`omdoc.sty/cls`: Semantic Markup for Open Mathematical Documents in LaTeX

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

November 10, 2015

Abstract

The `omdoc` 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 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.

Contents

1 Introduction

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). The package supports direct translation to the OMDoc format [Koh06]

The `omdoc` package supplies macros and environment that allow to label document fragments and to reference them later in the same document or in other documents. In essence, this enhances the document-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 only be used in the sTeX collection.

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]

EdN:1

2 The User Interface

The `omdoc` package generates four files: `omdoc.cls`, `omdoc.sty` and their LaTeXML bindings `omdoc.cls.ltxml` and `omdoc.sty.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. The OMDoc class is a minimally changed variant of the standard `article` class that includes the functionality provided by `omdoc.sty`. Most importantly, `omdoc.cls` sets up the LaTeXML infrastructure and thus should be used if OMDoc is to be generated from the sTeX sources. The rest of the documentation pertains to the functionality introduced by `omdoc.sty`.

2.1 Package and Class Options

The `omdoc` package and class accept the following options:

`report`	load `report.cls` instead of `article.cls`
`book`	load `book.cls` instead of `article.cls`
`showignores`	show the the contents of the `ignore` environment after all
`showmeta`	show the metadata; see `metakeys.sty`
`showmods`	show modules; see `modules.sty`
`extrefs`	allow external references; see `sref.sty`
`defindex`	index definienda; see `statements.sty`

[1] EdNote: integrate with latexml's XMRef in the Math mode.

2.2 Document Structure

document
id
omgroup
id
creators
contributors
short
loadmodules

The top-level `document` environment is augmented with an optional key/value argument that can be used to give metadata about the document. For the moment only the `id` key is used to give an identifier to the `omdoc` element resulting from the LaTeXML transformation.

The structure of the document is given by the `omgroup` environment just like in OMDoc. In the LaTeX route, the `omgroup` environment is flexibly mapped to sectioning commands, inducing the proper sectioning level from the nesting of `omgroup` environments. Correspondingly, the `omgroup` environment takes an optional key/value argument for metadata followed by a regular argument for the (section) title of the omgroup. The optional metadata argument has the keys `id` for an identifier, `creators` and `contributors` for the Dublin Core metadata [DCM03]; see [Koh15a] for details of the format. The `short` allows to give a short title for the generated section. If the title contains semantic macros, they need to be protected by `\protect`, and we need to give the `loadmodules` key it needs no value. For instance we would have

```
\begin{module}{foo}
\symdef{bar}{B^a_r}
 ...
\begin{omgroup}[id=barderiv,loadmodules]
  {Introducing $\protect\bar$ Derivations}
```

blindomgroup

STeX automatically computes the sectioning level, from the nesting of `omgroup` environments. But sometimes, we want to skip levels (e.g. to use a subsection* as an introduction for a chapter). Therefore the `omdoc` package provides a variant `blindomgroup` that does not produce markup, but increments the sectioning level and logically groups document parts that belong together, but where traditional document markup relies on convention rather than explicit markup. The `blindomgroup` environment is useful e.g. for creating frontmatter at the correct level. Example 1 shows a typical setup for the outer document structure of a book with parts and chapters. We use two levels of `blindomgroup`:

- The outer one groups the introductory parts of the book (which we assume to have a sectioning hierarchy topping at the part level). This `blindomgroup` makes sure that the introductory remarks become a "chapter" instead of a "part".
- Th inner one groups the frontmatter[1] and makes the preface of the book a section-level construct. Note that here the `display=flow` on the `omgroup` environment prevents numbering as is traditional for prefaces.

\currentsectionlevel
\CurrentSectionLevel

The `\currentsectionlevel` macro supplies the name of the current sectioning level, e.g. "chapter", or "subsection". `\CurrentSectionLevel` is the capitalized variant. They are useful to write something like "In this `\currentsectionlevel`, we will..." in an `omgroup` environment, where we do not know which sectioning level we will end up.

