l

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

Michael Kohlhase
FAU Erlangen-Nürnberg
`http://kwarc.info/kohlhase`

November 2, 2020

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 environments 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 two files: `omdoc.cls`, and `omdoc.sty`. The OMDoc class is a minimally changed variant of the standard `article` class that includes the functionality provided by `omdoc.sty`. The rest of the documentation pertains to the functionality introduced by `omdoc.sty`.

2.1 Package and Class Options

The `omdoc` class accept the following options:

`class=`⟨*name*⟩	load ⟨*name*⟩`.cls` instead of `article.cls`
`topsect=`⟨*sect*⟩	The top-level sectioning level; the default for ⟨*sect*⟩ is `section`
`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`
`mh`	MathHub support; see [Koh20b]
`minimal`	for testing; do not load any sTeX packages

The `omdoc` package accepts the same except the first two.

2.2 Document Structure

`document` The top-level `document` environment can be given key/value information by the

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

\documentkeys
id

\documentkeys macro in the preamble[1]. This 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.

omgroup

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

id
creators
contributors
short
loadmodules

keys id for an identifier, creators and contributors for the Dublin Core metadata [DCM03]; see [Koh20a] 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=sec.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[2] 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.

\skipomgroup

The \skipomgroup "skips an omgroup", i.e. it just steps the respective sectioning counter. This macro is useful, when we want to keep two documents in sync structurally, so that section numbers match up: Any section that is left out in one becomes a \skipomgroup.

\currentsectionlevel

The \currentsectionlevel macro supplies the name of the current sectioning

[1] We cannot patch the document environment to accept an optional argument, since other packages we load already do; pity.

[2] 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

`\CurrentSectionLevel` 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.

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 [Koh20d] 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`.

`\prematurestop` `\afterprematurestop` For prematurely stopping the formatting of a document, sTEX provides the `\prematurestop` macro. It can be used everywhere in a document and ignores all input after that – backing out of the omgroup environment as needed. After that – and before the implicit `\end{document}` it calls the internal `\afterprematurestop`, which can be customized to do additional cleanup or e.g.

print the bibliography.

\prematurestop is useful when one has a driver file, e.g. for a course taught multiple years and wants to generate course notes up to the current point in the lecture. Instead of commenting out the remaining parts, one can just move the \prematurestop macro. This is especially useful, if we need the rest of the file for processing, e.g. to generate a theory graph of the whole course with the already-covered parts marked up as an overview over the progress; see import_graph.py from the lmhtools utilities [LMH].

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]

EdN:2

2.5 Global Variables

Text fragments and modules can be made more re-usable by the use of global variables. For instance, the admin section of a course can be made course-independent (and therefore re-usable) by using variables (actually token registers) courseAcronym and courseTitle instead of the text itself. The variables can then be set in the STeX preamble of the course notes file. \setSGvar{⟨*vname*⟩}{⟨*text*⟩} to set the global variable ⟨*vname*⟩ to ⟨*text*⟩ and \useSGvar{⟨*vname*⟩} to reference it.

\setSGvar
\useSGvar

\ifSGvar

With \ifSGvar we can test for the contents of a global variable: the macro call \ifSGvar{⟨*vname*⟩}{⟨*val*⟩}{⟨*ctext*⟩} tests the content of the global variable ⟨*vname*⟩, only if (after expansion) it is equal to ⟨*val*⟩, the conditional text ⟨*ctext*⟩ is formatted.

2.6 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.

[2] EdNote: document LMID und LMXREf here if we decide to keep them.

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.

4.1 Class Options

To initialize the `omdoc` class, we declare and process the necessary options using the `kvoptions` package for key/value options handling. For `omdoc.cls` this is quite simple. We have options `report` and `book`, which set the `\omdoc@cls@class` macro and pass on the macro to `omdoc.sty` for further processing.

`\omdoc@cls@class`

```
⟨*cls⟩
\RequirePackage{etoolbox}
\RequirePackage{kvoptions}
\SetupKeyvalOptions{family=omdoc@cls,prefix=omdoc@cls@}
\DeclareStringOption[article]{class}
\AddToKeyvalOption*{class}{\PassOptionsToPackage{class=\omdoc@cls@class}{omdoc}}
```

the following options are deprecated.

```
\DeclareVoidOption{report}{\def\omdoc@cls@class{report}%
\ClassWarning{omdoc}{the option 'report' is deprecated, use 'class=report', instead}}
\DeclareVoidOption{book}{\def\omdoc@cls@class{book}%
\ClassWarning{omdoc}{the option 'part' is deprecated, use 'class=book', instead}}
\DeclareVoidOption{bookpart}{\def\omdoc@cls@class{book}%
\PassOptionsToPackage{topsect=chapter}{omdoc}%
\ClassWarning{omdoc}{the option 'bookpart' is deprecated, use 'class=book,topsect=chapter', ins
\DeclareBoolOption{minimal}
```

