

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

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

November 24, 2015

Abstract

The `omdoc` package is part of the \LaTeX collection, a version of $\text{\TeX}/\text{\LaTeX}$ that allows to markup $\text{\TeX}/\text{\LaTeX}$ documents semantically without leaving the document format, essentially turning $\text{\TeX}/\text{\LaTeX}$ into a document format for mathematical knowledge management (MKM).

This package supplies an infrastructure for writing OMDoc documents in \LaTeX . This includes a simple structure sharing mechanism for \LaTeX that allows 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 \LaTeX sources, or after translation.

Contents

1	Introduction	3
2	The User Interface	3
2.1	Package and Class Options	3
2.2	Document Structure	4
2.3	Ignoring Inputs	5
2.4	Structure Sharing	5
2.5	Colors	6
3	Limitations	6
4	Implementation: The OMDoc Class	7
4.1	Class Options	7
4.2	Setting up Namespaces and Schemata for LaTeXML	8
4.3	Beefing up the <code>document</code> environment	9
5	Implementation: OMDoc Package	10
5.1	Package Options	10
5.2	Document Structure	10
5.3	Front and Backmatter	13
5.4	Ignoring Inputs	15
5.5	Structure Sharing	15
5.6	Colors	17
5.7	L ^A T _E X Commands we interpret differently	17
5.8	Leftovers	17

1 Introduction

$\mathcal{S}\mathcal{T}\mathcal{E}\mathcal{X}$ is a version of $\mathcal{T}\mathcal{E}\mathcal{X}/\mathcal{L}\mathcal{A}\mathcal{T}\mathcal{E}\mathcal{X}$ that allows to markup $\mathcal{T}\mathcal{E}\mathcal{X}/\mathcal{L}\mathcal{A}\mathcal{T}\mathcal{E}\mathcal{X}$ documents semantically without leaving the document format, essentially turning $\mathcal{T}\mathcal{E}\mathcal{X}/\mathcal{L}\mathcal{A}\mathcal{T}\mathcal{E}\mathcal{X}$ 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 $\mathcal{S}\mathcal{T}\mathcal{E}\mathcal{X}$ sources, or after translation. Currently, trans-document referencing provided by this package can only be used in the $\mathcal{S}\mathcal{T}\mathcal{E}\mathcal{X}$ 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

2 The User Interface

The `omdoc` package generates four files: `omdoc.cls`, `omdoc.sty` and their $\mathcal{L}\mathcal{A}\mathcal{T}\mathcal{E}\mathcal{X}\mathcal{M}\mathcal{L}$ 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 $\mathcal{L}\mathcal{A}\mathcal{T}\mathcal{E}\mathcal{X}\mathcal{M}\mathcal{L}$ infrastructure and thus should be used if OMDoc is to be generated from the $\mathcal{S}\mathcal{T}\mathcal{E}\mathcal{X}$ 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:

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

¹EDNOTE: integrate with `latexml`’s `XMRef` in the Math mode.

2.2 Document Structure

document	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
id	only the id key is used to give an identifier to the omdoc element resulting from the L ^A T _E X _{ML} transformation.
omgroup	The structure of the document is given by the omgroup environment just like in OMDoc. In the L ^A T _E X 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
id	keys id for an identifier, creators and contributors for the Dublin Core meta-
creators	data [DCM03]; see [Koh15a] for details of the format. The short allows to give
contributors	a short title for the generated section. If the title contains semantic macros, they
short	need to be protected by <code>\protect</code> , and we need to give the loadmodules key it
loadmodules	needs no value. For instance we would have

```

\begin{module}{foo}
\symdef{bar}{Ba_r}
...
\begin{omgroup}[id=barderv,loadmodules]
{Introducing $\protect\bar$ Derivations}

```

blindomgroup	<p> \TeX 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: </p> <ul style="list-style-type: none"> • 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 frontmatter¹ 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	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.
\CurrentSectionLevel	

¹We 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 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 \LaTeX 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` The `\STRlabel` macro takes two arguments: a label and the content and stores 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 \LaTeX ML generate the correct reference.

