase','verbalizes','Semiverbatim');
140 DefKeyVal('phrase','type','Semiverbatim');
141 DefConstructor('\phrase OptionalKeyVals:phrase {}',
142     "<ltx:text %&KeyVals(#1)>#2</ltx:text>");
143 ⟨/ltxml⟩
```

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

```
144 ⟨∗package⟩
145 \def\nlex#1{\green{\sl{#1}}}
146 \def\nlcex#1{*\green{\sl{#1}}}
147 ⟨/package⟩
148 ⟨∗ltxml⟩
149 DefConstructor('\nlex{}',
150   "<ltx:text class='nlex'>#1</ltx:text>");
151 DefConstructor('\nlcex{}',
152   "<ltx:text type='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 [4]

```
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}
```

---

[4]EDNOTE: describe above

10

```
175 ⟨/package⟩
176 ⟨∗ltxml⟩
177 DefEnvironment('{sblockquote} []',
178                   "?#1(<omdoc:omtext type='quote'>"
179                 .      "<dc:source>#1</dc:source>"
180                 .      "#body"
181              . "   </omdoc:omtext>)"
182              .   "(<ltx:quote>#body</ltx:quote>)");
183 ⟨/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.

```
184 ⟨∗package⟩
185 \providecommand{\@@lec}[1]{(#1)}
186 \def\@lec#1{\strut\hfil\strut\null\nobreak\hfill\hbox{\@@lec{#1}}}
187 \def\lec#1{\@lec{#1}\par}
188 ⟨/package⟩
189 ⟨∗ltxml⟩
190 DefConstructor('\lec{}',
191    "\n<omdoc:note type='line-end-comment'>#1</omdoc:note>");
192 ⟨/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.

```
193 ⟨ltxml⟩RawTeX('
194 ⟨∗ltxml | package⟩
195 \newcommand\mygraphics[2][]{\includegraphics[#1]{#2}}
196 \newcommand\mycgraphics[2][]{\begin{center}\includegraphics[#1]{#2}\end{center}}
197 \newcommand\mybgraphics[2][]{\fbox{\includegraphics[#1]{#2}\end{center}}}
198 ⟨/ltxml | package⟩
199 ⟨ltxml⟩');
```

4.6 Index Markup

```
200 % this is the main internal indexing comman. It makes sure that the modules necessary for
201 % interpreting the math in the index entries are loaded.
202 ⟨∗package⟩
203 \newcommand{\omdoc@index}[2][]{\ifindex\def\@test{#1}%%
204 \ifx\@test\@empty\def\@@idx{#2}\else\def\@@idx{#1}\fi%
205 \@ifundefined{mod@id}{\index{\@@idx @#2}}%
206 {\index{\@@idx @{\importmodule{\mod@id} #2}}\fi}}
207 \newcommand{\indexalt}[3][]{{#2}\omdoc@index[#1]{#3}}              % word in text and index
208 \newcommand{\indextoo}[2][]{{#2}\omdoc@index[#1]{#2}}              % word in text and index
209 ⟨/package⟩
```

11

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

```
210 ⟨∗package⟩
211 \newcommand{\@twin}[3][]{\ifindex\def\@test{#1}%%
212 \ifx\@test\@empty\def\@@idx{#2}\else\def\@@idx{#1}\fi%
213 \@ifundefined{mod@id}
214 {\index{\@@idx @#2!#3}%
215 \ifx\@test\@empty\def\@@idx{#3}\else\def\@@idx{#1}\fi%
216 \index{\@@idx @#2!#3}}%
217 {\index{\@@idx @{\importmodule{\mod@id} #2}!{\importmodule{\mod@id} #3}}%
218 \ifx\@test\@empty\def\@@idx{#3}\else\def\@@idx{#1}\fi%
219 \index{\@@idx @{\importmodule{\mod@id} #3}!{\importmodule{\mod@id} #2}}\fi}}
220 \newcommand{\twinalt}[4][]{#2\@twin[#1]{#3}{#4}}
221 \newcommand{\twintoo}[3][]{{#2 #3}\@twin[#1]{#2}{#3}}                % and use the word compound t
222 ⟨/package⟩
```

