`smglom.cls/sty`: Semantic Multilingual Glossary for Math

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

April 20, 2015

Abstract

The `smglom` package is part of the SᴛEX 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 glossary entries.

Contents

1 Introduction

2 The User Interface

2.1 Package and Class Options

`smglom.cls` accepts all options of the `omdoc.cls` and `article.cls` and just passes them on to these.

3 Implementation: The SMGloM Class

3.1 Class Options

To initialize the `smglom` class, we pass on all options to `omdoc.cls`

```
⟨*cls⟩
\DeclareOption*{\PassOptionsToClass{\CurrentOption}{omdoc}}
\ProcessOptions
⟨/cls⟩
⟨*ltxml.cls | ltxml.sty⟩
# -*- CPERL -*-
package LaTeXML::Package::Pool;
use strict;
use warnings;
use LaTeXML::Package;

DeclareOption(undef,sub {PassOptions('omdoc','cls',ToString(Digest(T_CS('\CurrentOption')))); }
ProcessOptions();
⟨/ltxml.cls | ltxml.sty⟩
```

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

```
⟨*cls⟩
\LoadClass{omdoc}
\RequirePackage{smglom}
⟨/cls⟩
⟨*sty⟩
\RequirePackage{amstext}
\RequirePackage{modules}
\RequirePackage{dcm}
\RequirePackage{statements}
\RequirePackage{sproof}
\RequirePackage{cmath}
\RequirePackage[langfiles]{smultiling}
\RequirePackage{presentation}
\RequirePackage{amsfonts}
⟨/sty⟩
⟨*ltxml.cls⟩
LoadClass('omdoc');
RequirePackage('smglom');
⟨/ltxml.cls⟩
⟨*ltxml.sty⟩
RequirePackage('amstext');
RequirePackage('modules');
RequirePackage('dcm');
RequirePackage('statements');
RequirePackage('sproof');
RequirePackage('cmath');
RequirePackage('smultiling',options => ['langfiles']);
RequirePackage('presentation');
```

```
RequirePackage('amsfonts');
⟨/ltxml.sty⟩
```

3.2 For Module Definitions

\gimport just a shortcut, we have a starred and unstarred version, the first one is conservative.

```
⟨*sty⟩
\def\gimport{\@ifstar\@gimport@star\@gimport@nostar}
\newcommand\@gimport@star[2][]{\def\@test{#1}\edef\mh@@repos{\mh@currentrepos}%
\ifx\@test\@empty\importmhmodule[conservative,repos=\mh@@repos,ext=tex,path=#2]{#2}%
\else\importmhmodule[conservative,repos=#1,ext=tex,path=#2]{#2}\fi%
\mhcurrentrepos\mh@@repos\ignorespaces}
\newcommand\@gimport@nostar[2][]{\def\@test{#1}\edef\mh@@repos{\mh@currentrepos}%
\ifx\@test\@empty\importmhmodule[repos=\mh@@repos,ext=tex,path=#2]{#2}%
\else\importmhmodule[repos=#1,ext=tex,path=#2]{#2}\fi%
\mhcurrentrepos\mh@@repos\ignorespaces}
⟨/sty⟩
⟨*ltxml.sty⟩
DefMacro('\gimport',' \@ifstar\@gimport@star\@gimport@nostar');
DefMacro('\@gimport@star[]{}','\g@import[conservative=true,ext=tex,path=#2]{#1}{#2}');
DefMacro('\@gimport@nostar[]{}','\g@import[conservative=false,ext=tex,path=#2]{#1}{#2}');
DefConstructor('\g@import OptionalKeyVals:importmhmodule {}{}',
       "<omdoc:imports "
       . "from='?&GetKeyVal(#1,'load')(&canonical_omdoc_path(&GetKeyVal(#1,'load')))()\##2'  "
       . "conservative='&GetKeyVal(#1,'conservative')'/>",
  afterDigest => \&gimportI);
```

To make this work we need a sub that sets the respective values.

```
sub gimportI {
  my ($stomach,$whatsit) = @_;
  my $keyval = $whatsit->getArg(1);
  my $repos = ToString($whatsit->getArg(2));
  my $name = $whatsit->getArg(3);
  if ($repos) {
    $keyval->setValue('repos',$repos); }
  else {
    $keyval->setValue('repos',LookupValue('current_repos')); }
  # Mystery: Why does $whatsit->setArgs($keyval,$name) raise a warning for
  #          "odd numbers" in hash assignment? Workaround for now!
  $$whatsit{args}[1] = $name; # Intention: $whatsit->setArg(2,$name);
  undef $$whatsit{args}[2]; # Intention: $whatsit->deleteArg(3);
  importMHmoduleI($stomach,$whatsit);
  return; }#$
⟨/ltxml.sty⟩
```

guse just a shortcut

```
⟨*sty⟩
\newcommand\guse[2][]{\def\@test{#1}%
```

```
\edef\mh@@repos{\mh@currentrepos}%
\ifx\@test\@empty\usemhmodule[repos=\mh@@repos,ext=tex,path=#2]{#2}%
\else\usemhmodule[repos=#1,ext=tex,path=#2]{#2}\fi
\mhcurrentrepos\mh@@repos\ignorespaces}
⟨/sty⟩
⟨*ltxml.sty⟩
DefMacro('\guse[]{}','\g@use[ext=tex,path=#2]{#1}{#2}');
DefConstructor('\g@use OptionalKeyVals:importmhmodule {} {}',
  "<omdoc:uses from='?&GetKeyVal(#1,'load')(&canonical_omdoc_path(&GetKeyVal(#1,'load')))()\##2
  afterDigest => \&gimportI);
⟨/ltxml.sty⟩
```

