

smultiling.sty: Multilinguality Support for \TeX

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

November 17, 2015

Abstract

The `smultiling` package is part of the \TeX 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).

The `smultiling` package adds multilinguality support for \TeX , the idea is that multilingual modules in \TeX 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	\TeX Module Signatures	2
2	The User Interface	2
2.1	Multilingual Modules	3
2.2	Multilingual Definitions and Crossreferencing Terms	3
2.3	Multilingual Views	4
3	Implementation	6
3.1	Class Options	6
3.2	Signatures	6
3.3	Language Bindings	7
3.4	Multilingual Statements and Terms	9
3.5	Miscellaneous	9
3.6	Finale	9

1 Introduction

We have been using \TeX as the encoding for the Semantic Multilingual Glossary of Mathematics (SMGloM; see [IanJucKoh:sps14]). The SMGloM data model has been taxing the representational capabilities of \TeX with respect to multilingual support and verbalization definitions; see [Kohlhase:dmesmgm14], 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 $\ast.\text{sms}$ 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 the traditional monolingual markup in Figure 1, we we now advocate the divided style in Figure 2.

```
\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}
```

Example 1: A module with definition in monolingual \TeX

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 $\ast.\text{sms}$ 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

`langfiles` The `smultiling` package accepts the `langfiles` option that specifies – for a module $\langle mod \rangle$ that the module signature file has the name $\langle mod \rangle.\text{tex}$ and the language bindings of language with the ISO 639 language specifier $\langle lang \rangle$ have the file name $\langle mod \rangle.\langle lang \rangle.\text{tex}$.¹

¹EDNOTE: implement other schemes, e.g. the onefile scheme.

```

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

\begin{modnl}[creators=miko,primary]{foo}{en}
\begin{definition}
  A \defiii[bar]{big}{array}{raster} ( $\bar{}$ ) 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} ( $\bar{}$ ) ist ein\ldots, es
  ist viel gr"o"ser als ein \defiii[sar]{kleines}{Feld}{Raster}.
\end{definition}
\end{modnl}

```

Example 2: Multilingual \LaTeX for Figure 1.

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).²

\sympi There is another difference in the multilingual encoding: All symbols are introduced in the module signature, either by a `\symdef` or the new `\sympi` macro. `\sympi{<name>}` takes a symbol name `<name>` as an argument and reserves that name. The variant `\sympi*{<name>}` declares `<name>` to be a primary symbol; see [Kohlhase:dmesmgm14] for a discussion. \LaTeX provides variants `\symii` and `\symiii` – and their starred versions – for multi-part names.

2.2 Multilingual Definitions and Crossreferencing Terms

We do not need a new infrastructure for defining mathematical concepts, only the realization that symbols are language-independent. So we can use symbols for the coordination of corresponding verbalizations. As the example in Figure 2 already shows, we can just specify the symbol name in the optional argument of the `\defi` macro to establish that the language bindings provide different verbalizations of the same symbol.

²EdNOTE: ©DG: This needs to be implemented in \LaTeX ML

For multilingual term references the situation is more complex: For single-word verbalizations we could use `\atrefi` for language bindings. Say we have introduced a symbol `foo` in English by `\defi{foo}` and in German by `\defi[foo]{Foo}`. Then we can indeed reference it via `\trefi{foo}` and `\atrefi{Foo}{foo}`. But on the one hand this blurs the distinction between translation and “linguistic variants” and on the other hand does not scale to multi-word compounds as `bar` in Figure 2, which we would have to reference as `\atrefiii{gro"ses Feld Raster}{bar}`. To avoid this, the `smultiling` package provides the new macros `\mtrefi`, `\mtrefii`, and `\mtrefiii` for multilingual references. Using this, we can reference `bar` as `\mtrefiii[?bar]{gro"ses}{Feld}{Raster}`, where we use the (up to three) mandatory arguments to segment the lexical constituents.

`\mtref*`

The first argument is syntactically optional to keep the parallelity to `*def*` `*tref*` it specifies the symbol via its name $\langle name \rangle$ and module name $\langle mod \rangle$ in a MMT URI $\langle mod \rangle ? \langle name \rangle$. Note that MMT URIs can be relative:

1. `foo?bar` denotes the symbol `bar` from module `foo`
2. `foo` the module `foo` (the symbol name is induced from the remaining arguments of `\mtref*`)
3. `?bar` specifies symbol `bar` from the current module

Note that the number suffix `i`/`ii`/`iii` indicates the number of words in the actual language binding, not in the symbol name as in `\atref*`.

