Semantic Markup for Open Mathematical Documents in LaTeX*

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

May 7, 2008

Abstract

The omdoc packge 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 a the infrastructure for writing OMDoc documents in LaTeX. This includes a simple structure sharing mechanism for sTeX that allows to to move from a copy-and-paste document development model to a copy-and-reference model, which conserves space and simplifies document management. The augmented structure can be used by MKM systems for added-value services, either directly from the sTeX sources, or after translation.

*Version v0.3 (last revised 2007/09/09)

1 Introduction

The `omdoc` package supplies macros and environment that allow to label document fragements and to reference them later in the same document or in other documents. In essence, this enhances the docuent-as-trees model to documents-as-directed-acyclic-graphs (DAG) model. This structure can be used by MKM systems for added-value services, either directly from the sTEX sources, or after translation. Currently, trans-document referencing provided by this package can conly be used in the sTEX collection.

sTEX is 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).

DAG models of documents allow to replace the "Copy and Paste" in the source document with a label-and-reference model where document are shared in the document source and the formatter does the copying during document formatting/presentation.[1][2][3]

EdNote(1)
EdNote(2)
EdNote(3)

2 The User Interface

2.1 Document Structure

`omgroup` The structure of the document is given by the `omgroup` environment just like in OMDoc.

2.2 Providing IDs for OMDoc Elements

Some of the OMDoc elements need IDs to function corrrectly. The general strategy here is to equip the sTEX macros with keys, so that the author can specify meaningful ones, but to let the transformation give default ones if the author did not.

2.3 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 rhethorical 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

[1] EdNote: talk about the advantages and give an example.
[2] EdNote: is there a way to load documents at URIs in LaTeX?
[3] EdNote: integrate with latexml's XMRef in the Math mode.

"transition between two statements"), so additionally, a source should be specified
EdNote(4) using the `from` key.[4]

2.4 Structure Sharing

`\STRlabel` `\STRcopy` The `\STRlabel` macro takes two arguments: a label and the content and stores the the content for later use by `\STRcopy{label}`, which expands to the previously stored content.

`\STRsemantics` The `\STRlabel` macro has a variant `\STRsemantics`, where the label argument is optional, and which takes a third argument, which is ignored in LaTeX. This allows to specify the meaning of the content (whatever that may mean) in cases, where the source document is not formatted for presentation, but is transformed
EdNote(5) into some content markup format. [5]

2.5 Phrase-Level Markup

`phrase` The `phrase` enviornment allows to mark up phrases with semantic information. It takes an optional `KeyVal` argument with the keys

3 Implementation: The OMDoc Class

We load `article.cls`, and the desired packages. For the LaTeXML bindings, we make sure the right packages are loaded.

```
1 ⟨*cls⟩
2 \LoadClass{article}
3 \RequirePackage{omdoc}
4 ⟨/cls⟩
5 ⟨*ltxml.cls⟩
6 # -*- CPERL -*-
7 package LaTeXML::Package::Pool;
8 use strict;
9 use LaTeXML::Package;
10 LoadClass('article');
11 ⟨/ltxml.cls⟩
```

Now, we also need to register the namespace prefixes for LaTeXML to use.

```
12 ⟨*ltxml.cls⟩
13 RegisterNamespace('omdoc'=>"http://www.mathweb.org/omdoc"); # OMDoc namespace
14 RegisterNamespace(om=>"http://www.openmath.org/OpenMath");
15 RegisterNamespace('m'=>"http://www.w3.org/1998/Math/MathML");
16 RegisterNamespace('dc'=>"http://purl.org/dc/elements/1.1/");
17 RegisterNamespace('cc'=>"http://creativecommons.org/ns");
18 ⟨/ltxml.cls⟩
```

