Semantic Markup for Mathematical Statements[*]

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

November 16, 2015

Abstract

The `statements` 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 provides semantic markup facilities for mathematical statements like Theorems, Lemmata, Axioms, Definitions, etc. in STEX files. This structure can be used by MKM systems for added-value services, either directly from the STEX sources, or after translation.

Contents

---

[*]Version v1.2 (last revised 2015/04/03)

1 Introduction

The motivation for the `statements` package is very similar to that for semantic macros in the `modules` package: We want to annotate the structural semantic properties of statements in the source, but present them as usual in the formatted documents. In contrast to the case for mathematical objects, the repertoire of mathematical statements and their structure is more or less fixed.

This structure can be used by MKM systems for added-value services, either directly from the sTeX sources, or after translation. Even though it is part of the sTeX collection, it can be used independently, like it's sister package `sproofs`.

sTeX [**sTeX:online**; Koh08] is 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). Currently the OMDoc format [Koh06] is directly supported.

2 The User Interface

The `statements` package supplies a semantically oriented infrastructure for marking up mathematical statements: fragments of natural language that state properties of mathematical objects, e.g. axioms, definitions, or theorems. The `statement` package provides an infrastructure for marking up the semantic relations between statements for the OMDoc transformation and uses the `ntheorem` package [MS] for formatting (i.e. transformation to PDF).

2.1 Package Options

The `statements` package provides the `defindex` option to sTeX. If this is set, then definienda are automatically passed into the index of the document. Furthermore, showmeta the `statements` package passes the `showmeta` to the `metakeys` package. If this is set, then the metadata keys are shown (see [Koh15a] for details and customization options).

2.2 Statements

All the statements are marked up as environments, that take a `KeyVal` argument that allows to annotate semantic information. Generally, we distinguish two forms of statements:

block statements have explicit discourse markers that delimit their content in the surrounding text, e.g. the boldface word "**Theorem:**" as a start marker and a little line-end box as an end marker of a proof.

flow statements do not have explicit markers, they are interspersed with the surrounding text.

Since they have the same semantic status, they must both be marked up, but styled differently. We distinguish between these two presentational forms with the **display=** **display** key, which is allowed on all statement environments. If it has the value **block** (the default), then the statement will be presented in a paragraph of its own, have explicit discourse markers for its begin and end, possibly numbering, etc. If it has the value **flow**, then no extra presentation will be added the semantic information is invisible to the reader. Another key that is present on all statement **id=** environments in the **id** key it allows to identify the statement with a name and to reference it with the semantic referencing infrastructure provided by the **sref** package [Koh15c].

2.2.1   Axioms and Assertions

assertion The **assertion** environment is used for marking up statements that can be justified from previously existing knowledge (usually marked with the monikers "Theorem", "Lemma", "Proposition", etc. in mathematical vernacular). The environment **assertion** is used for all of them, and the particular subtype of as- **type=** sertion is given in the **type** key. So instead of \begin{Lemma}we have to write \begin{assertion}[type=lemma](see Example 1 for an example).

```
\begin{assertion}[id=sum-over-odds,type=lemma]
   $\sum_{i=1}^n{2i-1}=n^2$
\end{assertion}
```

will lead to the result
Lemma 2.1 $\sum_{i=1}^{n} 2i - 1 = n^2$

Example 1: Semantic Markup for a Lemma in a **module** context

Whether we will see the keyword "Lemma" will depend on the value of the optional **display** key. In all of the **assertion** environments, the presentation expectation is that the text will be presented in italic font. The presentation (keywords, spacing, and numbering) of the **assertion** environment is delegated to a theorem styles from the **ntheorem** environment. For an assertion of type $\langle type \rangle$ the **assertion** environment calls the ST$\langle type \rangle$**AssEnv** environment provided by the **statements** package; see Figure 2 for a list of provided assertion types. Their formatting can be customized by redefining the ST$\langle type \rangle$**AssEnv** environment via the \renewtheorem command from the **ntheorem** package; see [MS] for details.

axiom The **axiom** environment is similar to **assertion**, but the content has a different ontological status: axioms are assumed without (formal) justification, whereas assertions are expected to be justified from other assertions, axioms or definitions. This environment relegates the formatting to the **STaxiomEnv** environment, which can be redefined for configuration.

2.2.2   Symbols

symboldec The **symboldec** environment can be used for declaring concepts and symbols. Note

Value	Explanation
theorem, **proposition**	an important assertion with a proof
Note that the meaning of **theorem** (in this case the existence of a proof) is not enforced by OMDOC applications. It can be appropriate to give an assertion the **theorem**, if the author knows of a proof (e.g. in the literature), but has not formalized it in OMDOC yet.	
lemma	a less important assertion with a proof
The difference of importance specified here is even softer than the other ones, since e.g. reusing a mathematical paper as a chapter in a larger monograph, may make it necessary to downgrade a theorem (e.g. the main theorem of the paper) and give it the status of a lemma in the overall work.	
corollary	a simple consequence
An assertion is sometimes marked as a corollary to some other statement, if the proof is considered simple. This is often the case for important theorems that are simple to get from technical lemmata.	
postulate, **conjecture**	an assertion without proof or counter-example
Conjectures are assertions, whose semantic value is not yet decided, but which the author considers likely to be true. In particular, there is no proof or counter-example.	
false-conjecture	an assertion with a counter-example
A conjecture that has proven to be false, i.e. it has a counter-example. Such assertions are often kept for illustration and historical purposes.	
obligation, **assumption**	an assertion on which a proof of another depends
These kinds of assertions are convenient during the exploration of a mathematical theory. They can be used and proven later (or assumed as an axiom).	
rule	a normative assertion
These kinds of assertions can be interpreted procedurally to trigger actions	
observation	if everything else fails
This type is the catch-all if none of the others applies.	

Example 2: Types of Mathematical Assertions

the the `symdef` forms from the `modules` package will not do this automatically (but the `definition` environment and the `\inlinedef` macro will for all the definienda; see below). The `symboldec` environment takes an optional keywords argument with the keys `id`, `role`, `title` and `name`. The first is for general identification, the `role` specifies the OPENMATH/OMDOC role, which is one of `object`, `type`, `sort`, `binder`, `attribution`, `application`, `constant`, `semantic-attribution`, and `error` (see the OMDOC specification for details). The `name` key specifies the OPENMATH name of the symbol, it should coincide with the control sequence introduced by the corresponding `\symdef` (if one is present). The `title` key is for presenting the title of this symbol as in other statements. Usually, `axiom` and `symboldec` environments are used together as in Figure 3.

2.2.3 Types

In many cases, we can give additional information for symbols in the form of type assignments. STEX does not fix a type system, but allows types to be arbitrary mathematical objects that they can be defined in (imported) modules. The `\symtype`   `\symtype` macro can be used to assign a type to a symbol:

$$\texttt{\symtype[}\langle keys\rangle\texttt{]\{}\langle sym\rangle\texttt{\}\{}\langle type\rangle\texttt{\}}$$

assigns the type $\langle type \rangle$ to a symbol with name $\langle sym \rangle$. For instance

```
\symtype[id=plus-nat.type,system=sts]{plus}{\fntype{\Nat,\Nat}\Nat}
```

assigns the type $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$ (in the `sts` type system) to the symbol `plus`. This states (type assignments are statements epistemologically) that addition is a binary function on natural numbers. The `\symtype` macro supports the keys `id` (for identifiers) and `system` for the type system.

Often, type assignments occur in informal context, where the type assignment is given by a natural language sentence or phrase. For this, the `statements` `typedec` package supplies the `typedec` environment and the `\inlinetypedec` macro. Both `\inlinetypedec` take an optional keyval argument followed by the type. The phrase/sentence is the body of the `typedec` environment and the last argument of the `\inlinetypedec` macro. The symbol name is given in via the `for` key. For convenience, the macro `\thedectype` `\thedectype` is bound to the type. So we can use

```
\begin{typedec}[for=plus,id=plus-nat.type]{\fntype{\Nat,\Nat}\Nat}
  $+:\thedectype$ is a binary function on $\Nat$
\end{typedec}
```

instead of the `\symtype` above in an informal setting.

2.2.4 Definitions, and Definienda

`definition` The `definition` environment is used for marking up mathematical definitions. Its peculiarity is that it defines (i.e. gives a meaning to) new mathematical con- `\definiendum` cepts or objects. Theseare identified by the `\definiendum` macro, which is used

5

```
 \symdef{zero}{0}
 \begin{symboldec}[name=zero,title=The number zero,type=constant]
   The number zero, it is used as the base case of the inductive definition
   of natural numbers via the Peano Axioms.
 \end{symboldec}

 \symdef{succ}[1]{\prefix{s}{#1}}
 \begin{symboldec}[name=succ,title=The Successor Function,type=application]
   The successor function, it is used for the step case of the inductive
   definition of natural numbers via the Peano Axioms.
 \end{symboldec}

 \symdef{NaturalNumbers}{\mathbb{N}}
 \begin{symboldec}[name=succ,title=The Natural Numbers,type=constant]
   The natural numbers inductively defined via the Peano Axioms.
 \end{symboldec}

 \begin{axiom}[id=peano.P1,title=P1]
   $\zero$ is a natural number.
 \end{axiom}
 ...
 \begin{axiom}[id=peano.P5,title=P5]
   Any property $P$ such $P(\zero)$ and $P(\succ{k})$ whenever $P(k)$
   holds for all $n$ in $\NaturalNumbers$
 \end{axiom}
```

will lead to the result

[id=peano] zero0 zero: (**The number zero**)
The number zero, it is used as the base case of the inductive definition of natural
numbers via the Peano Axioms.
succ[1] s(1) succ: (**The Successor Function**)
The successor function, it is used for the step case of the inductive definition of
natural numbers via the Peano Axioms.
NaturalNumbersN succ: (**The Natural Numbers**)
The natural numbers inductively defined via the Peano Axioms.