`gadopt` just a shortcut

```
⟨*sty⟩
\newcommand\gadopt[2][]{\def\@test{#1}%
\edef\mh@@repos{\mh@currentrepos}%
\ifx\@test\@empty\adoptmhmodule[repos=\mh@@repos,ext=tex,path=#2]{#2}%
\else\adoptmhmodule[repos=#1,ext=tex,path=#2]{#2}\fi
\mhcurrentrepos\mh@@repos\ignorespaces}
⟨/sty⟩
⟨*ltxml.sty⟩
DefMacro('\gadopt[]{}','\g@adopt[ext=tex,path=#2]{#1}{#2}');
DefConstructor('\g@adopt OptionalKeyVals:importmhmodule {} {}',
  "<omdoc:adopts from='?&GetKeyVal(#1,'load')(&canonical_omdoc_path(&GetKeyVal(#1,'load')))()\#
  afterDigest => \&gimportI);
⟨/ltxml.sty⟩
```

`*nym`

```
⟨*sty⟩
\newcommand\hypernym[3][]{\if@importing\else\par\noindent #2 is a hypernym of #3\fi}
\newcommand\hyponym[3][]{\if@importing\else\par\noindent #2 is a hyponym of  #3\fi}
\newcommand\meronym[3][]{\if@importing\else\par\noindent #2 is a meronym of #3\fi}
⟨/sty⟩
⟨*ltxml.sty⟩
DefConstructor('\hypernym [] {}{}',"");
DefConstructor('\hyponym [] {}{}',"");
DefConstructor('\meronym [] {}{}',"");
⟨/ltxml.sty⟩
```

EdN:1 `\MSC` to define the Math Subject Classification, [1]

```
⟨*sty⟩
\newcommand\MSC[1]{\if@importing\else MSC: #1\fi}
⟨/sty⟩
⟨*ltxml.sty⟩
DefConstructor('\MSC{}',"");
⟨/ltxml.sty⟩
```

[1] EdNote: MK: what to do for the LaTeXML side?

3.3 For Language Bindings

Here we adapt the `smultiling` functionality to the special situation, where the module and file names are identical by design.

`gviewsig` The `gviewsig` environment is just a layer over the `viewsig` environment with the keys suitably adapted.

```
⟨ltxml.sty⟩RawTeX('
⟨*sty | ltxml.sty⟩
\newenvironment{gviewsig}[4][]{\def\test{#1}\ifx\@test\@empty%
\begin{mhviewsig}[frompath=#3,topath=#4]{#2}{#3}{#4}\else
\begin{mhviewsig}[frompath=#3,topath=#4,#1]{#2}{#3}{#4}\fi}
{\end{mhviewsig}}
```

`gviewnl` The `gve` environment is just a layer over the `viewnl` environment with the keys suitably adapted.

```
\newenvironment{gviewnl}[5][]{\def\@test{#1}\ifx\@test\@empty%
\begin{mhviewnl}[frompath=#4,topath=#5]{#2}{#3}{#4}{#5}\else%
\begin{mhviewnl}[#1,frompath=#4,topath=#5]{#2}{#3}{#4}{#5}\fi}
{\end{mhviewnl}}
⟨/sty | ltxml.sty⟩
⟨ltxml.sty⟩');
```

3.4 Authoring States

We add a key to the module environment.

```
⟨*sty⟩
\addmetakey{module}{state}
⟨/sty⟩
⟨*ltxml.sty⟩
DefKeyVal('modnl','state','Semiverbatim');
⟨/ltxml.sty⟩
```

3.5 Shadowing of repositories

`\repos@macro` `\repos@macro` parses a GitLab repository name ⟨*group*⟩`/`⟨*name*⟩ and creates an internal macro name from that, which will be used

```
⟨*sty⟩
\def\repos@macro#1/#2;{#1@shadows@#2}
```

`\shadow` `\shadow{`⟨*orig*⟩`}{`⟨*fork*⟩`}` declares a that the private repository ⟨*fork*⟩ shadows the MathHub repository ⟨*orig*⟩. Internally, it simply defines an internal macro with the shadowing information.

```
\def\shadow#1#2{\@namedef{\repos@macro#1;}{#2}}
⟨/sty⟩
⟨*ltxml.sty⟩
DefConstructor('\shadow{}{}','');
⟨/ltxml.sty⟩
```

`\MathHubPath` `\MathHubPath{`⟨*repos*⟩`}` computes the path of the fork that shadows the MathHub repository ⟨*repos*⟩ according to the current `\shadow` specification. The computed path can be used for loading modules from the private version of ⟨*repos*⟩.

```
148 ⟨*sty⟩
149 \def\MathHubPath#1{\@ifundefined{\repos@macro#1;}{#1}{\@nameuse{\repos@macro#1;}}}
150 ⟨/sty⟩
151 ⟨*ltxml.sty⟩
152 DefConstructor('\MathHubPath{}','');
153 ⟨/ltxml.sty⟩
```

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

```
154 ⟨ltxml.sty | ltxml.cls⟩1;
```