`smultiling.sty`: Multilinguality Support for sTeX

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

Deyan Ginev
Authorea

October 26, 2020

Abstract

The `smultiling` 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).

The `smultiling` package adds multilinguality support for sTeX, the idea is that multilingual modules in sTeX 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

We have been using sTeX as the encoding for the Semantic Multilingual Glossary of Mathematics (SMGloM; see [Gin+16; SMG]). The SMGloM data model has been taxing the representational capabilities of sTeX with respect to multilingual support and verbalization definitions; see [Koh14], which we assume as background reading for this note.

1.1 sTeX Module Signatures

(Monolingual) sTeX had the intuition that the symbol and and definitions (`\begin/\end{module}`, `\symdef`, and `\symvariant`) as well as the `\import/\usemodule` directives (we jointly call this information the **sTeX module signatures**) are interspersed with the text. In the SMGloM, we make the module signatures primary objects that contain the language-independent part of the formal structure conveyed by the sTeX documents and there may be multiple narrative "language bindings" that are translations of each other. 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} ($\bar$) 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 sTeX

We retain the old `module` environment as an intermediate stage. It is still useful for monolingual texts.

2 The User Interface

2.1 Package Options

langfiles   The `smultiling` package accepts the `langfiles` option that specifies – for a module ⟨*mod*⟩ that 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`.[1]

Furthermore, the `smultiling` package accepts the options of the `modules` package and passes them on to it.

mh   Finally, the `mh` option turns on MathHub support; see [Koh20].

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

```
\usepackage{multiling}
\begin{modsig}{foo}
  \symdef{bar}{BAR}
  \symi[gfc=N]{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 sTeX for Figure 1.

2.2   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 required 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. In the optional keyword argument we have the same keys as `modsig`, but we also add the `primary` key, which can specify the primary language binding (the one the others translate from; and which serves as the reference in case of translation conflicts).

\symi   There is another difference in the multilingual encoding: All symbols are introduced in the module signature, either by a `\symdef` or the new `\symi` macro. `\symi[`⟨*keys*⟩`]{`⟨*name*⟩`}` takes a symbol name ⟨*name*⟩ as an argument and re-
\symi*   serves that name. The variant `\symi*[`⟨*keys*⟩`]{`⟨*name*⟩`}` declares ⟨*name*⟩ to be
\symii   a primary symbol; see [Koh14] for a discussion. sTeX provides variants `\symii`,
\symiii   `\symiii`, and `\symiv` – and their starred versions – for multi-part names. The
\symiv   key-value interface ⟨*keys*⟩ does not have any effect on the LaTeX rendering, it can be used to embed metadata. See for instance Subsection 2.6.

2.3   Multilingual Definitions and Cross-referencing 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.

Finally note that hyperlinks on term references only have information on the underlying symbol and module names – i.e. signature information – and we need to cross-reference into the language bindings. To do this, we need to know the base language of the document. To ensure basic functionality we set this to `en` \sTeXlanguage and provide the `\sTeXlanguage` macro to set it.

2.4 Multilingual Views

Views receive a similar treatment as modules in the `smultiling` package. A multilingual view consists of

viewsig
1. 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 (`loadfrom` and `loadto`) and

viewnl
2. multiple **language bindings** marked up by the `viewnl` environment, which takes two required arguments: the view name and the language specifier. The optional first key/value argument takes the same keys as `viewsig` except the last two.

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

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

```
\begin{viewnl}[creators=miko]{norm-metric}{en}
  \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{viewnl}
```

2.5 Mathematical Keywords

For translations of the mathematical keywords, the `statements` and `sproofs` EdN:2 packages in sTeX define special language definition files, e.g. `statements-ngerman.ldf`.[23]

---

[2]EDNOTE: say more about this

There is currently only very limited support for this.

2.6 GF Metadata

Several sTEX macros and environments allow keys for syntactical information about the objects declared.

gfc   The symbol-declaring macros `\symi` and friends as well as `\symdef` allow gfc key allows to specify the grammatical category in terms of the Resource Grammar of the Grammatical Framework [GFR].

The verbalization-defining macros `\defi` and friends allow the `gfa` (GF apply) and `gfl` (GF linearization) keys.

A definiendum of the form `\defii[gfa=mkN]{empty}{set}` generates the GF linearization `empty_set = mkN "empty set"`. Some what less conveniently, `\defii[name=datum,gfl={mkN "Datum", "Daten"}]{Datum}` can be used if the GF linearization is more complex than simply applying a "make command" to the verbalization.

3 Limitations

We list the limitations of the `smultiling` package.

3.1 General `babel` Integration

There is currently no integration with the `babel` package that handles language-specific aspects in LaTeX. In particular, selecting the right language must be done manually. In particular, the example from Figure **??** would really have the form given in Figure 3 – see the `\usepackage[usenglish,ngerman]{babel}` in line 2, and the `\selectlanguage` statements in lines 6 and 13.

For the `langfiles` setup, which assumes that module signatures and language bindings are in separate files, `babel` integration can be simplified by providing a language-specific preamble file with `\usepackage{⟨language⟩}{babel}` which is pre-pended to all language binding files when formatted. This preamble can also contain the other language-specific packages (e.g. for font encodings, etc.).

3.2 PDF links on term references are language-dependent

Given the `langfiles` mode, we need the intended language to generate PDF links on term references. But we cannot infer this for top-level "papers" (we do in the language bindings). So it has to be specified via `\sTexlanguage`, and we do not really had a way to check that it is. Unfortunately, the only place it would be natural to do so is in `\mod@component`, but the `\PackageError` there had to be commented out, since it leads to serious errors. Thus we set the language to `en`

---

[3]EDNOTE: There is the translator package which belongs to beamer, maybe we should switch to that.