the rest of the options are only passed on to `omdoc.sty` and the class selected by the first options. We need to load the `etoolbox` package early for `\@xappto`.

```
\def\@omdoc@cls@docopt{}
\DeclareDefaultOption{%
\ifx\@omdoc@cls@docopt\@empty%
\xdef\@omdoc@cls@docopt{\CurrentOption}%
\else\xappto\@omdoc@cls@docopt{,\CurrentOption}%
\fi}%
\PassOptionsToPackage{\CurrentOption}{omdoc}
\PassOptionsToPackage{\CurrentOption}{stex}
\ProcessKeyvalOptions{omdoc@cls}
```

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

```
\LoadClass[\@omdoc@cls@docopt]{\omdoc@cls@class}
```

4.2 Beefing up the `document` environment

Now, – unless the option `minimal` is defined – we include the `stex` package

```
\ifomdoc@cls@minimal\else%
\RequirePackage{stex}
```

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

`document`

EdN:3

For the moment we do not use them on the LaTeX level, but the document identifier is picked up by LaTeXML.[3]

```
\srefaddidkey{document}
\newcommand\documentkeys[1]{\metasetkeys{document}{#1}}
\let\orig@document=\document
\renewcommand{\document}[1][]{\metasetkeys{document}{#1}\orig@document}
```

Finally, we end the test for the `minimal` option.

```
\fi% \ifomdoc@cls@minimal
⟨/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).

```
⟨*package⟩
\RequirePackage{kvoptions}
\SetupKeyvalOptions{family=omdoc@sty,prefix=omdoc@sty@}
\DeclareBoolOption{mh}
\DeclareStringOption[article]{class}
\DeclareBoolOption{showignores}
\DeclareStringOption[section]{topsect}
\newcount\section@level
\DeclareDefaultOption{\PassOptionsToPackage{\CurrentOption}{sref}}
\ProcessKeyvalOptions{omdoc@sty}
```

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

```
\ifomdoc@sty@mh\RequirePackage{omdoc-mh}\fi
\RequirePackage{sref}
\RequirePackage{xspace}
\RequirePackage{comment}
\RequirePackage{pathsuris}
\RequirePackage[base]{babel}
```

We set up triggers for the other languages, currently only German.

```
\AfterBabelLanguage{ngerman}{\input{omdoc-ngerman.ldf}}
```

\section@level Finally, we set the \section@level macro that governs sectioning. The default is two (corresponding to the article class), then we set the defaults for the standard classes book and report and then we take care of the levels passed in via the topsect option.

```
\section@level=2
\ifdefstring{\omdoc@sty@class}{book}{\section@level=0}{}
\ifdefstring{\omdoc@sty@class}{report}{\section@level=0}{}
\ifdefstring{\omdoc@sty@topsect}{part}{\section@level=0}{}
\ifdefstring{\omdoc@sty@topsect}{chapter}{\section@level=1}{}
```

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.[4]

EdN:4

```
\def\current@section@level{document}%
\newcommand\currentsectionlevel{\lowercase\expandafter{\current@section@level}\xspace}%
\newcommand\Currentsectionlevel{\expandafter\MakeUppercase\current@section@level\xspace}%
```

\skipomgroup

```
\newcommand\skipomgroup{%
  \ifcase\section@level%
  \or\stepcounter{chapter}%
  \or\stepcounter{section}%
  \or\stepcounter{subsection}%
  \or\stepcounter{subsubsection}%
  \or\stepcounter{paragraph}%
  \or\stepcounter{subparagraph}%
  \fi}% \ifcase
```

blindomgroup

```
\newcommand\at@begin@blindomgroup[1]{}
\newenvironment{blindomgroup}
{\advance\section@level by 1\at@begin@blindomgroup\setion@level}
{\advance\section@level by -1}
```

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

[3] EdNote: faking documentkeys for now. @HANG, please implement

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

```
\newcommand\omgroup@nonum[2]{%
\ifx\hyper@anchor\@undefined\else\phantomsection\fi%
\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. In the end we call `\sref@label@id` to enable crossreferencing.

```
\newcommand\omgroup@num[2]{%
\edef\@@ID{\sref@id}
\ifx\omgroup@short\@empty% no short title
\@nameuse{#1}{#2}%
\else% we have a short title
\@ifundefined{rdfmeta@sectioning}%
  {\@nameuse{#1}[\omgroup@short]{#2}}%
  {\@nameuse{rdfmeta@#1@old}[\omgroup@short]{#2}}%
\fi%
\sref@label@id@arg{\omdoc@sect@name~\@nameuse{the#1}}\@@ID}
```

omgroup

```
\def\@true{true}
\def\@false{false}
\srefaddidkey{omgroup}
\addmetakey{omgroup}{date}
\addmetakey{omgroup}{creators}
\addmetakey{omgroup}{contributors}
\addmetakey{omgroup}{srccite}
\addmetakey{omgroup}{type}
\addmetakey*{omgroup}{short}
\addmetakey*{omgroup}{display}
\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.

