

Semantic Markup for Mathematical Statements^{*}

Michael Kohlhasse
Jacobs University, Bremen
<http://kwarc.info/kohlhasse>

November 10, 2015

Abstract

The `statements` package is part of the \LaTeX collection, a version of $\text{\TeX}/\text{\LaTeX}$ that allows to markup $\text{\TeX}/\text{\LaTeX}$ documents semantically without leaving the document format, essentially turning $\text{\TeX}/\text{\LaTeX}$ into a document format for mathematical knowledge management (MKM).

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

Contents

1	Introduction	2
2	The User Interface	2
2.1	Package Options	2
2.2	Statements	2
2.3	Cross-Referencing Symbols and Concepts	9
3	Configuration of the Presentation	10
4	Limitations	11
5	The Implementation	11
5.1	Package Options	11
5.2	Statements	13
5.3	Cross-Referencing Symbols and Concepts	26
5.4	Providing IDs for OMDoc Elements	27
5.5	Auxiliary Functionality	28
5.6	Deprecated Functionality	29
5.7	Finale	29

^{*}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 $\text{\S}\text{\TeX}$ sources, or after translation. Even though it is part of the $\text{\S}\text{\TeX}$ collection, it can be used independently, like its sister package `sproofs`.

$\text{\S}\text{\TeX}$ [`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 $\text{\S}\text{\TeX}$. If this is set, then definienda are automatically passed into the index of the document. Furthermore, 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=` 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 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 assertion 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. \TeX 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:

```
\symtype[\langle keys \rangle]{\langle sym \rangle}{\langle type \rangle}
```

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 $\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ (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` package supplies the `typedec` environment and the `\inlinetypedec` macro. Both 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

`typedec`
`\inlinetypedec`

`\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 concepts or objects. These are identified by the `\definiendum` macro, which is used

`\definiendum`

```

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

```

will lead to the result

Symbol 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.

Symbol 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.

Symbol succ: (The Natural Numbers)

The natural numbers inductively defined via the Peano Axioms.

Axiom 2.2 (P1) 0 is a natural number.

...

Axiom 2.6 (P5) Any property P such $P(0)$ and $P(\succ k)$ whenever $P(k)$ holds for all n in \mathbb{N}

Example 3: Semantic Markup for the Peano Axioms

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

Definition 2.7 1 is the successor of 0 (formally: $1: =\succ 0$)

Example 4: A Definition based on Figure 3

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

```
\defi[⟨name⟩]{⟨word⟩}[⟨indexkeys⟩]
```

to markup a `\definiendum` $\langle word \rangle$ with system name $\langle name \rangle$ that appear in the index (where $\langle indexkeys \rangle$ 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 compounds. Note that if the `\definiendum` contains semantic macros, then we need to
`\defiii` specify the `loadmodules` key and also protect the semantic macro. For instance if `\eset` is the semantic macro for \emptyset , 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. 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
<code>\defi{concept}</code>		
concept	concept	concept
<code>\defi[csymbol]{concept}</code>		
csymbol	concept	concept
<code>\adefi[csymbol]{concepts}{concept}</code>		
csymbol	concepts	concept
<code>\defii{concept}{group}</code>		
concept-group	concept group	concept group, group - , concept
<code>\adefii{small}{concept}{group}</code>		
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 phrases¹. Therefore, the `\termref` can be used to make this information explicit. It takes the keys

`\termref`

`cdbase` to specify a URI (a path actually, since L^AT_EX 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 by `\trefi{<name>}`. Here the link text is just `<name>`. Concepts defined via `\defii{<first>}{<second>}` can be referenced by `\trefii{<first>}{<second>}` (with link text “`<first> <second>`”) and analogously for `\defiii` and `\trefiii`.

`\trefi`

`\trefii`

`\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`

¹We 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 `statements` package supplies the `\symref` macro. Like `\termref`, and invocation of `\symref{<csseq>}{<text>}` will just typeset `<text>` with a hyperlink to the relevant definition (i.e. the one that has the declaration `for=<csseq>` in the metadata argument.)

`\symref`

`\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 with `\definiendum` macro and appropriate metadata).¹

EdN: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 `\termemph` 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 preset this in the class file.²

EdN: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 be achieved by redefining the `\STpresent` macro, which is applied to the keyword of the `ST*Env` theorem environments.³

`\STpresent`

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.