```
\usepackage{multiling}
\usepackage[usenglish,ngerman]{babel}% babel support
\begin{modsig}{foo}
  \importmodule{arrays}
  \symdef{bar}{BAR}
  \symi{sar}
\end{modsig}
\selectlanguage{english}% english version follows
\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}
\selectlanguage{german}% german umlauts please
\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 3: Multilingual STEX with `babel`

by default, which is sub-optimal. Maybe there is a way to infer the document language from the babel settings.

3.3 Language-Specific Limitations

Some languages have more problems than others

Turkish makes = an active character (to give better spacing); this interacts unfavourably with the `keyval` package which needs = as key/value separator (and gives it a different category code). Therefore we need to prohibit this by restricting the `shorthands` option: use `\usepackage[turkish,shorthands=:!]{babel}`.

Chinese needs special fonts and `xelatex`[4].

---

4 Implementation

4.1 Package Options

```
1 ⟨*sty⟩
2 \newif\if@smultiling@mh@\@smultiling@mh@false
3 \DeclareOption{mh}{\@smultiling@mh@true}
4 \newif\if@langfiles\@langfilesfalse
5 \DeclareOption{langfiles}{\@langfilestrue}
6 \DeclareOption*{\PassOptionsToPackage{\CurrentOption}{modules}}
7 \ProcessOptions
```

We load the packages referenced here.
```
8 \if@smultiling@mh@\RequirePackage{smultiling-mh}\fi
9 \RequirePackage{etoolbox}
10 \RequirePackage{structview}
```

4.2 Module 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. Finally, we set the flag \mod@⟨mod⟩@multiling to true.

```
11 \newenvironment{modsig}[2][]{\def\@test{#1}%
12 \ifx\@test\@empty\begin{module}[id=#2]\else\begin{module}[id=#2,#1]\fi%
13 \expandafter\gdef\csname mod@#2@multiling\endcsname{true}%
14 \ignorespacesandpars}
15 {\end{module}\ignorespacesandparsafterend}
```

\mod@component We redefine the macro from the modules package that computes the module component identifier for external links on term references. If \mod@⟨mod⟩@multiling is true, then we make the component identifier .⟨lang⟩, which can be customized by the next macro below.

```
16 \renewcommand\mod@component[1]{%
17 \expandafter\ifx\csname mod@#1@multiling\endcsname\@true%
18 \@ifundefined{smultiling@language}{}
19 % for some reason this error message bombs big time; so we leave it out.
20 % {\PackageError{smultiling}%
21 %    {No document language specified for term reference links}
22 %    {use \protect\sTeXlanguage to specify it!}}
23 {.\smultiling@language}%
24 \fi}
```

\sTeXlanguage This macro sets the internal flag \smultiling@language, we set the default to en, since otherwise hyper-references on term references do not work.

```
25 \newcommand\sTeXlanguage[1]{\def\smultiling@language{#1}}
26 \sTeXlanguage{en}
```

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

```
27 \newenvironment{viewsig}[4][]{\def\@test{#1}\ifx\@test\@empty%
```

```
28    \begin{view}[id=#2]{#3}{#4}\else\begin{view}[id=#2,#1]{#3}{#4}\fi%
29    \ignorespacesandpars}
30    {\end{view}\ignorespacesandparsafterend}
```