Axiom 2.2 (P1) is a natural number.

. . .

Axiom 2.6 (P5) Any property P such $P()$ and $P(\succ k)$ whenever $P(k)$ holds for
all n in

Example 3: Semantic Markup for the Peano Axioms

6

as `\definiendum[⟨sysname⟩]{⟨text⟩}`. Here, ⟨text⟩ is the text that is to be emphasized in the presentation and the optional ⟨sysname⟩ is a system name of the symbol defined (for reference via `\termref`, see Section 2.3). If ⟨sysname⟩ is not given, then ⟨text⟩ is used as a system name instead, which is usually sufficient for most situations.

```
\symdef{one}{1}
\begin{definition}[id=one.def,for=one]
  $\notatiendum[one]{\one}$ is the successor of $\zero$
  (formally: $\one\colon=\succ\zero$)
\end{definition}
```

will lead to the result

peano one1

Definition 2.7   is the successor of (formally: : =≻)

<div align="center">Example 4: A Definition based on Figure 3</div>

defi   The `\defi{⟨word⟩}` macro combines the functionality of the `\definiendum` macro with index markup from the `omdoc` package [Koh15b]: use

$$\texttt{\textbackslash defi[⟨name⟩]\{⟨word⟩\}[⟨indexkeys⟩]}$$

to markup a definiendum ⟨word⟩ with system name ⟨name⟩ that appear in the index (where ⟨indexkeys⟩ are passed to the `\omdoc@index` macro from the `omdoc` package) — in other words in almost all definitions of single-word concepts. We

\defii   also have the variants `\defii` and `\defiii` for (adjectivized) two-word com-
\defiii   pounds. Note that if the definiendum contains sematnic macros, then we need to specify the `loadmodules` key and also protect the semantic macro. For instance if `\eset` is the semantic macro for ∅, then we would use

```
\defii[eset-comp]{$\protect\eset$}{compatible}[loadmodules]
```

\adefi   for the definiendum markup. Finally, the variants `\adefi`, `\adefii`,and `\adefiii`
\adefii   have an additional first argument that allows to specify an alternative text; see
\adefiii Figure 5. The main use of these is to mark up inflected forms as in

```
we speak of \adefi{lemmata}{lemma} in this case.
```

          As the greatest number of these are plurals, which tends to be regular (e.g.
\defis   adding a trailing "s" in English), we provide the variants `\defis`, `\defiis`, and
\defiis   `\defiiis` for that case: `\defiis{simple}{group}` is equivalent to much longer
\defiiis `\adefii{simple groups}{simple}{group}`.
          Note that the `\definiendum`, `\defi`, `\defii`, and `\defiii` macros can only be used inside the definitional situation, i.e. in a `definition` or `symboldec` environment or a `\inlinedef` macro. If you find yourself in a situation where you want to

source		
system name	result	index
`\defi{concept}`		
`concept`	concept	concept
`\defi[csymbol]{concept}`		
`csymbol`	concept	concept
`\adefi[csymbol]{concepts}{concept}`		
`csymbol`	concepts	concept
`\defii{concept}{group}`		
`concept-group`	concept group	concept group, group - , concept
`\adefii{small}{concept}{group}`		
`small-concept-group`	small concept group	small concept group, concept group - , small

Example 5: Some definienda with Index

use it outside, you will most likely want to wrap the appropriate text fragment in a `\begin{definition}[display=flow]` ... and `\end{definition}`. For instance, we could continue the example in Figure 3 with the `definition` environment in Figure 4.

`\inlinedef`   Sometimes we define mathematical concepts in passing, e.g. in a phrase like "...$s(o)$ which we call **one**.". For this we cannot use the `definition` environment, which presupposes that its content gives all that is needed to understand the definition. But we do want to make use of the infrastructure introduced for the `definition` environment. In this situation, we just wrap the phrase in an `\inlinedef` macro that makes them available. The `\inlinedef` macro accepts the same `id` and `for` keys in its optional argument, and additionally the `verbalizes` key which can be used to point to a full definition of the concept somewhere else.

Note that definienda can only be referenced via a `\term` element, if they are only allowed inside a named module, i.e. a `module` environment with a name given by the `id=` key or the `theory=` key on is specified on the definitional environment.

2.2.5   Examples

`example`   The `example` environment is a generic statement environment, except that the `for` key should be given to specify the identifier what this is an example for. The `example` environment also expects a `type` key to be specified, so that we know whether this is an example or a counterexample.

`\inlineex`   The `\inlineex` is analogous to `\inlinedef`, only that it is used for inline examples, e.g. "...mammals, e.g. goats". Note that we have used an inline example for an inline example.

As examples need to import foreign vocabularies (those used to construct the example), the example environment provides the `\usevocab` command, a special

variant of \importmodule that is only available in the example environment and the argument of \inlineex.

2.3 Cross-Referencing Symbols and Concepts

If we have defined a concept with the \definiendum macro, then we can mark up other occurrences of the term as referring to this concept. Note that this process cannot be fully automatized yet, since that would need advanced language technology to get around problems of disambiguation, inflection, and non-contiguous phrases[1]. Therefore, the \termref can be used to make this information explicit.

\termref

It takes the keys

cdbase to specify a URI (a path actually, since LaTeX cannot load from URIs) where the module can be found.

cd to specify the module in which the term is defined. If the cd key is not given, then the current module is assumed. If no cdbase is specified (this is the usual case), then the CD has to be imported via a \importmodule from the modules package [KGA15].

name to specify the name of the definiendum (which is given in the body of the \definiendum or the optional argument). If the name key is not specified, then argument of the \termref macro is used.

role is currently unused.

\termref[cd=⟨cd⟩,name=⟨name⟩]{⟨text⟩} will just typeset the link text ⟨text⟩ with (if the hyperref package is loaded) a hyperlink to the definition in module ⟨cd⟩ that defines the concept ⟨name⟩, e.g. that contains \defi[⟨name⟩]{⟨text⟩}.

Just as the \definiendum macro has the convenience variants \defi, \defii and \defiii, the \termref has variants \trefi, \trefii, and \trefiii that take two and three arguments for the parts of the compositum. In the same module, concepts that are marked up by \defi{⟨name⟩} in the definition can be referenced

\trefi   by \trefi{⟨name⟩}. Here the link text is just ⟨name⟩. Concepts defined via
\trefii   \defii{⟨first⟩}{⟨second⟩} can be referenced by \trefii{⟨first⟩}{⟨second⟩} (with
\trefiii   link text "⟨first⟩ ⟨second⟩") and analogously for \defiii and \trefiii.
\atref*   We have variants \atrefi, \atrefii, and \atrefiii with alternative link text. For instance \atrefii{⟨text⟩}{⟨first⟩}{⟨second⟩} references a concept introduced by \defii{⟨first⟩}{⟨second⟩} but with link text ⟨text⟩. Of course, if the system identifier is given explicitly in the optional argument of the definition form, as in \defii[⟨name⟩]{⟨first⟩}{⟨second⟩}, then the terms are referenced by \trefi{⟨name⟩}.

For referencing terms outside the current module, the module name can be specified in the first optional argument of the *tref* macros. To specify the cdbase, we have to resort to the \termref macro with the keyval arguments.

Note that the \termref treatment above is natural for "concepts" declared by the \termdef macro from the modules package [KGA15]. Concepts are natural language names for mathematical objects. For "symbols", i.e. symbolic identifiers for mathematical objects used in mathematical formulae, we use the \symdef

---

[1]We do have a program that helps annotate larger text collections spotting the easy cases; see http://kwarc.info/projects/stex and look for the program termin.

macro from the `modules` package. Sometimes, symbols also have an associated natural language concept, and we want to use the symbol name to reference it (instead of specifying `cd` and `name` which is more inconvenient). For this the

`\symref` `statements` package supplies the `\symref` macro. Like `\termref`, and invocation of `\symref{`⟨*cseq*⟩`}{`⟨*text*⟩`}` will just typeset ⟨*text*⟩ with a hyperlink to the relevant definition (i.e. the one that has the declaration `for=`⟨*cseq*⟩ in the metadata argument.)

`\term` The `\term` macro is a variant of the `\termref` macro that marks up a phrase as a (possible) term reference, which does not have a link *yet*. This macro is a convenient placeholder for authoring, where a `\termref` annotation is (currently) too tedious or the link target has not been authored yet. It facilitates lazy flexiformalization workflows, where definitions for mathematical concepts are supplied or marked up by need (e.g. after a `grep` shows that the number of `\term` annotations of a concept is above a threshold). Editors or active documents can also support the `\term` macro like a wiki-like dangling link: a click on `\term{`⟨*phrase*⟩`}` could generate a new editor buffer with a stub definition (an `definition` environment

EdN:1 with `\definiendum` macro and appropriate metadata).[1]

3   Configuration of the Presentation

`\defemph` The `\defemph` macro is a configuration hook that allows to specify the style of presentation of the definiendum. By default, it is set to `\bf` as a fallback, since we can be sure that this is always available. It can be customized by redefinition: For instance `\renewcommand{\defemph}[1]{\emph{#1}}`, changes the default behavior to italics.