`\STRcopy`

`\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

2.5 Colors

For convenience, the `omdoc` package defines a couple of color macros for the `color` package: For instance `\blue` abbreviates `\textcolor{blue}`, so that `\blue{\textit{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.

²EdNOTE: document LMID und LMXRef here if we decide to keep them.

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 \LaTeX ML bindings (between `<*ltxml.cls>` and `</ltxml.cls>` and `<*ltxml.sty>` and `</ltxml.sty>` respectively). 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 \LaTeX ML (both package and class actually):

```
1 <*ltxml.sty | ltxml.cls>
2 # -*- CPERL -*-
3 package LaTeXML::Package::Pool;
4 use strict;
5 use LaTeXML::Package;
6 use LaTeXML::Util::Pathname;
7 use Cwd qw(abs_path);
8 </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`

```
9 <*cls>
10 \def\omdoc@class{article}
11 \DeclareOption{report}{\def\omdoc@class{report}%
12 \PassOptionsToPackage{\CurrentOption}{omdoc}}
13 \newif\ifclass@book\class@bookfalse
14 \DeclareOption{book}{\def\omdoc@class{book}\class@booktrue%
15 \PassOptionsToPackage{\CurrentOption}{omdoc}}
```

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

```
16 \DeclareOption*{\PassOptionsToClass{\CurrentOption}{\omdoc@class}
17 \PassOptionsToPackage{\CurrentOption}{omdoc}}
18 \ProcessOptions
19 </cls>
20 <*ltxml.cls>
21 AssignValue('omdoc@class'=>'article');
22 DeclareOption('report', sub {AssignValue('omdoc@class' => 'report'); });
23 DeclareOption('book', sub {AssignValue('omdoc@class' => 'book'); });
24 DeclareOption(undef, sub {PassOptions('article','cls',ToString(Digest(T_CS('\CurrentOption'))))
25 PassOptions('omdoc','sty',ToString(Digest(T_CS('\Curr
```

```

26 ProcessOptions();
27 </ltxml.cls>

```

We load `article.cls`, and the desired packages. For the L^AT_EX_{ML} bindings, we make sure the right packages are loaded.

```

28 <*cls>
29 \LoadClass{\omdoc@class}
30 \RequirePackage{etoolbox}
31 \RequirePackage{omdoc}
32 </cls>
33 <*ltxml.cls>
34 my $class = LookupValue('omdoc@class');
35 LoadClass($class);
36 </ltxml.cls>

```

4.2 Setting up Namespaces and Schemata for L^AT_EX_{ML}

Now, we also need to register the namespace prefixes for L^AT_EX_{ML} to use.

```

37 <*ltxml.cls>
38 RegisterNamespace('omdoc'=>"http://omdoc.org/ns");
39 RegisterNamespace('om'=>"http://www.openmath.org/OpenMath");
40 RegisterNamespace('m'=>"http://www.w3.org/1998/Math/MathML");
41 RegisterNamespace('dc'=>"http://purl.org/dc/elements/1.1/");
42 RegisterNamespace('cc'=>"http://creativecommons.org/ns");
43 RegisterNamespace('stex'=>"http://kwarc.info/ns/sTeX");
44 RegisterNamespace('ltx'=>"http://dlmf.nist.gov/LaTeXML");
45 </ltxml.cls>

```

Since we are dealing with a class, we need to set up the document type in the L^AT_EX_{ML} bindings.