[1] We shied away from redefining the `frontmatter` to induce a blindomgroup, but this may be the "right" way to go in the future.

```
\begin{document}
\begin{blindomgroup}
\begin{blindomgroup}
\begin{frontmatter}
\maketitle\newpage
\begin{omgroup}[display=flow]{Preface}
... <<preface>> ...
\end{omgroup}
\clearpage\setcounter{tocdepth}{4}\tableofcontents\clearpage
\end{frontmatter}
\end{blindomgroup}
... <<introductory remarks>> ...
\end{blindomgroup}
\begin{omgroup}{Introduction}
... <<intro>> ...
\end{omgroup}
... <<more chapters>> ...
\bibliographystyle{alpha}\bibliography{kwarc}
\end{document}
```

Example 1: A typical Document Structure of a Book

2.3 Ignoring Inputs

ignore
showignores

The `ignore` environment can be used for hiding text parts from the document structure. The body of the environment is not PDF or DVI output unless the `showignores` option is given to the `omdoc` class or `package`. But in the generated OMDoc result, the body is marked up with a `ignore` element. This is useful in two situations. For

editing One may want to hide unfinished or obsolete parts of a document

narrative/content markup In sTEX we mark up narrative-structured documents. In the generated OMDoc documents we want to be able to cache content objects that are not directly visible. For instance in the `statements` package [Koh15c] we use the `\inlinedef` macro to mark up phrase-level definitions, which verbalize more formal definitions. The latter can be hidden by an ignore and referenced by the `verbalizes` key in `\inlinedef`.

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[`⟨*URL*⟩`]{`⟨*label*⟩`}`, which expands to the previously stored content. If the `\STRlabel` macro was in a different file, then we can give a URL ⟨*URL*⟩ that lets LaTeXML generate the correct reference.

\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 into some content markup format.

2.5 Colors

\blue
\red
...
\black

For convenience, the `omdoc` package defines a couple of color macros for the `color` package: For instance `\blue` abbreviates `\textcolor{blue}`, so that `\blue{`⟨*something*⟩`}` writes ⟨*something*⟩ in blue. The macros `\red` `\green`, `\cyan`, `\magenta`, `\brown`, `\yellow`, `\orange`, `\gray`, and finally `\black` are analogous.

3 Limitations

In this section we document known limitations. If you want to help alleviate them, please feel free to contact the package author. Some of them are currently discussed in the sTEX GitHub repository [sTeX].

1. when option `book` which uses `\pagestyle{headings}` is given and semantic macros are given in the `omgroup` titles, then they sometimes are not defined by the time the heading is formatted. Need to look into how the headings are made.

4 Implementation: The OMDoc Class

The functionality is spread over the `omdoc` class and package. The class provides the `document` environment and the `omdoc` element corresponds to it, whereas the package provides the concrete functionality.

`omdoc.dtx` generates four files: `omdoc.cls` (all the code between ⟨*cls⟩ and ⟨/cls⟩), `omdoc.sty` (between ⟨*package⟩ and ⟨/package⟩) and their LaTeXML bindings (between ⟨*ltxml.cls⟩ and ⟨/ltxml.cls⟩ and ⟨*ltxml.sty⟩ and ⟨/ltxml.sty⟩ respetively). We keep the corresponding code fragments together, since the documentation applies to both of them and to prevent them from getting out of sync.

The initial setup for LaTeXML (both package and class actually):

```
⟨*ltxml.sty | ltxml.cls⟩
# -*- CPERL -*-
package LaTeXML::Package::Pool;
use strict;
use LaTeXML::Package;
use LaTeXML::Util::Pathname;
use Cwd qw(abs_path);
⟨/ltxml.sty | ltxml.cls⟩
```

4.1 Class Options

To initialize the `omdoc` class, we declare and process the necessary options. For `omdoc.cls` this is quite simple. We have options `report` and `book`, which set the `\omdoc@class` macro and pass on the macro to `omdoc.sty` for further processing. The `book` option also sets the conditional to true for the frontmatter handling later.