`\termemph` The `\termenph` macro does the same for the style for `\termref`, it is empty by default. Note the term might carry an implicit hyper-reference to the defining occurrence and that the presentation engine might mark this up, changing this behavior.

`\stDMemph` The `\stDMemph` macro does the same for the style for the markup of the discourse markers like "Theorem". If it is not defined, it is set to `\bf`; that allows to
EdN:2 preset this in the class file. [2]

Some authors like to lowercase the semantic references, i.e. use "axiom 2.6" instead of the default "Axiom 2.6" to refer to the last axiom in Figure 3. This can
`\STpresent` be achieved by redefining the `\STpresent` macro, which is applied to the keyword
EdN:3 of the `ST*Env` theorem environments.[3]

Finally, we provide configuration hooks in Figure 6 for the statement types provided by the `statement` package. These are mainly intended for package authors building on `statements`, e.g. for multi-language support. The language bindings are given in the `smultiling` [KG15] package not in `statements` itself.

---

[1] EDNOTE: MK: we probably need multi-part variants for *tref*
[2] EDNOTE: function declarations
[3] EDNOTE: this does not quite work as yet, since **STpresent** is applied when the label is written. But we would really like to have it applied when the reference is constructed. But for that we need to split the label into keyword and number in package `sref`.

Environment	configuration macro	value
`STtheoremAssEnv`	`\st@theorem@kw`	Theorem
`STlemmaAssEnv`	`\st@lemma@kw`	Lemma
`STpropositionAssEnv`	`\st@proposition@kw`	Proposition
`STcorollaryAssEnv`	`\st@corollary@kw`	Corollary
`STconjectureAssEnv`	`\st@conjecture@kw`	Conjecture
`STfalseconjectureAssEnv`	`\st@falseconjecture@kw`	Conjecture (false)
`STpostulateAssEnv`	`\st@postulate@kw`	Postulate
`STobligationAssEnv`	`\st@obligation@kw`	Obligation
`STassumptionAssEnv`	`\st@assumption@kw`	Assumption
`STobservationAssEnv`	`\st@observation@kw`	Observation
`STruleAssEnv`	`\st@rule@kw`	Rule
`STexampleEnv`	`\st@example@kw`	Example
`STaxiomEnv`	`\st@axiom@kw`	Axiom
`STdefinitionEnv`	`\st@definition@kw`	Definition
`STnotationEnv`	`\st@notation@kw`	Notation

Example 6: Configuration Hooks for statement types

4 Limitations

In this section we document known limitations. If you want to help alleviate them, please feel free to contact the package author. Some of them are currently discussed in the sTeX GitHub repository [sTeX].

1. none reported yet

5 The Implementation

The `statements` package generates two files: the LaTeX package (all the code between ⟨*package⟩ and ⟨/package⟩) and the LaTeXML bindings (between ⟨*ltxml⟩ and ⟨/ltxml⟩). We keep the corresponding code fragments together, since the documentation applies to both of them and to prevent them from getting out of sync.

The general preamble for LaTeXML:

```
1 ⟨∗ltxml⟩
2 # -*- CPERL -*-
3 package LaTeXML::Package::Pool;
4 use strict;
5 use LaTeXML::Package;
6 ⟨/ltxml⟩
```

5.1 Package Options

We declare some switches which will modify the behavior according to the package options. Generally, an option `xxx` will just set the appropriate switches to true

(otherwise they stay false). First we have the general options

```
 7 ⟨∗package⟩
 8 \newif\if@statements@mh@\@statements@mh@false
 9 \DeclareOption{mh}{\@statements@mh@true}
10 \newif\ifdef@index\def@indexfalse
11 \DeclareOption{defindex}{\def@indextrue}
12 \DeclareOption*{\PassOptionsToPackage{\CurrentOption}{omtext}}
13 \ProcessOptions
14 ⟨/package⟩
15 ⟨∗ltxml⟩
16 DefConditional('\if@defindex');
17 DefConditional('\if@statements');
18 DeclareOption('mh', sub {Digest(T_CS('\@statementstrue')); });
19 DeclareOption('defindex', sub{Digest(T_CS('\@defindextrue')); });
20 DeclareOption(undef,sub{PassOptions('omtext','sty',ToString(Digest(T_CS('\CurrentOption'))));
21                                      PassOptions('modules','sty',ToString(Digest(T_CS('\Curr
22 ProcessOptions();
23 ⟨/ltxml⟩
```

The next measure is to ensure that some STEX packages are loaded: `omdoc` for the statement keys, `modules` since we need module identifiers for referencing. Furthermore, we need the `ntheorem` package for presenting statements. For LATEXML, we also initialize the package inclusions, there we do not need `ntheorem`, since the XML does not do the presentation.

```
24 ⟨∗package⟩
25 \if@statements@mh@\RequirePackage{statements-mh}\fi
26 \RequirePackage{omtext}
27 \RequirePackage[base]{babel}
28 \RequirePackage[hyperref]{ntheorem}
29 \theoremstyle{plain}
30 ⟨/package⟩
31 ⟨∗ltxml⟩
32 if(IfCondition(T_CS('\if@statements'))) {RequirePackage('statements-mh');}
33 RequirePackage('omtext');
34 RequirePackage('modules');
35 ⟨/ltxml⟩
```

Now, we define an auxiliary function that lowercases strings

```
36 ⟨∗ltxml⟩
37 sub lowcase {my ($string) = @_; $string ? return lc(ToString($string)) : return('')}#$
38 sub dashed { join('-',map($_->toString,@_));}#$
39 ⟨/ltxml⟩
```

Sometimes it is necessary to fallback to symbol names in order to generate xml:id attributes. For this purpose, we define an auxiliary function which ensures the name receives a unique NCName equivalent.[4]

```
40 ⟨∗ltxml⟩
41 sub makeNCName {
```

EdN:4

---

[4]EDNOTE: Hard to be unique here, e.g. the names "foo_bar" and "foo bar" would receive the same xml:id attributes… of course we can devise a more complex scheme for the symbol replacement.

```
42   my ($name) = @_;
43   my $ncname=$name;
44   $ncname=~s/\s/_/g;   #Spaces to underscores
45   $ncname="_$ncname" if $ncname!~/^(\w|_)/;   #Ensure start with letter or underscore
46   ##More to come...
47   $ncname;
48 }
49 ⟨/ltxml⟩
```

The following functions are strictly utility functions that makes our life easier later on

```
50 ⟨∗ltxml⟩
51 sub simple_wrapper {
52   #Deref if array reference
53   my @input;
54   foreach (@_) {
55   if (ref $_ && $_ =~ /ARRAY/ && $_ !~ /LaTeXML/) {
56       @input=(@input,@$_);
57     } else
58       { push (@input,$_); }
59   }
60   return '' if (!@input);
61   @input = map(split(/\s*,\s*/,ToString($_)),@input);
62   my $output=join(" ",@input);
63   $output=~s/(^ )|[{}]//g; #remove leading space and list separator brackets
64   $output||'';
65 }
66 sub hash_wrapper{
67   #Deref if array reference
68   my @input;
69   foreach (@_) {
70   if (ref $_ && $_ =~ /ARRAY/ && $_ !~ /LaTeXML/) {
71       @input=(@input,@$_);
72     } else
73       { push (@input,$_); }
74   }
75   return '' if (!@input);
76   @input = sort map(split(/\s*,\s*/,ToString($_)),@input);
77   my $output=join(".sym #",@input);
78   $output=~s/(^\.sym )|[{}]//g; #remove leading space and list separator brackets
79   "#$output"||'';
80 }
81 ⟨/ltxml⟩
```

For the other languages, we set up triggers

```
82 ⟨∗package⟩
83 \AfterBabelLanguage{ngerman}{\input{statements-ngerman.ldf}}
84 ⟨/package⟩
```