```

46 <*ltxml.cls>
47 RelaxNGSchema('omdoc+ltxml',
48     '#default'=>"http://omdoc.org/ns",
49     'om'=>"http://www.openmath.org/OpenMath",
50     'm'=>"http://www.w3.org/1998/Math/MathML",
51     'dc'=>"http://purl.org/dc/elements/1.1/",
52     'cc'=>"http://creativecommons.org/ns",
53     'stex'=>"http://kwarc.info/ns/sTeX",
54     'ltx'=>"http://dlmf.nist.gov/LaTeXML");
55 </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.

```

56 <*ltxml.cls>
57 RequirePackage('omdoc');
58 </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 L^AT_EX level, but the document identifier is picked up by L^AT_EXML.

```

59 <*cls>
60 \srefaddidkey{document}
61 \newcommand\documentkeys[1]{\metasetkeys{document}{#1}}
62 \let\orig@document=\document
63 \srefaddidkey{document}
64 \renewcommand{\document}[1][\metasetkeys{document}{#1}\orig@document}
65 </cls>
66 <*txml.cls>
67 sub xmlBase {
68   my $baseuri = LookupValue('URLBASE');
69   $baseuri =~ s/\$//g; # No trailing slashes
70   Tokenize($baseuri); }
71 DefEnvironment('{document} OptionalKeyVals:omdoc',
72   "<omdoc:omdoc "
73     . "&GetKeyVal{#1,'id'}(xml:id='&GetKeyVal{#1,'id'}))"
74     . "(?&Tokenize(&LookupValue('SOURCEBASE'))"
75     . "(xml:id='&Tokenize(&LookupValue('SOURCEBASE')).omdoc'))() "
76     . "&Tokenize(&LookupValue('URLBASE'))"
77     . "(xml:base='&xmlBase()')()>"
78   . "#body"
79   . "</omdoc:omdoc>",
80   beforeDigest=> sub { AssignValue(inPreamble=>0); },
81   afterDigest=> sub { $_[0]->getGullet->flush; return; },
82   afterDigestBegin => sub {
83     $_[1]->setProperty(id => Expand(T_CS('\thedocument@ID')));
84     if (my $ops = LookupValue('@at@begin@document')) {
85       Digest(Tokens(@$ops)); }
86     else {
87       return; } },
88   beforeDigestEnd => sub {
89     $_[0]->getGullet->flush;
90     if (my $ops = LookupValue('@at@end@document')) {
91       Digest(Tokens(@$ops)); }
92     else {
93       return; } },
94   mode => 'text');
95 Tag('omdoc:omdoc', 'afterOpen:late'=>\&insertFrontMatter,
96   afterOpen=>\&numberIt,afterClose=>\&locateIt);
97 </txml.cls>%$

```

5 Implementation: OMDoc Package

5.1 Package Options

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`

```
98 <*package>
99 \newif\ifshow@ignores\show@ignorefalse
100 \DeclareOption{showignores}{\show@ignorestrue}
101 \newcount\section@level\section@level=2
102 \DeclareOption{report}{\section@level=0}
103 \DeclareOption{book}{\section@level=0}
104 \DeclareOption*{\PassOptionsToPackage{\CurrentOption}{sref}}
105 \ProcessOptions
106 </package>
107 <*ltxml.sty>
108 \DeclareOption(undef, sub {PassOptions('sref','sty',ToString(Digest(T_CS('\CurrentOption')))); }
109 \ProcessOptions();
110 </ltxml.sty>
```

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

```
111 <*package>
112 \RequirePackage{sref}
113 \RequirePackage{xspace}
114 \RequirePackage{comment}
115 \RequirePackage{etoolbox}
116 </package>
117 <*ltxml.sty>
118 \RequirePackage('sref');
119 \RequirePackage('xspace');
120 </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`, which will be instantiated by CSS later.³