[4] EdNote: describe the keys more fully
[5] EdNote: make an example

Since we are dealing with a class, we need to set up the document type in the LaTeXML bindings.

```
⟨*ltxml.cls⟩
RelaxNGSchema('/Users/kohlhase/stex/rnc/omdoc+ltxml',
        '#default'=>"http://www.mathweb.org/omdoc",
        'om'=>"http://www.openmath.org/OpenMath",
        'm'=>"http://www.w3.org/1998/Math/MathML",
        'dc'=>"http://purl.org/dc/elements/1.1/",
        'cc'=>"http://creativecommons.org/ns",
        'ltx'=>"http://dlmf.nist.gov/LaTeXML");
⟨/ltxml.cls⟩
```

Then we load the `omdoc` package, which we define separately in the next section so that it can be loaded separately[6]

EdNote(6)

```
⟨*ltxml.cls⟩
RequirePackage('omdoc');
⟨/ltxml.cls⟩
```

Now, we will define the environments we need. The top-level one is the `document` environment, which we need to provide with an identifier.[7]

EdNote(7)

```
⟨*ltxml.cls⟩
DefEnvironment('{document}','<omdoc:omdoc>#body</omdoc:omdoc>',
       beforeDigest=> sub { AssignValue(inPreamble=>0); },
       afterDigest=> sub { $_[0]->getGullet->flush; return; });
⟨/ltxml.cls⟩\iffalse$\fi
```

4 Implementation: OMDoc Package

We first need to set up the packages by requiring the `KeyVal` package to be loaded (in the ritht version).

```
⟨package⟩\RequirePackage{keyval}[1997/11/10]
⟨*ltxml.sty⟩
package LaTeXML::Package::Pool;
use strict;
use LaTeXML::Package;
RequirePackage('keyval');
⟨/ltxml.sty⟩
```

4.1 Document Structure

The structure of the document is given by the `omgroup` environment just like in OMDoc. The hierarchy is adjusted automatically[8]

EdNote(8)

```
⟨*package⟩
```

[6] EdNote: reword

[7] EdNote: probably move to the omdoc package

[8] EdNote: maybe define the toplevel according to a param, need to know how to detect that the chapter macro exists.

```
\define@key{omgroup}{id}{\label{#1}}
\define@key{omgroup}{type}{\def\omgroup@type{#1}}
\define@key{omgroup}{display}{\def\omgroup@display{#1}}
\def\omgroup@flow{flow}
\newcount\section@level\section@level=1
\newenvironment{omgroup}[2][]% title
{\setkeys{omgroup}{#1}
\ifx\omgroup@display\omgroup@flow{#2}\else
\ifnum\section@level=1\section{#2}\fi
\ifnum\section@level=2\subsection{#2}\fi
\ifnum\section@level=3\subsubsection{#2}\fi
\ifnum\section@level=4\paragraph{#2}\fi
\advance\section@level by 1\fi}{}
\def\tgroup{\omgroup}\def\endtgroup{\endomgroup}
⟨/package⟩
⟨*ltxml.sty⟩
DefKeyVal('omgroup','id','Semiverbatim');
DefKeyVal('omgroup','type','Semiverbatim');
DefKeyVal('omgroup','display','Semiverbatim');
DefEnvironment('{omgroup} OptionalKeyVals:omgroup {}',
               "<omdoc:omgroup ?&KeyVal(#1,'id')(xml:id='#1')() ?&KeyVal(#1,'type')(type='#1')(
          . "?#1(<omdoc:metadata><dc:title>#1</dc:title></omdoc:metadata>\n)()"
       . "#body\n"
     . "</omdoc:omgroup>");

DefEnvironment('{tgroup} OptionalKeyVals:omgroup {}',
               "<omdoc:tgroup ?&KeyVal(#1,'id')(xml:id='#1')() ?&KeyVal(#1,'type')(type='#1')()
          . "?#1(<omdoc:metadata><dc:title>#1</dc:title></omdoc:metadata>\n)()"
       . "#body\n"
     . "</omdoc:tgroup>");
⟨/ltxml.sty⟩
```

the `itemize`, `description`, and `enumerate` environments do double duty in OMDoc,

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

2. inside a `CMP`, they are transformed into `<ul>`, `<dl>`, and `<ol>` elements like in html.