5.2 Statements

```
85 ⟨*package⟩
86 \providecommand\STpresent[1]{#1}
87 ⟨/package⟩
```

\define@statement@env We define a meta-macro that allows us to define several variants of statements. Upon beginning this environment, we first set the KeyVal attributes, then we decide whether to print the discourse marker based on the value of the display key, then (given the right Options were set), we show the semantic annotations, and finally initialize the environment using the appropriate macro. Upon ending the environment, we just run the respective termination macro.

```
88 ⟨*package⟩
89 \def\define@statement@env#1{%
90 \newenvironment{#1}[1][]{\metasetkeys{omtext}{##1}\sref@target%
91 \@in@omtexttrue%
92 \ifx\omtext@display\st@flow\else%
93 \ifx\omtext@title\@empty\begin{ST#1Env}\else\begin{ST#1Env}[\omtext@title]\fi%
94 \ifx\sref@id\@empty\else\label{#1.\sref@id}\fi
95 \csname st@#1@initialize\endcsname\fi% display
96 \ifx\sref@id\@empty\sref@label@id{here}\else%
97 \sref@label@id{\STpresent{\csname ST#1EnvKeyword\endcsname}~\@currentlabel}\fi%
98 \ignorespaces}
99 {\csname st@#1@terminate\endcsname\ifx\omtext@display\st@flow\else\end{ST#1Env}\fi%
100 \omtext@post@skip\@in@omtextfalse}}
101 ⟨/package⟩
```

assertion

```
102 ⟨*package⟩
103 \newenvironment{assertion}[1][]{\metasetkeys{omtext}{#1}\sref@target%
104 \@in@omtexttrue%
105 \ifx\omtext@display\st@flow\itshape\noindent\ignorespaces%
106 \else% display!=flow
107 \ifx\omtext@title\@empty\begin{ST\omtext@type AssEnv}%
108 \else\begin{ST\omtext@type AssEnv}[\omtext@title]\fi\fi%
109 \ifx\omtext@type\@empty\sref@label@id{here}\else%
110 \sref@label@id{\STpresent{\csname ST\omtext@type AssEnvKeyword\endcsname}~\@currentlabel}
111 \fi}%display=flow
112 {\ifx\omtext@display\st@flow\else\end{ST\omtext@type AssEnv}\@in@omtextfalse\fi}
113 ⟨/package⟩
114 ⟨*ltxml⟩
115 DefStatement('{assertion} OptionalKeyVals:omtext',
116   "<omdoc:assertion "
117   .   "?&GetKeyVal(#1,'id')(xml:id='&GetKeyVal(#1,'id')')() "
118   .   "?&GetKeyVal(#1,'theory')(theory='&GetKeyVal(#1,'theory')')() "
119   .   "type='&lowcase(&GetKeyVal(#1,'type'))'>"
120   .   "?&GetKeyVal(#1,'title')(<dc:title>&GetKeyVal(#1,'title')</dc:title>)()"
121   .   "<omdoc:CMP>#body"
```

```
122    ."</omdoc:assertion>\n");
123 ⟨/ltxml⟩
```

\st@*@kw   We configure the default keywords for the various theorem environments.

```
124 ⟨*package⟩
125 \def\st@theorem@kw{Theorem}
126 \def\st@lemma@kw{Lemma}
127 \def\st@proposition@kw{Proposition}
128 \def\st@corollary@kw{Corollary}
129 \def\st@conjecture@kw{Conjecture}
130 \def\st@falseconjecture@kw{Conjecture (false)}
131 \def\st@postulate@kw{Postulate}
132 \def\st@obligation@kw{Obligation}
133 \def\st@assumption@kw{Assumption}
134 \def\st@rule@kw{Rule}
135 \def\st@observation@kw{Observation}
```

Then we configure the presentation of the theorem environments

```
136 \theorembodyfont{\itshape}
137 \theoremheaderfont{\normalfont\bfseries}
```

and then we finally define the theorem environments in terms of the statement keywords defined above. They are all numbered together with the section counter.

ST*AssEnv

```
138 \newtheorem{STtheoremAssEnv}{\st@theorem@kw}[section]
139 \newtheorem{STlemmaAssEnv}[STtheoremAssEnv]{\st@lemma@kw}
140 \newtheorem{STpropositionAssEnv}[STtheoremAssEnv]{\st@proposition@kw}
141 \newtheorem{STcorollaryAssEnv}[STtheoremAssEnv]{\st@corollary@kw}
142 \newtheorem{STconjectureAssEnv}[STtheoremAssEnv]{\st@conjecture@kw}
143 \newtheorem{STfalseconjectureAssEnv}[STtheoremAssEnv]{\st@falseconjecture@kw}
144 \newtheorem{STpostulateAssEnv}[STtheoremAssEnv]{\st@postulate@kw}
145 \newtheorem{STobligationAssEnv}[STtheoremAssEnv]{\st@obligation@kw}
146 \newtheorem{STassumptionAssEnv}[STtheoremAssEnv]{\st@assumption@kw}
147 \newtheorem{STobservationAssEnv}[STtheoremAssEnv]{\st@observation@kw}
148 \newtheorem{STruleAssEnv}[STtheoremAssEnv]{\st@rule@kw}
149 ⟨/package⟩
```

EdN:5   example   [5]

```
150 ⟨*package⟩
151 \let\usevocab=\usemodule
152 \def\st@example@initialize{}\def\st@example@terminate{}
153 \define@statement@env{example}
154 \def\st@example@kw{Example}
155 \theorembodyfont{\upshape}
156 \newtheorem{STexampleEnv}[STtheoremAssEnv]{\st@example@kw}
157 ⟨/package⟩
```

---

[5]EDNOTE: need to do something clever for the OMDoc representation of examples, in particular, the usevocab should only be defined in example

```
158 ⟨*ltxml⟩
159 DefMacro('\usevocab','\usemodule');
160 DefStatement('{example} OptionalKeyVals:omtext',
161         "<omdoc:example "
162       . "?&GetKeyVal(#1,'id')(xml:id='&GetKeyVal(#1,'id')')() "
163       . "?&GetKeyVal(#1,'for')(for='&hash_wrapper(&GetKeyVal(#1,'for'))')()>"
164       . "?&GetKeyVal(#1,'title')(<dc:title>&GetKeyVal(#1,'title')</dc:title>)()"
165      . "#body"
166     . "</omdoc:example>\n");
167 ⟨/ltxml⟩
```

axiom

```
168 ⟨*package⟩
169 \def\st@axiom@initialize{}\def\st@axiom@terminate{}
170 \define@statement@env{axiom}
171 \def\st@axiom@kw{Axiom}
172 \theorembodyfont{\upshape}
173 \newtheorem{STaxiomEnv}[STtheoremAssEnv]{\st@axiom@kw}
174 ⟨/package⟩
175 ⟨*ltxml⟩
176 DefStatement('{axiom} OptionalKeyVals:omtext',
177   "<omdoc:axiom "
178   .   "?&GetKeyVal(#1,'id')(xml:id='&GetKeyVal(#1,'id')')()>"
179   .   "?&GetKeyVal(#1,'title')(<dc:title>&GetKeyVal(#1,'title')</dc:title>)()"
180  . "<omdoc:CMP>#body"
181   . "</omdoc:axiom>\n");
182 ⟨/ltxml⟩
```

symboldec We use \symdef@type from the modules package as the visual cue.

```
183 ⟨*package⟩
184 \srefaddidkey{symboldec}
185 \addmetakey{symboldec}{functions}
186 \addmetakey{symboldec}{role}
187 \addmetakey*{symboldec}{title}
188 \addmetakey*{symboldec}{name}
189 \addmetakey{symboldec}{subject}
190 \addmetakey*{symboldec}{display}
191 \newenvironment{symboldec}[1][]{\metasetkeys{symboldec}{#1}\sref@target\st@indeftrue%
192 \ifx\symboldec@display\st@flow\else{\noindent\stDMemph{\symdef@type} \symboldec@name:}\fi%
193 \ifx\symboldec@title\@empty~\else~(\stDMemph{\symboldec@title})\par\fi}{}
194 ⟨/package⟩
195 ⟨*ltxml⟩
196 DefStatement('{symboldec} OptionalKeyVals:symboldec',
197         "<omdoc:symbol "
198       .   "?&GetKeyVal(#1,'id')(xml:id='&GetKeyVal(#1,'id')')"
199       .                   "(xml:id='&makeNCName(&GetKeyVal(#1,'name')).def.sym')"
200       .              "name='&GetKeyVal(#1,'name')'>"
201       .   "?&GetKeyVal(#1,'title')(<dc:title>&GetKeyVal(#1,'title')</dc:title>)()"
202       .   "<dc:description>#body"
203      ."</omdoc:symbol>\n");
```

5.2.1 Types

\symtype [6]

```
205 ⟨∗package⟩
206 \srefaddidkey{symtype}
207 \addmetakey*{symtype}{system}
208 \addmetakey*{symtype}{for}
209 \newcommand\type@type{Type}
210 \newcommand\symtype[3][]{\metasetkeys{symtype}{#1}\sref@target%
211 \noindent\type@type \ifx\symtype@\@empty\else (\symtype@system)\fi #2: $#3$}
212 ⟨/package⟩
213 ⟨∗ltxml⟩
214 DefConstructor('\symtype OptionalKeyVals:omtext {}{}',
215   "<omdoc:type for='#2'"
216 .    "?&GetKeyVal(#1,'id')(xml:id='&GetKeyVal(#1,'id').not')()"
217 .    "?&GetKeyVal(#1,'system')(xml:id='&GetKeyVal(#1,'system')')()>"
218 .    "<ltx:Math><ltx:XMath>#3</ltx:XMath></ltx:Math>"
219 ."</omdoc:type>");
220 ⟨/ltxml⟩
```

\inlinetypedec

```
221 ⟨∗package⟩
222 \newcommand\inlinetypedec[3][]{\metasetkeys{symtype}{#1}\sref@target{\def\thedectype{#2}#3}}
223 ⟨/package⟩
224 ⟨∗ltxml⟩
225 DefConstructor('\inlinetypedec OptionalKeyVals:omtext {}{}',
226   "<omdoc:type for='&GetKeyVal(#1,'for')'"
227 .    "?&GetKeyVal(#1,'id')(xml:id='&GetKeyVal(#1,'id').not')()"
228 .    "?&GetKeyVal(#1,'system')(xml:id='&GetKeyVal(#1,'system')')()>"
229 .    "<ltx:Math><ltx:XMath>#2</ltx:XMath></ltx:Math>"
230 .    "<omdoc:CMP>#body"
231 ."</omdoc:type>");
232 ⟨/ltxml⟩
```

typedec   We first define a theorem environment

```
233 ⟨∗package⟩
234 \def\st@typedec@kw{Type Declaration}
235 \theorembodyfont{\upshape}
236 \newtheorem{STtypedecEnv}[STtheoremAssEnv]{\st@typedec@kw}
```

 and then the environment itself.

```
237 \newenvironment{typedec}[2][]{\metasetkeys{omtext}{#1}\sref@target%
238 \def\thedectype{#2}%
239 \ifx\omtext@display\st@flow\else%
240 \ifx\omtext@title\@empty\begin{STtypedecEnv}\else\begin{STtypedecEnv}[\omtext@title]\fi%
241 \ifx\sref@id\@empty\else\label{typedec.\sref@id}\fi
```

