smglom.cls/sty: Semantic Multilingual Glossary for Math

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

May 21, 2018

Abstract

The smglom 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.

2.2 Convenience Macros for SMGloM Modules

EdN:1 [1]

2.3 Terminological Relations

EdN:2 [2]

2.4 Namespaces and Alignments

EdN:3 [3] In SMGloM, we often want to align the content of glossary modules to formalizations, e.g. to take advantage of type declarations there. The `\symalign` macro takes two regular arguments: the first is the name symbol declared in the current module (e.g. by a `\symi`), and the second the URI name of a symbol in an external theory in the form ⟨*theory*⟩`?`⟨*name*⟩.

`\symalign`

As full MMT URIs are of the form ⟨*URI*⟩`?`⟨*theory*⟩`?`⟨*name*⟩, we need a way to specify the ⟨*URI*⟩. We adopt the system of **namespaces** of in MMT: the macro declares a namespace URI. If the optional argument is given, then this is a namespace abbreviation declaration, which can be used later, for instance in `\symalign` that takes an optional first argument: the namespace of the external theory.

`\namespace`

The situation below is typical. We first declare the namespace abbreviation `sets` and then use the `\modalign` macro to specify that the external theory `sets:?ESet` is the default alignment target, i.e. any symbol that in the local `emptyset` module is aligned by default to the symbol with the same name in the external `sets:?ESet` theory.

`\modalign`

```
\begin{modsig}[creators=miko]{emptyset}
  \gimport{set}
  \namespace[sets]{http://mathhub.info/MitM/smglom/sets}
  \modalign[sets]{ESet}

 \symdef{eset}{\emptyset}
```

[1] EdNote: document them
[2] EdNote: document them
[3] EdNote: MK: maybe this should go into some other module; it seems awfully foundational.

```
  \symi{non-empty}
  \symalign{non-empty}{ESet?non_empty}
 \end{modsig}
```

The default alignment breaks down for the symbol `non-empty`, so we specify an alignment to the symbol `Eset?non_empty` via `\symalign`.

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.

```
⟨*cls⟩
\DeclareOption*{\PassOptionsToClass{\CurrentOption}{omdoc}
                          \PassOptionsToPackage{\CurrentOption}{stex}
                          \PassOptionsToPackage{\CurrentOption}{smglom}}
\ProcessOptions
```

We load `omdoc.cls`, the `smglom` package that provides the SMGloM-specific functionality[4], and the `stex` package to allow OMDoc compatibility.

EdN:4

```
\LoadClass{omdoc}
\RequirePackage{smglom}
\RequirePackage{stex}
\RequirePackage{amstext}
\RequirePackage{amsfonts}
⟨/cls⟩
```

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

```
⟨*sty⟩
\DeclareOption*{\PassOptionsToPackage{\CurrentOption}{statements}
                          \PassOptionsToPackage{\CurrentOption}{structview}
                          \PassOptionsToPackage{\CurrentOption}{smultiling}}
\ProcessOptions
```

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

```
\RequirePackage{statements}
\RequirePackage[langfiles]{smultiling}
\RequirePackage{structview}
```

3.2 Convenience Macros for SMGloM Modules

`\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

[4] 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=⟨*the repo's path*⟩. Finally we use `\mhcurrentrepos`(defined in `module.sty`) to change the `\mh@currentrepos`.

```
\def\gimport{\@ifstar\@gimport@star\@gimport@nostar}%
\newrobustcmd\@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}%
  \ignorespacesandpars%
}%
\newrobustcmd\@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}%
  \ignorespacesandpars%
}%
```

`guse` just a shortcut

```
\newrobustcmd\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}%
  \ignorespacesandpars%
}%
```

`gstructure` we essentially copy over the definition of `mhstructure`, but adapt it to the SMGloM situation.

```
\newenvironment{gstructure}[3][]{\def\@test{#1}%
  \xdef\mh@@@repos{\mh@currentrepos}%
  \ifx\@test\@empty%
    \gdef\@@doit{\importmhmodule[repos=\mh@@@repos,path=#3,ext=tex]{#3}}%
  \else%
    \gdef\@@doit{\importmhmodule[repos=#1,path=#3,ext=tex]{#3}}%
  \fi%
```

```
  \ifmod@show\par\noindent structure import "#2" from module #3 \@@doit\fi%
  \ignorespacesandpars}
{\aftergroup\@@doit\ifmod@show end import\fi%
  \ignorespacesandparsafterend}
```

3.3 Terminological Relations

*nym

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

EdN:5 \MSC to define the Math Subject Classification, [5]

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

3.4 Namespaces and Alignments

\namespace

```
\newcommand\namespace[2][]{\ignorespaces}
```

\modalign

```
\newcommand\modalign[2][]{\ignorespaces}
```

\symalign

```
\newcommand\symalign[3][]{\ignorespaces}
```

3.5 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.

```
\newenvironment{gviewsig}[4][]{% keys, id, from, to
  \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%
  \ignorespacesandpars%
}{%
  \end{mhviewsig}%
  \ignorespacesandparsafterend%
}%
```

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

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

```
\newenvironment{gviewnl}[5][]{% keys, id, lang, from, to
  \def\@test{#1}\ifx\@test\@empty%
    \begin{mhviewnl}[frompath=#4,topath=#5]{#2}{#3}{#4}{#5}%
  \else%
    \begin{mhviewnl}[frompath=#4,topath=#5,#1]{#2}{#3}{#4}{#5}%
  \fi%
\ignorespacesandpars%
}{%
  \end{mhviewnl}%
  \ignorespacesandparsafterend%
}%
```

EdN:6 \gincludeview [6]

```
\newcommand\gincludeview[2][]{\ignorespacesandpars}%
```

3.6 Authoring States, etc

We add a key to the module environment.

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

3.7 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

```
\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}}%
```

\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*⟩.

```
\def\MathHubPath#1{\@ifundefined{\repos@macro#1;}{#1}{\@nameuse{\repos@macro#1;}}}%
⟨/sty⟩
```

[6] EdNote: This is fake for now, needs to be implemented and documented