this puts adjectivized two-compound words into the index in various permuta-
tions[5]

```
223 ⟨∗package⟩
224 \newcommand{\@atwin}[4][]{\ifindex\def\@test{#1}%%
225 \ifx\@test\@empty\def\@@idx{#2}\else\def\@@idx{#1}\fi%
226 \@ifundefined{mod@id}%
227 {\index{\@@idx @#2!#3!#4}%
228 \ifx\@test\@empty\def\@@idx{#3}\else\def\@@idx{#1}\fi
229 \index{\@@idx @#3!#2 (#4)}}%
230 {\index{\@@idx @{\importmodule{\mod@id} #2}!{\importmodule{\mod@id} #3}!{\importmodule{\mod@id}
231 \ifx\@test\@empty\def\@@idx{#3}\else\def\@@idx{#1}\fi%
232 \index{\@@idx @{\importmodule{\mod@id} #3}!{\importmodule{\mod@id} #2} ({\importmodule{\mod@id}
233 \newcommand{\atwinalt}[5][]{#2\@atwin[#1]{#3}{#4}{#4}}
234 \newcommand{\atwintoo}[4][]{{#2 #3 #4}\@atwin[#1]{#2}{#3}{#4}}            % and use it too
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 DefConstructor('\indexalt[]{}{}',
245         "<omdoc:idx>"
246      .  "<omdoc:idt>#2</omdoc:idt>"
247      .  "<omdoc:ide ?#1(sort-by='#1')()>"
248      .    "<omdoc:idp>#3</omdoc:idp>"
249      .  "</omdoc:ide>"
250      ."</omdoc:idx>");
251 ⟨/ltxml⟩

252 ⟨∗ltxml⟩
253 DefConstructor('\twintoo[]{}{}',
```

---

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

```
254      "<omdoc:idx>"
255    .  "<omdoc:idt>#2 #3</omdoc:idt>"
256    .  "<omdoc:ide ?#1(sort-by='#1')()>"
257    .    "<omdoc:idp>#2</omdoc:idp>"
258    .    "<omdoc:idp>#3</omdoc:idp>"
259    .  "</omdoc:ide>"
260    ."</omdoc:idx>");
261 DefConstructor('\twinalt[]{}{}{}',
262      "<omdoc:idx>"
263    .  "<omdoc:idt>#2</omdoc:idt>"
264    .  "<omdoc:ide ?#1(sort-by='#1')()>"
265    .    "<omdoc:idp>#2</omdoc:idp>"
266    .    "<omdoc:idp>#3</omdoc:idp>"
267    .  "</omdoc:ide>"
268    ."</omdoc:idx>");
```
269 ⟨/ltxml⟩

270 ⟨∗ltxml⟩
```
271 DefConstructor('\atwintoo[]{}{}{}',
272      "<omdoc:idx>"
273    .  "<omdoc:idt>#2 #3</omdoc:idt>"
274    .  "<omdoc:ide ?#1(sort-by='#1')()>"
275    .    "<omdoc:idp>#2</omdoc:idp>"
276    .    "<omdoc:idp>#3</omdoc:idp>"
277    .    "<omdoc:idp>#4</omdoc:idp>"
278    .  "</omdoc:ide>"
279    ."</omdoc:idx>");
280
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⟩

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 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.

291 ⟨∗ltxml⟩
```
292 # needed? DefConstructor('\par',sub { $_[0]->maybeCloseElement('ltx:p'); },alias=>"\\par\n");
293 Tag('omdoc:CMP', autoClose=>1, autoOpen=>1);
```

13

```
294 Tag('omdoc:omtext', autoClose=>1, autoOpen=>1);
295 ⟨/ltxml⟩#$
```

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

```
296 ⟨package⟩\def\omspace#1{\hspace*{#1}}
297 ⟨*ltxml⟩
298 DefConstructor('\footnote[]{}',
299        "<omdoc:note type='foot' ?#1(mark='#1')>#2</omdoc:note>");
300 DefConstructor('\footnotemark[]',"");
301 DefConstructor('\footnotetext[]{}',
302        "<omdoc:note class='foot' ?#1(mark='#1')>#2</omdoc:note>");
303 ⟨/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.

```
304 ⟨*ltxml⟩
305 Tag('omdoc:omtext',afterOpen=>\&numberIt,afterClose=>\&locateIt);
306 Tag('omdoc:omgroup',afterOpen=>\&numberIt,afterClose=>\&locateIt);
307 Tag('omdoc:CMP',afterOpen=>\&numberIt,afterClose=>\&locateIt);
308 Tag('omdoc:idx',afterOpen=>\&numberIt,afterClose=>\&locateIt);
309 Tag('omdoc:ide',afterOpen=>\&numberIt,afterClose=>\&locateIt);
310 Tag('omdoc:idt',afterOpen=>\&numberIt,afterClose=>\&locateIt);
311 Tag('omdoc:note',afterOpen=>\&numberIt,afterClose=>\&locateIt);
312 Tag('omdoc:metadata',afterOpen=>\&numberIt,afterClose=>\&locateIt);
313 Tag('omdoc:meta',afterOpen=>\&numberIt,afterClose=>\&locateIt);
314 Tag('omdoc:resource',afterOpen=>\&numberIt,afterClose=>\&locateIt);
315 Tag('ltx:p',afterOpen=>\&numberIt,afterClose=>\&locateIt);
316 ⟨/ltxml⟩
```