---

[6]EdNote: MK@DG; the type element should percolate up.

```
242 \ifx\sref@id\@empty\sref@label@id{here}\else%
243 \sref@label@id{\STpresent{\csname STtypedecEnvKeyword\endcsname}~\@currentlabel}\fi%
244 \ignorespaces}
245 {\ifx\omtext@display\st@flow\else\end{STtypedecEnv}\fi\omtext@post@skip}
246 ⟨/package⟩
247 ⟨∗ltxml⟩
248 DefStatement('{typedec} OptionalKeyVals:omtext {}',
249    "<omdoc:type for='&GetKeyVal(#1,'for')'"
250    .    "?&GetKeyVal(#1,'id')(xml:id='&GetKeyVal(#1,'id').not')()"
251    .    "?&GetKeyVal(#1,'system')(xml:id='&GetKeyVal(#1,'system')')()>"
252    .    "?&GetKeyVal(#1,'title')(<dc:title>&GetKeyVal(#1,'title')</dc:title>)()"
253    .    "<ltx:Math><ltx:XMath>#2</ltx:XMath></ltx:Math>"
254    .    "<omdoc:CMP>#body"
255    ."</omdoc:type>");
256 ⟨/ltxml⟩
```

definition The `definition` environment itself is quite similar to the other's but we need to set the `\st@indef` switch to suppress warnings from `\st@def@target`.

```
257 ⟨∗package⟩
258 \newif\ifst@indef\st@indeffalse
259 \newenvironment{definition}[1][]{\metasetkeys{omtext}{#1}\sref@target\st@indeftrue%
260 \ifx\omtext@display\st@flow\else%
261 \ifx\omtext@title\@empty\begin{STdefinitionEnv}\else\begin{STdefinitionEnv}[\omtext@title]\fi\f
262 \ifx\sref@id\@empty\sref@label@id{here}\else%
263 \sref@label@id{\STpresent{\csname STdefinitionEnvKeyword\endcsname}~\@currentlabel}\fi%
264 \ignorespaces}
265 {\ifx\omtext@display\st@flow\else\end{STdefinitionEnv}\fi}
266 \def\st@definition@kw{Definition}
267 \theorembodyfont{\upshape}
268 \newtheorem{STdefinitionEnv}[STtheoremAssEnv]{\st@definition@kw}
269 ⟨/package⟩
270 ⟨∗ltxml⟩
271 sub definitionBody {
272     my ($doc, $keyvals, %props) = @_;
273     my $for = $keyvals->getValue('for') if $keyvals;
274     my $type = $keyvals->getValue('type') if $keyvals;
275     my %for_attr=();
276     if (ToString($for)) {
277       $for = ToString($for);
278       $for =~ s/^{(.+)}$/$1/eg;
279       foreach (split(/,\s*/,$for)) {
280         $for_attr{$_}=1;
281     }}
282     if ($props{theory}) {
283       my @symbols = @{$props{defs} || []};
284       my $signature = $props{signature};
285       foreach my $symb(@symbols) {
286         next if $for_attr{$symb};
287         my $qualified_symbol = $signature ? "$signature?$symb" : $symb;
288         $for_attr{$qualified_symbol}=1;
```

18

```
289        if (!$props{multiling}) {
290            $doc->insertElement('omdoc:symbol', undef, (name=>$symb, "xml:id"=>makeNCName("$symb.
291        }
292        }
293      my %attrs = ();
294      $for = join(" ",(sort keys %for_attr));
295      $attrs{'for'} = $for if $for;
296      my $id = $keyvals->getValue('id') if $keyvals;
297      $attrs{'xml:id'} = $id if $id;
298      $attrs{'type'} = $type if $type;
299      if ($props{theory}) {
300        $doc->openElement('omdoc:definition', %attrs);
301      } else {
302        $attrs{'type'}='definition';
303        $doc->openElement('omdoc:omtext', %attrs);
304      }
305      my $title = $keyvals->getValue('title') if $keyvals;
306      if ($title) {
307        $doc->openElement('omdoc:metadata');
308        $doc->openElement('dc:title');
309        $doc->absorb($title);
310        $doc->closeElement('dc:title');}
311      $doc->openElement('omdoc:CMP');
312      $doc->absorb($props{body}) if $props{body};
313      $doc->maybeCloseElement('omdoc:CMP');
314      if ($props{theory}) {
315        $doc->closeElement('omdoc:definition');
316      } else {
317        $doc->closeElement('omdoc:omtext');
318      }
319      return; }
320 # We use the standard DefEnvironment here, since
321 # afterDigestBegins would collide otherwise
322 DefEnvironment('{definition} OptionalKeyVals:omtext', \&definitionBody,
323    afterDigestBegin=>sub {
324      my ($stomach, $whatsit) = @_;
325      my @symbols = ();
326      $whatsit->setProperty(multiling=>LookupValue('multiling'));
327      $whatsit->setProperty(theory=>(LookupValue('modnl_signature') || LookupValue('current_modul
328      $whatsit->setProperty(defs=>\@symbols);
329      $whatsit->setProperty(signature=>LookupValue('modnl_signature'));
330      AssignValue('defs', \@symbols);
331      declareFunctions($stomach,$whatsit);
332      return; },
333    afterDigest => sub { AssignValue('defs', undef); return; });
334 ⟨/ltxml⟩%$
```

notation   We initialize the \def\st@notation@initialize{} here, and extend it with func-
            tionality below.

335 ⟨∗package⟩

```
336 \def\notemph#1{#1}
337 \def\st@notation@terminate{}
338 \def\st@notation@initialize{}
339 \define@statement@env{notation}
340 \def\st@notation@kw{Notation}
341 \theorembodyfont{\upshape}
342 \newtheorem{STnotationEnv}[STtheoremAssEnv]{\st@notation@kw}
343 ⟨/package⟩
344 ⟨∗ltxml⟩
345 DefStatement('{notation} OptionalKeyVals:omtext',
346   "<omdoc:definition "
347   .  "?&GetKeyVal(#1,'id')(xml:id='&GetKeyVal(#1,'id').not')()"
348   .  "?&GetKeyVal(#1,'for')(for='&simple_wrapper(&GetKeyVal(#1,'for'))')()>"
349   . "?&GetKeyVal(#1,'title')(<dc:title>&GetKeyVal(#1,'title')</dc:title>)()"
350   . "<omdoc:CMP>#body"
351   . "</omdoc:definition>\n");
352 DefConstructor('\notatiendum OptionalKeyVals:notation {}',
353               "<ltx:text class='notatiendum'>#2</ltx:text>");
354 ⟨/ltxml⟩
```

\st@def@target the next macro is a variant of the \sref@target macro provided by the sref
package specialized for the use in the \definiendum, \defi, \defii, and
\defiii macros. \st@def@target{⟨opt⟩}{⟨name⟩} makes a target with label
sref@⟨opt⟩@⟨modulename⟩@target, if ⟨opt⟩ is non-empty, else with the label
sref@⟨name⟩@⟨modulename⟩@target. Also it generates the necessary warnings
for a definiendum-like macro.

```
355 ⟨∗package⟩
356 \def\st@def@target#1#2{\def\@test{#1}%
357 \ifst@indef% if we are in a definition or such
358 \@ifundefined{mod@id}% if we are not in a module
359 {\PackageWarning{statements}{definiendum in unidentified module\MessageBreak
360 \protect\definiendum, \protect\defi,
361 \protect\defii, \protect\defiii\MessageBreak
362 can only be referenced when called in a module with id key}}%
363 {\edef\@@cd{\ifx\omtext@theory\@empty\mod@id\else\omtext@theory\fi}%
364 \edef\@@name{\ifx\@test\@empty{#2}\else{#1}\fi}%
365 \expandafter\sref@target@ifh{sref@\@@name @\@@cd @target}{}%
366 \ifmetakeys@showmeta\metakeys@show@keys{\@@cd}{name:\@@name}\fi}%
367 \else% st@indef
368 \PackageError{statements}%
369 {definiendum outside definition context\MessageBreak
370 \protect\definiendum, \protect\defi,
371 \protect\defii, \protect\defiii\MessageBreak
372 do not make sense semantically outside a definition.\MessageBreak
373 Consider wrapping the defining phrase in a \protect\inlinedef}%
374 \fi}
375 ⟨/package⟩
```

The \definiendum and \notatiendum macros are very simple.