`\omdoc@class`
`\ifclass@book`

```
⟨*cls⟩
\def\omdoc@class{article}
\DeclareOption{report}{\def\omdoc@class{report}%
\PassOptionsToPackage{\CurrentOption}{omdoc}}
\newif\ifclass@book\class@bookfalse
\DeclareOption{book}{\def\omdoc@class{book}\class@booktrue%
\PassOptionsToPackage{\CurrentOption}{omdoc}}
```

the rest of the options are only passed on to `omdoc.sty` and the class selected by the first options.

```
\DeclareOption*{\PassOptionsToClass{\CurrentOption}{\omdoc@class}
                          \PassOptionsToPackage{\CurrentOption}{omdoc}}
\ProcessOptions
⟨/cls⟩
⟨*ltxml.cls⟩
DeclareOption('report',sub {PassOptions('omdoc','sty',ToString(Digest(T_CS('\CurrentOption'))))
DeclareOption('book',sub {PassOptions('omdoc','sty',ToString(Digest(T_CS('\CurrentOption'))));
DeclareOption(undef,sub
{PassOptions('article','cls',ToString(Digest(T_CS('\CurrentOption'))));
  PassOptions('omdoc','sty',ToString(Digest(T_CS('\CurrentOption'))));});
```

```
ProcessOptions();
```
⟨/ltxml.cls⟩

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

```
⟨*cls⟩
\LoadClass{\omdoc@class}
\RequirePackage{etoolbox}
\RequirePackage{omdoc}
⟨/cls⟩
⟨*ltxml.cls⟩
LoadClass('article');
⟨/ltxml.cls⟩
```

4.2 Setting up Namespaces and Schemata for LaTeXML

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

```
⟨*ltxml.cls⟩
RegisterNamespace('omdoc'=>"http://omdoc.org/ns");
RegisterNamespace('om'=>"http://www.openmath.org/OpenMath");
RegisterNamespace('m'=>"http://www.w3.org/1998/Math/MathML");
RegisterNamespace('dc'=>"http://purl.org/dc/elements/1.1/");
RegisterNamespace('cc'=>"http://creativecommons.org/ns");
RegisterNamespace('stex'=>"http://kwarc.info/ns/sTeX");
RegisterNamespace('ltx'=>"http://dlmf.nist.gov/LaTeXML");
⟨/ltxml.cls⟩
```

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

```
⟨*ltxml.cls⟩
RelaxNGSchema('omdoc+ltxml',
        '#default'=>"http://omdoc.org/ns",
        '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",
       'stex'=>"http://kwarc.info/ns/sTeX",
       'ltx'=>"http://dlmf.nist.gov/LaTeXML");
⟨/ltxml.cls⟩
```

Then we load the `omdoc` package `omdoc.sty`, which contains the main body of functionality (e.g.sectioning/grouping). It can be loaded by classes other than `omdoc.cls` as well.

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

4.3 Beefing up the `document` environment

Now, we will define the environments we need. The top-level one is the `document` environment, which we redefined so that we can provide keyval arguments.

`document`

For the moment we do not use them on the LaTeX level, but the document identifier is picked up by LaTeXML.

```
⟨*cls⟩
\let\orig@document=\document
\srefaddidkey{document}
\renewcommand{\document}[1][]{\metasetkeys{document}{#1}\orig@document}
⟨/cls⟩
⟨*ltxml.cls⟩
sub xmlBase {
  my $baseuri = LookupValue('URLBASE');
  $baseuri =~ s/\/$//g; # No trailing slashes
  Tokenize($baseuri); }
DefEnvironment('{document} OptionalKeyVals:omdoc',
       "<omdoc:omdoc "
      .    "?&GetKeyVal(#1,'id')(xml:id='&GetKeyVal(#1,'id')')"
      .     "(?&Tokenize(&LookupValue('SOURCEBASE'))"
      .      "(xml:id='&Tokenize(&LookupValue('SOURCEBASE')).omdoc')()) "
      .    "?&Tokenize(&LookupValue('URLBASE'))"
      .    "(xml:base='&xmlBase()')()>"
     .  "#body"
      ."</omdoc:omdoc>",
  beforeDigest=> sub { AssignValue(inPreamble=>0); },
  afterDigest=> sub { $_[0]->getGullet->flush; return; },
  afterDigestBegin => sub {
    $_[1]->setProperty(id => Expand(T_CS('\thedocument@ID')));
    if (my $ops = LookupValue('@at@begin@document')) {
      Digest(Tokens(@$ops)); }
    else {
      return; } },
  beforeDigestEnd => sub {
    $_[0]->getGullet->flush;
    if (my $ops = LookupValue('@at@end@document')) {
      Digest(Tokens(@$ops)); }
    else {
      return; } },
  mode => 'text');
Tag('omdoc:omdoc', 'afterOpen:late'=>\&insertFrontMatter,
       afterOpen=>\&numberIt,afterClose=>\&locateIt);
⟨/ltxml.cls⟩%$
```