EdN:3

¹EdNOTE: MK: we probably need multi-part variants for `*tref*`

²EdNOTE: function declarations

³EdNOTE: 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 \LaTeX 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 `\endpackage`) and the \LaTeX ML bindings (between `\ltxml` and `\endltxml`). 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 \LaTeX ML:

```

1 \ltxml
2 # -*- PERL -*-
3 package LaTeXML::Package::Pool;
4 use strict;
5 use LaTeXML::Package;
6 \endltxml

```

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\ifdef@index\def@indexfalse
9 \DeclareOption{defindex}{\def@indextrue}
10 \DeclareOption*{\PassOptionsToPackage{\CurrentOption}{omtext}}
11 \ProcessOptions
12 </package>
13 <*ltxml>
14 DefConditional('\if@defindex');
15 DeclareOption('defindex', sub{Digest(T_CS('\@defindextrue'))});
16 DeclareOption(undef, sub{PassOptions('omtext', 'sty', ToString(Digest(T_CS('\CurrentOption'))));
17                                     PassOptions('modules', 'sty', ToString(Digest(T_CS('\Curr
18 ProcessOptions();
19 </ltxml>

```

The next measure is to ensure that some \TeX 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 \LaTeX XML, we also initialize the package inclusions, there we do not need `ntheorem`, since the XML does not do the presentation.

```

20 <*package>
21 \RequirePackage{omtext}
22 \RequirePackage[base]{babel}
23 \RequirePackage[hyperref]{ntheorem}
24 \theoremstyle{plain}
25 </package>
26 <*ltxml>
27 RequirePackage('omtext');
28 RequirePackage('modules');
29 </ltxml>

```

Now, we define an auxiliary function that lowercases strings

```

30 <*ltxml>
31 sub lowercase {my ($string) = @_; $string ? return lc(ToString($string)) : return('')}##$
32 sub dashed { join('-', map($_->toString, @_));}##$
33 </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

```

34 <*ltxml>
35 sub makeNCName {
36   my ($name) = @_;
37   my $ncname=$name;
38   $ncname=~s/\s/_/g; #Spaces to underscores
39   $ncname="_$ncname" if $ncname!~/^\w|_/; #Ensure start with letter or underscore
40   ##More to come...
41   $ncname;

```

⁴EDNOTE: 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 }
43 </ltxml>
The following functions are strictly utility functions that makes our life easier later
on
44 <*ltxml>
45 sub simple_wrapper {
46   #Deref if array reference
47   my @input;
48   foreach (@_) {
49     if (ref $_ && $_ =~ /ARRAY/ && $_ !~ /LaTeXML/) {
50       @input=(@input,@$_);
51     } else
52       { push (@input,$_); }
53   }
54   return '' if (!@input);
55   @input = map(split(/\s*,\s*/,ToString($_)),@input);
56   my $output=join(" ",@input);
57   $output=~s/(\^ )|[\{\}]/g; #remove leading space and list separator brackets
58   $output||'';
59 }
60 sub hash_wrapper{
61   #Deref if array reference
62   my @input;
63   foreach (@_) {
64     if (ref $_ && $_ =~ /ARRAY/ && $_ !~ /LaTeXML/) {
65       @input=(@input,@$_);
66     } else
67       { push (@input,$_); }
68   }
69   return '' if (!@input);
70   @input = sort map(split(/\s*,\s*/,ToString($_)),@input);
71   my $output=join(".sym #",@input);
72   $output=~s/(\^\.sym )|[\{\}]/g; #remove leading space and list separator brackets
73   "$#$output"||'';
74 }
75 </ltxml>

```

For the other languages, we set up triggers

```

76 <*package>
77 \AfterBabelLanguage{ngerman}{\input{statements-ngerman.ldf}}
78 </package>

```

5.2 Statements

\STpresent

```

79 <*package>
80 \providecommand\STpresent[1]{#1}
81 </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.