\@termdef This macro is experimental, it is supposed to be invoked in `\definiendum` to define a macro with the definiendum text, so that can be re-used later in term assignments (see the `modules` package). But in the current context, where we rely on TeX groupings for visibility, this does not work, since the invocations of `\definiendum` are in `definition` environments and thus one group level too low. Keeping this for future reference.

```
376 ⟨∗package⟩
377 \newcommand\@termdef[2][]{\def\@test{#1}%
378 \@ifundefined{mod@id}{}{\ifx\@test\@empty\def\@@name{#2}\else\def\@@name{#1}\fi%
379 \termdef{\mod@id @\@@name}{#2}}}
380 ⟨/package⟩
```

\definiendum

```
381 ⟨∗package⟩
382 %\newcommand\definiendum[2][]{\st@def@target{#1}{#2}\@termdef[#1]{#2}\defemph{#2}}
383 \newcommand\definiendum[2][]{\st@def@target{#1}{#2}\defemph{#2}}
384 ⟨/package⟩
385 ⟨∗ltxml⟩
386 DefConstructor('\definiendum [] {}',
387       "<omdoc:term role='definiendum' name='#name' cd='#theory'>#2</omdoc:term>",
388       afterDigest => sub { defHelper(@_, 'definiendum'); });
389 ⟨/ltxml⟩
```

\notatiendum the `notatiendum` macro also needs to be visible in the `notation` and `definition` environments

```
390 ⟨∗package⟩
391 \newcommand\notatiendum[2][]{\notemph{#2}}
392 ⟨/package⟩
```

We expand the LaTeXML bindings for `\defi`, `\defii` and `\defiii` into two instances one will be used for the definition and the other for indexing.

\defi We split the `\defi` macro in two: `\defi` does the definiendum bit and `\@defi` handles the last optional argument and does the indexing. The information flow between them goes via the local `\@phrase` macro.

```
393 ⟨∗package⟩
394 \newcommand\defi[2][]{\st@def@target{#1}{#2}\defemph{#2}\def\@phrase{#2}\@defi}
395 \newcommand\@defi[1][]{\ifdef@index\omdoc@index[#1]{\@phrase}\fi\xspace}
396 \newcommand\defis[2][]{\st@def@target{#1}{#2}\defemph{#2s}\def\@phrase{#2}\@defi}
397 ⟨/package⟩
398 ⟨∗ltxml⟩
399 DefConstructor('\defi[]{} OptionalKeyVals:DEF',
400   "?#defindex(<omdoc:idx><omdoc:idt>)"
401   .      "<omdoc:term role='definiendum' name='?#1(#1)(#2)' cd='#theory'>#2</omdoc:term>"
402   ."?#defindex(</omdoc:idt><omdoc:ide index='default'><omdoc:idp>#2</omdoc:idp></omdoc:ide></omd
403   afterDigest => sub { defHelper(@_, 'defi'); },
404   alias=>'\defi');
405 DefConstructor('\defis[]{} OptionalKeyVals:DEF',
```

```
406  "?#defindex(<omdoc:idx><omdoc:idt>)"
407  .       "<omdoc:term role='definiendum' name='?#1(#1)(#2)' cd='#theory'>#2s</omdoc:term>"
408  ."?#defindex(</omdoc:idt><omdoc:ide index='default'><omdoc:idp>#2</omdoc:idp></omdoc:ide></omd
409  afterDigest => sub { defHelper(@_, 'defi'); },
410  alias=>'\defi');
411 ⟨/ltxml⟩
```

\adefi Again we split the \adefi macro into two parts: \adef does the definiendum bit
and \@adefi handles the last optional argument and does the indexing.

```
412 ⟨∗package⟩
413 \newcommand\adefi[3][]{\def\@name{#1}\def\@verb{#3}%
414 \st@def@target{#1}{#3}\defemph{#2}\@adefi}
415 \newcommand\@adefi[1][]{%
416 \ifdef@index%
417 \ifx\@name\@empty\omdoc@index[#1]{\@verb}%
418 \else\omdoc@index[at=\@name,#1]{\@verb}\fi%
419 \fi\xspace}
420 ⟨/package⟩
421 ⟨∗ltxml⟩
422  DefConstructor('\adefi[]{}{} OptionalKeyVals:DEF',
423  "?#defindex(<omdoc:idx><omdoc:idt>)"
424         .       "<omdoc:term role='definiendum' name='?#1(#1)(#3)' cd='#theory'>#2</omdoc:term>"
425         ."?#defindex(</omdoc:idt><omdoc:ide index='default'><omdoc:idp>#3</omdoc:idp></omdoc:ide
426         afterDigest => sub { defHelper(@_, 'adefi'); },
427         alias=>'\adefi');
428 ⟨/ltxml⟩
```

\defii

```
429 ⟨∗package⟩
430 \newcommand\@defii[1][]{\ifdef@index\@twin[#1]{\@pone}{\@ptwo}\fi\xspace}
431 \newcommand\defii[3][]{\def\@pone{#2}\def\@ptwo{#3}%
432 \st@def@target{#1}{#2-#3}\defemph{#2 #3}\@defii}
433 \newcommand\defiis[3][]{\def\@pone{#2}\def\@ptwo{#3}%
434 \st@def@target{#1}{#2-#3}\defemph{#2 #3s}\@defii}
435 ⟨/package⟩
436 ⟨∗ltxml⟩
437 DefConstructor('\defii[]{}{} OptionalKeyVals:DEF',
438  "?#defindex(<omdoc:idx><omdoc:idt>)"
439  .    "<omdoc:term role='definiendum' name='?#1(#1)(&dashed(#2,#3))' cd='#theory'>"
440  .       "#2 #3"
441  .    "</omdoc:term>"
442 ."?#defindex(</omdoc:idt><omdoc:ide index='default'><omdoc:idp>#2</omdoc:idp><omdoc:idp>#3</om
443         afterDigest => sub { defHelper(@_, 'defii'); },
444         alias=>'\defii');
445 DefConstructor('\defiis[]{}{} OptionalKeyVals:DEF',
446  "?#defindex(<omdoc:idx><omdoc:idt>)"
447  .    "<omdoc:term role='definiendum' name='?#1(#1)(&dashed(#2,#3))' cd='#theory'>"
448  .       "#2 #3s"
449  .    "</omdoc:term>"
450 ."?#defindex(</omdoc:idt><omdoc:ide index='default'><omdoc:idp>#2</omdoc:idp><omdoc:idp>#3</om
```

```
451        afterDigest => sub { defHelper(@_, 'defiii'); },
452        alias=>'\defiii');
453 ⟨/ltxml⟩
```

\adefii   analogous to \adefi

```
454 ⟨*package⟩
455 \newcommand\adefii[4][]{\def\@name{#1}\def\@pone{#3}\def\@ptwo{#4}%
456 \st@def@target{#1}{#3-#4}\defemph{#2}\@adefii}
457 \newcommand\@adefii[1][]{%
458 \ifdef@index%
459 \ifx\@name\@empty\@twin[#1]{\@pone}{\@ptwo}%
460 \else\@twin[at=\@name,#1]{\@pone}{\@ptwo}\fi%
461 \fi\xspace}
462 ⟨/package⟩
463 ⟨*ltxml⟩
464 DefConstructor('\adefii[]{}{}{} OptionalKeyVals:DEF',
465   "?#defindex(<omdoc:idx><omdoc:idt>)"
466         .   "<omdoc:term role='definiendum' name='?#1(#1)(&dashed(#3,#4))' cd='#theory'>"
467         .       "#2"
468         .       "</omdoc:term>"
469 ."?#defindex(</omdoc:idt><omdoc:ide index='default'><omdoc:idp>#3</omdoc:idp><omdoc:idp>#4</om
470        afterDigest => sub { defHelper(@_, 'adefii'); },
471        alias=>'\adefii');
472 ⟨/ltxml⟩
```

\defiii   similar to \defii

```
473 ⟨*package⟩
474 \newcommand\defiii[4][]{\def\@pone{#2}\def\@ptwo{#3}\def\@pthree{#4}%
475 \st@def@target{#1}{#2-#3-#4}\defemph{#2 #3 #4}\@defiii}
476 \newcommand\@defiii[1][]{\ifdef@index\@atwin[#1]{\@pone}{\@ptwo}{\@pthree}\fi\xspace}
477 \newcommand\defiiis[4][]{\def\@pone{#2}\def\@ptwo{#3}\def\@pthree{#4}%
478 \st@def@target{#1}{#2-#3-#4}\defemph{#2 #3 #4s}\@defiii}
479 ⟨/package⟩
480 ⟨*ltxml⟩
481 DefConstructor('\defiii[]{}{}{} OptionalKeyVals:DEF',
482   "?#defindex(<omdoc:idx><omdoc:idt>)"
483         . "<omdoc:term role='definiendum' cd='#theory' name='?#1(#1)(&dashed(#2,#3,#4))'>#2 #3
484 ."?#defindex(</omdoc:idt><omdoc:ide index='default'><omdoc:idp>#2</omdoc:idp><omdoc:idp>#3</om
485        afterDigest => sub { defHelper(@_, 'defiii'); },
486        alias=>'\defiii');
487 DefConstructor('\defiii[]{}{}{} OptionalKeyVals:DEF',
488   "?#defindex(<omdoc:idx><omdoc:idt>)"
489         . "<omdoc:term role='definiendum' cd='#theory' name='?#1(#1)(&dashed(#2,#3,#4))'>#2 #3
490 ."?#defindex(</omdoc:idt><omdoc:ide index='default'><omdoc:idp>#2</omdoc:idp><omdoc:idp>#3</om
491        afterDigest => sub { defHelper(@_, 'defiii'); },
492        alias=>'\defiii');
493 ⟨/ltxml⟩
```