5 Implementation: OMDoc Package

5.1 Package Options

The package options come in two parts: the first we only pass on to the various other sTEX packages.

```
⟨*package⟩
\DeclareOption{showmeta}{\PassOptionsToPackage{\CurrentOption}{metakeys}}
\DeclareOption{showmods}{\PassOptionsToPackage{\CurrentOption}{modules}}
\DeclareOption{extrefs}{\PassOptionsToPackage{\CurrentOption}{sref}}
\DeclareOption{noauxreq}{\PassOptionsToPackage{\CurrentOption}{modules}}
\DeclareOption{defindex}{\PassOptionsToPackage{\CurrentOption}{statements}}
⟨/package⟩
⟨*ltxml.sty⟩
DeclareOption('showmeta',sub {PassOptions('metakeys','sty',ToString(Digest(T_CS('\CurrentOption
DeclareOption('showmods',sub {PassOptions('modules','sty',ToString(Digest(T_CS('\CurrentOption'
DeclareOption('extrefs',sub {PassOptions('sref','sty',ToString(Digest(T_CS('\CurrentOption'))))
DeclareOption('noauxreq',sub {PassOptions('modules','sty',ToString(Digest(T_CS('\CurrentOption'
DeclareOption('defindex',sub {PassOptions('statements','sty',ToString(Digest(T_CS('\CurrentOpti
⟨/ltxml.sty⟩
```

For the rest 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). The `report` and `book` options affect the sectioning behavior of the `omgroup` environment via the `\section@level` macro later.

`\section@level`

```
⟨*package⟩
\newif\ifshow@ignores\show@ignoresfalse
\DeclareOption{showignores}{\show@ignorestrue}
\newcount\section@level\section@level=2
\DeclareOption{report}{\section@level=0}
\DeclareOption{book}{\section@level=0}
\DeclareOption*{\PassOptionsToPackage{\CurrentOption}{sref}}
\ProcessOptions
⟨/package⟩
⟨*ltxml.sty⟩
DeclareOption('showignores','');
DeclareOption('report','');
DeclareOption('book','');
DeclareOption(undef, '');
ProcessOptions();
⟨/ltxml.sty⟩
```

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

```
⟨*package⟩
\RequirePackage{sref}
\RequirePackage{xspace}
\RequirePackage{comment}
```

```
\RequirePackage{etoolbox}
⟨/package⟩
⟨*ltxml.sty⟩
RequirePackage('sref');
RequirePackage('xspace');
RequirePackage('omtext');
⟨/ltxml.sty⟩
```

5.2 Document Structure

The structure of the document is given by the `omgroup` environment just like in OMDoc. The hierarchy is adjusted automatically according to the LaTeX class in effect.

`\currentsectionlevel` For the `\currentsectionlevel` and `\Currentsectionlevel` macros we use an internal macro `\current@section@level` that only contains the keyword (no markup). We initialize it with "document" as a default. In the generated OMDoc, we only generate a text element of class `omdoc_currentsectionlevel`, wich will be instantiated by CSS later.[2]