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

```
317 ⟨*ltxml⟩
318 Tag('ltx:tabular',afterOpen=>\&numberIt,afterClose=>\&locateIt);
319 Tag('ltx:thead',afterOpen=>\&numberIt,afterClose=>\&locateIt);
320 Tag('ltx:td',afterOpen=>\&numberIt,afterClose=>\&locateIt);
321 Tag('ltx:tr',afterOpen=>\&numberIt,afterClose=>\&locateIt);
322 Tag('ltx:caption',afterOpen=>\&numberIt,afterClose=>\&locateIt);
323 ⟨/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

---

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

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.

```
324 ⟨*ltxml⟩
325 sub numberIt {
326   my($document,$node,$whatsit)=@_;
327   my(@parents)=$document->findnodes('ancestor::*[@xml:id]',$node);
328   my $prefix= (@parents ? $parents[$#parents]->getAttribute('xml:id')."." : '');
329   my(@siblings)=$document->findnodes('preceding-sibling::*[@xml:id]',$node);
330   my $n = scalar(@siblings)+1;
331   my $id = ($node -> getAttribute('xml:id'));
332   my $localname = $node->localname;
333   $node->setAttribute('xml:id'=>$prefix."$localname$n") unless $id;
334   my $about = $node -> getAttribute('about');
335   $node->setAttribute('about'=>'#'.$node->getAttribute('xml:id')) unless $about;
336   #Also, provide locators:
337   my $locator = $whatsit->getProperty('locator');
338   #Need to inherit locators if missing:
339   $locator = (@parents ? $parents[$#parents]->getAttribute('stex:srcref') : '') unless $locator
340   if ($locator) {
341     # There is a BUG with namespace declarations (or am I using the API wrongly??) which
342     # does not recognize the stex namespace. Hence, I need to redeclare it...
343     my $parent=$document->getNode;
344     if(! defined $parent->lookupNamespacePrefix("http://kwarc.info/ns/sTeX"))
345       { # namespace not already declared?
346         $document->getDocument->documentElement->setNamespace("http://kwarc.info/ns/sTeX","stex
347       }
348     $node->setAttribute('stex:srcref'=>$locator);
349   }return;}
350
351 sub locateIt {
352   my($document,$node,$whatsit)=@_;
353   #Estimate trailer locator:
354   my $trailer = $whatsit->getProperty('trailer');
355   return unless $trailer; #Nothing we can do if the trailer isn't defined
356   $trailer = $trailer->getLocator;
357   return unless ($trailer && $trailer!~/^\s*$/); #Useless if broken
358   my $locator = $node->getAttribute('stex:srcref');
359   if ($locator) {
360     $locator =~ /^(.+from=\d+;\d+)/;
361     my $from = $1;
362     $trailer =~ /(,to=\d+;\d+.+)$/;
363     my $to = $1;
364     $locator = $from.$to;
365   } else {
366     $locator = $trailer; #This should never happen
367   }
368   my $parent = $document->getNode;
```

15

```
369    if(! defined $parent->lookupNamespacePrefix("http://kwarc.info/ns/sTeX"))
370      { # namespace not already declared?
371        $document->getDocument->documentElement->setNamespace("http://kwarc.info/ns/sTeX","stex",
372      }
373    $node->setAttribute('stex:srcref' => $locator);
374    return;
375 }
376 ⟨/ltxml⟩#$
```

4.9   Finale

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

```
377 ⟨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.

References

[KGA10] Michael Kohlhase, Deyan Ginev, and Rares Ambrus. *modules.sty: Semantic Macros and Module Scoping in sTeX*. Self-documenting LaTeX package. 2010. URL: `https://svn.kwarc.info/repos/stex/trunk/sty/modules/modules.pdf`.

[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`.

[Koh10a] Michael Kohlhase. *metakeys.sty: A generic framework for extensible Metadata in LaTeX*. Self-documenting LaTeX package. Comprehensive TeX Archive Network (CTAN), 2010. URL: `http://www.ctan.org/tex-archive/macros/latex/contrib/stex/metakeys/metakeys.pdf`.

[Koh10b] Michael Kohlhase. *statements.sty: Structural Markup for Mathematical Statements*. Self-documenting LaTeX package. 2010. URL: `https://svn.kwarc.info/repos/stex/trunk/sty/statements/statements.pdf`.

[Ste] *Semantic Markup for LaTeX*. Project Homepage. URL: `http://trac.kwarc.info/sTeX/` (visited on 02/22/2011).