\adefiii

```
494 ⟨∗package⟩
495 \newcommand\adefiii[5][]{\def\@name{#1}\def\@pone{#3}\def\@ptwo{#4}\def\@pthree{#3}%
496 \st@def@target{#1}{#3-#4-#5}\defemph{#2}\@adefiii}
497 \newcommand\@adefiii[1][]{%
498 \ifdef@index%
499 \ifx\@name\@empty\@atwin[#1]{\@pone}{\@ptwo}{\@pthree}%
500 \else\@atwin[at=\@name,#1]{\@pone}{\@ptwo}{\@pthree}\fi%
501 \fi\xspace}
502 ⟨/package⟩
503 ⟨∗ltxml⟩
504 DefConstructor('\adefiii[]{}{}{}{} OptionalKeyVals:DEF',
505   "?#defindex(<omdoc:idx><omdoc:idt>)"
506   . "<omdoc:term role='definiendum' cd='#theory' name='?#1(#1)(&dashed(#3,#4,#5))'>#2</omdoc:ter
507   ."?#defindex(</omdoc:idt><omdoc:ide index='default'><omdoc:idp>#3</omdoc:idp><omdoc:idp>#4</om
508        afterDigest => sub { defHelper(@_, 'adefiii')},
509        alias=>'\adefiii');
510 ⟨/ltxml⟩
```

\inlineex

```
511 ⟨∗package⟩
512 \newcommand\inlineex[2][]{\metasetkeys{omtext}{#1}%
513 \sref@target\sref@label@id{here}#2}
514 ⟨/package⟩
515 ⟨∗ltxml⟩
516 DefConstructor('\inlineex OptionalKeyVals:omtext {}',
517                "<ltx:text class='example'>#2</ltx:text>");
518 ⟨/ltxml⟩
```

\inlineass

```
519 ⟨∗package⟩
520 \newcommand\inlineass[2][]{\metasetkeys{omtext}{#1}%
521 \sref@target\sref@label@id{here}#2}
522 ⟨/package⟩
523 ⟨∗ltxml⟩
524 DefConstructor('\inlineass OptionalKeyVals:omtext {}',
525                "<ltx:text "
526                .  "?&GetKeyVal(#1,'type')(class='&GetKeyVal(#1,'type') assertion')(class='assert
527                .  "#2"
528                . "</ltx:text>");
529 ⟨/ltxml⟩
```

\inlinedef

```
530 ⟨∗package⟩
531 \newcommand\inlinedef[2][]{\metasetkeys{omtext}{#1}%
532 \if@in@omtext\else% we are not in an omtext or statement
533 \PackageError{modules}{\protect\inlinedef\space outside a statement!}%
534 {Try wrapping the paragraph in a\MessageBreak
535 \protect\begin{omtext}, \protect\begin{assertion}, \protect\begin{axiom}, ... \MessageBreak
536 whatever is suitable semantically}\fi%
```

```
537 \sref@target\sref@label@id{here}\st@indeftrue #2}
538 ⟨/package⟩
539 ⟨∗ltxml⟩
540 DefConstructor('\inlinedef OptionalKeyVals:omtext {}', sub {
541 my ($document, $keyvals, $body, %props) = @_;
542 my $for = $keyvals->getValue('for') if $keyvals;
543 my %for_attr=();
544 if (ToString($for)) {
545   $for = ToString($for);
546   $for =~ s/^{(.+)}$/$1/eg;
547   foreach (split(/,\s*/,$for)) {
548     $for_attr{$_}=1;
549   }}
550 my @symbols = @{$props{defs} || []};
551 #Prepare for symbol insertion -insert before the parent of the closest ancestor CMP element
552 my $original_node = $document->getNode;
553 my $statement_ancestor = $document->findnode('./ancestor::omdoc:CMP/..', $original_node);
554 foreach my $symb(@symbols) {
555   next if $for_attr{$symb};
556   $for_attr{$symb}=1;
557   my $symbolnode = XML::LibXML::Element->new('symbol');
558   $symbolnode->setAttribute(name=>$symb);
559   $symbolnode->setAttribute("xml:id"=>makeNCName("$symb.def.sym"));
560 if ($statement_ancestor) {
561   $statement_ancestor->parentNode->insertBefore($symbolnode,$statement_ancestor);
562  } else {
563    Error('malformed', $statement_ancestor, $original_node, "\\inlinedef outside a statement!
564 Try wrapping the paragraph in a begin{omtext}, \\begin{assertion}, \\begin{axiom}...\nwhatever
565 #Restore the insertion point
566 $document->setNode($original_node);
567 my %attrs = ();
568 $for = join(" ",(sort keys %for_attr));
569 $attrs{'for'} = $for if $for;
570 my $id = $keyvals->getValue('id') if $keyvals;
571 $attrs{'xml:id'} = $id if $id;
572 $attrs{'class'} = 'inlinedef';
573 $document->openElement('ltx:text',%attrs);
574 $document->absorb($body);
575 $document->closeElement('ltx:text'); },
576 #Prepare 'defs' hooks for \defi and \definiendum symbol names
577  beforeDigest=>sub {
578    my @symbols = ();
579    AssignValue('defs', \@symbols); return; },
580 #Adopt collected names as 'defs' property, remove hooks
581  afterDigest=>sub {
582    my ($stomach, $whatsit) = @_;
583    my $defsref = LookupValue('defs');
584    my @defs = @$defsref;
585    $whatsit->setProperty('defs',\@defs);
586    AssignValue('defs',undef);
```

```
587  return; });#$
```
588 ⟨/ltxml⟩

5.3   Cross-Referencing Symbols and Concepts

\termref   We delegate to the worker macro `\st@termref` after setting the default for the `cd` key.

589 ⟨∗package⟩
590 `\addmetakey*{termref}{cd}`
591 `\addmetakey*{termref}{cdbase}`
592 `\addmetakey*{termref}{name}`
593 `\addmetakey*{termref}{role}`
594 `\newcommand\termref[2][]{\metasetkeys{termref}{#1}%`
595 `\ifx\termref@cd\@empty\def\termref@cd{\mod@id}\fi%`
596 `\st@termref{#2}}`
597 ⟨/package⟩
598 ⟨∗ltxml⟩
599 `DefConstructor('\termref OptionalKeyVals:termref {}',`
600 `              "<omdoc:term "`
601 `            . "?&GetKeyVal(#1,'cdbase')(cdbase='&GetKeyVal(#1,'cdbase')')()" `
602 `            . "cd='?&GetKeyVal(#1,'cd')(&GetKeyVal(#1,'cd'))(#module)' "`
603 `            . "name='&GetKeyVal(#1,'name')'>"`
604 `            . "#2"`
605 `            ."</omdoc:term>",`
606 `              afterDigest=>sub{$_[1]->setProperty(module=>(LookupValue('modnl_signature') || Lo`
607 ⟨/ltxml⟩%$

The next macro is where the actual work is done.

\st@termref   If the `cdbase` is given, then we make a hyper-reference, otherwise we punt to `\mod@termref`, which can deal with the case where the cdbase is given by the imported cd.

608 ⟨∗package⟩
609 `\newcommand\st@termref[1]{\ifx\termref@name\@empty\def\termref@name{#1}\fi%`
610 `\ifx\termref@cdbase\@empty\mod@termref\termref@cd\termref@name{#1}%`
611 `\else\sref@href@ifh\termref@cdbase{#1}\fi}`
612 ⟨/package⟩

\tref*

613 ⟨ltxml⟩`RawTeX('`
614 ⟨∗package | ltxml⟩
615 `\newcommand\atrefi[3][]{\def\@test{#1}%`
616 `\ifx\@test\@empty\termref[name=#3]{#2}\else\termref[cd=#1,name=#3]{#2}\fi}`
617 `\newcommand\atrefii[4][]{\atrefi[#1]{#2}{#3-#4}}`
618 `\newcommand\atrefiii[5][]{\atrefi[#1]{#2}{#3-#4-#5}}`

\tref*

619 `\newcommand\trefi[2][]{\atrefi[#1]{#2}{#2}}`
620 `\newcommand\trefii[3][]{\atrefi[#1]{#2 #3}{#2-#3}}`

```
621 \newcommand\trefiii[4][]{\atrefi[#1]{#2 #3 #4}{#2-#3-#4}}
622 \newcommand\trefis[2][]{\atrefi[#1]{#2s}{#2}}
623 \newcommand\trefiis[3][]{\atrefi[#1]{#2 #3s}{#2-#3}}
624 \newcommand\trefiiis[4][]{\atrefi[#1]{#2 #3 #4s}{#2-#3-#4}}
```
625 ⟨/package | ltxml⟩
626 ⟨ltxml⟩');

Now we care about the configuration switches, they are set to sensible values, if they are not defined already. These are just configuration parameters, which should not appear in documents, therefore we do not provide LATEXML bindings for them.