3. Outside a `CMP`, `\item` is turned into `\ignoe@item`

```
⟨*ltxml.sty⟩
DefParameterType('IfBeginFollows', sub {
   my ($gullet) = @_;
   $gullet->skipSpaces;
   return 1 unless $gullet->ifNext(T_CS('\begin'));
```

```
  return; },
 reversion=>'', optional=>1);

Let('\group@item@maybe@unwrap','\relax');

DefMacro('\group@item[] IfBeginFollows', sub {
  my($gullet,$tag,$needwrapper)=@_;
  ( T_CS('\group@item@maybe@unwrap'),
    ($needwrapper ? (Invocation(T_CS('\group@item@wrap'),$tag)->unlist) : ()) ); });

DefConstructor('\group@item@wrap {}',
       "<omdoc:omtext>"
       . "?#1(<omdoc:metadata><dc:title>#1</dc:title></omdoc:metadata>)()"
              . "<omdoc:CMP><omdoc:p>",
       beforeDigest=>sub {
 Let('\group@item@maybe@unwrap','\group@item@unwrap');
 $_[0]->bgroup;
 useCMPItemizations();
 return; },
       properties=>sub{ RefStepItemCounter(); });
DefConstructor('\group@item@unwrap',
       "</omdoc:p></omdoc:CMP></omdoc:omtext>",
       beforeDigest=>sub {
 $_[0]->egroup;
 Let('\group@item@maybe@unwrap','\relax'); });

Let('group@item@maybe@unwrap','\relax');

Let('\itemize@item'=>'\group@item');
Let('\enumerate@item'=>'\group@item');
Let('\description@item'=>'\group@item');

DefEnvironment('{itemize}',
       "<omdoc:omgroup xml:id='#id' type='itemize'>#body</omdoc:omgroup>",
       properties=>sub { beginItemize('itemize'); },
       beforeDigestEnd=>sub { Digest(T_CS('\group@item@maybe@unwrap')); });
DefEnvironment('{enumerate}',
       "<omdoc:omgroup xml:id='#id' type='enumerate'>#body</omdoc:omgroup>",
       properties=>sub { beginItemize('enumerate'); },
       beforeDigestEnd=>sub { Digest(T_CS('\group@item@maybe@unwrap')); });
DefEnvironment('{description}',
       "<omdoc:omgroup xml:id='#id' type='description'>#body</omdoc:omgroup>",
       properties=>sub { beginItemize('description'); },
       beforeDigestEnd=>sub { Digest(T_CS('\group@item@maybe@unwrap')); });
⟨/ltxml.sty⟩
```

Inside a `theory` element , use `tgroup` instead of `omgroup`, otherwise, same as default case

```
⟨*ltxml.sty⟩
Let('\tgroup@itemize@item'=>'\group@item');
```

```
Let('\tgroup@enumerate@item'=>'\group@item');
Let('\tgroup@description@item'=>'\group@item');

DefEnvironment('{tgroup@itemize}',
       "<omdoc:tgroup xml:id='#id' type='itemize'>#body</omdoc:tgroup>",
       properties=>sub { beginItemize('tgroup@itemize'); },
       beforeDigest=>sub { Let(T_CS('\end{itemize}') =>T_CS('\end{tgroup@itemize}')); },
       beforeDigestEnd=>sub { Digest(T_CS('\group@item@maybe@unwrap')); });
DefEnvironment('{tgroup@enumerate}',
       "<omdoc:tgroup xml:id='#id' type='enumerate'>#body</omdoc:tgroup>",
       properties=>sub { beginItemize('tgroup@enumerate'); },
       beforeDigest=>sub{ Let(T_CS('\end{enumerate}') =>T_CS('\end{tgroup@enumerate}'));},
       beforeDigestEnd=>sub { Digest(T_CS('\group@item@maybe@unwrap')); });
DefEnvironment('{tgroup@description}',
       "<omdoc:tgroup xml:id='#id' type='description'>#body</omdoc:tgroup>",
       properties=>sub { beginItemize('tgroup@description'); },
       beforeDigest=>sub{ Let(T_CS('\end{description}') =>T_CS('\end{tgroup@description}'));},
       beforeDigestEnd=>sub { Digest(T_CS('\group@item@maybe@unwrap')); });

sub useTheoryItemizations {
    Let(T_CS('\begin{itemize}')    =>T_CS('\begin{tgroup@itemize}'));
    Let(T_CS('\begin{enumerate}')  =>T_CS('\begin{tgroup@enumerate}'));
    Let(T_CS('\begin{description}')=>T_CS('\begin{tgroup@description}'));
    return; }
⟨/ltxml.sty⟩
```