82 \package
83 \def\define@statement@env#1{%
84 \newenvironment{#1}[1][\metasetkeys{omtext}{#1}\sref@target%
85 \in@omtexttrue%
86 \ifx\omtext@display\st@flow\else%
87 \ifx\omtext@title\@empty\begin{ST#1Env}\else\begin{ST#1Env}[\omtext@title]\fi%
88 \ifx\sref@id\@empty\else\label{#1.\sref@id}\fi
89 \csname st@#1@initialize\endcsname\fi% display
90 \ifx\sref@id\@empty\sref@label@id{here}\else%
91 \sref@label@id{\STpresent{\csname ST#1EnvKeyword\endcsname}~\@currentlabel}\fi%
92 \ignorespaces}
93 {\csname st@#1@terminate\endcsname\ifx\omtext@display\st@flow\else\end{ST#1Env}\fi%
94 \omtext@post@skip\in@omtextfalse}}
95 \package

assertion

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

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

118 \package
119 \def\st@theorem@kw{Theorem}
120 \def\st@lemma@kw{Lemma}

```

```

121 \def\st@proposition@kw{Proposition}
122 \def\st@corollary@kw{Corollary}
123 \def\st@conjecture@kw{Conjecture}
124 \def\st@falseconjecture@kw{Conjecture (false)}
125 \def\st@postulate@kw{Postulate}
126 \def\st@obligation@kw{Obligation}
127 \def\st@assumption@kw{Assumption}
128 \def\st@rule@kw{Rule}
129 \def\st@observation@kw{Observation}

```

Then we configure the presentation of the theorem environments

```

130 \theorembodyfont{\itshape}
131 \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

```

132 \newtheorem{STtheoremAssEnv}{\st@theorem@kw}[section]
133 \newtheorem{STlemmaAssEnv}[STtheoremAssEnv]{\st@lemma@kw}
134 \newtheorem{STpropositionAssEnv}[STtheoremAssEnv]{\st@proposition@kw}
135 \newtheorem{STcorollaryAssEnv}[STtheoremAssEnv]{\st@corollary@kw}
136 \newtheorem{STconjectureAssEnv}[STtheoremAssEnv]{\st@conjecture@kw}
137 \newtheorem{STfalseconjectureAssEnv}[STtheoremAssEnv]{\st@falseconjecture@kw}
138 \newtheorem{STpostulateAssEnv}[STtheoremAssEnv]{\st@postulate@kw}
139 \newtheorem{STobligationAssEnv}[STtheoremAssEnv]{\st@obligation@kw}
140 \newtheorem{STassumptionAssEnv}[STtheoremAssEnv]{\st@assumption@kw}
141 \newtheorem{STobservationAssEnv}[STtheoremAssEnv]{\st@observation@kw}
142 \newtheorem{STruleAssEnv}[STtheoremAssEnv]{\st@rule@kw}
143 \end{package}

```

EdN:5

example

5

```

144 \begin{package}
145 \let\usevocab=\usemodule
146 \def\st@example@initialize{}\def\st@example@terminate{}
147 \define@statement@env{example}
148 \def\st@example@kw{Example}
149 \theorembodyfont{\upshape}
150 \newtheorem{STexampleEnv}[STtheoremAssEnv]{\st@example@kw}
151 \end{package}
152 \begin{xml}
153 DefMacro('usevocab', 'usemodule');
154 DefStatement('{example} OptionalKeyVals:omtext',
155             "<omdoc:example "
156             . "&GetKeyVal(#1,'id')(xml:id='&GetKeyVal(#1,'id'))()"
157             . "&GetKeyVal(#1,'for')(for='&hash_wrapper(&GetKeyVal(#1,'for'))')>"
158             . "&GetKeyVal(#1,'title')(<dc:title>&GetKeyVal(#1,'title')</dc:title>)"
159             . "#body"

```

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

```

160 . "</omdoc:example>\n");
161 </ltxml>

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

```

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

```

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

```

5.2.1 Types

EdN:6 `\symtype` ⁶

⁶EdNOTE: MK@DG; the type element should percolate up.


```

199 <*package>
200 \srefaddidkey{syntype}
201 \addmetakey*{syntype}{system}
202 \addmetakey*{syntype}{for}
203 \newcommand\type@type{Type}
204 \newcommand\syntype[3] [] {\metasetkeys{syntype}{#1}\sref@target%
205 \noindent\type@type \ifx\syntype@{}empty\else (\syntype@system)\fi #2: $#3$}
206 </package>
207 <*txml>
208 DefConstructor(' \syntype OptionalKeyVals:omtext {}{}',
209 " <omdoc:type for=' #2' "
210 . " ?&GetKeyVal(#1,'id')(xml:id='&GetKeyVal(#1,'id').not')() "
211 . " ?&GetKeyVal(#1,'system')(xml:id='&GetKeyVal(#1,'system'))()>"
212 . " <ltx:Math><ltx:XMath>#3</ltx:XMath></ltx:Math>"
213 . " </omdoc:type>");
214 </txml>