EdN:2

```
⟨*package⟩
\def\current@section@level{document}%
\newcommand\currentsectionlevel{\lowercase\expandafter{\current@section@level}\xspace}%
\newcommand\Currentsectionlevel{\expandafter\MakeUppercase\current@section@level\xspace}%
⟨/package⟩
⟨*ltxml.sty⟩
DefMacro('\currentsectionlevel','\@currentsectionlevel\xspace');
DefMacro('\Currentsectionlevel','\@Currentsectionlevel\xspace');
DefConstructor('\@currentsectionlevel',
               "<ltx:text class='omdoc-currentsectionlevel'>section</ltx:text>");
DefConstructor('\@Currentsectionlevel',
               "<ltx:text class='omdoc-Currentsectionlevel'>Section</ltx:text>");
⟨/ltxml.sty⟩
```

`blindomgroup`

```
⟨*package⟩
\newcommand\at@begin@blindomgroup[1]{}
\newenvironment{blindomgroup}
{\advance\section@level by 1\at@begin@blindomgroup\setion@level}
{\advance\section@level by -1}
⟨/package⟩
⟨*ltxml.sty⟩
DefEnvironment('{blindomgroup} OptionalKeyVals:omgroup',
               "<omdoc:omgroup layout='invisible'"
               .   "?&GetKeyVal(#1,'id')(xml:id='&GetKeyVal(#1,'id')')()"
               .   "?&GetKeyVal(#1,'type')(type='&GetKeyVal(#1,'type')')()>\n"
       . "#body\n"
```

[2] EdNote: MK: we may have to experiment with the more powerful uppercasing macro from `mfirstuc.sty` once we internationalize.

```
161       . "</omdoc:omgroup>");
162 ⟨/ltxml.sty⟩
```

`\omgroup@nonum` convenience macro: `\omgroup@nonum{`⟨*level*⟩`}{`⟨*title*⟩`}` makes an unnumbered sectioning with title ⟨*title*⟩ at level ⟨*level*⟩.

```
163 ⟨*package⟩
164 \newcommand\omgroup@nonum[2]{%
165 \ifx\hyper@anchor\@undefined\else\phantomsection\fi%
166 \addcontentsline{toc}{#1}{#2}\@nameuse{#1}*{#2}}
```

`\omgroup@num` convenience macro: `\omgroup@nonum{`⟨*level*⟩`}{`⟨*title*⟩`}` makes numbered sectioning with title ⟨*title*⟩ at level ⟨*level*⟩. We have to check the `short` key was given in the `omgroup` environment and – if it is use it. But how to do that depends on whether the `rdfmeta` package has been loaded.

```
167 \newcommand\omgroup@num[2]{%
168 \sref@label@id{\omdoc@sect@name~\@nameuse{the#1}}%
169 \ifx\omgroup@short\@empty\@nameuse{#1}{#2}%
170 \else\@ifundefined{rdfmeta@sectioning}{\@nameuse{#1}[\omgroup@short]{#2}}%
171 {\@nameuse{rdfmeta@#1@old}[\omgroup@short]{#2}}%
172 \fi}
173 ⟨/package⟩
```

`omgroup`

```
174 ⟨*package⟩
175 \def\@true{true}
176 \def\@false{false}
177 \srefaddidkey{omgroup}
178 \addmetakey{omgroup}{date}
179 \addmetakey{omgroup}{creators}
180 \addmetakey{omgroup}{contributors}
181 \addmetakey{omgroup}{srccite}
182 \addmetakey{omgroup}{type}
183 \addmetakey*{omgroup}{short}
184 \addmetakey*{omgroup}{display}
185 \addmetakey[false]{omgroup}{loadmodules}[true]
```

`\at@begin@omgroup` we define a switch for numbering lines and a hook for the beginning of groups: The `\at@begin@omgroup` macro allows customization. It is run at the beginning of the `omgroup`, i.e. after the section heading.

```
186 \newif\if@@num\@@numtrue
187 \newif\if@frontmatter\@frontmatterfalse
188 \newif\if@backmatter\@backmatterfalse
189 \newcommand\at@begin@omgroup[3][]{}
```

