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

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

April 19, 2014

**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 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. [1]

EdN:1

[1] EdNote: describe them

# 3 Implementation: The OMDoc Class

## 3.1 Class Options

To initialize the `omdoc` class, we declare and process the necessary options.

```
⟨*cls⟩
\DeclareOption*{\PassOptionsToClass{\CurrentOption}{omdoc}}
\ProcessOptions
⟨/cls⟩
⟨*ltxml.cls | ltxml.sty⟩
# -*- CPERL -*-
package LaTeXML::Package::Pool;
use strict;
use LaTeXML::Package;
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{amstext}
\RequirePackage{modules}
\RequirePackage{dcm}
\RequirePackage{statements}
\RequirePackage{sproof}
\RequirePackage{cmath}
\RequirePackage{presentation}
\RequirePackage{amsfonts}
\RequirePackage[english,ngerman]{babel}
\RequirePackage{smglom}
⟨/cls⟩
⟨*ltxml.cls⟩
LoadClass('omdoc');
RequirePackage('amstext');
RequirePackage('modules');
RequirePackage('dcm');
RequirePackage('statements');
RequirePackage('cmath');
RequirePackage('presentation');
RequirePackage('amsfonts');
RequirePackage('babel',options=>['english','ngerman']);
RequirePackage('smglom');
⟨/ltxml.cls⟩
```

## 3.2 Input

`ginput` iterates over the language bindings.

```
⟨ltxml.sty⟩RawTeX('
⟨*sty | ltxml.sty⟩
\newcommand\ginput[2][]{\input{#2}\@for\@I:=#1\do{\input{#2.\@I}}}
```

### 3.3 For Module Definitions

`gimport` just a shortcut

```
\newcommand\gimport[2][]{\def\@test{#1}%
\edef\mh@@repos{\mh@currentrepos}%
\ifx\@test\@empty\importmhmodule[\mh@@repos]{#2}{#2}%
\else\importmhmodule[#1]{#2}{#2}\fi}
```

`guse` just a shortcut

```
\newcommand\guse[2][]{\def\@test{#1}%
\edef\mh@@repos{\mh@currentrepos}%
\ifx\@test\@empty\usemhmodule[\mh@@repos]{#2}{#2}%
\else\usemhmodule[#1]{#2}{#2}\fi}
```

`gadopt` just a shortcut

```
\newcommand\gadopt[2][]{\def\@test{#1}%
\edef\mh@@repos{\mh@currentrepos}%
\ifx\@test\@empty\adoptmhmodule[\mh@@repos]{#2}{#2}%
\else\adoptmhmodule[#1]{#2}{#2}\fi}
```

`gview` The `gview` environment is just a layer over the `view` environment with the keys suitably adapted.

```
\newenvironment{gview}[3][]{\def\@test{#1}%
\ifx\@test\@empty%
\begin{view}[from=#2,to=#3]{#2}{#3}\else%
\begin{view}[from=#2,to=#3,#1]{#2}{#3}\fi}
{\end{view}}
```

`symbol` has a starred form for primary symbols. Both do nothing.

```
⟨*sty⟩
\def\symbol{\@ifstar\@gobble\@gobble}
⟨/sty⟩
⟨*ltxml.sty⟩
DefConstructor('\symbol OptionalMatch:* {}',
     "<omdoc:symbol ?#1(role='primary')(role='secondary') name='#2'/>");
⟨/ltxml.sty⟩
```

`*nym`

```
⟨*cls⟩
\newcommand\hypernym[3][]{#2 is a hypernym of #3}
\newcommand\hyponym[3][]{#2 is a hyponym of  #3}
\newcommand\meronym[3][]{#2 is a meronym of #3}
⟨/cls⟩
⟨*ltxml.cls⟩
```

```
DefConstructor('\hypernym [] {}{}',"");
DefConstructor('\hyponym [] {}{}',"");
DefConstructor('\meronym [] {}{}',"");
⟨/ltxml.cls⟩
```

EdN:2 `\MSC` to define the Math Subject Classification, [2]

```
⟨*cls⟩
\newcommand\MSC{\@gobble}
⟨/cls⟩
⟨*ltxml.cls⟩
DefConstructor('\MSC{}',"");
⟨/ltxml.cls⟩
```

## 3.4 For Language Bindings

`\smg@select@language` this internal macro selects one of the registered languages by its langauage code. Here we only initialize it, the actual selection code is generated by the `\registerlanguage` macro.

```
⟨ltxml.sty⟩RawTeX('
⟨*sty | ltxml.sty⟩
\newcommand\smg@select@lang{}
```

`\registerlanguage` `\registerlanguage{`⟨*lang*⟩`}{`⟨*babel*⟩`}` registers the `babel` language name ⟨*babel*⟩ with its ISO 639 languge code ⟨*lang*⟩ by extending the `\smg@select@language` macro.

```
\newcommand\registerlanguage[2]%
{\appto\smg@select@lang%
{\expandafter\ifstrequal\expandafter\thelang{#1}{\selectlanguage{#2}}{}}}
```

`gle` The `gle` environment is just a layer over the `module` environment with the keys and language suitably adapted.

```
\newenvironment{gle}[3][]{\def\@test{#1}%
\ifx\@test\@empty\begin{module}[id=#2.#3]\else\begin{module}[id=#2.#3,#1]\fi
\edef\mh@@repos{\mh@currentrepos}%
\gimport[\mh@@repos]{#2}\def\@test{#3}%
\smg@select@lang}
{\end{module}}
```

`gviewsketch` The `gviewsketch` environment is just a layer over the `viewsketch` environment with the keys suitably adapted.

```
\newenvironment{gviewsketch}[3][]{\def\@test{#1}%
\ifx\@test\@empty%
\begin{viewsketch}[from=#2,to=#3]{#2}{#3}\else%
\begin{viewsketch}[from=#2,to=#3,#1]{#2}{#3}\fi}
{\end{viewsketch}}
```

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

`gve` The `gve` environment is just a layer over the `gviewsketch` environment with the keys and language suitably adapted.

```
\def\@@en{en}\def\@@de{de}
\newenvironment{gve}[5][]{\def\@test{#1}%
\ifx\@test\@empty%
\begin{gviewsketch}[id=#2.#3]{#4}{#5}\else%
\begin{gviewsketch}[id=#2.#3,#1]{#4}{#5}\fi
\def\@test{#3}%
\smg@select@lang}
{\end{gviewsketch}}
⟨/sty | ltxml.sty⟩
⟨ltxml.sty⟩');
```

`noun`

```
⟨*cls⟩
\newcommand\noun[2]{}
⟨/cls⟩
⟨*ltxml.cls⟩
DefMacro('\noun {}{}','');
⟨/ltxml.cls⟩
```

`qualifier`

```
⟨*cls⟩
\newcommand\qualifier[3]{}
⟨/cls⟩
⟨*ltxml.cls⟩
DefMacro('\qualifier {}{}{}','');
⟨/ltxml.cls⟩
```