# `omdoc.sty/cls`: Semantic Markup for Open Mathematical Documents in LaTeX[*]

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

July 20, 2010

**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.

[*]Version v1.0 (last revised 2010/06/25)

## Contents

## 1 Introduction

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.

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]

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

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

## 2 The User Interface

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

noindex
extrefs

omdoc.sty has the noindex package option, which allows to suppress the creation of index entries. The option can be set to activate multifile support, see [Koh] for details.

omdoc.cls accepts all options of the omdoc.sty (see Subsection 2.1) and article.cls and just passes them on to these. [4]

EdNote(4)

### 2.2 Document Structure

document

The top-level document environment is augmented with an optional key/value

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

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.

`id`

`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 keys `id` for an identifier, `creators` and `contributors` for the Dublin Core metadata [DUB03]; see [Koh10a] for details of the format. The `short` allows to give a short title for the generated section.

`id` `creators` `contributors` `short`

## 2.3 Mathematical Text

`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

`showignores`

**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 [Koh10b] 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{label}`, which expands to the previously stored content.

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

EdNote(5)

## 2.5 Phrase-Level Markup

`phrase` The `phrase` environment allows to mark up phrases with semantic information. It takes an optional `KeyVal` argument with the keys[6]

EdNote(6)

[5]EdNote: make an example
[6]EdNote: continue

`\sinlinequote` The `sinlinequote` macro allows to mark up quotes inline and attribute them. The quote itself is given as the argument, possibly preceded by the a specification of the source in a an optional argument. For instance, we would quote Hamlet with

```
\sinlinequote[Hamlet, \cite{Shak:1603:Hamlet}]{To be or not to be}
```

which would appear as "*To be or not to be*" Hamlet, (Shakespeare 1603) in the text. The style in which inline quotations appear in the text can be adapted by specializing the macros `\@sinlinequote` — for quotations without source and `\@@sinlinequote` — for quotations with source.

`\@sinlinequote`
`\@@sinlinequote`

## 2.6 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{`⟨*something*⟩`}` writes ⟨*something*⟩ in blue. The macros `\red` `\green` `\cyan` `\magenta` `\brown` `\yellow`, and finally `\black` are analogous.

`\blue`
`\red`
`\green`
`\cyan`
`\magenta`
`\brown`
`\yellow`
`\black`

## 3 Implementation: The OMDoc Class

The `omdoc` package 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.

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

```
⟨*cls⟩
\def\omdoc@class{article}
\DeclareOption{report}{\def\omdoc@class{report}\PassOptionsToPackage{\CurrentOption}{omdoc}}
\DeclareOption{book}{\def\omdoc@class{book}\PassOptionsToPackage{\CurrentOption}{omdoc}}
\DeclareOption{chapter}{\PassOptionsToPackage{\CurrentOption}{omdoc}}
\DeclareOption{part}{\PassOptionsToPackage{\CurrentOption}{omdoc}}
\DeclareOption{showignores}{\PassOptionsToPackage{\CurrentOption}{omdoc}}
\DeclareOption{extrefs}{\PassOptionsToPackage{\CurrentOption}{sref}}
\DeclareOption*{\PassOptionsToClass{\CurrentOption}{article}}
\ProcessOptions
\LoadClass{\omdoc@class}
\RequirePackage{omdoc}
⟨/cls⟩
⟨*ltxml.cls⟩
# -*- CPERL -*-
package LaTeXML::Package::Pool;
use strict;
use LaTeXML::Package;
use LaTeXML::Util::Pathname;
use Cwd qw(cwd abs_path);
DeclareOption('report',sub {PassOptions('omdoc','sty',ToString(Digest(T_CS('\CurrentOption'))))
DeclareOption('book',sub {PassOptions('omdoc','sty',ToString(Digest(T_CS('\CurrentOption'))));
DeclareOption('chapter',sub {PassOptions('omdoc','sty',ToString(Digest(T_CS('\CurrentOption')))
DeclareOption('part',sub {PassOptions('omdoc','sty',ToString(Digest(T_CS('\CurrentOption'))));
DeclareOption('showignores',sub {PassOptions('omdoc','sty',ToString(Digest(T_CS('\CurrentOption
DeclareOption('extrefs',sub {PassOptions('sref','sty',ToString(Digest(T_CS('\CurrentOption'))))
DeclareOption(undef,sub {PassOptions('article','cls',ToString(Digest(T_CS('\CurrentOption'))));
ProcessOptions();
LoadClass('article');
RequirePackage('sref');
⟨/ltxml.cls⟩
```