³EDNOTE: MK: we may have to experiment with the more powerful uppercasing macro from `mfirstuc.sty` once we internationalize.

```

121 <*package>
122 \def\current@section@level{document}%
123 \newcommand\currentsectionlevel{\lowercase\expandafter\current@section@level\xspace}%
124 \newcommand\Currentsectionlevel{\expandafter\MakeUppercase\current@section@level\xspace}%
125 </package>
126 <*ltxml.sty>
127 DefMacro('currentsectionlevel', '@currentsectionlevel\xspace');
128 DefMacro('Currentsectionlevel', '@Currentsectionlevel\xspace');
129 DefConstructor('@currentsectionlevel',
130               "<ltx:text class='omdoc-currentsectionlevel'>section</ltx:text>");
131 DefConstructor('@Currentsectionlevel',
132               "<ltx:text class='omdoc-Currentsectionlevel'>Section</ltx:text>");
133 </ltxml.sty>

```

blindomgroup

```

134 <*package>
135 \newcommand\at@begin@blindomgroup[1]{%
136 \newenvironment{blindomgroup}
137 {\advance\section@level by 1\at@begin@blindomgroup\section@level}
138 {\advance\section@level by -1}
139 </package>
140 <*ltxml.sty>
141 DefEnvironment('{blindomgroup} OptionalKeyVals:omgroup',
142               "<omdoc:omgroup layout='invisible'"
143               . "?&GetKeyVal(#1,'id')(xml:id='&GetKeyVal(#1,'id')')()"
144               . "?&GetKeyVal(#1,'type')(type='&GetKeyVal(#1,'type')')()>\n"
145               . "#body\n"
146               . "</omdoc:omgroup>");
147 </ltxml.sty>

```

`\omgroup@nonum` convenience macro: `\omgroup@nonum{<level>}{<title>}` makes an unnumbered sectioning with title `<title>` at level `<level>`.

```

148 <*package>
149 \newcommand\omgroup@nonum[2]{%
150 \ifx\hyper@anchor\undefined\else\phantomsection\fi%
151 \addcontentsline{toc}{#1}{#2}\@nameuse{#1}*{#2}}

```

`\omgroup@num` convenience macro: `\omgroup@num{<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 `rdfmata` package has been loaded.

```

152 \newcommand\omgroup@num[2]{%
153 \sref@label{id}{\omdoc@sect@name~\@nameuse{the#1}}%
154 \ifx\omgroup@short\empty\@nameuse{#1}{#2}%
155 \else\ifundefined{rdfmata@sectioning}{\@nameuse{#1}[\omgroup@short]{#2}}%
156 {\@nameuse{rdfmata@#1@old}[\omgroup@short]{#2}}%
157 \fi}
158 </package>

```

omgroup

```

159 <*package>
160 \def\@true{true}
161 \def\@false{false}
162 \srefaddidkey{omgroup}
163 \addmetakey{omgroup}{date}
164 \addmetakey{omgroup}{creators}
165 \addmetakey{omgroup}{contributors}
166 \addmetakey{omgroup}{srccite}
167 \addmetakey{omgroup}{type}
168 \addmetakey*{omgroup}{short}
169 \addmetakey*{omgroup}{display}
170 \addmetakey[false]{omgroup}{loadmodules}[true]

```

we define a switch for numbering lines and a hook for the beginning of groups:

`\at@begin@omgroup` The `\at@begin@omgroup` macro allows customization. It is run at the beginning of the `omgroup`, i.e. after the section heading.

```

171 \newif\if@num\@numtrue
172 \newif\if@frontmatter\@frontmatterfalse
173 \newif\if@backmatter\@backmatterfalse
174 \newcommand\at@begin@omgroup[3][]{\fi}

```

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

```

175 \addmetakey{omdoc@sect}{name}
176 \addmetakey[false]{omdoc@sect}{clear}[true]
177 \addmetakey{omdoc@sect}{ref}
178 \addmetakey[false]{omdoc@sect}{num}[true]
179 \newcommand\omdoc@sectioning[3][]{\metasetkeys{omdoc@sect}{#1}%
180 \ifx\omdoc@sect@clear\@true\cleardoublepage\fi%
181 \if@num% numbering not overridden by frontmatter, etc.
182 \ifx\omdoc@sect@num\@true\omgroup@num{#2}{#3}\else\omgroup@nonum{#2}{#3}\fi%
183 \def\current@section@level{\omdoc@sect@name}%
184 \else\omgroup@nonum{#2}{#3}\fi}

```

and another one, if redefines the `\addtocontentsline` macro of L^AT_EX to import the respective macros. It takes as an argument a list of module names.⁴

```

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

```

⁴EdNOTE: MK: the extension `sms` is hard-coded here, but should not be. This will not work in multilingual settings.

