

smultiling.sty: Multilinguality Support for S_TE_X

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

April 28, 2014

Abstract

The **smultiling** package is part of the S_TE_X collection, a version of T_EX/L^AT_EX that allows to markup T_EX/L^AT_EX documents semantically without leaving the document format, essentially turning T_EX/L^AT_EX into a document format for mathematical knowledge management (MKM).

The **smultiling** package adds multilinguality support for S_TE_X, the idea is that multilingual modules in S_TE_X consist of a module signature together with multiple language bindings that inherit symbols from it, which also account for cross-language coordination.

Contents

1	Introduction	2
1.1	S _T E _X Module Signatures	2
2	The User Interface	3
2.1	Multilingual Modules	3
2.2	Multilingual Views	3
3	Implementation	5
3.1	Class Options	5
3.2	Handling Languages	5
3.3	Signatures	7
3.4	Language Bindings	7

1 Introduction

We have been using \TeX as the encoding for the Semantic Multilingual Glossary of Mathematics (SMGloM; see [Gin+14]). The SMGloM data model has been taxing the representational capabilities of \TeX with respect to multilingual support and verbalization definitions; see [Koh14], which we assume as background reading for this note.

1.1 \TeX Module Signatures

(monolingual) \TeX had the intuition that the symbol definitions ($\text{\textbackslash symdef}$ and $\text{\textbackslash symvariant}$) are interspersed with the text and we generate \TeX module signatures (SMS *.sms files) from the \TeX files. The SMS duplicate “formal” information from the “narrative” \TeX files. In the SMGloM, we extend this idea by making the the SMS primary objects that contain the language-independent part of the formal structure conveyed by the \TeX documents and there may be multiple narrative “language bindings” that are translations of each other – and as we do not want to duplicate the formal parts, those are inherited from the SMS rather than written down in the language binding itself. So instead of

```
\begin{module}[id=foo]
\symdef{bar}{BAR}
\begin{definition}[for=bar]
  A \defiii{big}{array}{raster} ( $\bar{\phantom{x}}$ ) is a\ldots, it is much bigger
  than a \defiii[sar]{small}{array}{raster}.
\end{definition}
\end{module}
```

we now advocate the divided style in the listing below.

```
\usepackage[english,ngerman]{multiling}
\begin{modsig}{foo}
\symdef{bar}{BAR}
\symbol{sar}
\end{modsig}

\begin{modnl}[creators=miko,primary]{foo}{en}
\begin{definition}
  A \defiii[bar]{big}{array}{raster} ( $\bar{\phantom{x}}$ ) is a\ldots, it is much bigger
  than a \defiii[sar]{small}{array}{raster}.
\end{definition}
\end{modnl}

\begin{modnl}[creators=miko]{foo}{de}
\begin{definition}
  Ein \defiii[bar]{gro"ses}{Feld}{Raster} ( $\bar{\phantom{x}}$ ) ist ein\ldots, es
  ist viel gr"o"ser als ein \defiii[sar]{kleines}{Feld}{Raster}.
\end{definition}
\end{modnl}
```

`\end{modnl}`

We retain the old `module` environment as an intermediate stage. It is still useful for monolingual texts. Note that for files with a module, we still have to extract `*.sms` files. It is not completely clear yet, how to adapt the workflows. We clearly need a `lmh` or editor command that transfers an old-style module into a new-style signature/binding combo to prepare it for multilingual treatment.

2 The User Interface

The `smultiling` package accepts all options of the `babel.sty` and just passes them on to it. The options specify which languages can be used in the `gTeX` language bindings.

2.1 Multilingual Modules

`modsig` There the `modsig` environment works exactly like the old `module` environment, only that the `id` attribute has moved into the required argument – anonymous module signatures do not make sense.

`modnl` The `modnl` environment takes two arguments the first is the name of the module signature it provides language bindings for and the second the ISO 639 language specifier of the content language. We add the `primary` key `modnl`, which can specify the primary language binding (the one the others translate from; and which serves as the reference in case of translation conflicts).¹

`\symbol` There is another difference in the multilingual encoding: All symbols are introduced in the module signature, either by a `\symdef` or the new `\symbol` macro. `\symbol{<name>}` takes a symbol name `<name>` as an argument and reserves that name. The variant `\symbol*{<name>}` declares `<name>` to be a primary symbol; see [Koh14] for a discussion.