```
\newif\if@mainmatter\@mainmattertrue
\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.

```
\addmetakey{omdoc@sect}{name}
\addmetakey[false]{omdoc@sect}{clear}[true]
\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@mainmatter% 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}% if@mainmatter
```

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.

```
\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}}}}
%\ifx\hyper@anchor\@undefined% hyperref.sty loaded?
%\def\addcontentsline##1##2##3{%
%\addtocontents{##1}{\protect\contentsline{##2}{\string\withusedmodules{#1}{##3}}{\thepage}}}
%\else% hyperref.sty not loaded
%\def\addcontentsline##1##2##3{%
%\addtocontents{##1}{\protect\contentsline{##2}{\string\withusedmodules{#1}{##3}}{\thepage}{\@c
%\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`. It also registeres the current level of omgroups in the `\omgroup@level` counter.

```
\newcount\omgroup@level
\newenvironment{omgroup}[2][]% keys, title
{\metasetkeys{omgroup}{#1}\sref@target%
\advance\omgroup@level by 1\relax%
```

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{module@id}\used@modules%
{\@ifundefined{module@\module@id @path}{\used@modules}\module@id}}\fi%
```

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

```
\advance\section@level by 1\relax%
\ifcase\section@level%
\or\omdoc@sectioning[name=\omdoc@part@kw,clear,num]{part}{#2}%
\or\omdoc@sectioning[name=\omdoc@chapter@kw,clear,num]{chapter}{#2}%
\or\omdoc@sectioning[name=\omdoc@section@kw,num]{section}{#2}%
\or\omdoc@sectioning[name=\omdoc@subsection@kw,num]{subsection}{#2}%
\or\omdoc@sectioning[name=\omdoc@subsubsection@kw,num]{subsubsection}{#2}%
\or\omdoc@sectioning[name=\omdoc@paragraph@kw,ref=this \omdoc@paragraph@kw]{paragraph}{#2}%
\or\omdoc@sectioning[name=\omdoc@subparagraph@kw,ref=this \omdoc@subparagraph@kw]{paragraph}{#2
\fi% \ifcase
\at@begin@omgroup[#1]\section@level{#2}}% for customization
{\advance\section@level by -1\advance\omgroup@level by -1}
```

and finally, we localize the sections

```
\newcommand\omdoc@part@kw{Part}
\newcommand\omdoc@chapter@kw{Chapter}
\newcommand\omdoc@section@kw{Section}
\newcommand\omdoc@subsection@kw{Subsection}
\newcommand\omdoc@subsubsection@kw{Subsubsection}
\newcommand\omdoc@paragraph@kw{paragraph}
\newcommand\omdoc@subparagraph@kw{subparagraph}
```

5.3 Front and Backmatter

Index markup is provided by the `omtext` package [Koh20c], so in the `omdoc` package we only need to supply the corresponding `\printindex` command, if it is not already defined

`\printindex`

```
\providecommand\printindex{\IfFileExists{\jobname.ind}{\input{\jobname.ind}}{}}
```

some classes (e.g. `book.cls`) already have `\frontmatter`, `\mainmatter`, and `\backmatter` macros. As we want to define `frontmatter` and `backmatter` environments, we save their behavior (possibly defining it) in `orig@*matter` macros and make them undefined (so that we can define the environments).

```
\ifcsdef{frontmatter}% to redefine if necessary
  {\cslet{orig@frontmatter}{\frontmatter}\cslet{frontmatter}{\relax}}
  {\cslet{orig@frontmatter}{\clearpage\@mainmatterfalse\pagenumbering{roman}}}
\ifcsdef{backmatter}% to redefine if necessary
  {\cslet{orig@backmatter}{\backmatter}\cslet{backmatter}{\relax}}
  {\cslet{orig@backmatter}{\clearpage\@mainmatterfalse\pagenumbering{roman}}}
```

Using these, we can now define the `frontmatter` and `backmatter` environments

`frontmatter` we use the `\orig@frontmatter` macro defined above and `\mainmatter` if it exists, otherwise we define it.

```
\newenvironment{frontmatter}
{\orig@frontmatter}
{\ifcsdef{mainmatter}{\mainmatter}{\clearpage\@mainmattertrue\pagenumbering{arabic}}}
```