Then we define a helper macro that takes care of the sectioning magic. It comes with its own key/value interface for customization.

```
190 \addmetakey{omdoc@sect}{name}
191 \addmetakey[false]{omdoc@sect}{clear}[true]
192 \addmetakey{omdoc@sect}{ref}
```

```
\addmetakey[false]{omdoc@sect}{num}[true]
\newcommand\omdoc@sectioning[3][]{\metasetkeys{omdoc@sect}{#1}%
\ifx\omdoc@sect@clear\@true\cleardoublepage\fi%
\if@@num% numbering not overridden by frontmatter, etc.
\ifx\omdoc@sect@num\@true\omgroup@num{#2}{#3}\else\omgroup@nonum{#2}{#3}\fi%
\def\current@section@level{\omdoc@sect@name}%
\else\omgroup@nonum{#2}{#3}\fi}
```

and another one, if redefines the `\addtocontentsline` macro of LaTeX to import the respective macros. It takes as an argument a list of module names.[3]

EdN:3

```
\newcommand\omgroup@redefine@addtocontents[1]{\edef\@@import{#1}%
\@for\@I:=\@@import\do{\edef\@path{\csname module@\@I  @path\endcsname}%
\@ifundefined{tf@toc}\relax{\protected@write\tf@toc{}{\string\@requiremodules{\@path}{sms}}}}
\ifx\hyper@anchor\@undefined% hyperref.sty loaded?
\def\addcontentsline##1##2##3{%
\addtocontents{##1}{\protect\contentsline{##2}{\string\importmodules{#1}##3}{\thepage}}}
\else\def\addcontentsline##1##2##3{%
\addtocontents{##1}{\protect\contentsline{##2}{\string\importmodules{#1}##3}{\thepage}{\@curren
\fi}% hypreref.sty loaded?
```

now the `omgroup` environment itself. This takes care of the table of contents via the helper macro above and then selects the appropriate sectioning command from `article.cls`.

```
\newenvironment{omgroup}[2][]% keys, title
{\metasetkeys{omgroup}{#1}\sref@target%
\ifx\omgroup@display\st@flow\@@numfalse\fi
\if@frontmatter\@@numfalse\fi
```

If the `loadmodules` key is set on `\begin{omgroup}`, we redefine the `\addcontetsline` macro that determines how the sectioning commands below construct the entries for the table of contents.

```
\ifx\omgroup@loadmodules\@true%
\omgroup@redefine@addtocontents{\@ifundefined{mod@id}\imported@modules%
{\@ifundefined{module@\mod@id @path}{\imported@modules}\mod@id}}\fi%
```

now we only need to construct the right sectioning depending on the value of `\section@level`.

```
\advance\section@level by 1
\ifcase\section@level%
\or\omdoc@sectioning[name=Part,clear,num]{part}{#2}%
\or\omdoc@sectioning[name=Chapter,clear,num]{chapter}{#2}%
\or\omdoc@sectioning[name=Section,num]{section}{#2}%
\or\omdoc@sectioning[name=Subsection,num]{subsection}{#2}%
\or\omdoc@sectioning[name=Subsubsection,num]{subsubsection}{#2}%
\or\omdoc@sectioning[name=Paragraph,ref=this paragraph]{paragraph}{#2}%
\or\omdoc@sectioning[name=Subparagraph,ref=this subparagraph]{paragraph}{#2}%
\fi% \ifcase
\at@begin@omgroup[#1]\section@level{#2}}% for customization
```

[3] EdNote: MK: the extension sms is hard-coded here, but should not be. This will not work in multilingual settings.

```
{\advance\section@level by -1}
⟨/package⟩
⟨*ltxml.sty⟩
DefEnvironment('{omgroup} OptionalKeyVals:omgroup {}',
               "<omdoc:omgroup layout='sectioning'"
               .   "?&GetKeyVal(#1,'id')(xml:id='&GetKeyVal(#1,'id')')()"
               .   "?&GetKeyVal(#1,'type')(type='&GetKeyVal(#1,'type')')()>\n"
          . "<dc:title>#2</dc:title>\n"
       . "#body\n"
     . "</omdoc:omgroup>");
⟨/ltxml.sty⟩
```

5.3 Front and Backmatter

Index markup is provided by the `omtext` package [Koh15b], so in the `omdoc` package we only need to supply the corresponding `\printindex` command, if it is not already defined