2.2 Multilingual Views

`viewsig` Views receive a similar treatment as modules in the `smultiling` package. A multilingual view consists of a view signature marked up with the `viewsig` environment. This takes three required arguments: a view name, the source module, and the target module. The optional first argument is for metadata (`display`, `title`, `creators`, and `contributors`) and load information (`frompath`, `fromrepos`, `topath`, and `torepos`).²

```
\begin{viewsig}[creators=miko,]{norm-metric}{metric-space}{norm}
  \vassign{base-set}{base-set}
```

¹EdNOTE: ©DG: This needs to be implemented in LaTeXML

²EdNOTE: MK: that does not work yet, what we describe here is `mhviewig`; we need to refactor further.

```

\assign{metric}{\funcdot{x,y}{\norm{x-y}}}
\end{viewsig}

```

Views have language bindings just as modules do, in our case, we have

```

\begin{gviewnl}[creators=miko]{norm-metric}{en}{norm}{metric-space}
\obligation{metric-space}{obl.norm-metric.en}
\begin{assertion}[type=obligation,id=obl.norm-metric.en]
  $\defeq{d(x,y)}{\norm{x-y}}$ is a \trefii[metric-space]{distance}{function}
\end{assertion}
\begin{sproof}[for=obl.norm-metric.en]{we prove the three conditions for a distance function:}
  ...
\end{sproof}
\end{gviewnl}

```

3 Implementation

Technically, the `smultiling` package is essentially a wrapper around the `babel` package but allows specification of languages by their ISO 639 language codes.

3.1 Class Options

To initialize the `smultiling` class, we pass on all options to `babel.cls` and record which languages are loaded by defining `\smul@⟨language⟩@loaded` macros.³

`langfiles` The `langfiles` option specifies that for a module `⟨mod⟩`, the module signature file has the name `⟨mod⟩.tex` and the language bindings of language with the ISO 639 language specifier `⟨lang⟩` have the file name `⟨mod⟩.⟨lang⟩.tex`.⁴

```

1 ⟨*sty⟩
2 \newif\if@langfiles\@langfilesfalse
3 \DeclareOption{langfiles}{\@langfilestrue}
4 \DeclareOption*{\PassOptionsToPackage{\CurrentOption}{babel}}
5 \@namedef{smul@\CurrentOption @loaded}{yes}
6 \ProcessOptions
7 ⟨/sty⟩
8 ⟨*ltxml⟩
9 # -*- CPERL -*-
10 package LaTeXML::Package::Pool;
11 use strict;
12 use LaTeXML::Package;
13 DeclareOption('langfiles',sub {AssignValue('smultiling_langfiles',1,'global')});
14 DeclareOption(undef,sub {PassOptions('babel','sty',ToString(Digest(T_CS('\CurrentOption')))); });
15 ProcessOptions();
16 ⟨/ltxml⟩

    We load babel.sty

17 ⟨*sty⟩
18 \RequirePackage{etoolbox}
19 \RequirePackage{babel}
20 \RequirePackage{modules}
21 ⟨/sty⟩
22 ⟨*ltxml⟩
23 RequirePackage('babel');
24 RequirePackage('modules');
25 ⟨/ltxml⟩

```

3.2 Handling Languages

`\smg@select@language` This macro selects one of the registered languages by its language code by setting the internal `\smg@lang` macro to the argument and then runs the actual selection

³EdNOTE: @DG: We also want to do that in \LaTeX ML

⁴EdNOTE: implement other schemes, e.g. the onefile scheme.

code in `\smg@select@lang`. This internal code register is only initialized there, the code is generated by the `\smg@register@language` macro below.

```

26 <ltxml>RawTeX('
27 <*sty | ltxml>
28 \newcommand\smg@select@lang{}
29 \newcommand\smg@select@language[1]{\def\smg@lang{#1}\smg@select@lang}

