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

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

July 20, 2010

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 2010/06/25)

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

`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`, `annote` are recommended. 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.

3 Block-Level Markup

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

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

and would render as

> *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.

4 Index Markup

The `omtext` package provides some extensions for the well-known indexing macros of LaTeX. The main reason for introducing these macros is that index markup in OMDoc wraps the indexed terms rather than just marking the spot for cross-referencing. Furthermore the index commands only indexes words unless the`noindex` option is set in the `\usepackage`. The `omtext` package and class make the usual `\index` macro undefined[1].

noindex

EdNote(1)

`\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:

`\twintoo`

```
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.

`\atwintoo`

`\atwinalt`

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

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

5 Implementation

5.1 Package Options

The initial setup for LaTeXML:

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

EdNote(2)

```
⟨*package⟩
\newif\ifindex\indextrue
\DeclareOption{noindex}{\indexfalse}
\ProcessOptions
\ifindex\makeindex\fi
⟨/package⟩
⟨*ltxml⟩
DeclareOption('noindex','');
⟨/ltxml⟩
```

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

```
⟨*package⟩
\RequirePackage{sref}
\RequirePackage{comment}
⟨/package⟩
⟨*ltxml⟩
RequirePackage('sref');
⟨/ltxml⟩
```

5.2 Metadata

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

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

[2] EdNote: need an implementation for LaTeXML

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`.

```
⟨*ltxml⟩
DefConstructor('\CMP@itemize@item[]',
       "<omdoc:li>?#1(<dc:title ?#locator(stex:srcref='#locator')()>#1</dc:title>)()",
       properties=>sub{ RefStepItemCounter(); });
DefConstructor('\CMP@enumerate@item[]',
       "<omdoc:li>?#1(<dc:title ?#locator(stex:srcref='#locator')()>#1</dc:title>)()",
       properties=>sub{ RefStepItemCounter(); });
DefConstructor('\CMP@description@item[]',
       "<omdoc:di>"
       . "?#1(<omdoc:dt>#1</omdoc:dt>)()<omdoc:dd>", # trust di and dt to autoclose
       properties=>sub{ RefStepItemCounter(); });
DefEnvironment('{CMP@itemize}',
       "<omdoc:ul>#body</omdoc:ul>",
       properties=>sub { beginItemize('CMP@itemize'); },
       beforeDigest=>sub { Let(T_CS('\end{itemize}') =>T_CS('\end{CMP@itemize}')); });
DefEnvironment('{CMP@enumerate}',
       "<omdoc:ol xml:id='#id'>#body</omdoc:ol>",
       properties=>sub { beginItemize('CMP@enumerate'); },
       beforeDigest=>sub { Let(T_CS('\end{enumerate}') =>T_CS('\end{CMP@enumerate}')); });
DefEnvironment('{CMP@description}',
 "<omdoc:dl xml:id='#id'>#body</omdoc:dl>",
       properties=>sub { beginItemize('CMP@description'); },
       beforeDigest=>sub { Let(T_CS('\end{description}') =>T_CS('\end{CMP@description}'));});
sub useCMPItemizations {
    Let(T_CS('\begin{itemize}')    =>T_CS('\begin{CMP@itemize}'));
    Let(T_CS('\begin{enumerate}')  =>T_CS('\begin{CMP@enumerate}'));
    Let(T_CS('\begin{description}')=>T_CS('\begin{CMP@description}'));
    return; }
sub declareFunctions{
  my ($stomach,$whatsit) = @_;
  my $keyval = $whatsit->getArg(1);
  my $funval = KeyVal($keyval,'functions') if KeyVal($keyval,'functions');
  my @funsymbs = ParseKeyValList($funval);
  #Unread the function declarations at the Gullet
  foreach (@funsymbs) {
    $stomach->getGullet->unread(Tokenize('\lxDeclare[role=FUNCTION]{$'.$_.'$}')->unlist);
  }
  return;
}
Tag('omdoc:CMP', afterOpen => sub {AssignValue('_LastSeenCMP', $_[1], 'global');return;});#$
Tag('omdoc:li', autoClose=>1);
Tag('omdoc:dd', autoClose=>1);
Tag('omdoc:di', autoClose=>1);
⟨/ltxml⟩
```

6 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 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).