`\@sym*` has a starred form for primary symbols. The key/value interface has no effect on the LATEX side. We read the to check whether only allowed ones are used.

```
31 \define@key{symi}{noverb}[all]{}%
32 \define@key{symi}{align}[WithTheSymbolOfTheSameName]{}%
33 \define@key{symi}{specializes}{}%
34 \define@key{symi}{noalign}[true]{}%
35 \newcommand\symi{\@ifstar\@symi@star\@symi}
36 \newcommand\@symi[2][]{\metasetkeys{symi}{#1}%
37   \usemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2}\fi\ignorespacesan
38 \newcommand\@symi@star[2][]{\metasetkeys{symi}{#1}%
39   \usemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2}\fi\ignore
40 \newcommand\symii{\@ifstar\@symii@star\@symii}
41 \newcommand\@symii[3][]{\metasetkeys{symi}{#1}%
42   \usemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3}\fi\ignorespace
43 \newcommand\@symii@star[3][]{\metasetkeys{symi}{#1}%
44   \usemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2-#3}\fi\ign
45 \newcommand\symiii{\@ifstar\@symiii@star\@symiii}
46 \newcommand\@symiii[4][]{\metasetkeys{symi}{#1}%
47   \usemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3-#4}\fi\ignoresp
48 \newcommand\@symiii@star[4][]{\metasetkeys{symi}{#1}%
49   \usemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2-#3-#4}\fi\
50 \newcommand\symiv{\@ifstar\@symiv@star\@symiv}
51 \newcommand\@symiv[5][]{\metasetkeys{symi}{#1}%
52   \usemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3-#4-#5}\fi\ignor
53 \newcommand\@symiv@star[5][]{\metasetkeys{symi}{#1}%
54   \usemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2-#3-#4-#5}\
```

4.3   Language Bindings

`modnl:*`

```
55 \addmetakey{modnl}{load}
56 \addmetakey{modnl}{path}% ignored, specified to simplify keyval argument passing
57 \addmetakey*{modnl}{title}
58 \addmetakey*{modnl}{creators}
59 \addmetakey*{modnl}{contributors}
60 \addmetakey{modnl}{srccite}
61 \addmetakey{modnl}{primary}[yes]
```

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

```
62 \newenvironment{modnl}[3][]{\metasetkeys{modnl}{#1}%
63   \def\@test{#1}\ifx\@test\@empty\begin{module}[id=#2.#3]\else\begin{module}[id=#2.#3,#1]\fi%
64   \def\smultiling@language{#3}%
65   \if@langfiles
66   \ifx\modnl@load\@empty\importmodule[load=#2]{#2}\else\importmodule[load=\modnl@load]{#2}\fi
```

```
67    \else
68    \ifx\modnl@load\@empty\importmodule{#2}\else\importmodule[load=\modnl@load]{#2}\fi%
69    \fi%
70    \ignorespacesandpars}
71 {\end{module}\ignorespacesandparsafterend}
```

viewnl The `viewnl` environment is just a layer over the `view` environment with the keys and language suitably adapted.[5]

```
72 \newenvironment{viewnl}[5][]{\def\@test{#1}\ifx\@test\@empty%
73    \begin{view}[id=#2.#3]{#4}{#5}\else%
74    \begin{view}[id=#2.#3,#1]{#4}{#5}\fi%
75    \ignorespacesandpars}
76    {\end{view}\ignorespacesandparsafterend}
```

4.4   GF Metadata

gfc We add the `gfc` key to various symbol declaration macros.

```
77 \addmetakey{symi}{gfc}
78 \addmetakey{symdef}{gfc}%
```

gfa/l

```
79 \addmetakey{definiendum}{gfa}
80 \addmetakey{definiendum}{gfl}
```

4.5   Miscellaneneous

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

\ttl

```
81 \newcommand\ttl[1]{\red{TTL: #1}}
82 ⟨/sty⟩
```

---

[5]EDNOTE: 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*⟩!

Change History

References

[GFR]   B. Bringert, T. Hallgren, and A. Ranta. *GF Resource Grammar Library: Synopsis.* URL: https://www.grammaticalframework.org/lib/doc/synopsis/ (visited on 03/11/2020).

[Gin+16]   Deyan Ginev et al. "The SMGloM Project and System. Towards a Terminology and Ontology for Mathematics". In: *Mathematical Software - ICMS 2016 - 5th International Congress.* Ed. by Gert-Martin Greuel et al. Vol. 9725. LNCS. Springer, 2016. DOI: 10.1007/978-3-319-42432-3. URL: https://kwarc.info/kohlhase/papers/icms16-smglom.pdf.

[Koh14]   Michael Kohlhase. "A Data Model and Encoding for a Semantic, Multilingual Terminology of Mathematics". In: *Intelligent Computer Mathematics.* Conferences on Intelligent Computer Mathematics (Coimbra, Portugal, July 7–11, 2014). Ed. by Stephan Watt et al. LNCS 8543. Springer, 2014, pp. 169–183. ISBN: 978-3-319-08433-6. URL: https://kwarc.info/kohlhase/papers/cicm14-smglom.pdf.

[Koh20]   Michael Kohlhase. *MathHub Support for sTeX.* Tech. rep. 2020. URL: https://github.com/sLaTeX/sTeX/raw/master/sty/mathhub/mathhub.pdf.

[SMG]   *SMGloM Glossary.* URL: http://mathhub.info/applications/glossary (visited on 11/21/2019).