```

`\smg@register@language` `\smg@register@language{<lang>}{<babel>}` registers the `babel` language name `<babel>` with its ISO 639 language code `<lang>` by extending the `\smg@select@language` macro.

```

30 \newcommand\smg@register@language[2]%
31 {\@ifundefined{smul@#1@loaded}{\appto\smg@select@lang%
32 {\expandafter\ifstrequal\expandafter\smg@lang{#1}{\selectlanguage{#2}}{}}}}

```

Now we register a couple of languages for which we have `babel` support. Maybe we have to extend this list with others. But then we have to extend the mechanisms.

```

33 \smg@register@language{af}{afrikaans}
34 \smg@register@language{de}{ngerman}
35 \smg@register@language{fr}{french}%
36 \smg@register@language{he}{hebrew}
37 \smg@register@language{hu}{hungarian}
38 \smg@register@language{id}{indonesian}
39 \smg@register@language{ms}{malay}
40 \smg@register@language{nn}{nynorsk}
41 \smg@register@language{pt}{portuguese}
42 \smg@register@language{ru}{russian}
43 \smg@register@language{uk}{ukrainian}
44 \smg@register@language{en}{english}
45 \smg@register@language{es}{spanish}
46 \smg@register@language{sq}{albanian}
47 \smg@register@language{bg}{bulgarian}
48 \smg@register@language{ca}{catalan}
49 \smg@register@language{hr}{croatian}
50 \smg@register@language{cs}{czech}
51 \smg@register@language{da}{danish}
52 \smg@register@language{nl}{dutch}
53 \smg@register@language{eo}{esperanto}
54 \smg@register@language{et}{estonian}
55 \smg@register@language{fi}{finnish}
56 \smg@register@language{ka}{georgian}
57 \smg@register@language{el}{greek}
58 \smg@register@language{is}{icelandic}
59 \smg@register@language{it}{italian}
60 \smg@register@language{la}{latin}
61 \smg@register@language{no}{norsk}
62 \smg@register@language{pl}{polish}
63 \smg@register@language{sr}{serbian}
64 \smg@register@language{sk}{slovak}
65 \smg@register@language{sl}{slovenian}

```

```

66 \smg@register@language{sv}{swedish}
67 \smg@register@language{th}{thai}
68 \smg@register@language{tr}{turkish}
69 \smg@register@language{vi}{vietnamese}
70 \smg@register@language{cy}{welsh}
71 \smg@register@language{hi}{hindi}

```

3.3 Signatures

modsig The `modsig` environment is just a layer over the `module` environment. We also redefine macros that may occur in module signatures so that they do not create markup.

```

72 \newenvironment{modsig}[2][]{%
73 \def\@test{#1}\ifx\@test\@empty\begin{module}[id=#2]\else\begin{module}[id=#2,#1]\fi}
74 {\end{module}}

```

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

```

75 \newenvironment{viewsig}[4][]{\def\@test{#1}\ifx\@test\@empty%
76 \begin{view}[id=#2,ext=tex]{#3}{#4}\else\begin{view}[id=#2,#1,ext=tex]{#3}{#4}\fi}
77 {\end{view}}

```

mhviewsig The `mhviewsig` environment is just a layer over the `mhview` environment with the keys suitably adapted.

```

78 \newenvironment{mhviewsig}[4][]{\def\@test{#1}\ifx\@test\@empty%
79 \begin{mhview}[id=#2,ext=tex]{#3}{#4}\else\begin{mhview}[id=#2,#1,ext=tex]{#3}{#4}\fi}
80 {\end{mhview}}
81 \<sty | ltxml>
82 \<ltxml>');

```

symbol has a starred form for primary symbols.

```

83 \<sty>
84 \def\symbol{\@ifstar{\@symbol}{\@symbol@star}}
85 \def\@symbol#1{\if@importing\else Symbol: \textsf{#1}\fi}
86 \def\@symbol@star#1{\if@importing\else Primary Symbol: \textsf{#1}\fi}
87 \</sty>
88 \<ltxml.sty>
89 DefConstructor('\symbol OptionalMatch:* {}',
90               "<omdoc:symbol ?#1(role='primary')(role='secondary') name='#2'/>");
91 \</ltxml.sty>