```
⟨*package⟩
\srefaddidkey{omtext}
\omdaddkey[]{omtext}{functions}
\omdaddkey{omtext}{display}
\omdaddkey{omtext}{for}
\omdaddkey{omtext}{from}
\omdaddkey{omtext}{type}
\omdaddkey{omtext}{title}
\omdaddkey{omtext}{theory}
\omdaddkey{omtext}{continues}
\omdaddkey{omtext}{verbalizes}
⟨/package⟩
⟨*ltxml⟩
DefKeyVal('omtext','functions','Semiverbatim');
DefKeyVal('omtext','display','Semiverbatim');
DefKeyVal('omtext','for','Semiverbatim');
DefKeyVal('omtext','from','Semiverbatim');
DefKeyVal('omtext','type','Semiverbatim');
DefKeyVal('omtext','title','Plain'); #Math mode in titles.
DefKeyVal('omtext','theory','Semiverbatim');
DefKeyVal('omtext','continues','Semiverbatim');
DefKeyVal('omtext','verbalizes','Semiverbatim');
⟨/ltxml⟩
```

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

```
⟨*package⟩
\def\st@flow{flow}
⟨/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.

```
⟨*package⟩
\def\omtext@pre@skip{\smallskip}
\def\omtext@post@skip{}
\providecommand{\stDMemph}[1]{\textbf{#1}}
\newenvironment{omtext}[1][]{\bgroup\omdsetkeys{omtext}{#1}\sref@label@id{this paragraph}%
\def\lec##1{\@lec{##1}}
\ifx\omtext@display\st@flow\else\omtext@pre@skip\par\noindent%
\ifx\omtext@title\@empty\else\stDMemph{\omtext@title}: \fi\fi\ignorespaces}
{\egroup\omtext@post@skip}
⟨/package⟩
```

```
⟨*ltxml⟩
DefCMPEnvironment('{omtext} OptionalKeyVals:omtext',
  "<omdoc:omtext "
     . "?&KeyVal(#1,'id')(xml:id='&KeyVal(#1,'id')')() "
     . "?&KeyVal(#1,'type')(type='&KeyVal(#1,'type')')() "
     . "?&KeyVal(#1,'for')(for='&KeyVal(#1,'for')')() "
             . "?&KeyVal(#1,'from')(from='&KeyVal(#1,'from')')()>"
  . "?&KeyVal(#1,'title')(<dc:title>&KeyVal(#1,'title')</dc:title>)()"
  . "<omdoc:CMP>"
                  .   "<omdoc:p>"
                  .      "#body");
⟨/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

```
⟨*ltxml⟩
sub DefCMPEnvironment {
  my ($proto, $replacement, %options) = @_;
  my @before = $options{beforeDigest} ? ($options{beforeDigest}) : ();
  push(@before, \&useCMPItemizations);
  $options{beforeDigest} = \@before;
  my @after = $options{afterDigestBegin} ? ($options{afterDigestBegin}) : ();
  push(@after, \&declareFunctions);
  $options{afterDigestBegin} = \@after;
  DefEnvironment($proto, $replacement, %options);
}
sub DefCMPConstructor {
  my ($proto, $replacement, %options) = @_;
  my @before = $options{beforeDigest} ? ($options{beforeDigest}) : ();
  push(@before, \&useCMPItemizations);
  $options{beforeDigest} = \@before;
  DefConstructor($proto, $replacement, %options);
}
⟨/ltxml⟩
```

6.1 Phrase-level Markup

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

```
⟨*package⟩
\srefaddidkey{phrase}
\omdaddkey{phrase}{style}
\omdaddkey{phrase}{class}
\omdaddkey{phrase}{index}
\omdaddkey{phrase}{verbalizes}
\omdaddkey{phrase}{type}
\newenvironment{phrase}[1][]{\omdsetkeys{phrase}{#1}}{}
⟨/package⟩
⟨*ltxml⟩
DefKeyVal('phrase','id','Semiverbatim');
```