```

\inlinetypedec

```

215 <*package>
216 \newcommand\inlinetypedec[3] [] {\metasetkeys{syntype}{#1}\sref@target{\def\thedectype{#2}#3}}
217 </package>
218 <*txml>
219 DefConstructor(' \inlinetypedec OptionalKeyVals:omtext {}{}',
220 " <omdoc:type for='&GetKeyVal(#1,'for')' "
221 . " ?&GetKeyVal(#1,'id')(xml:id='&GetKeyVal(#1,'id').not')() "
222 . " ?&GetKeyVal(#1,'system')(xml:id='&GetKeyVal(#1,'system'))()>"
223 . " <ltx:Math><ltx:XMath>#2</ltx:XMath></ltx:Math>"
224 . " <omdoc:CMPT>#body"
225 . " </omdoc:type>");
226 </txml>

```

typedec We first define a theorem environment

```

227 <*package>
228 \def\st@typedec@kw{Type Declaration}
229 \theorembodyfont{\upshape}
230 \newtheorem{STtypedecEnv}[STtheoremAssEnv]{\st@typedec@kw}
231 and then the environment itself.
232 \newenvironment{typedec}[2] [] {\metasetkeys{omtext}{#1}\sref@target%
233 \def\thedectype{#2}%
234 \ifx\omtext@display\st@flow\else%
235 \ifx\omtext@title{}empty\begin{STtypedecEnv}\else\begin{STtypedecEnv}[\omtext@title]\fi%
236 \ifx\sref@id{}empty\else\label{typedec.\sref@id}\fi
237 \ifx\sref@id{}empty\sref@label@id{here}\else%
238 \sref@label@id{\STpresent{\csname STtypedecEnvKeyword\endcsname}~\@currentlabel}\fi%
239 \ignorespaces}
240 {\ifx\omtext@display\st@flow\else\end{STtypedecEnv}\fi\omtext@post@skip}
241 </package>
242 <*txml>
243 DefStatement(' {typedec} OptionalKeyVals:omtext {}',

```

```

243 "<omdoc:type for='&GetKeyVal(#1,'for')'"
244 . "?&GetKeyVal(#1,'id')(xml:id='&GetKeyVal(#1,'id').not')()"
245 . "?&GetKeyVal(#1,'system')(xml:id='&GetKeyVal(#1,'system')')()>"
246 . "?&GetKeyVal(#1,'title')(<dc:title>&GetKeyVal(#1,'title')</dc:title>())"
247 . "<ltx:Math><ltx:XMath>#2</ltx:XMath></ltx:Math>"
248 . "<omdoc:CMP>#body"
249 . "</omdoc:type>");
250 </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`.

```

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

```

```

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

```

notation We initialize the `\def\st@notation@initialize{}` here, and extend it with functionality below.

```

329 <*package>
330 \def\notemph#1{#1}
331 \def\st@notation@terminate{}
332 \def\st@notation@initialize{}
333 \define@statement@env{notation}
334 \def\st@notation@kw{Notation}
335 \theorembodyfont{\upshape}
336 \newtheorem{STnotationEnv}[STtheoremAssEnv]{\st@notation@kw}

```

```

337 </package>
338 <*ltxml>
339 DefStatement('{notation} OptionalKeyVals:omtext',
340   "<omdoc:definition "
341   .   "?&GetKeyVal(#1,'id')(xml:id='&GetKeyVal(#1,'id').not')()"
342   .   "?&GetKeyVal(#1,'for')(for='&simple_wrapper(&GetKeyVal(#1,'for'))')()>"
343   .   "?&GetKeyVal(#1,'title')(<dc:title>&GetKeyVal(#1,'title')</dc:title>())"
344   .   "<omdoc:CMP>#body"
345   .   "</omdoc:definition>\n");
346 DefConstructor('\notatiendum OptionalKeyVals:notation {}',
347   "<ltx:text class='notatiendum'>#2</ltx:text>");
348 </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>@<module name>@target`, if `<opt>` is non-empty, else with the label `sref@<name>@<module name>@target`. Also it generates the necessary warnings for a `definiendum`-like macro.

```

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