`backmatter` As backmatter is at the end of the document, we do nothing for `\endbackmatter`.

```
\newenvironment{backmatter}
{\orig@backmatter}
{\ifcsdef{mainmatter}{\mainmatter}{\clearpage\@mainmattertrue\pagenumbering{arabic}}}
```

finally, we make sure that page numbering is arabic and we have main matter as the default

```
\@mainmattertrue\pagenumbering{arabic}
```

5.4 Ignoring Inputs

ignore

```
\ifomdoc@sty@showignores
\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
```

\prematurestop We initialize \afterprematurestop, and provide \prematurestop@endomgroup which looks up \omgroup@level and recursively ends enough {omgroup}s.

```
\newcommand\afterprematurestop{}
\def\prematurestop@endomgroup{\ifnum\omgroup@level=0\else%
\end{omgroup}\advance\omgroup@level by -1\prematurestop@endomgroup\fi}
\providecommand\prematurestop{%
\message{Stopping sTeX processing prematurely}
\prematurestop@endomgroup\afterprematurestop
\end{document}}
```

5.5 Structure Sharing

EdN:5 [5]

```
\providecommand{\lxDocumentID}[1]{}%
\def\LXMID#1#2{\expandafter\gdef\csname xmarg#1\endcsname{#2}\csname xmarg#1\endcsname}
\def\LXMRef#1{\csname xmarg#1\endcsname}
```

\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.

```
\long\def\STRlabel#1#2{\STRlabeldef{#1}{#2}{#2}}
```

\STRcopy The \STRcopy macro is used to call the expansion of a given label. In case the EdN:6 label is not defined it will issue a warning.[6]

```
\newcommand\STRcopy[2][]{\expandafter\ifx\csname STR@#2\endcsname\relax
\message{STR warning: reference #2 undefined!}
\else\csname STR@#2\endcsname\fi}
```

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

```
\newcommand\STRsemantics[3][]{#2\def\@test{#1}\ifx\@test\@empty\STRlabeldef{#1}{#2}\fi}
```

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

```
\def\STRlabeldef#1{\expandafter\gdef\csname STR@#1\endcsname}
```

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

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

5.6 Global Variables

`\setSGvar` set a global variable

```
\newcommand\setSGvar[1]{\@namedef{sTeX@Gvar@#1}}
```

`\useSGvar` use a global variable

```
\newrobustcmd\useSGvar[1]{%
  \@ifundefined{sTeX@Gvar@#1}
  {\PackageError{omdoc}
    {The sTeX Global variable #1 is undefined}
    {set it with \protect\setSGvar}}
\@nameuse{sTeX@Gvar@#1}}
```

`\ifSGvar` execute something conditionally based on the state of the global variable.

```
\newrobustcmd\ifSGvar[3]{\def\@test{#2}%
  \@ifundefined{sTeX@Gvar@#1}
  {\PackageError{omdoc}
    {The sTeX Global variable #1 is undefined}
    {set it with \protect\setSGvar}}
  {\expandafter\ifx\csname sTeX@Gvar@#1\endcsname\@test #3\fi}}
```

5.7 Colors

`blue, red, green, magenta` We will use the following abbreviations for colors from `color.sty`

```
\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⟩
```

Change History

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.

[Koh20a] Michael Kohlhase. *dcm.sty: An Infrastructure for marking up Dublin Core Metadata in LaTeX documents*. Tech. rep. 2020. URL: https://github.com/sLaTeX/sTeX/raw/master/sty/stex/dcm/dcm.pdf.

[Koh20b] Michael Kohlhase. *MathHub Support for sTeX*. Tech. rep. 2020. URL: https://github.com/sLaTeX/sTeX/raw/master/sty/mathhub/mathhub.pdf.

[Koh20c] Michael Kohlhase. *omtext: Semantic Markup for Mathematical Text Fragments in LaTeX*. Tech. rep. 2020. URL: https://github.com/sLaTeX/sTeX/raw/master/sty/omtext/omtext.pdf.

[Koh20d] Michael Kohlhase. *statements.sty: Structural Markup for Mathematical Statements*. Tech. rep. 2020. URL: https://github.com/sLaTeX/sTeX/raw/master/sty/statements/statements.pdf.

[LMH] *LMH Scripts*. URL: https://github.com/sLaTeX/lmhtools.

[sTeX] *sTeX: A semantic Extension of TeX/LaTeX*. URL: https://github.com/sLaTeX/sTeX (visited on 05/11/2020).