```
DefKeyVal('phrase','style','Semiverbatim');
DefKeyVal('phrase','class','Semiverbatim');
DefKeyVal('phrase','index','Semiverbatim');
DefKeyVal('phrase','verbalizes','Semiverbatim');
DefKeyVal('phrase','type','Semiverbatim');
DefConstructor('\phrase OptionalKeyVals:phrase {}',
       "<omdoc:phrase %&KeyVals(#1)>#2</omdoc:phrase>");
⟨/ltxml⟩
```

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

```
⟨*package⟩
\def\nlex#1{\green{\sl{#1}}}
\def\nlcex#1{*\green{\sl{#1}}}
⟨/package⟩
⟨*ltxml⟩
DefConstructor('\nlex{}',
   "<omdoc:phrase type='nlex'>#1</omdoc:phrase>");
DefConstructor('\nlcex{}',
   "<omdoc:phrase type='nlcex'>#1</omdoc:phrase>");
⟨/ltxml⟩
```

`sinlinequote`

```
⟨*package⟩
\def\@sinlinequote#1{``{\sl{#1}}''}
\def\@@sinlinequote#1#2{\@sinlinequote{#2}~#1}
\newcommand{\sinlinequote}[2][]
{\def\@opt{#1}\ifx\@opt\@empty\@sinlinequote{#2}\else\@@sinlinequote\@opt{#2}\fi}
⟨/package⟩
⟨*ltxml⟩
DefConstructor('\sinlinequote [] {}',
               "<omdoc:phrase type='inlinequote'>"
               . "?#1(<dc:source ?#locator(stex:srcref='#locator')()>#1</dc:source>\n)()"
               . "#2"
              . "</omdoc:phrase>");
⟨/ltxml⟩
```

6.2 Block-Level Markup

EdNote(3) `sblockquote` [3]

```
⟨*package⟩
\def\begin@sblockquote{\begin{quote}\sl}
\def\end@sblockquote{\end{quote}}
\def\begin@@sblockquote#1{\begin@sblockquote}
\def\end@@sblockquote#1{\def\@@lec##1{{\rm ##1}}\@lec{#1}\end@sblockquote}
\newenvironment{sblockquote}[1][]
  {\def\@opt{#1}\ifx\@opt\@empty\begin@sblockquote\else\begin@@sblockquote\@opt\fi}
  {\ifx\@opt\@empty\end@sblockquote\else\end@@sblockquote\@opt\fi}
```

[3] EdNote: describe above

```
⟨/package⟩
⟨*ltxml⟩
DefEnvironment('{sblockquote} []',
               "<omdoc:omgroup type='quote'"
               . "?#1(<dc:source>#1</dc:source>\n)()"
               . "#body"
              . "</omdoc:omgroup>");
⟨/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.

```
⟨*package⟩
\providecommand{\@@lec}[1]{(#1)}
\def\@lec#1{\strut\hfil\strut\null\nobreak\hfill\hbox{\@@lec{#1}}}
\def\lec#1{\@lec{#1}\par}
⟨/package⟩
⟨*ltxml⟩
DefConstructor('\lec{}',
   "\n<omdoc:note type='line-end-comment'>#1</omdoc:note>");
⟨/ltxml⟩
```

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

```
⟨ltxml⟩RawTeX('
⟨*ltxml | package⟩
\newcommand\mygraphics[2][]{\includegraphics[#1]{#2}}
\newcommand\mycgraphics[2][]{\begin{center}\includegraphics[#1]{#2}\end{center}}
\newcommand\mybgraphics[2][]{\fbox{\includegraphics[#1]{#2}\end{center}}}
⟨/ltxml | package⟩
⟨ltxml⟩');
```