```

370 (*package)
371 \newcommand\@termdef[2] [] {\def\@test{#1}%
372 \ifundefined{mod@id}{\ifx\@test\empty\def\@name{#2}\else\def\@name{#1}\fi%
373 \termdef\mod@id \@name{#2}}
374 \}

```

```

375 \package
376 \newcommand\definiendum[2] [] {\st@def@target{#1}{#2}\@termdef{#1}{#2}\defemph{#2}}
377 \newcommand\definiendum[2] [] {\st@def@target{#1}{#2}\defemph{#2}}
378 \endpackage
379 \let\ltxml
380 DefConstructor('\definiendum [] {}',
381               "<omdoc:term role='definiendum' name='#name' cd='#theory'>#2</omdoc:term>",
382               afterDigest => sub { defHelper(@_, 'definiendum'); });
383 \let\ltxml

```

```

384 \package
385 \newcommand\notatiendum[2][\notemph{#2}]
386 \endpackage

```

```

387 \package)
388 \newcommand\defi[2] [] {\st@def@target{#1}{#2}\defemph{#2}\def \@phrase{#2}\@defi}
389 \newcommand\@defi[1] [] {\ifdef@index\omdoc@index[#1]{\@phrase}\fi\xspace}
390 \newcommand\defis[2] [] {\st@def@target{#1}{#2}\defemph{#2s}\def \@phrase{#2}\@defi}
391 \end{package}
392 \end{txml}

393 DefConstructor(' \defi[]{} OptionalKeyVals:DEF',
394   " ?#defindex(<omdoc:idx><omdoc:idt>)"
395   .    "<omdoc:term role='definiendum' name='?#1(#1)(#2)' cd='#theory'#2</omdoc:term>"
396   ." ?#defindex(</omdoc:idt><omdoc:ide index='default'><omdoc:idp>#2</omdoc:idp></omdoc:ide></omdoc:"
397 afterDigest => sub { defHelper(@_, 'defi'); },
398 alias=>' \defi');

399 DefConstructor(' \defis[]{} OptionalKeyVals:DEF',
400   " ?#defindex(<omdoc:idx><omdoc:idt>)"
401   .    "<omdoc:term role='definiendum' name='?#1(#1)(#2)' cd='#theory'#2s</omdoc:term>"
402   ." ?#defindex(</omdoc:idt><omdoc:ide index='default'><omdoc:idp>#2</omdoc:idp></omdoc:ide></omdoc:"
403 afterDigest => sub { defHelper(@_, 'defi'); },
404 alias=>' \defi');

405 \end{txml}

```

`\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.

```

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

```

`\defii`

```

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

```

`\adefii` analogous to `\adefi`

```

448 <*package>

```

```

449 \newcommand\adezii[4] [] {\def\@name{#1}\def\@pone{#3}\def\@ptwo{#4}%
450 \st@def@target{#1}{#3-#4}\defemph{#2}\@adezii}
451 \newcommand\@adezii[1] [] {%
452 \ifdef@index%
453 \ifx\@name\@empty\@twin[#1]{\@pone}{\@ptwo}%
454 \else\@twin[at=\@name,#1]{\@pone}{\@ptwo}\fi%
455 \fi\xspace}
456 \end{package}
457 \end{*txml}
458 DefConstructor('\adezii[]{}{} OptionalKeyVals:DEF',
459 "?#defindex(<omdoc:idx><omdoc:idt>)"
460 . "<omdoc:term role='definiendum' name='?#1(#1)&dashed(#3,#4)'' cd='#theory'>"
461 . "#2"
462 . "</omdoc:term>"
463 . "?#defindex(</omdoc:idt><omdoc:ide index='default'><omdoc:idx>#3</omdoc:idx><omdoc:idx>#4</omdoc:idx>"
464 . afterDigest => sub { defHelper(@_, 'adezii'); },
465 . alias=>'adezii');
466 \end{*txml}