2.3 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).³

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

³EDNOTE: MK: that does not work yet, what we describe here is `mhviewig`; we need to refactor further.

```

...
\end{sproof}
\end{gviewnl}

```

3 Implementation

The general preamble for L^AT_EXML

```

1 <*lxml>
2 # -*- CPERL -*-
3 package LaTeXML::Package::Pool;
4 use strict;
5 use LaTeXML::Package;
6 </lxml>

```

3.1 Class Options

```

7 <*sty>
8 \newif\if@smultiling@mh@\@smultiling@mh@false
9 \DeclareOption{mh}{\@smultiling@mh@true}
10 \newif\if@langfiles@\@langfiles@false
11 \DeclareOption{langfiles}{\@langfiles@true}
12 \DeclareOption*{\PassOptionsToPackage{\CurrentOption}{modules}}
13 \ProcessOptions
14 </sty>
15 <*lxml>
16 DeclareOption('mh', sub {RequirePackage('smultiling-mh')});
17 DeclareOption('langfiles', sub {AssignValue('smultiling_langfiles', 1, 'global')});
18 DeclareOption(undef, sub {PassOptions('modules', 'sty', ToString(Digest(T_CS('\CurrentOption')))});
19 ProcessOptions();
20 </lxml>

```

We load the packages referenced here.

```

21 <*sty>
22 \if@smultiling@mh\RequirePackage{smultiling-mh}\fi
23 \RequirePackage{etoolbox}
24 \RequirePackage{structview}
25 </sty>
26 <*lxml>
27 RequirePackage('structview');
28 </lxml>

```

3.2 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.

```

29 <lxml>RawTeX('
30 <*sty | lxml>
31 \newenvironment{modsig}[2][{}]{%
32 \def\@test{#1}\ifx\@test\@empty\begin{module}[id=#2]\else\begin{module}[id=#2,#1]\fi}
33 {\end{module}}

```

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

```

34 \newenvironment{viewsig}[4][]{\def\@test{#1}\ifx\@test\@empty%
35 \begin{view}[id=#2,ext=tex]{#3}{#4}\else\begin{view}[id=#2,#1,ext=tex]{#3}{#4}\fi}
36 {\end{view}}
37 \</sty | ltxml>
38 \ltxml');

```

`\@sym*` has a starred form for primary symbols.

```

39 \<sty>
40 \newcommand\symi{\@ifstar\@symi@star\@symi}
41 \newcommand\@symi[1]{\if@importing\else Symbol: \textsf{#1}\fi}
42 \newcommand\@symi@star[1]{\if@importing\else Primary Symbol: \textsf{#1}\fi}
43 \newcommand\symii{\@ifstar\@symii@star\@symii}
44 \newcommand\@symii[2]{\if@importing\else Symbol: \textsf{#1-#2}\fi}
45 \newcommand\@symii@star[2]{\if@importing\else Primary Symbol: \textsf{#1-#2}\fi}
46 \newcommand\symiii{\@ifstar\@symiii@star\@symiii}
47 \newcommand\@symiii[3]{\if@importing\else Symbol: \textsf{#1-#2-#3}\fi}
48 \newcommand\@symiii@star[3]{\if@importing\else Primary Symbol: \textsf{#1-#2-#3}\fi}
49 \</sty>
50 \<ltxml>
51 DefConstructor('\symi OptionalMatch:* {}',
52     "<omdoc:symbol ?#1(role='primary')(role='secondary') name='#2'/>");
53 DefConstructor('\symii OptionalMatch:* {} {}',
54     "<omdoc:symbol ?#1(role='primary')(role='secondary') name='#2-#3'/>");
55 DefConstructor('\symiii OptionalMatch:* {} {} {}',
56     "<omdoc:symbol ?#1(role='primary')(role='secondary') name='#2-#3-#4'/>");
57 \</ltxml>

```

3.3 Language Bindings

`modnl:*`

```

58 \<sty>
59 \addmetakey{modnl}{load}
60 \addmetakey*{modnl}{title}
61 \addmetakey*{modnl}{creators}
62 \addmetakey*{modnl}{contributors}
63 \addmetakey{modnl}{srccite}
64 \addmetakey{modnl}{primary}[yes]
65 \</sty>
66 \<ltxml>
67 DefKeyVal('modnl','title','Semiverbatim');
68 DefKeyVal('modnl','load','Semiverbatim');
69 DefKeyVal('modnl','creators','Semiverbatim');
70 DefKeyVal('modnl','contributors','Semiverbatim');
71 DefKeyVal('modnl','primary','Semiverbatim');
72 \</ltxml>