Inside a CMP, we generate omdoc:li, omdoc:di with autoclose

```
⟨*ltxml.sty⟩
DefConstructor('\CMP@itemize@item[]',
       '<omdoc:li>'
       . '?#1(<omdoc:metadata><dc:title>#1</dc:title></omdoc:metadata>)()',
       properties=>sub{ RefStepItemCounter(); });

DefConstructor('\CMP@enumerate@item[]',
       '<omdoc:li>'
       . '?#1(<omdoc:metadata><dc:title>#1</dc:title></omdoc:metadata>)()',
       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 xml:id='#id'>#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; }

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.sty⟩
```

4.2 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.

```
⟨*package⟩
\define@key{stat}{id}{\def\st@id{#1}}
\define@key{stat}{display}{\def\st@display{#1}}
\define@key{stat}{for}{\def\stat@for{#1}}
\define@key{stat}{from}{\def\stat@from{#1}}
\define@key{stat}{type}{\def\stat@type{#1}}
\define@key{stat}{title}{\def\stat@title{#1}}
\define@key{stat}{continues}{\def\stat@continues{#1}}
⟨/package⟩
⟨*ltxml.sty⟩
DefKeyVal('stat','id','Semiverbatim');
DefKeyVal('stat','for','Semiverbatim');
DefKeyVal('stat','title','Semiverbatim');
DefKeyVal('stat','from','Semiverbatim');
DefKeyVal('stat','type','Semiverbatim');
DefKeyVal('stat','display','Semiverbatim'); # not used at the moment
⟨/ltxml.sty⟩
```

`\show@st@keys@aux` We now define a macro that shows the `st` keys, if in draft mode, they annotate the document with key/value pairs.

```
⟨*package⟩
\def\show@st@keys@aux{%
\@ifundefined{st@id}{}{\ifst@id{id=\st@id},\fi}%
```

```
\@ifundefined{st@display}{}{\ifst@display{display=\st@display}\fi}}
\def\clear@st@keys{\let\st@id=\relax\let\st@display=\relax}
⟨/package⟩
```

`\show@stat@keys@aux` we do the same for the `stat` keys.

```
⟨*package⟩
\def\show@stat@keys@aux{%
\@ifundefined{st@for}{}{\ifstat@for{for=\stat@for},\fi}%
\@ifundefined{st@from}{}{\ifstat@from{from=\stat@from},\fi}%
\@ifundefined{st@type}{}{\ifstat@type{type=\stat@type},\fi}%
\@ifundefined{st@title}{}{\ifstat@title{title=\stat@title},\fi}%
\@ifundefined{st@continues}{}{\ifstat@continues{continues=\stat@continues}\fi}}
⟨/package⟩
```

`\show@stat@keys` and combine them, so that the code is more readable.

```
⟨*package⟩
\def\show@stat@keys#1{\footnote{#1[\show@st@keys@aux\show@stat@keys@aux]}%
\clear@st@keys\clear@stat@keys}
\def\clear@stat@keys{\let\stat@for=\relax\let\stat@from=\relax%
\let\stat@type=\relax\let\stat@title=\relax\let\stat@continues=\relax}
⟨/package⟩
```

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

```
⟨package⟩\def\st@flow{flow}
```

