

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

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

April 3, 2016

## Abstract

The `smglom` package is part of the  $\text{\LaTeX}$  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 glossary entries.

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>The User Interface</b>	<b>2</b>
2.1	Package and Class Options . . . . .	2
<b>3</b>	<b>Implementation: The SMGloM Class</b>	<b>3</b>
3.1	Class Options . . . . .	3
3.2	For Module Definitions . . . . .	3
3.3	For Language Bindings . . . . .	5
3.4	Authoring States . . . . .	5
3.5	Shadowing of repositories . . . . .	5

## **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` as well as the `stex` and `smglom` packages.

```
1 <*cls>
2 \DeclareOption*{\PassOptionsToClass{\CurrentOption}{omdoc}}
3                                     \PassOptionsToPackage{\CurrentOption}{stex}
4                                     \PassOptionsToPackage{\CurrentOption}{smglom}}
5 \ProcessOptions
```

We load `omdoc.cls`, the `smglom` package that provides the SMGloM-specific functionality<sup>1</sup>, and the `stex` package to allow OMDoc compatibility.

```
6 \LoadClass{omdoc}
7 \RequirePackage{smglom}
8 \RequirePackage{stex}
9 \RequirePackage{amstext}
10 \RequirePackage{amsfonts}
11 </cls>
```

Now we do the same thing for the package; first the options, which we just pass on to the `stex` package.

```
12 <*sty>
13 \DeclareOption*{\PassOptionsToPackage{\CurrentOption}{statements}}
14                                     \PassOptionsToPackage{\CurrentOption}{structview}
15                                     \PassOptionsToPackage{\CurrentOption}{smultiling}}
16 \ProcessOptions
```

We load `omdoc.cls`, and the desired packages. For the L<sup>A</sup>T<sub>E</sub>XML bindings, we make sure the right packages are loaded.

```
17 \RequirePackage{statements}
18 \RequirePackage[langfiles]{smultiling}
19 \RequirePackage{structview}
```

### 3.2 For Module Definitions

`\gimport` Just a shortcut, we have a starred and unstarred version, the first one is conservative. For example, if we execute:

```
\gimport[smglom/numberfields]{naturalnumbers}
```

First we are redirected to `\@gimport@nostar`, we store the `smglom/numberfields` (*the repo's path*) in `\@test`, then store `\mh@currentrepos` (*current directory*) in `\mh@repos`. If no repo's path is offered, that means the module to import is

---

<sup>1</sup>EDNOTE: MK:describe that above

under the same directory, so we let `repos=\mh@repos` and pass bunch of parameters to `\importmhmodule`, which is defined in `module.sty`. If there's a repo's path, then we let `repos=\langle the repo's path \rangle`. Finally we use `\mhcurrentrepos` (defined in `module.sty`) to change the `\mh@currentrepos`.

```

20 \def\gimport{\@ifstar\@gimport@star\@gimport@nostar}%
21 \newrobustcmd\gimport@star[2] [] {%
22   \def\@test{#1}%
23   \edef\mh@@repos{\mh@currentrepos}%
24   \ifx\@test\@empty%
25     \importmhmodule[conservative, repos=\mh@@repos, ext=tex, path=#2]{#2}%
26   \else%
27     \importmhmodule[conservative, repos=#1, ext=tex, path=#2]{#2}%
28   \fi%
29   \mhcurrentrepos{\mh@@repos}%
30   \ignorespaces%
31 }%
32 \newrobustcmd\gimport@nostar[2] [] {%
33   \def\@test{#1}%
34   \edef\mh@@repos{\mh@currentrepos}%
35   \ifx\@test\@empty%
36     \importmhmodule[repos=\mh@@repos, ext=tex, path=#2]{#2}%
37   \else%
38     \importmhmodule[repos=#1, ext=tex, path=#2]{#2}%
39   \fi%
40   \mhcurrentrepos{\mh@@repos}%
41   \ignorespaces%
42 }%

```

`guse` just a shortcut

```

43 \newrobustcmd\guse[2] [] {%
44   \def\@test{#1}%
45   \edef\mh@@repos{\mh@currentrepos}%
46   \ifx\@test\@empty%
47     \usemhmodule[repos=\mh@@repos, ext=tex, path=#2]{#2}%
48   \else%
49     \usemhmodule[repos=#1, ext=tex, path=#2]{#2}%
50   \fi%
51   \mhcurrentrepos{\mh@@repos}%
52   \ignorespaces%
53 }%

```

`*nym`

```

54 \newrobustcmd\hypernym[3] [] {\if@importing\else\par\noindent #2 is a hypernym of #3\fi}%
55 \newrobustcmd\hyponym[3] [] {\if@importing\else\par\noindent #2 is a hyponym of #3\fi}%
56 \newrobustcmd\meronym[3] [] {\if@importing\else\par\noindent #2 is a meronym of #3\fi}%

```

EdN:2

`\MSC` to define the Math Subject Classification,<sup>2</sup>

```

57 \newrobustcmd\MSC[1] {\if@importing\else MSC: #1\fi}%

```

---

<sup>2</sup>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 `mhviewsig` environment with the keys suitably adapted.

```
58 \newenvironment{gviewsig}[4] []{%
59   \def\test{#1}%
60   \ifx\@test\@empty%
61     \begin{mhviewsig}[frompath=#3,topath=#4]{#2}{#3}{#4}%
62   \else%
63     \begin{mhviewsig}[frompath=#3,topath=#4,#1]{#2}{#3}{#4}%
64   \fi%
65 }{%
66   \end{mhviewsig}%
67 }%
```

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

```
68 \newenvironment{gviewnl}[5] []{%
69   \def\@test{#1}\ifx\@test\@empty%
70     \begin{mhviewnl}[frompath=#4,topath=#5]{#2}{#3}{#4}{#5}%
71   \else%
72     \begin{mhviewnl}[frompath=#4,topath=#5,#1]{#2}{#3}{#4}{#5}%
73   \fi%
74 }{%
75   \end{mhviewnl}%
76 }%
```

EdN:3

```
\gincludeview 3
77 \newcommand\gincludeview[2] []{}%
```

### 3.4 Authoring States

We add a key to the module environment.

```
78 \addmetakey{module}{state}%
```

### 3.5 Shadowing of repositories

`\repos@macro` `\repos@macro` parses a GitLab repository name  $\langle group \rangle / \langle name \rangle$  and creates an internal macro name from that, which will be used

```
79 \def\repos@macro#1/#2;{#1@shadows@#2}%
```

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

```
80 \def\shadow#1#2{\@namedef{\repos@macro#1;}{#2}}%
```

---

<sup>3</sup>EDNOTE: This is fake for now, needs to be implemented and documented

`\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⟩`.

```

81 \def\MathHubPath#1{\@ifundefined{\repos@macro#1;}{#1}{\@nameuse{\repos@macro#1;}}%
82 \</sty>

```