`\printindex`

```
⟨*package⟩
\providecommand\printindex{\IfFileExists{\jobname.ind}{\input{\jobname.ind}}{}}
⟨/package⟩
⟨*ltxml.sty⟩
DefConstructor('\printindex','<omdoc:index/>');
Tag('omdoc:index',afterOpen=>\&numberIt,afterClose=>\&locateIt);
⟨/ltxml.sty⟩
```

`\tableofcontents` The table of contents already exists in LaTeX, so we only need to provide a LaTeXML binding for it.

```
⟨*ltxml.sty⟩
DefConstructor('\tableofcontents',
               "<omdoc:tableofcontents level='&ToString(&CounterValue('tocdepth'))'/>");
Tag('omdoc:tableofcontents',afterOpen=>\&numberIt,afterClose=>\&locateIt);
⟨/ltxml.sty⟩
```

The case of the `\bibliography` command is similar

`\bibliography`

```
⟨*ltxml.sty⟩
DefConstructor('\bibliography{}',"<omdoc:bibliography files='#1'/>");
Tag('omdoc:bibliography',afterOpen=>\&numberIt,afterClose=>\&locateIt);
⟨/ltxml.sty⟩
```

`frontmatter` `book.cls` already has a `\frontmatter` macro, so we have to redefine the front matter environment in this case.

```
⟨*cls⟩
\ifclass@book
\renewenvironment{frontmatter}
```

```
{\@frontmattertrue\cleardoublepage\@mainmatterfalse\pagenumbering{roman}}
{\@frontmatterfalse\setcounter{page}{1}\pagenumbering{arabic}}
\else
\newenvironment{frontmatter}
{\@frontmattertrue\pagenumbering{roman}}
{\@frontmatterfalse\setcounter{page}{1}\pagenumbering{arabic}}
\fi
⟨/cls⟩
⟨*ltxml.cls⟩
DefEnvironment('{frontmatter}','#body');
⟨/ltxml.cls⟩
%    \End{macrocode}
% \end{environment}
%
% \begin{environment}{backmatter}
%   |book.cls| already has a |\backmatter| macro, so we have to redefine the back
%   matter environment in this case.
%    \begin{macrocode}
⟨*cls⟩
\ifclass@book
\renewenvironment{backmatter}
{\cleardoublepage\@mainmatterfalse\@backmattertrue}
{\@backmatterfalse}
\else
\newenvironment{backmatter}{\@backmattertrue}{\@backmatterfalse}
\fi
⟨/cls⟩
⟨*ltxml.cls⟩
DefEnvironment('{backmatter}','#body');
⟨/ltxml.cls⟩
```

5.4 Ignoring Inputs

ignore

```
⟨*package⟩
\ifshow@ignores
\addmetakey{ignore}{type}
\addmetakey{ignore}{comment}
\newenvironment{ignore}[1][]
{\metasetkeys{ignore}{#1}\textless\ignore@type\textgreater\bgroup\itshape}
{\egroup\textless/\ignore@type\textgreater}
\renewenvironment{ignore}{}{}\else\excludecomment{ignore}\fi
⟨/package⟩
⟨*ltxml.sty⟩
DefKeyVal('ignore','type','Semiverbatim');
DefKeyVal('ignore','comment','Semiverbatim');
DefEnvironment('{ignore} OptionalKeyVals:ignore',
               "<omdoc:ignore  %&GetKeyVals(#1)>#body</omdoc:ignore>");
Tag('omdoc:ignore',afterOpen=>\&numberIt,afterClose=>\&locateIt);
```

```
⟨/ltxml.sty⟩
```

5.5 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.[4]

EdN:4

```
⟨*package⟩
\newcommand\STRcopy[2][]{\expandafter\ifx\csname STR@#2\endcsname\relax
\message{STR warning: reference #2 undefined!}
\else\csname STR@#2\endcsname\fi}
⟨/package⟩
⟨*ltxml.sty⟩
DefConstructor('\STRcopy[]{}',"<omdoc:ref xref='#1##2'/>");
Tag('omdoc:ref',afterOpen=>\&numberIt,afterClose=>\&locateIt);
⟨/ltxml.sty⟩
```