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",
        'ltx'=>"http://dlmf.nist.gov/LaTeXML",
        'stex'=>"http://kwarc.info/ns/sTeX");
⟨/ltxml.cls⟩
```

EdNote(7) Then we load the omdoc package, which we define separately in the next section so that it can be loaded separately[7]

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

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
\renewcommand{\document}[1][]{\orig@document}
⟨/cls⟩
⟨*ltxml.cls⟩
sub xmlBase {
  my $baseuri = LookupValue('baseuri');
  my $baselocal = LookupValue('baselocal');
  my $cdir = abs_path(cwd());
  $cdir =~ s/^$baselocal//;
  my ($d,$f,$t) = pathname_split(LookupValue('SOURCEFILE'));
  $t = '' if LookupValue('cooluri');
  Tokenize($baseuri.$cdir.'/'.$f.$t); }
DefEnvironment('{document} OptionalKeyVals:omdoc',
       "<omdoc:omdoc "
      .    "?&KeyVal(#1,'id')(xml:id='&KeyVal(#1,'id')')"
      .     "(?&Tokenize(&LookupValue('SOURCEBASE'))"
      .      "(xml:id='&Tokenize(&LookupValue('SOURCEBASE')).omdoc')()) "
      .    "?&Tokenize(&LookupValue('baseuri'))"
      .    "(xml:base='&xmlBase()')() "
      .    "?#locator(stex:srcref='#locator')()>"
```

[7] EdNote: reword

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

## 4 Implementation: OMDoc Package

### 4.1 Package Options

The initial setup for LaTeXML:

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

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).[8]

EdNote(8)

```
⟨*package⟩
\newif\if@chapter\@chapterfalse
\newif\if@part\@partfalse
\newcount\section@level\section@level=3
\newif\ifshow@ignores\show@ignoresfalse
\def\omdoc@class{article}
\DeclareOption{report}{\def\omdoc@class{report}\section@level=2}
\DeclareOption{book}{\def\omdoc@class{book}\section@level=1}
\DeclareOption{chapter}{\section@level=2\@chaptertrue}
\DeclareOption{part}{\section@level=1\@chaptertrue\@parttrue}
\DeclareOption{showignores}{\show@ignorestrue}
\DeclareOption{extrefs}{\PassOptionsToPackage{\CurrentOption}{sref}}
\ProcessOptions
⟨/package⟩
⟨*ltxml.sty⟩
DeclareOption('report','');
DeclareOption('book','');
DeclareOption('chapter','');
DeclareOption('part','');
DeclareOption('showignores','');
DeclareOption('extrefs','');
⟨/ltxml.sty⟩
```

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

```
⟨*package⟩
```

[8]EdNote: need an implementation for LaTeXML

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

## 4.2 Document Structure

EdNote(9) The structure of the document is given by the omgroup environment just like in OMDoc. The hierarchy is adjusted automatically[9]

omgroup

```
⟨*package⟩
\srefaddidkey{omgroup}
\omdaddkey{omgroup}{creators}
\omdaddkey{omgroup}{contributors}
\omdaddkey{omgroup}{type}
\omdaddkey{omgroup}{short}
\omdaddkey{omgroup}{display}
\newenvironment{omgroup}[2][]% title
{\bgroup\omdsetkeys{omgroup}{#1}\sref@target
\ifx\omgroup@display\st@flow\noindent{\Large\textbf{#2}\\[.3ex]\noindent\ignorespaces}
\else
\if@part\ifnum\section@level=1\part{#2}\sref@label@id{Part \thepart}\fi\fi
\if@chapter\ifnum\section@level=2\chapter{#2}\sref@label@id{Chapter \thechapter}\fi\fi
\ifnum\section@level=3\section{#2}\sref@label@id{Section \thesection}\fi
\ifnum\section@level=4\subsection{#2}\sref@label@id{Subsection \thesubsection}\fi
\ifnum\section@level=5\subsubsection{#2}\sref@label@id{Subsubsection \thesubsubsection}\fi
\ifnum\section@level=6\paragraph{#2}\sref@label@id{this paragraph}\fi
\ifnum\section@level=7\subparagraph{#2}\sref@label@id{this subparagraph}\fi
\advance\section@level by 1
\fi}{\egroup}
⟨/package⟩
⟨*ltxml.sty⟩
DefEnvironment('{omgroup} OptionalKeyVals:omgroup {}',
               "<omdoc:omgroup layout='sectioning'"
               .   "?&KeyVal(#1,'id')(xml:id='&KeyVal(#1,'id')')()"
               .   "?&KeyVal(#1,'type')(type='&KeyVal(#1,'type')')()>\n"
           . "<dc:title ?#locator(stex:srcref='#locator')()>#2</dc:title>\n"
       . "#body\n"
     . "</omdoc:omgroup>");
⟨/ltxml.sty⟩
```

the itemize, description, and enumerate environments originally introduced in the omtext package do double duty in OMDoc, outside a CMP they are