*emph

627 ⟨∗package⟩
```
628 \providecommand{\termemph}[1]{#1}
629 \providecommand{\defemph}[1]{{\textbf{#1}}}
630 \providecommand{\stDMemph}[1]{{\textbf{#1}}}
```
631 ⟨/package⟩

\term The \term macro is used for wiki-style dangling links with editor support.[7]

632 ⟨∗package⟩
```
633 \newcommand\term[2][]{\def\@test{#1}%
634 \ifx\@test\@empty\else
635 \@ifundefined{module@defs@#1}{\PackageWarning{statements}%
636 {{\protect\term} specifies module #1 which is not in
637   scope\MessageBreak import it via e.g. via \protect\importmhmodule}}{}
638 \fi%
639 \PackageWarning{statements}%
640 {Dangling link (\protect\term) for "#2" still needs to be specified}%
641 \textcolor{blue}{\underline{#2}}}
```
642 ⟨/package⟩
643 ⟨∗ltxml⟩
644 DefConstructor('\term{}',"<omdoc:term class='dangling-term-link' ?#1(cd='#1')()>#1</omdoc:term>
645 ⟨/ltxml⟩

\symref The \symref macros is quite simple, since we have done all the heavy lifting in the modules package: we simply apply \mod@symref@⟨arg1⟩ to ⟨arg2⟩.

646 ⟨∗package⟩
```
647 \newcommand\symref[2]{\@nameuse{mod@symref@#1}{#2}}
```
648 ⟨/package⟩
649 ⟨∗ltxml⟩
```
650 DefConstructor('\symref{}{}',
651                "<omdoc:term cd='&LookupValue('symdef.#1.cd')' name='&LookupValue('symdef.#1.nam
652                . "#2"
653                ."</omdoc:term>");
```
654 ⟨/ltxml⟩

---
[7]EDNOTE: MK: document above

27

5.4 Providing IDs for OMDoc Elements

To provide default identifiers, we tag all OMDoc elements that allow `xml:id` attributes by executing the `numberIt` procedure from `omdoc.sty.ltxml`.

```
655 ⟨∗ltxml⟩
656 Tag('omdoc:assertion',afterOpen=>\&numberIt,afterClose=>\&locateIt);
657 Tag('omdoc:definition',afterOpen=>\&numberIt,afterClose=>\&locateIt);
658 Tag('omdoc:example',afterOpen=>\&numberIt,afterClose=>\&locateIt);
659 Tag('omdoc:requation',afterOpen=>\&numberIt,afterClose=>\&locateIt);
660 Tag('omdoc:axiom',afterOpen=>\&numberIt,afterClose=>\&locateIt);
661 Tag('omdoc:symbol',afterOpen=>\&numberIt,afterClose=>\&locateIt);
662 Tag('omdoc:type',afterOpen=>\&numberIt,afterClose=>\&locateIt);
663 Tag('omdoc:term',afterOpen=>\&numberIt,afterClose=>\&locateIt);
664 ⟨/ltxml⟩
```

5.5 Auxiliary Functionality

```
665 ⟨∗ltxml⟩
666 # ========================================================
667 #  Auxiliary Functions:                                   #
668 # ========================================================
669 sub DefStatement {
670   my ($definition,$replacement,%properties)=@_;
671   DefEnvironment($definition,$replacement,%properties,
672       afterDigestBegin=>\&declareFunctions,
673 );}
674
675 sub declareFunctions{
676   my ($stomach,$whatsit) = @_;
677   my $keyval = $whatsit->getArg(1);
678   my $funval = GetKeyVal($keyval,'functions') if GetKeyVal($keyval,'functions');
679   return unless $funval;
680   my @funsymbs = $funval->unlist;
681   #Unread the function declarations at the Gullet
682   foreach (@funsymbs) {
683     my $symb = UnTeX($_);
684     $stomach->getGullet->unread(Tokenize('\lxDeclare[role=FUNCTION]{$'.$symb.'$}')->unlist);
685   }
686   return; }
687 sub defHelper{
688   my ($stomach, $whatsit, $defOption) = @_;
689   my $addr = LookupValue('defs');
690   my $name = $whatsit->getArg(1);
691   $name = $name->toString if $name;
692   my %choose_Option = (
693                        definiendum => sub {$whatsit->getArg(2);},
694                        defi => sub {$whatsit->getArg(2);},
695                        adefi => sub {$whatsit->getArg(3);},
696                        defii => sub {$whatsit->getArg(2)->toString.'-'.$whatsit->getArg(3)->toSt
```

```
697                              adefii => sub {$whatsit->getArg(3)->toString.'-'.$whatsit->getArg(4)->toS
698                              defiii => sub {$whatsit->getArg(2)->toString.'-'.$whatsit->getArg(3)->toS
699                              adefiii => sub {$whatsit->getArg(3)->toString.'-'.$whatsit->getArg(4)->to
700                              );
701   $name = $name || $choose_Option{$defOption}->();
702   $whatsit->setProperty(name=>$name->toString) if ref($name);
703   push(@$addr, $name) if ($addr and $name);
704   $whatsit->setProperty('defindex', IfCondition(T_CS('\if@defindex')));
705   $whatsit->setProperty(theory=>(LookupValue('modnl_signature') || LookupValue('current_module'
706  return;}#$
707 ⟨/ltxml⟩
```

5.6   Deprecated Functionality

In this section we centralize old interfaces that are only partially supported any
more.

def

```
708 ⟨ltxml⟩####### Deprecated functionality:
709 ⟨ltxml⟩RawTeX('
710 ⟨∗package | ltxml⟩
711 \newcommand\defin[2][]{\defi[#1]{#2}%
712 \PackageWarning{statements}{\protect\defin\space is deprecated, use \protect\defi\space instead
713 \newcommand\twindef[3][]{\defii[#1]{#2}{#3}%
714 \PackageWarning{statements}{\protect\twindef\space is deprecated, use \protect\defii\space inst
715 \newcommand\atwindef[4][]{\defiii[#1]{#2}{#3}{#4}%
716 \PackageWarning{statements}{\protect\atwindef\space is deprecated, use \protect\defiii\space in
717 \newcommand\definalt[3][]{\adefi[#1]{#2}{#3}%
718 \PackageWarning{statements}{\protect\definalt\space is deprecated, use \protect\adefi\space ins
719 \newcommand\twindefalt[4][]{\adefii[#1]{#2}{#3}{#4}%
720 \PackageWarning{statements}{\protect\twindefalt\space is deprecated, use \protect\adefii\space
721 \newcommand\atwindefalt[5][]{\adefiii[#1]{#2}{#3}{#4}{#5}%
722 \PackageWarning{statements}{\protect\atwindefalt\space is deprecated, use \protect\adefiii\spac
```

def

```
723 \newcommand\twinref[3][]{\trefii[#1]{#2}{#3}%
724 \PackageWarning{statements}{\protect\twinref\space is deprecated, use \protect\trefii\space ins
725 \newcommand\atwinref[4][]{\atrefiii[#1]{#2}{#3}{#4}%
726 \PackageWarning{statements}{\protect\atwindef\space is deprecated, use \protect\trefiii\space i
727 ⟨/package | ltxml⟩
728 ⟨ltxml⟩');
```

5.7   Finale

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

```
729 ⟨ltxml⟩1;
```

References

[KG15]   Michael Kohlhase and Deyan Ginev. *smultiling.sty: Multilinguality Support for sTeX*. Tech. rep. 2015. URL: https://github.com/KWARC/sTeX/raw/master/sty/smultiling/smultiling.pdf.

[KGA15]   Michael Kohlhase, Deyan Ginev, and Rares Ambrus. *modules.sty: Semantic Macros and Module Scoping in sTeX*. Tech. rep. Comprehensive TeX Archive Network (CTAN), 2015. URL: http://www.ctan.org/get/macros/latex/contrib/stex/modules/modules.pdf.

[Koh06]   Michael Kohlhase. OMDoc – *An open markup format for mathematical documents [Version 1.2]*. LNAI 4180. Springer Verlag, Aug. 2006. URL: http://omdoc.org/pubs/omdoc1.2.pdf.

[Koh08]   Michael Kohlhase. "Using LaTeX as a Semantic Markup Format". In: *Mathematics in Computer Science* 2.2 (2008), pp. 279–304. URL: https://svn.kwarc.info/repos/stex/doc/mcs08/stex.pdf.

[Koh15a]   Michael Kohlhase. *metakeys.sty: A generic framework for extensible Metadata in LaTeX*. Tech. rep. Comprehensive TeX Archive Network (CTAN), 2015. URL: http://www.ctan.org/tex-archive/macros/latex/contrib/stex/metakeys/metakeys.pdf.

[Koh15b]   Michael Kohlhase. *omdoc.sty/cls: Semantic Markup for Open Mathematical Documents in LaTeX*. Tech. rep. Comprehensive TeX Archive Network (CTAN), 2015. URL: http://www.ctan.org/tex-archive/macros/latex/contrib/stex/omdoc/omdoc.pdf.

[Koh15c]   Michael Kohlhase. *sref.sty: Semantic Crossreferencing in LaTeX*. Tech. rep. Comprehensive TeX Archive Network (CTAN), 2015. URL: http://www.ctan.org/tex-archive/macros/latex/contrib/stex/sref/sref.pdf.

[MS]   Wolfgang May and Andreas Schedler. *An Extension of the LATEX-Theorem Evironment*. Self-documenting LaTeX package. URL: http://dante.ctan.org/tex-archive/macros/latex/contrib/ntheorem/ntheorem.pdf (visited on 01/11/2010).

[sTeX]   *KWARC/sTeX*. URL: https://svn.kwarc.info/repos/stex (visited on 05/15/2015).