\STRsemantics if we have a presentation form and a semantic form, then we can use

```
⟨*package⟩
\newcommand\STRsemantics[3][]{#2\def\@test{#1}\ifx\@test\@empty\STRlabeldef{#1}{#2}\fi}
⟨/package⟩
⟨*ltxml.sty⟩
DefConstructor('\STRsemantics[]{}{}', sub {
  my($document,$label,$ignore,$object)=@_;
  $document->absorb($object);
  $document->addAttribute('xml:id'=>ToString($label)) if $label; });
⟨/ltxml.sty⟩#$
```

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

```
⟨*package⟩
\def\STRlabeldef#1{\expandafter\gdef\csname STR@#1\endcsname}
⟨/package⟩
⟨*ltxml.sty⟩
DefMacro('\STRlabeldef{}{}', "");
⟨/ltxml.sty⟩
```

[4] EdNote: MK: we need to do something about the ref!

5.6 Colors

blue, red, green, magenta

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

```
⟨*package⟩
\def\black#1{\textcolor{black}{#1}}
\def\gray#1{\textcolor{gray}{#1}}
\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}}
\def\orange#1{\textcolor{orange}{#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)(<ltx:text ?#1(style='color:#1')()>#2</ltx:text>)");
foreach my $color(qw(black gray blue red green cyan magenta brown yellow orange)) {
  DefMacro("\\".$color.'{}', sub { omdocColorMacro($color, @_); }); }#$
⟨/ltxml.sty⟩
```

5.7 LaTeX Commands we interpret differently

The reinterpretations are quite simple, we either disregard presentational markup or we re-interpret it in terms of OMDoc.

```
⟨*ltxml.sty⟩
DefConstructor('\newpage','');
⟨/ltxml.sty⟩
```

5.8 Leftovers

```
⟨*package⟩
```

```
\newcommand\baseURI[2][]{}
⟨/package⟩
⟨*ltxml.sty⟩
DefMacro('\baseURI []Semiverbatim', sub {
   my $baselocal = ToString(Digest($_[1]));
   $baselocal = abs_path($baselocal) unless $baselocal=~/^(\w+):\/\//;
   AssignValue('BASELOCAL'=>$baselocal,'global');
   AssignValue('URLBASE'=>ToString(Digest($_[2])),'global');
 });
⟨/ltxml.sty⟩%$
```

EdN:5

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

```
⟨ltxml.sty | ltxml.cls⟩1;
```

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

References

[DCM03] The DCMI Usage Board. *DCMI Metadata Terms*. DCMI Recommendation. Dublin Core Metadata Initiative, 2003. URL: http://dublincore.org/documents/dcmi-terms/.

[Koh06] Michael Kohlhase. OMDoc – *An open markup format for mathematical documents [Version 1.2]*. LNAI 4180. Springer Verlag, Aug. 2006. URL: http://omdoc.org/pubs/omdoc1.2.pdf.

[Koh15a] Michael Kohlhase. *dcm.sty: An Infrastructure for marking up Dublin Core Metadata in LaTeX documents*. Tech. rep. Comprehensive TeX Archive Network (CTAN), 2015. URL: http://www.ctan.org/tex-archive/macros/latex/contrib/stex/dcm/dcm.pdf.

[Koh15b] Michael Kohlhase. *omtext: Semantic Markup for Mathematical Text Fragments in LaTeX*. Tech. rep. Comprehensive TeX Archive Network (CTAN), 2015. URL: http://www.ctan.org/tex-archive/macros/latex/contrib/stex/omtext/omtext.pdf.

[Koh15c] Michael Kohlhase. *statements.sty: Structural Markup for Mathematical Statements*. Tech. rep. Comprehensive TeX Archive Network (CTAN), 2015. URL: http://www.ctan.org/tex-archive/macros/latex/contrib/stex/statements/statements.pdf.

[sTeX] *KWARC/sTeX*. URL: https://svn.kwarc.info/repos/stex (visited on 05/15/2015).