6.3 Index Markup

```
⟨*package⟩
\newcommand{\omdoc@index}[2][]{\def\@test{#1}%
\ifindex\ifx\@test\@empty\index{#2}\else\index{#1@#2}\fi\fi}
\newcommand{\indexalt}[3][]{{#2}\omdoc@index[#1]{#3}}              % word in text and index
\newcommand{\indextoo}[2][]{{#2}\omdoc@index[#1]{#2}}              % word in text and index
⟨/package⟩
```

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

```
⟨*package⟩
\newcommand{\@twin}[3][]{\def\@test{#1}%
\ifindex\ifx\@test\@empty\index{#2!#3}\else\index{#1@#2!#3}\fi\index{#3!#2}\fi}
\newcommand{\twinalt}[4][]{#2\@twin[#1]{#3}{#4}}
```

```
\newcommand{\twintoo}[3][]{{#2 #3}\@twin[#1]{#2}{#3}}             % and use the word compound t
⟨/package⟩
```

this puts adjectivized two-compound words into the index in various permutations[4]

EdNote(4)

```
⟨*package⟩
\newcommand{\@atwin}[4][]{\def\@test{#1}%
\ifindex\ifx\@test\@empty\index{#2!#3!#4}\else\index{#1@#2!#3!#4}\fi\index{#3!#2 (#1)}\fi}
\newcommand{\atwinalt}[5][]{#2\@atwin[#1]{#3}{#4}{#4}}
\newcommand{\atwintoo}[4][]{{#2 #3 #4}\@atwin[#1]{#2}{#3}{#4}}         % and use it too
⟨/package⟩
⟨*ltxml⟩
DefConstructor('\indextoo[]{}',
       "<omdoc:idx>"
      .  "<omdoc:idt>#2</omdoc:idt>"
      .  "<omdoc:ide ?#1(sort-by='#1')()>"
      .    "<omdoc:idp>#2</omdoc:idp>"
      .  "</omdoc:ide>"
      ."</omdoc:idx>");
DefConstructor('\indexalt[]{}{}',
       "<omdoc:idx>"
      .  "<omdoc:idt>#2</omdoc:idt>"
      .  "<omdoc:ide ?#1(sort-by='#1')()>"
      .    "<omdoc:idp>#3</omdoc:idp>"
      .  "</omdoc:ide>"
      ."</omdoc:idx>");
⟨/ltxml⟩
⟨*ltxml⟩
DefConstructor('\twintoo[]{}{}',
       "<omdoc:idx>"
      .  "<omdoc:idt>#2 #3</omdoc:idt>"
      .  "<omdoc:ide ?#1(sort-by='#1')()>"
      .    "<omdoc:idp>#2</omdoc:idp>"
      .    "<omdoc:idp>#3</omdoc:idp>"
      .  "</omdoc:ide>"
      ."</omdoc:idx>");
DefConstructor('\twinalt[]{}{}{}',
       "<omdoc:idx>"
      .  "<omdoc:idt>#2</omdoc:idt>"
      .  "<omdoc:ide ?#1(sort-by='#1')()>"
      .    "<omdoc:idp>#2</omdoc:idp>"
      .    "<omdoc:idp>#3</omdoc:idp>"
      .  "</omdoc:ide>"
      ."</omdoc:idx>");
⟨/ltxml⟩
⟨*ltxml⟩
DefConstructor('\atwintoo[]{}{}{}',
       "<omdoc:idx>"
```

[4] EdNote: what to do with the optional argument here and below?

```
      .  "<omdoc:idt>#2 #3</omdoc:idt>"
      .  "<omdoc:ide ?#1(sort-by='#1')()>"
      .    "<omdoc:idp>#2</omdoc:idp>"
      .    "<omdoc:idp>#3</omdoc:idp>"
      .    "<omdoc:idp>#4</omdoc:idp>"
      .  "</omdoc:ide>"
      ."</omdoc:idx>");

DefConstructor('\atwinalt[]{}{}{}{}',
       "<omdoc:idx>"
      .  "<omdoc:idt>#2</omdoc:idt>"
      .  "<omdoc:ide ?#1(sort-by='#1')()>"
      .    "<omdoc:idp>#2</omdoc:idp>"
      .    "<omdoc:idp>#3</omdoc:idp>"
      .    "<omdoc:idp>#4</omdoc:idp>"
      .  "</omdoc:ide>"
      ."</omdoc:idx>");
⟨/ltxml⟩
```