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`.

```

194 \newenvironment{omgroup}[2][ ]% keys, title
195 {\metasetkeys{omgroup}{#1}\sref@target%
196 \ifx\omgroup@display\st@flow\@numfalse\fi
197 \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.

198 \ifx\omgroup@loadmodules\@true%
199 \omgroup@redefine@addtocontents{\@ifundefined{mod@id}\imported@modules%
200 {\@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.

201 \advance\section@level by 1
202 \ifcase\section@level%
203 \or\omdoc@sectioning[name=Part,clear,num]{part}{#2}%
204 \or\omdoc@sectioning[name=Chapter,clear,num]{chapter}{#2}%
205 \or\omdoc@sectioning[name=Section,num]{section}{#2}%
206 \or\omdoc@sectioning[name=Subsection,num]{subsection}{#2}%
207 \or\omdoc@sectioning[name=Subsubsection,num]{subsubsection}{#2}%
208 \or\omdoc@sectioning[name=Paragraph,ref=this paragraph]{paragraph}{#2}%
209 \or\omdoc@sectioning[name=Subparagraph,ref=this subparagraph]{paragraph}{#2}%
210 \fi% \ifcase
211 \at@begin@omgroup[#1]\section@level{#2}}% for customization
212 {\advance\section@level by -1}
213 \end{package}
214 \end{*ltxml.sty}

215 DefEnvironment('{omgroup} OptionalKeyVals:omgroup {}',
216               "<omdoc:omgroup layout='sectioning'"
217               . " ?&GetKeyVal(#1,'id')(xml:id='&GetKeyVal(#1,'id')')()"
218               . " ?&GetKeyVal(#1,'type')(type='&GetKeyVal(#1,'type')')()>\n"
219               . "<dc:title>#2</dc:title>\n"
220               . "#body\n"
221               . "</omdoc:omgroup>");
222 \end{*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
223 \providecommand\printindex{\IfFileExists{\jobname.ind}{\input{\jobname.ind}}{}}

```

```

225 </package>
226 <*lxml.sty>
227 DefConstructor('\printindex','<omdoc:index/>');
228 Tag('omdoc:index',afterOpen=>\&numberIt,afterClose=>\&locateIt);
229 </lxml.sty>

```

`\tableofcontents` The table of contents already exists in L^AT_EX, so we only need to provide a L^AT_EX_{ML} binding for it.

```

230 <*lxml.sty>
231 DefConstructor('\tableofcontents',
232               "<omdoc:tableofcontents level='&ToString(&CounterValue('tocdepth'))' />");
233 Tag('omdoc:tableofcontents',afterOpen=>\&numberIt,afterClose=>\&locateIt);
234 </lxml.sty>

```

The case of the `\bibliography` command is similar

`\bibliography`

```

235 <*lxml.sty>
236 DefConstructor('\bibliography{}','<omdoc:bibliography files='#1' />");
237 Tag('omdoc:bibliography',afterOpen=>\&numberIt,afterClose=>\&locateIt);
238 </lxml.sty>