```

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

```

73 \<sty>

```

```

74 \newenvironment{modnl}[3] [] {\metasetkeys{modnl}{#1}%
75 \def\@test{#1}\ifx\@test\@empty\begin{module}[id=#2.#3]\else\begin{module}[id=#2.#3,#1]\fi%
76 \if@langfiles\importmodule[load=#2,ext=tex]{#2}\else
77 \ifx\modnl@load\@empty\importmodule{#2}\else\importmodule[ext=tex,load=\modnl@load]{#2}\fi%
78 \fi}
79 {\end{module}}
80 \</sty>
81 \<*txml>
82 DefEnvironment(' {modnl} OptionalKeyVals:modnl {}{ }',
83     "?#excluded()(<omdoc:theory xml:id='#2.#3'>"
84     . " ?&defined(&GetKeyVal(#1,'creators'))(<dc:creator>&GetKeyVal(#1,'creators')</dc:cr
85     . " ?&defined(&GetKeyVal(#1,'title'))(<dc:title>&GetKeyVal(#1,'title')</dc:title>())"
86     . " ?&defined(&GetKeyVal(#1,'contributors'))(<dc:contributor>&GetKeyVal(#1,'contribut
87     . " <omdoc:imports from='?&GetKeyVal(#1,'load')(&canonical_omdoc_path(&GetKeyVal(#1,'
88     . " #body"
89     . "</omdoc:theory>)",
90     afterDigestBegin=>sub {
91     my ($stomach, $whatsit) = @_;
92     my $keyval = $whatsit->getArg(1);
93     my $signature = ToString($whatsit->getArg(2));
94     my $language = ToString($whatsit->getArg(3));
95     if ($keyval) {
96     # If we're not given load, AND the langfiles option is in effect,
97     # default to #2
98     if ((! $keyval->getValue('load')) && (LookupValue('smultiling_langfiles')) {
99     $keyval->setValue('load',$signature); }
100    # Always load a TeX file
101    $keyval->setValue('ext','tex');
102    AssignValue('modnl_signature',$signature);
103    $keyval->setValue('id'," $signature.$language"); }
104    module_afterDigestBegin(@_);
105    importmoduleI(@_);
106    AssignValue(multiling => 1);
107    return; },
108    afterDigest =>\&module_afterDigest );
109 \</txml>%$

```

viewnl The `viewnl` environment is just a layer over the `viewsketch` environment with the keys and language suitably adapted.⁴

```

110 \<txml>RawTeX(
111 \<sty | txml>
112 \newenvironment{viewnl}[5] [] {\def\@test{#1}\ifx\@test\@empty%
113 \begin{viewsketch}[id=#2.#3,ext=tex]{#4}{#5}\else%
114 \begin{viewsketch}[id=#2.#3,#1,ext=tex]{#4}{#5}\fi}
115 {\end{viewsketch}}

```

⁴EDNOTE: 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`!

3.4 Multilingual Statements and Terms

`\mtref*` we first first define an auxiliary conditional `\@instring` that checks if `?` is in the first argument. `\mtrefi` uses it, if there is one, it just calls `\termref`, otherwise it calls `\@mtrefi`, which assembles the `\termref` after splitting at the `?`.

```

116 \def\@instring#1#2{TT\fi\begin{group}\edef\x{\end{group}\noexpand\in@{#1}{#2}}\x\ifin@
117 \newcommand\mtrefi[2][]{\if\@instring{?}{#1}\@mtref #1\relax{#2}\else\termref[cd=#1]{#2}\fi}
118 \def\@mtref#1?#2\relax{\termref[cd=#1,name=#2]}
119 \newcommand\mtrefis[2][]{\mtrefi[#1]{#2s}}
120 \newcommand\mtrefiis[3][]{\mtrefi[#1]{#2 #3}}
121 \newcommand\mtrefiis[3][]{\mtrefi[#1]{#2 #3s}}
122 \newcommand\mtrefiis[4][]{\mtrefi[#1]{#2 #3 #4}}
123 \newcommand\mtrefiis[4][]{\mtrefi[#1]{#2 #3 #4s}}
124 \</sty | ltxml>
125 \<ltxml>' );

```

3.5 Miscellaneous

the `\ttl` macro (to-translate) is used to mark untranslated stuff. We need a better L^AT_EX MLtreatment of this eventually that is integrated with MathHub.info.

```

\ttl
126 \<ltxml>RawTeX( '
127 \<*sty | ltxml>
128 \newcommand\ttl[1]{\red{TTL: #1}}
129 \</sty | ltxml>
130 \<ltxml>' );

```

3.6 Finale

Finally, we need to terminate the file with a success mark for perl.

```

131 \<ltxml>1;

```