and finally we supply the `\printindex` command

```
⟨*package⟩
\def\printindex{\IfFileExists{\jobname.ind}{\input{\jobname.ind}}{}}
⟨/package⟩
⟨*ltxml⟩
DefConstructor('\printindex','<omdoc:index/>');
⟨/ltxml⟩
```

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

```
⟨*ltxml⟩
DefConstructor('\par',sub { $_[0]->maybeCloseElement('omdoc:p'); },
                      alias=>"\\par\n");
Tag('omdoc:p', autoClose=>1, autoOpen=>1);
Tag('omdoc:CMP', autoClose=>1, autoOpen=>1);
Tag('omdoc:omtext', autoClose=>1, autoOpen=>1);
⟨/ltxml⟩#$
```

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

```
⟨package⟩\def\omspace#1{\hspace*{#1}}
⟨*ltxml⟩
DefConstructor('\omspace{}','');
DefConstructor('\emph{}',"<omdoc:phrase class='emphasis'>#1</omdoc:phrase>");
DefConstructor('\em',"<omdoc:phrase class='emphasis'>");
```

```
DefConstructor('\texttt{}',"<omdoc:phrase class='code'>#1</omdoc:phrase>");
DefConstructor('\textbf{}',"<omdoc:phrase class='bold'>#1</omdoc:phrase>");
Tag('omdoc:phrase', autoClose=>1);
DefEnvironment('{center}','#body');
DefEnvironment('{flushleft}','#body');
DefEnvironment('{flushright}','#body');
DefEnvironment('{minipage}[]{}','#body');
DefEnvironment('{quote}',
    "<omdoc:phrase type='quote' style='display:block'>"
   .  "#body"
   ."</omdoc:phrase>");
DefEnvironment('{quotation}',
    "<omdoc:phrase type='quote' style='display:block'>
   .  "#body"
   ."</omdoc:phrase>");
DefEnvironment('{LARGE}','#body');
DefEnvironment('{Large}','#body');
DefEnvironment('{large}','#body');
DefEnvironment('{small}','#body');
DefEnvironment('{footnotesize}','#body');
DefEnvironment('{tiny}','#body');
DefEnvironment('{scriptsize}','#body');
DefConstructor('\LARGE','');
DefConstructor('\Large','');
DefConstructor('\large','');
DefConstructor('\small','');
DefConstructor('\footnotesize','');
DefConstructor('\scriptsize','');
DefConstructor('\tiny','');
DefConstructor('\fbox{}','#1');
DefConstructor('\footnote[]{}',
       "<omdoc:note class='foot' ?#1(mark='#1')>#2</omdoc:note>");
DefConstructor('\footnotemark[]',"");
DefConstructor('\footnotetext[]{}',
       "<omdoc:note class='foot' ?#1(mark='#1')>#2</omdoc:note>");
DefConstructor('\sf', '');
DefConstructor('\sc', '');
⟨/ltxml⟩
```