\defiii similar to \defii
467 \end{package}
468 \newcommand\defiii[4] [] {\def\@pone{#2}\def\@ptwo{#3}\def\@pthree{#4}%
469 \st@def@target{#1}{#2-#3-#4}\defemph{#2 #3 #4}\@defiii}
470 \newcommand\@defiii[1] [] {\ifdef@index\@twin[#1]{\@pone}{\@ptwo}{\@pthree}\fi\xspace}
471 \newcommand\defiiis[4] [] {\def\@pone{#2}\def\@ptwo{#3}\def\@pthree{#4}%
472 \st@def@target{#1}{#2-#3-#4}\defemph{#2 #3 #4s}\@defiii}
473 \end{package}
474 \end{*txml}
475 DefConstructor('\defiii[]{}{} OptionalKeyVals:DEF',
476 "?#defindex(<omdoc:idx><omdoc:idt>)"
477 . "<omdoc:term role='definiendum' cd='#theory' name='?#1(#1)&dashed(#2,#3,#4)''>#2 #3"
478 . "?#defindex(</omdoc:idt><omdoc:ide index='default'><omdoc:idx>#2</omdoc:idx><omdoc:idx>#3</omdoc:idx>"
479 . afterDigest => sub { defHelper(@_, 'defiii'); },
480 . alias=>'defiii');
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:idx>#2</omdoc:idx><omdoc:idx>#3</omdoc:idx>"
485 . afterDigest => sub { defHelper(@_, 'defiii'); },
486 . alias=>'defiii');
487 \end{*txml}

\adeiiiii
488 \end{package}
489 \newcommand\adeiiiii[5] [] {\def\@name{#1}\def\@pone{#3}\def\@ptwo{#4}\def\@pthree{#3}%
490 \st@def@target{#1}{#3-#4-#5}\defemph{#2}\@adeiiiii}
491 \newcommand\@adeiiiii[1] [] {%
492 \ifdef@index%
493 \ifx\@name\@empty\@twin[#1]{\@pone}{\@ptwo}{\@pthree}%
494 \else\@twin[at=\@name,#1]{\@pone}{\@ptwo}{\@pthree}\fi%

```

```

495 \fi\xspace}
496 \end{package}
497 \end{*xml}
498 DefConstructor('\adeftiii []{}{}{} OptionalKeyVals:DEF',
499 "?#defindex(<omdoc:idx><omdoc:idt>)"
500 . "<omdoc:term role='definiendum' cd='#theory' name='?#1(#1)(&dashed(#3,#4,#5))'>#2</omdoc:term>"
501 . "?#defindex(</omdoc:idt><omdoc:ide index='default'><omdoc:idp>#3</omdoc:idp><omdoc:idp>#4</omdoc:ide>"
502 . afterDigest => sub { defHelper(@_, 'adeftiii')},
503 . alias=>'adeftiii');
504 \end{*xml}

\inlineex
505 \begin{package}
506 \newcommand\inlineex[2] [] {\metasetkeys{omtext}{#1}%
507 \sref@target\sref@label{id{here}}#2}
508 \end{package}
509 \end{*xml}
510 DefConstructor('\inlineex OptionalKeyVals:omtext {}',
511 . "<ltx:text class='example'>#2</ltx:text>");
512 \end{*xml}

\inlineass
513 \begin{package}
514 \newcommand\inlineass[2] [] {\metasetkeys{omtext}{#1}%
515 \sref@target\sref@label{id{here}}#2}
516 \end{package}
517 \end{*xml}
518 DefConstructor('\inlineass OptionalKeyVals:omtext {}',
519 . "<ltx:text "
520 . " ?&GetKeyVal(#1,'type')(class='&GetKeyVal(#1,'type') assertion')(class='assert"
521 . "#2"
522 . "</ltx:text>");
523 \end{*xml}

\inlinedef
524 \begin{package}
525 \newcommand\inlinedef[2] [] {\metasetkeys{omtext}{#1}%
526 \if@in@omtext\else% we are not in an omtext or statement
527 \PackageError{modules}{\protect\inlinedef\space outside a statement!}%
528 {Try wrapping the paragraph in a\MessageBreak
529 \protect\begin{omtext}, \protect\begin{assertion}, \protect\begin{axiom}, ... \MessageBreak
530 whatever is suitable semantically}\fi%
531 \sref@target\sref@label{id{here}}\st@indeftrue #2}
532 \end{package}
533 \end{*xml}
534 DefConstructor('\inlinedef OptionalKeyVals:omtext {}', sub {
535 my ($document, $keyvals, $body, %props) = @_;
536 my $for = $keyvals->getValue('for') if $keyvals;
537 my %for_attr=();

```