```

3.4 Language Bindings

modnl:

```

92 \<sty>
93 \addmetakey{modnl}{load}
94 \addmetakey*{modnl}{title}

```

```

95 \addmetakey*{modnl}{creators}
96 \addmetakey*{modnl}{contributors}
97 \addmetakey{primary}{contributors}[yes]
98 \</sty>
99 \<*ltxml>
100 DefKeyVal('modnl','title','Semiverbatim');
101 DefKeyVal('modnl','load','Semiverbatim');
102 DefKeyVal('modnl','creators','Semiverbatim');
103 DefKeyVal('modnl','contributors','Semiverbatim');
104 DefKeyVal('modnl','primary','Semiverbatim');
105 \</ltxml>

```

modnl The modnl environment is just a layer over the module environment and the \importmodule macro with the keys and language suitably adapted.

```

106 \<*sty>
107 \newenvironment{modnl}[3][\metasetkeys{modnl}{#1}%
108 \smg@select@language{#3}%
109 \def\@test{#1}\ifx\@test\@empty\begin{module}[id=#2.#3]\else\begin{module}[id=#2.#3,#1]\fi%
110 \if@langfiles\importmodule[load=#2,ext=tex]{#2}\else
111 \ifx\modnl@load\@empty\importmodule{#2}\else\importmodule[ext=tex,load=\modnl@load]{#2}\fi%
112 \fi}
113 {\end{module}}
114 \</sty>
115 \<*ltxml>
116 DefEnvironment('modnl OptionalKeyVals:modnl {}{}',
117 " <omdoc:theory "
118 . 'xml:id="#2.#3">'
119 . " ?&defined(&GetKeyVal{#1,'creators'})(<dc:creator>&GetKeyVal{#1,'creators'})</dc:cr
120 . " ?&defined(&GetKeyVal{#1,'title'})(<dc:title>&GetKeyVal{#1,'title'})</dc:title>())"
121 . " ?&defined(&GetKeyVal{#1,'contributors'})(<dc:contributor>&GetKeyVal{#1,'contribut
122 . "#body"
123 . "</omdoc:theory>",
124 afterDigestBegin=>sub {
125 my ($stomach, $whatsit) = @_;
126 my $keyval = $whatsit->getArg(1);
127 my $signature = ToString($whatsit->getArg(2));
128 if ($keyval) {
129 # If we're not given load, AND the langfiles option is in effect,
130 # default to #2
131 if ((! $keyval->getValue('load')) && (LookupValue('smultiling_langfiles'))){
132 $keyval->setValue('load',$signature); }
133 # Always load a TeX file
134 $keyval->setValue('ext','tex'); }
135 importmoduleI($stomach,$whatsit)});
136 \</ltxml>)%$

```

mhmodnl:*

```

137 \<*sty>
138 \addmetakey{mhmodnl}{repos}
139 \addmetakey{mhmodnl}{path}

```



```

140 \addmetakey*{mhmodnl}{title}
141 \addmetakey*{mhmodnl}{creators}
142 \addmetakey*{mhmodnl}{contributors}
143 \addmetakey{primary}{contributors}[yes]
144 \</sty>
145 \<*txml>
146 DefKeyVal('mhmodnl','title','Semiverbatim');
147 DefKeyVal('mhmodnl','repos','Semiverbatim');
148 DefKeyVal('mhmodnl','path','Semiverbatim');
149 DefKeyVal('mhmodnl','creators','Semiverbatim');
150 DefKeyVal('mhmodnl','contributors','Semiverbatim');
151 DefKeyVal('mhmodnl','primary','Semiverbatim');
152 \</txml>