```

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

```

239 <*cls>
240 \ifclass@book
241 \renewenvironment{frontmatter}
242 {\@frontmattertrue\cleardoublepage\@mainmatterfalse\pagenumbering{roman}}
243 {\@frontmatterfalse\setcounter{page}{1}\pagenumbering{arabic}}
244 \else
245 \newenvironment{frontmatter}
246 {\@frontmattertrue\pagenumbering{roman}}
247 {\@frontmatterfalse\setcounter{page}{1}\pagenumbering{arabic}}
248 \fi
249 </cls>
250 <*lxml.cls>
251 DefEnvironment('{frontmatter}','#body');
252 </lxml.cls>
253 % \End{macrocode}
254 % \end{environment}
255 %
256 % \begin{environment}{backmatter}
257 % |book.cls| already has a |\backmatter| macro, so we have to redefine the back
258 % matter environment in this case.
259 % \begin{macrocode}
260 <*cls>
261 \ifclass@book
262 \renewenvironment{backmatter}
263 {\cleardoublepage\@mainmatterfalse\@backmattertrue}

```

```

264 {\@backmatterfalse}
265 \else
266 \newenvironment{backmatter}{\@backmattertrue}{\@backmatterfalse}
267 \fi
268 \</cls>
269 \<*ltxml.cls>
270 DefEnvironment('{backmatter}', '#body');
271 \</ltxml.cls>

```

5.4 Ignoring Inputs

ignore

```

272 \<*package>
273 \ifshow@ignores
274 \addmetakey{ignore}{type}
275 \addmetakey{ignore}{comment}
276 \newenvironment{ignore}[1]{}
277 {\metasetkeys{ignore}{#1}\textless\ignore@type\textgreater\bgroup\itshape}
278 {\egroup\textless\ignore@type\textgreater}
279 \renewenvironment{ignore}{}{}\else\excluedecomment{ignore}\fi
280 \</package>
281 \<*ltxml.sty>
282 DefKeyVal('ignore', 'type', 'Semiverbatim');
283 DefKeyVal('ignore', 'comment', 'Semiverbatim');
284 DefEnvironment('{ignore} OptionalKeyVals:ignore',
285               "<omdoc:ignore  %&GetKeyVals(#1)>#body</omdoc:ignore>");
286 Tag('omdoc:ignore', afterOpen=>\&numberIt, afterClose=>\&locateIt);
287 \</ltxml.sty>

```

5.5 Structure Sharing

EdN:5

5

```

288 \<*package>
289 \providecommand{\lxDocumentID}[1]{}%
290 \def\XLMID#1#2{\expandafter\gdef\csname xmargin#1\endcsname{#2}\csname xmargin#1\endcsname}
291 \def\XLMRef#1{\csname xmargin#1\endcsname}
292 \</package>
293 \<*ltxml>
294 # Set the id to used for the top-level document
295 DefMacro(' \lxDocumentID{ }', '\def\thedocument@ID{#1}');
296
297 # \XLMID{id}{math} Associate the identifier id with the given math expression.
298 DefMacro(' \XLMID{ }{ }', '\@XMArg{#1}{#2}');
299
300 # \XLMRef{id} Refer to the math expression associated with id.

```

⁵EDNOTE: The following is simply copied over from the latexml package, which we eliminated, we should integrate better.

```

301 # In presentation, this is similar to using a shorthand macro.
302 # In content situations, an XMRef is generated.
303 DefMacro('\LXMRef{#1}', '\@XMRef{#1}');
304 </lxml>

```

`\STRlabel` The main macro, it is 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.

```

305 <*package>
306 \long\def\STRlabel#1#2{\STRlabeldef{#1}{#2}{#2}}
307 </package>
308 <*lxml.sty>
309 DefConstructor('\STRlabel{}{}', sub {
310   my($document,$label,$object)=@_;
311   $document->absorb($object);
312   $document->addAttribute('xml:id'=>ToString($label)) if $label; });
313 </lxml.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.⁶

```

314 <*package>
315 \newcommand\STRcopy[2] [] {\expandafter\ifx\csname STR@#2\endcsname\relax
316 \message{STR warning: reference #2 undefined!}
317 \else\csname STR@#2\endcsname\fi}
318 </package>
319 <*lxml.sty>
320 DefConstructor('\STRcopy[]{}', "<omdoc:ref xref='#1#2' />");
321 Tag('omdoc:ref', afterOpen=>\&numberIt, afterClose=>\&locateIt);
322 </lxml.sty>

```

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