```

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

```

583 <*package>
584 \addmetakey*{termref}{cd}
585 \addmetakey*{termref}{cibase}
586 \addmetakey*{termref}{name}
587 \addmetakey*{termref}{role}
588 \newcommand\termref[2] [] {\metasetkeys{termref}{#1}%
589 \ifx\termref@cd\@empty\def\termref@cd{\mod@id}\fi%
590 \st@termref{#2}}
591 </package>
592 <*lxml>
593 DefConstructor('\termref OptionalKeyVals:termref {}',
594               "<omdoc:term "
595               . "?&GetKeyVal(#1,'cibase')(cibase='&GetKeyVal(#1,'cibase')')() "
596               . "cd='&GetKeyVal(#1,'cd')(&GetKeyVal(#1,'cd'))(#module)' "
597               . "name='&GetKeyVal(#1,'name')'>"
598               . "#2"
599               . "</omdoc:term>",
600               afterDigest=>sub{$_[1]->setProperty(module=>(LookupValue('modnl_signature') || Lo
601 </lxml>)%$

```

The next macro is where the actual work is done.

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

```

602 <*package>
603 \newcommand\st@termref[1] {\ifx\termref@name\@empty\def\termref@name{#1}\fi%
604 \ifx\termref@cibase\@empty\mod@termref\termref@cd\termref@name{#1}%
605 \else\sref@href@ifh\termref@cibase{#1}\fi}
606 </package>

```

`\tref*`

```

607 <lxml>RawTeX('
608 <*package | lxml>
609 \newcommand\atrefi[3] [] {\def\@test{#1}%
610 \ifx\@test\@empty\termref[name=#3]{#2}\else\termref[cd=#1,name=#3]{#2}\fi}
611 \newcommand\atrefii[4] [] {\atrefi[#1]{#2}{#3-#4}}
612 \newcommand\atrefiii[5] [] {\atrefi[#1]{#2}{#3-#4-#5}}

```

`\tref*`

```

613 \newcommand\trefi[2] [] {\atrefi[#1]{#2}{#2}}
614 \newcommand\trefii[3] [] {\atrefi[#1]{#2 #3}{#2-#3}}
615 \newcommand\trefiii[4] [] {\atrefi[#1]{#2 #3 #4}{#2-#3-#4}}
616 \newcommand\trefis[2] [] {\atrefi[#1]{#2s}{#2}}
617 \newcommand\trefiis[3] [] {\atrefi[#1]{#2 #3s}{#2-#3}}

```

```

618 \newcommand\trefiiiis[4] [] {\atrefi[#1]{#2 #3 #4s}{#2-#3-#4}}
619 \</package | ltxml>
620 \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 L^AT_EX_ML bindings for them.

***emph**

```

621 \<*package>
622 \providecommand{\termemph}[1]{#1}
623 \providecommand{\defemph}[1]{\textbf{#1}}
624 \providecommand{\stDMemph}[1]{\textbf{#1}}
625 \</package>

```

EdN:7

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

```

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

```

640 \<*package>
641 \newcommand\symref[2]{\@nameuse{mod@symref@#1}{#2}}
642 \</package>
643 \<*ltxml>
644 DefConstructor('\symref{ }',
645                 "<omdoc:term cd='&LookupValue('symdef.#1.cd')' name='&LookupValue('symdef.#1.nam
646                 . \"#2"
647                 . "</omdoc:term>");
648 \</ltxml>

```

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`.

⁷EdNOTE: MK: document above

```

649 <*!xml>
650 Tag('omdoc:assertion',afterOpen=>\&numberIt,afterClose=>\&locateIt);
651 Tag('omdoc:definition',afterOpen=>\&numberIt,afterClose=>\&locateIt);
652 Tag('omdoc:example',afterOpen=>\&numberIt,afterClose=>\&locateIt);
653 Tag('omdoc:equation',afterOpen=>\&numberIt,afterClose=>\&locateIt);
654 Tag('omdoc:axiom',afterOpen=>\&numberIt,afterClose=>\&locateIt);
655 Tag('omdoc:symbol',afterOpen=>\&numberIt,afterClose=>\&locateIt);
656 Tag('omdoc:type',afterOpen=>\&numberIt,afterClose=>\&locateIt);
657 Tag('omdoc:term',afterOpen=>\&numberIt,afterClose=>\&locateIt);
658 </!xml>