6.5 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.

```
⟨*ltxml⟩
Tag('omdoc:p',afterOpen=>\&numberIt,afterClose=>\&locateIt);
Tag('omdoc:omtext',afterOpen=>\&numberIt,afterClose=>\&locateIt);
Tag('omdoc:omgroup',afterOpen=>\&numberIt,afterClose=>\&locateIt);
Tag('omdoc:CMP',afterOpen=>\&numberIt,afterClose=>\&locateIt);
Tag('omdoc:link',afterOpen=>\&numberIt,afterClose=>\&locateIt);
```

```
Tag('omdoc:meta',afterOpen=>\&numberIt,afterClose=>\&locateIt);
Tag('omdoc:resource',afterOpen=>\&numberIt,afterClose=>\&locateIt);
Tag('omdoc:ul',afterOpen=>\&numberIt,afterClose=>\&locateIt);
Tag('omdoc:li',afterOpen=>\&numberIt,afterClose=>\&locateIt);
Tag('omdoc:di',afterOpen=>\&numberIt,afterClose=>\&locateIt);
Tag('omdoc:dt',afterOpen=>\&numberIt,afterClose=>\&locateIt);
Tag('omdoc:dd',afterOpen=>\&numberIt,afterClose=>\&locateIt);
Tag('omdoc:ol',afterOpen=>\&numberIt,afterClose=>\&locateIt);
Tag('omdoc:dl',afterOpen=>\&numberIt,afterClose=>\&locateIt);
Tag('omdoc:idx',afterOpen=>\&numberIt,afterClose=>\&locateIt);
Tag('omdoc:phrase',afterOpen=>\&numberIt,afterClose=>\&locateIt);
Tag('omdoc:note',afterOpen=>\&numberIt,afterClose=>\&locateIt);
⟨/ltxml⟩
```

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

```
⟨*ltxml⟩
Tag('ltx:tabular',afterOpen=>\&numberIt,afterClose=>\&locateIt);
Tag('ltx:thead',afterOpen=>\&numberIt,afterClose=>\&locateIt);
Tag('ltx:td',afterOpen=>\&numberIt,afterClose=>\&locateIt);
Tag('ltx:tr',afterOpen=>\&numberIt,afterClose=>\&locateIt);
Tag('ltx:caption',afterOpen=>\&numberIt,afterClose=>\&locateIt);
⟨/ltxml⟩
```

The `numberIt` procedure gets the prefix from first parent with an `xml:id` attribute and then extends it with a label that reflects the number of preceding siblings, provided that there is not already an identifier. Additionally, it estimates an XPointer position in the original document of the command sequence which produced the tag. The `locateIt` subroutine is a sibling of `numberIt` as it is required as an `afterClose` handle for tags produced by LaTeX environments, as opposed to commands. `locateIt` estimates an XPointer end position of the LaTeX environment, allowing to meaningfully locate the entire environment at the source.

```
⟨*ltxml⟩
sub numberIt {
  my($document,$node,$whatsit)=@_;
  my(@parents)=$document->findnodes('ancestor::*[@xml:id]',$node);
  my $prefix= (@parents ? $parents[$#parents]->getAttribute('xml:id')."." : '');
  my(@siblings)=$document->findnodes('preceding-sibling::*[@xml:id]',$node);
  my $n = scalar(@siblings)+1;
  my $id = ($node -> getAttribute('xml:id'));
  $node->setAttribute('xml:id'=>$prefix."p$n") unless $id;
  my $about = $node -> getAttribute('about');
  $node->setAttribute('about'=>'#'.$node->getAttribute('xml:id')) unless $about;
  #Also, provide locators:
  my $locator = $whatsit->getProperty('locator');
  #Need to inherit locators if missing:
  $locator = (@parents ? $parents[$#parents]->getAttribute('stex:srcref') : '') unless $locator
  $node->setAttribute('stex:srcref'=>$locator) if $locator; }
sub locateIt {
  my($document,$node,$whatsit)=@_;
```

```
  #Estimate trailer locator:
  my $trailer = $whatsit->getProperty('trailer');
  return unless $trailer; #Nothing we can do if the trailer isn't defined
  $trailer = $trailer->getLocator;
  return unless ($trailer &&  $trailer!~/^\s*$/); #Useless if broken
  my $locator = $node->getAttribute('stex:srcref');
  if ($locator) {
    $locator =~ /^(.+from=\d+;\d+)/;
    my $from = $1;
    $trailer =~ /(,to=\d+;\d+.+)$/;
    my $to = $1;
    $locator = $from.$to;
  } else {
    $locator = $trailer; #This should never happen
  }
  $node->setAttribute('stex:srcref' => $locator);
}
```
⟨/ltxml⟩#$

6.6 Finale

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

```
⟨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

[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.