

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

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

October 4, 2015

Abstract

The `smglom` package is part of the \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

1	Introduction	2
2	The User Interface	2
2.1	Package and Class Options	2
3	Implementation: The SMGloM Class	3
3.1	Class Options	3
3.2	For Module Definitions	4
3.3	For Language Bindings	6
3.4	Authoring States	7
3.5	Shadowing of repositories	7

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`

```
1 <*cls>
2 \DeclareOption*{\PassOptionsToClass{\CurrentOption}{omdoc}}
3 \ProcessOptions
4 </cls>
5 <*ltxml.cls | ltxml.sty>
6 # -*- CPERL -*-
7 package LaTeXML::Package::Pool;
8 use strict;
9 use warnings;
10 use LaTeXML::Package;
11
12 DeclareOption(undef,sub {PassOptions('omdoc','cls',ToString(Digest(T_CS('\CurrentOption')))); }
13 ProcessOptions();
14 </ltxml.cls | ltxml.sty>
```

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

```
15 <*cls>
16 \LoadClass{omdoc}
17 \RequirePackage{smglom}
18 </cls>
19 <*sty>
20 \RequirePackage{amstext}
21 \RequirePackage{modules}
22 \RequirePackage{dcm}
23 \RequirePackage{statements}
24 \RequirePackage{sproof}
25 \RequirePackage{cmath}
26 \RequirePackage[langfiles]{smultiling}
27 \RequirePackage{presentation}
28 \RequirePackage{amsfonts}
29 </sty>
30 <*ltxml.cls>
31 LoadClass('omdoc');
32 RequirePackage('smglom');
33 </ltxml.cls>
34 <*ltxml.sty>
35 RequirePackage('amstext');
36 RequirePackage('modules');
37 RequirePackage('dcm');
38 RequirePackage('statements');
39 RequirePackage('sproof');
40 RequirePackage('cmath');
41 RequirePackage('smultiling',options => ['langfiles']);
42 RequirePackage('presentation');
```

```

43 RequirePackage('amsfonts');
44 </ltxml.sty>

```

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 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`.

```

45 <*sty>
46 \def\gimport{\@ifstar\@gimport@star\@gimport@nostar}%
47 \newrobustcmd\@gimport@star[2][]{%
48   \def\@test{#1}%
49   \edef\mh@repos{\mh@currentrepos}%
50   \ifx\@test\@empty%
51     \importmhmodule[conservative,repos=\mh@repos,ext=tex,path=#2]{#2}%
52   \else%
53     \importmhmodule[conservative,repos=#1,ext=tex,path=#2]{#2}%
54   \fi%
55   \mhcurrentrepos{\mh@repos}%
56   \ignorespaces%
57 }%
58 \newrobustcmd\@gimport@nostar[2][]{%
59   \def\@test{#1}%
60   \edef\mh@repos{\mh@currentrepos}%
61   \ifx\@test\@empty%
62     \importmhmodule[repos=\mh@repos,ext=tex,path=#2]{#2}%
63   \else%
64     \importmhmodule[repos=#1,ext=tex,path=#2]{#2}%
65   \fi%
66   \mhcurrentrepos{\mh@repos}%
67   \ignorespaces%
68 }%
69 </sty>
70 <*ltxml.sty>
71 DefMacro(' \gimport', ' \@ifstar\@gimport@star\@gimport@nostar');
72 DefMacro(' \@gimport@star[]{}', ' \gimport[conservative=true,ext=tex,path=#2]{#1}{#2}');
73 DefMacro(' \@gimport@nostar[]{}', ' \gimport[conservative=false,ext=tex,path=#2]{#1}{#2}');
74 DefConstructor(' \gimport OptionalKeyVals:importmhmodule {}{}',

```

```

75      "<omdoc:imports "
76      . "from='?&GetKeyVal(#1,'load')(&canonical_omdoc_path(&GetKeyVal(#1,'load')))(###2' "
77      . "conservative='&GetKeyVal(#1,'conservative')'>",
78      afterDigest => \&gimportI);

```

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

```

79 sub gimportI {
80   my ($stomach,$whatsit) = @_;
81   my $keyval = $whatsit->getArg(1);
82   my $repos = ToString($whatsit->getArg(2));
83   my $name = $whatsit->getArg(3);
84   if ($repos) {
85     $keyval->setValue('repos',$repos); }
86   else {
87     $keyval->setValue('repos',LookupValue('current_repos')); }
88   # Mystery: Why does $whatsit->setArgs($keyval,$name) raise a warning for
89   #           "odd numbers" in hash assignment? Workaround for now!
90   $$whatsit{args}[1] = $name; # Intention: $whatsit->setArg(2,$name);
91   undef $$whatsit{args}[2]; # Intention: $whatsit->deleteArg(3);
92   importMHmoduleI($stomach,$whatsit);
93   return; }##$
94 </ltxml.sty>