`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.

```
⟨*package⟩
\def\omtext@pre@skip{\smallskip}
\def\omtext@post@skip{}
\newenvironment{omtext}[1][]{\setkeys{stat}{#1}{% keyval args
\def\lec#1{\@lec{#1}}% so the trailing \par does not get into the way
\omtext@pre@skip\par\noindent
\@ifundefined{stat@title}{}{\ifx\st@display\st@flow\else\stDMemph{\stat@title}:\fi}}
\ifst@env\show@stat@keys{omtext:\stat@title}\fi
\omtext@post@skip}{}
⟨/package⟩
⟨*ltxml.sty⟩
DefCMPEnvironment('{omtext} OptionalKeyVals:stat',
  "<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')(<omdoc:metadata><dc:title>&KeyVal(#1,'title')</dc:title></omdoc:metad
  . "<omdoc:CMP><omdoc:p>#body</omdoc:p></omdoc:CMP>"
  . "</omdoc:omtext>\n");
⟨/ltxml.sty⟩
```

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

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

4.3 Structure Sharing

`\STRlabel` The main macro, it it used to attach a label to some text expansion. Later on, using the `\STRcopy` macro, the author can use this label to get the expansion originally assigned.

```
⟨*package⟩
\long\def\STRlabel#1#2{\STRlabeldef{#1}{#2}{#2}}
⟨/package⟩
⟨*ltxml.sty⟩
DefConstructor('\STRlabel{}{}', sub {
  my($document,$label,$object)=@_;
  $document->absorb($object);
  $document->addAttribute('xml:id'=>ToString($label)) if $label; });
⟨/ltxml.sty⟩
```

`\STRcopy` The `\STRcopy` macro is used to call the expansion of a given label. In case the label is not defined it will issue a warning.

```
⟨*package⟩
\def\STRcopy#1{\expandafter\ifx\csname STR@#1\endcsname\relax
\message{STR warning: reference #1 undefined!}
\else\csname STR@#1\endcsname\fi}
⟨/package⟩
⟨*ltxml.sty⟩
DefConstructor('\STRcopy{}',"<omdoc:ref xref='#1'/>");
⟨/ltxml.sty⟩
```

`\@semantics` if we have a presentation form and a semantic form, then we can use

```
⟨*package⟩
\long\def\@semantics[#1]#2#3{#2\STRlabeldef{#1}{#2}}
⟨/package⟩
```

`\STRlabeldef` This is the macro that does the actual labelling. Is it called inside `\STRlabel`

```
⟨*package⟩
\def\STRlabeldef#1{\expandafter\gdef\csname STR@#1\endcsname}
⟨/package⟩
⟨*ltxml.sty⟩
```

```
DefMacro('\STRlabeldef{}{}', "");
⟨/ltxml.sty⟩
```

EdNote(9) `\STRsemantics` [9]

```
⟨*package⟩
\def\STRsemantics{\@ifnextchar[\@semantics{\@semantics[]}}
⟨/package⟩
⟨*ltxml.sty⟩
DefConstructor('\STRsemantics[]{}{}', sub {
  my($document,$label,$ignore,$object)=@_;
  $document->absorb($object);
  $document->addAttribute('xml:id'=>ToString($label)) if $label; });
⟨/ltxml.sty⟩
```

4.4 Phrase-level Markup

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

```
⟨*package⟩
\define@key{phrase}{id}{}
\define@key{phrase}{style}{}
\define@key{phrase}{class}{}
\define@key{phrase}{index}{}
\define@key{phrase}{verbalizes}{}
\define@key{phrase}{type}{}
\newenvironment{phrase}[1][]{\setkeys{phrase}{#1}}{}
⟨/package⟩
⟨*ltxml.sty⟩
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)>#body</omdoc:phrase>");
⟨/ltxml.sty⟩
```

`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.sty⟩
DefConstructor('\nlex{}',"<omdoc:phrase type='nlex'>#1</omdoc:phrase>");
DefConstructor('\nlcex{}',"<omdoc:phrase type='nlcex'>#1</omdoc:phrase>");
⟨/ltxml.sty⟩
```

EdNote(10) `inlinequote` [10]

[9] EdNote: some explanation here
[10] EdNote: describe above

```
⟨package⟩\def\inlinequote#1{‘‘{\sl{#1}}’’}
⟨ltxml.sty⟩DefConstructor('\inlinequote{}',"<omdoc:phrase type='inlinequote'>#1</omdoc:phrase>");
```

4.5 Colors

`blue, red, green, magenta`

We will use the following abbreviations for colors from `color.sty`