```

mhmodnl The mhmodnl environment is just a layer over the module environment and the `\importmhmodule` macro with the keys and language suitably adapted.

```

153 \<*sty>
154 \newenvironment{mhmodnl}[3][\metasetkeys{mhmodnl}{#1}%
155 \smg@select@language{#3}%
156 \def\@test{#1}\ifx\@test\@empty\begin{module}[id=#2.#3]\else\begin{module}[id=#2.#3,#1]\fi%
157 \edef\@repos{\ifx\mhmodnl@repos\@empty\mh@currentrepos\else\mhmodnl@repos}
158 \if@langfiles\importmhmodule[repos=\@repos,load=#2,ext=tex]{#2}\else
159 \ifx\mhmodnl@load\@empty\importmodule{#2}\else\importmodule[ext=tex,load=\mhmodnl@load]{#2}\fi%
160 \fi}
161 \<end{module}>
162 \</sty>
163 \<*txml>
164 DefEnvironment('mhmodnl OptionalKeyVals:mhmodnl {}{}',
165     "<omdoc:theory "
166     . 'xml:id="#2.#3">'
167     . "&defined(&GetKeyVal(#1,'creators'))(<dc:creator>&GetKeyVal(#1,'creators')</dc:cr
168     . "&defined(&GetKeyVal(#1,'title'))(<dc:title>&GetKeyVal(#1,'title')</dc:title>())"
169     . "&defined(&GetKeyVal(#1,'contributors'))(<dc:contributor>&GetKeyVal(#1,'contribut
170     . "#body"
171     . "</omdoc:theory>",
172 afterDigestBegin=>sub {
173     my ($stomach, $whatsit) = @_;
174     my $keyval = $whatsit->getArg(1);
175     my $signature = ToString($whatsit->getArg(2));
176     my $repos = ToString(GetKeyVal($keyval,'torepos'));
177     my $current_repos = LookupValue('current_repos');
178     if (!$repos) { $repos = $current_repos; }
179     my $defpaths = LookupValue('defpath');
180     my $load_path = ($$defpaths{MathHub}).$repos.'/source/'. $signature;
181
182     if ($keyval) {
183         # If we're not given load, AND the langfiles option is in effect,
184         # default to #2
185         if ((! $keyval->getValue('path')) && (LookupValue('smultiling_langfiles')) {
186             $keyval->setValue('load',$load_path); }

```

```

187      # Always load a TeX file
188      $keyval->setValue('ext','tex'); }
189      importmoduleI($stomach,$whatsit));
190 </ltxml)%$

```

EdN:5

viewn1 The **viewn1** environment is just a layer over the **viewsketch** environment with the keys and language suitably adapted.⁵

```

191 <ltxml>RawTeX('
192 <*sty | ltxml>
193 \newenvironment{viewn1}[5][\def@test{#1}\ifx@test@empty%
194 \begin{viewsketch}[id=#2.#3,ext=tex]{#4}{#5}\else%
195 \begin{viewsketch}[id=#2.#3,#1,ext=tex]{#4}{#5}\fi%
196 \smg@select@language{#3}}
197 {\end{viewsketch}}

```

EdN:6

mhviewn1 The **mhviewn1** environment is just a layer over the **mhviewsketch** environment with the keys and language suitably adapted.⁶

```

198 \newenvironment{mhviewn1}[5][\def@test{#1}\ifx@test@empty%
199 \begin{mhviewsketch}[id=#2.#3,ext=tex]{#4}{#5}\else%
200 \begin{mhviewsketch}[id=#2.#3,#1,ext=tex]{#4}{#5}\fi%
201 \smg@select@language{#3}}
202 {\end{mhviewsketch}}
203 </sty | ltxml>
204 <ltxml>');

```

⁵EdNOTE: MK: we have to do something about the if@langfiles situation here. But this is non-trivial, since we do not know the current path, to which we could append `.<lang>!`

⁶EdNOTE: MK: we have to do something about the if@langfiles situation here. But this is non-trivial, since we do not know the current path, to which we could append `.<lang>!`

References

- [Gin+14] Deyan Ginev et al. “The SMGLoM Project and System”. 2014. URL: <http://kwarc.info/kohlhase/submit/cicm14-smglom-system.pdf>.
- [Koh14] Michael Kohlhase. “A Data Model and Encoding for a Semantic, Multilingual Glossary of Mathematics”. In: *Intelligent Computer Mathematics*. Conferences on Intelligent Computer Mathematics. (Coimbra, Portugal, July 7–11, 2014). Ed. by Stephan Watt et al. Lecture Notes in Computer Science. accepted. Springer, 2014. URL: <http://kwarc.info/kohlhase/submit/cicm14-smglom-datamd1.pdf>. Forthcoming.

␣ltxml␣1;