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

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

```
⟨*ltxml.sty⟩
DefParameterType('IfBeginFollows', sub {
   my ($gullet) = @_;
   $gullet->skipSpaces;
                   my $next = $gullet->readToken;
                   $gullet->unread($next);
                   $next = ToString($next);
                   #Hm, falling back to regexp handling, the $gullet->ifNext approach didn't wo
                   return 1 unless ($next=~/^\\begin/);
                   return;
                },
 reversion=>'', optional=>1);#$
Let('\group@item@maybe@unwrap','\relax');
DefMacro('\group@item[] IfBeginFollows', sub {
  my($gullet,$tag,$needswrapper)=@_;
  ( T_CS('\group@item@maybe@unwrap'),
    ($needswrapper ? (Invocation(T_CS('\group@item@wrap'),$tag)->unlist) : ()) ); });
DefConstructor('\group@item@wrap {}',
       "<omdoc:omtext>"
       . "?#1(<dc:title>#1</dc:title>)()"
               . "<omdoc:CMP><omdoc:p>",
       beforeDigest=>sub {
 Let('\group@item@maybe@unwrap','\group@item@unwrap');
 #$_[0]->bgroup;
 useCMPItemizations();
 return; },
       properties=>sub{ RefStepItemCounter(); });
DefConstructor('\group@item@unwrap',
               "",
               beforeDigest=>sub {
                 # $_[0]->egroup;#$
                 Let('\group@item@maybe@unwrap','\relax'); },
               beforeConstruct=>sub {
                 $_[0]->maybeCloseElement('omdoc:p');
                 $_[0]->maybeCloseElement('omdoc:CMP');
                 $_[0]->maybeCloseElement('omdoc:omtext');
               });
Let('group@item@maybe@unwrap','\relax');
Let('\itemize@item'=>'\group@item');
Let('\enumerate@item'=>'\group@item');
Let('\description@item'=>'\group@item');
DefEnvironment('{itemize}',
       "<omdoc:omgroup xml:id='#id' layout='itemize'>"
              .  "#body"
              ."</omdoc:omgroup>",
       properties=>sub { beginItemize('itemize'); },
       beforeDigestEnd=>sub { Digest(T_CS('\group@item@maybe@unwrap')); });
DefEnvironment('{enumerate}',
       "<omdoc:omgroup xml:id='#id' layout='enumerate'>#body</omdoc:omgroup>",
       properties=>sub { beginItemize('enumerate'); },
       beforeDigestEnd=>sub { Digest(T_CS('\group@item@maybe@unwrap')); });
DefEnvironment('{description}',
       "<omdoc:omgroup xml:id='#id' layout='description'>"
              .  "#body"
              ."</omdoc:omgroup>",
       properties=>sub { beginItemize('description'); },
       beforeDigestEnd=>sub { Digest(T_CS('\group@item@maybe@unwrap')); });
⟨/ltxml.sty⟩
```

ignore

```
⟨*package⟩
\ifshow@ignores
\omdaddkey{ignore}{type}
\omdaddkey{ignore}{comment}
\newenvironment{ignore}[1][]
{\omdsetkeys{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  %&KeyVals(#1)>#body</omdoc:ignore>");
⟨/ltxml.sty⟩
```

## 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.

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

\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⟩
```

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

## 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⟩
```

## 8 Providing IDs for OMDoc Elements

To provide default identifiers, we tag all OMDoc elements that allow `xml:id` attributes by executing the `numberIt` procedure below.

```
⟨*ltxml.sty⟩
Tag('omdoc:ignore',afterOpen=>\&numberIt,afterClose=>\&locateIt);
Tag('omdoc:ref',afterOpen=>\&numberIt,afterClose=>\&locateIt);
⟨/ltxml.sty⟩
```

## 9 Leftovers

```
⟨*package⟩
\newcommand{\baseURI}[2][]{}
⟨/package⟩
⟨*ltxml.sty⟩
DefMacro('\baseURI []Semiverbatim', sub {
    AssignValue('baselocal'=>abs_path(ToString(Expand($_[1]))));
    AssignValue('baseuri'=>ToString(Expand($_[2])));});
```

```
DefConstructor('\url Semiverbatim',"<omdoc:link href='#1'>#1</omdoc:link>");
DefConstructor('\href Semiverbatim {}',"<omdoc:link href='#1'>#2</omdoc:link>");
⟨/ltxml.sty⟩
```

EdNote(10)

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

```
⟨ltxml.sty | ltxml.cls⟩1;
```

[10] 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

[DUB03] The DCMI Usage Board. *DCMI Metadata Terms*. DCMI Recommendation. Dublin Core Metadata Initiative, 2003. URL: `http://dublincore.org/documents/dcmi-terms/`.

[Koh] Tech. rep. Comprehensive TEX Archive Network (CTAN), URL: `http://www.ctan.org/tex-archive/macros/latex/contrib/stex/sref/sref.pdf`.

[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`.

[Koh10a] Michael Kohlhase. *`dcm.sty`: An Infrastructure for marking up Dublin Core Metadata in LATEX documents*. Self-documenting LATEX package. Comprehensive TEX Archive Network (CTAN), 2010. URL: `http://www.ctan.org/tex-archive/macros/latex/contrib/stex/dcm/dcm.pdf`.

[Koh10b] Michael Kohlhase. *`statements.sty`: Structural Markup for Mathematical Statements*. Self-documenting LATEX package. Comprehensive TEX Archive Network (CTAN), 2010. URL: `http://www.ctan.org/tex-archive/macros/latex/contrib/stex/statements/statements.pdf`.