```

guse just a shortcut

```

95 <*sty>
96 \newrobustcmd\guse[2] [] {%
97   \def\@test{#1}%
98   \edef\mh@@repos{\mh@currentrepos}%
99   \ifx\@test\@empty%
100     \usemhmodule[repos=\mh@@repos,ext=tex,path=#2]{#2}%
101   \else%
102     \usemhmodule[repos=#1,ext=tex,path=#2]{#2}%
103   \fi%
104   \mhcurrentrepos{\mh@@repos}%
105   \ignorespaces%
106 }%
107 </sty>
108 <*ltxml.sty>
109 DefMacro('\guse[]{}', '\g@use[ext=tex,path=#2]{#1}{#2}');
110 DefConstructor('\g@use OptionalKeyVals:importmhmodule {} {}',
111   "<omdoc:uses
112   from='?&GetKeyVal(#1,'load')(&canonical_omdoc_path(&GetKeyVal(#1,'load')))(###2'>",
113   afterDigest => \&gimportI);
114 </ltxml.sty>

```

gadopt just a shortcut

```

115 <*sty>
116 \newrobustcmd\gadopt[2] [] {%
117   \def\@test{#1}%
118   \edef\mh@@repos{\mh@currentrepos}%

```

```

119 \ifx\@test\@empty%
120 \adoptmhmodule[repos=\mh@@repos,ext=tex,path=#2]{#2}%
121 \else%
122 \adoptmhmodule[repos=#1,ext=tex,path=#2]{#2}%
123 \fi%
124 \mhcurrentrepos{\mh@@repos}%
125 \ignorespaces%
126 }%
127 \</sty>
128 \<*lxml.sty>
129 DefMacro('gadopt[]{}','g@adopt[ext=tex,path=#2]{#1}{#2}');
130 DefConstructor('g@adopt OptionalKeyVals:importmhmodule {} {}',
131 " <omdoc:adopts
132 from='?&GetKeyVal(#1,'load')(&canonical_omdoc_path(&GetKeyVal(#1,'load'))())###2' />",
133 afterDigest => \&gimportI);
134 \</lxml.sty>

*nym
135 \<*sty>
136 \newrobustcmd\hypernym[3] [] {\if@importing\else\par\noindent #2 is a hypernym of #3\fi}%
137 \newrobustcmd\hyponym[3] [] {\if@importing\else\par\noindent #2 is a hyponym of #3\fi}%
138 \newrobustcmd\meronym[3] [] {\if@importing\else\par\noindent #2 is a meronym of #3\fi}%
139 \</sty>
140 \<*lxml.sty>
141 DefConstructor('hypernym [] {}{}', "");
142 DefConstructor('hyponym [] {}{}', "");
143 DefConstructor('meronym [] {}{}', "");
144 \</lxml.sty>

```

EdN:1

```

\MSC to define the Math Subject Classification, 1
145 \<*sty>
146 \newrobustcmd\MSC[1] {\if@importing\else MSC: #1\fi}%
147 \</sty>
148 \<*lxml.sty>
149 DefConstructor('MSC{}', "");
150 \</lxml.sty>

```

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.

```

151 \<lxml.sty>RawTeX(
152 \<*sty | lxml.sty>
153 \newenvironment{gviewsig}[4] [] {%

```

¹EdNOTE: MK: what to do for the LaTeXML side?

```

154 \def\test{#1}%
155 \ifx\@test\@empty%
156   \begin{mhviewsig}[frompath=#3,topath=#4]{#2}{#3}{#4}%
157   \else%
158     \begin{mhviewsig}[frompath=#3,topath=#4,#1]{#2}{#3}{#4}%
159   \fi%
160 }{%
161 \end{mhviewsig}%
162 }%

```

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

```

163 \newenvironment{gviewnl}[5][]{%
164   \def\@test{#1}\ifx\@test\@empty%
165     \begin{mhviewnl}[frompath=#4,topath=#5]{#2}{#3}{#4}{#5}%
166     \else%
167       \begin{mhviewnl}[#1,frompath=#4,topath=#5]{#2}{#3}{#4}{#5}%
168     \fi%
169 }{%
170 \end{mhviewnl}%
171 }%
172 \</sty | ltxml.sty>
173 \<ltxml.sty>');

```

3.4 Authoring States

We add a key to the module environment.

```

174 \<sty>
175 \addmetakey{module}{state}%
176 \</sty>
177 \<ltxml.sty>
178 DefKeyVal('modnl','state','Semiverbatim');
179 \</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

```

180 \<sty>
181 \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.

```

182 \def\shadow#1#2{\@namedef{\repos@macro#1;}{#2}}%
183 \</sty>
184 \<ltxml.sty>
185 DefConstructor('\shadow{}{}','');
186 \</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⟩`.

```

187 ⟨*sty⟩
188 \def\MathHubPath#1{\ifundefined{\repos@macro#1;}{#1}{\@nameuse{\repos@macro#1;}}}%
189 ⟨/sty⟩
190 ⟨*ltxml.sty⟩
191 DefConstructor('MathHubPath{','');
192 ⟨/ltxml.sty⟩

and finally, we need to terminate the file with a success mark for perl.
193 ⟨ltxml.sty | ltxml.cls⟩1;

```