```

323 <*package>
324 \newcommand\STRsemantics[3] [] {#2\def\@test{#1}\ifx\@test\@empty\STRlabeldef{#1}{#2}\fi}
325 </package>
326 <*lxml.sty>
327 DefConstructor('\STRsemantics[]{}{}', sub {
328   my($document,$label,$ignore,$object)=@_;
329   $document->absorb($object);
330   $document->addAttribute('xml:id'=>ToString($label)) if $label; });
331 </lxml.sty>##$

```

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

```

332 <*package>
333 \def\STRlabeldef#1{\expandafter\gdef\csname STR@#1\endcsname}
334 </package>
335 <*lxml.sty>
336 DefMacro('\STRlabeldef{}{}', "");
337 </lxml.sty>

```

⁶EDNOTE: 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`

```
338 <*package>
339 \def\black#1{\textcolor{black}{#1}}
340 \def\gray#1{\textcolor{gray}{#1}}
341 \def\blue#1{\textcolor{blue}{#1}}
342 \def\red#1{\textcolor{red}{#1}}
343 \def\green#1{\textcolor{green}{#1}}
344 \def\cyan#1{\textcolor{cyan}{#1}}
345 \def\magenta#1{\textcolor{magenta}{#1}}
346 \def\brown#1{\textcolor{brown}{#1}}
347 \def\yellow#1{\textcolor{yellow}{#1}}
348 \def\orange#1{\textcolor{orange}{#1}}
349 </package>
```

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

```
350 <*ltxml.sty>
351 sub omdocColorMacro {
352   my ($color, @args) = @_;
353   my $tok_color = TokenizeInternal($color);
354   (T_BEGIN, T_CS('\@omdoc@color'), T_BEGIN, $tok_color->unlist,
355    T_END, T_CS('\@omdoc@color@content'), T_OTHER(''), $tok_color->unlist, T_OTHER('')),
356   T_BEGIN, $args[1]->unlist, T_END, T_END); }
357 DefMacro('\@omdoc@color{', sub { MergeFont(color=>$_[1]->toString); return; });#$
358 </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 ?

```
359 <*ltxml.sty>
360 DefConstructor('\@omdoc@color@content[]{}',
361   "?#isMath(#2)(<ltx:text ?#1(style='color:#1')()>#2</ltx:text>)" );
362 foreach my $color(qw(black gray blue red green cyan magenta brown yellow orange)) {
363   DefMacro("\\".$color.'{}', sub { omdocColorMacro($color, @_); }); }#$
364 </ltxml.sty>
```

5.7 L^AT_EX Commands we interpret differently

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

```
365 <*ltxml.sty>
366 DefConstructor('\newpage', '');
367 </ltxml.sty>
```

5.8 Leftovers

```
368 <*package>
```

```

369 \newcommand\baseUrl[2] [] {}
370 \end{package}
371 \end{*ltxml.sty}
372 DefMacro('\baseUrl []Semiverbatim', sub {
373   my $baselocal = ToString(Digest($_[1]));
374   $baselocal = abs_path($baselocal) unless $baselocal =~ /^(w+):\/\//;
375   AssignValue('BASELOCAL'=>$baselocal,'global');
376   AssignValue('URLBASE'=>ToString(Digest($_[2])), 'global');
377 });
378 \end{ltxml.sty}%$

```

EdN:7 ⁷ and finally, we need to terminate the file with a success mark for perl.

```

379 \end{ltxml.sty | ltxml.cls}1;

```

⁷EDNOTE: 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 L^AT_EX documents*. Tech. rep. Comprehensive T_EX 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 L^AT_EX*. Tech. rep. Comprehensive T_EX 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 T_EX 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).