```
⟨*package⟩
\def\blue#1{\textcolor{blue}{#1}}
\def\red#1{\textcolor{red}{#1}}
\def\green#1{\textcolor{green}{#1}}
\def\cyan#1{\textcolor{cyan}{#1}}
\def\magenta#1{\textcolor{magenta}{#1}}
\def\brown#1{\textcolor{brown}{#1}}
\def\yellow#1{\textcolor{yellow}{#1}}
⟨/package⟩
```

For the LaTeXML bindings, we go a generic route, we replace `\blue{#1}` by `{\@omdoc@color{blue}\@omdoc@color@content{#1}}`.

```
⟨*ltxml.sty⟩
sub omdocColorMacro {
  my ($color, @args) = @_;
  my $tok_color = TokenizeInternal($color);
  (T_BEGIN, T_CS('\@omdoc@color'), T_BEGIN, $tok_color->unlist,
   T_END, T_CS('\@omdoc@color@content'), T_OTHER('['), $tok_color->unlist, T_OTHER(']'),
   T_BEGIN, $args[1]->unlist, T_END, T_END); }
DefMacro('\@omdoc@color{}', sub { MergeFont(color=>$_[1]->toString); return; });
⟨/ltxml.sty⟩
```

Ideally, here we will remove the optional argument and have a conversion module add the attribute at the end (or maybe add it just for math?) or, we can take the attributes for style from the current font ?

```
⟨*ltxml.sty⟩
DefConstructor('\@omdoc@color@content[]{}',
       "?#isMath(#2)(<omdoc:phrase ?#1(style='color:#1')()>#2</omdoc:phrase>)");
foreach my $color(qw(blue red green magenta cyan brown yellow)) {
  DefMacro("\\".$color.'{}', sub { omdocColorMacro($color, @_); }); }
⟨/ltxml.sty⟩
```

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

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

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.sty⟩
DefConstructor('\omspace{}','');
DefConstructor('\emph{}',"<omdoc:phrase class='emphasis'>#1</omdoc:phrase>");
DefConstructor('\em',"<omdoc:phrase class='emphasis'>");
Tag('omdoc:phrase', autoClose=>1);
DefEnvironment('{center}','#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:phr
DefEnvironment('{small}','#body');
DefEnvironment('{footnotesize}','#body');
DefEnvironment('{tiny}','#body');
DefEnvironment('{scriptsize}','#body');
DefConstructor('\LARGE','');
DefConstructor('\Large','');
DefConstructor('\large','');
DefConstructor('\small','');

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.sty⟩
```

4.7 Providing IDs for OMDoc Elements

To provide default identifiers (see section 2.2), we tag all OMDoc elements that allow `xml:id` attributes by excuting the `numberIt` procedure below.

```
⟨*ltxml.sty⟩
Tag('omdoc:p',afterOpen=>\&numberIt);
Tag('omdoc:omtext',afterOpen=>\&numberIt);
Tag('omdoc:omgroup',afterOpen=>\&numberIt);
Tag('omdoc:tgroup',afterOpen=>\&numberIt);
Tag('omdoc:CMP',afterOpen=>\&numberIt);
⟨/ltxml.sty⟩
```

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.

```
⟨*ltxml.sty⟩
```

```
sub numberIt {
  my($document,$node,$whatsit)=@_;
  my(@parents)=$document->findnodes('ancestor::*[@xml:id]',$node); # find 1st id'd parent.
  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; }
⟨/ltxml.sty⟩
```

4.8 Leftovers

```
⟨*ltxml.sty⟩
DefConstructor('\url Semiverbatim',"<omdoc:link href='#1'>#1</omdoc:link>");
⟨/ltxml.sty⟩
```

EdNote(11)

[11] and finally, we need to terminate the file with a success mark for perl.

```
⟨ltxml.sty | ltxml.cls⟩1;
```

[11] EdNote: this should be handled differently, omdoc.sty should include url and give a new macro for it, which we then use in omdoc