```

5.5 Auxiliary Functionality

```

659 <*!xml>
660 # =====
661 # Auxiliary Functions: #
662 # =====
663 sub DefStatement {
664   my ($definition,$replacement,%properties)=@_;
665   DefEnvironment($definition,$replacement,%properties,
666     afterDigestBegin=>\&declareFunctions,
667   );}
668
669 sub declareFunctions{
670   my ($stomach,$whatsit) = @_;
671   my $keyval = $whatsit->getArg(1);
672   my $funval = GetKeyVal($keyval,'functions') if GetKeyVal($keyval,'functions');
673   return unless $funval;
674   my @funssyms = $funval->unlist;
675   #Unread the function declarations at the Gullet
676   foreach (@funssyms) {
677     my $symb = UnTeX($_);
678     $stomach->getGullet->unread(Tokenize('\lxDeclare[role=FUNCTION]{$'.$symb.'$}')->unlist);
679   }
680   return; }
681 sub defHelper{
682   my ($stomach, $whatsit, $defOption) = @_;
683   my $addr = LookupValue('defs');
684   my $name = $whatsit->getArg(1);
685   $name = $name->toString if $name;
686   my %choose_Option = (
687     definiendum => sub {$whatsit->getArg(2);},
688     defi => sub {$whatsit->getArg(2);},
689     adefi => sub {$whatsit->getArg(3);},
690     defii => sub {$whatsit->getArg(2)->toString.'-'. $whatsit->getArg(3)->toString},
691     adefii => sub {$whatsit->getArg(3)->toString.'-'. $whatsit->getArg(4)->toString},
692     defiii => sub {$whatsit->getArg(2)->toString.'-'. $whatsit->getArg(3)->toString},
693     adefiii => sub {$whatsit->getArg(3)->toString.'-'. $whatsit->getArg(4)->toString},
694   );
695   $name = $name || $choose_Option{$defOption}->();

```

```

696 $whatsit->setProperty(name=>$name->toString) if ref($name);
697 push(@$addr, $name) if ($addr and $name);
698 $whatsit->setProperty('defindex', IfCondition(T_CS('if@defindex')));
699 $whatsit->setProperty(theory=>(LookupValue('modnl_signature') || LookupValue('current_module'
700 return;}##$
701 </ltxml>

```

5.6 Deprecated Functionality

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

def

```

702 <ltxml>##### Deprecated functionality:
703 <ltxml>RawTeX('
704 <*package | ltxml>
705 \newcommand\defin[2] [] {\defi[#1]{#2}%
706 \PackageWarning{statements}{\protect\defin\space is deprecated, use \protect\defi\space instead
707 \newcommand\twindef[3] [] {\defii[#1]{#2}{#3}%
708 \PackageWarning{statements}{\protect\twindef\space is deprecated, use \protect\defii\space inst
709 \newcommand\atwindef[4] [] {\defiii[#1]{#2}{#3}{#4}%
710 \PackageWarning{statements}{\protect\atwindef\space is deprecated, use \protect\defiii\space in
711 \newcommand\definalt[3] [] {\adefi[#1]{#2}{#3}%
712 \PackageWarning{statements}{\protect\definalt\space is deprecated, use \protect\adefi\space ins
713 \newcommand\twindefalt[4] [] {\adefii[#1]{#2}{#3}{#4}%
714 \PackageWarning{statements}{\protect\twindefalt\space is deprecated, use \protect\adefii\space
715 \newcommand\atwindefalt[5] [] {\adefiii[#1]{#2}{#3}{#4}{#5}%
716 \PackageWarning{statements}{\protect\atwindefalt\space is deprecated, use \protect\adefiii\space

```

def

```

717 \newcommand\twinref[3] [] {\trefii[#1]{#2}{#3}%
718 \PackageWarning{statements}{\protect\twinref\space is deprecated, use \protect\trefii\space ins
719 \newcommand\atwinref[4] [] {\atrefiii[#1]{#2}{#3}{#4}%
720 \PackageWarning{statements}{\protect\atwinref\space is deprecated, use \protect\trefiii\space i
721 </package | ltxml>
722 <ltxml>');

```

5.7 Finale

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

```

723 <ltxml>1;

```

Index

Numbers written in *italic* refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in *roman* refer to the code lines where the entry is used.

<i>*</i> ,	10	statement,	2	OPENMATH,	5
block					
statement,	2	LATEXML, 11, 12, 21, 27	statement		
			block,		2
flow		OMDoc, 2, 4, 5, 27	flow,		2

References

- [KG15] Michael Kohlhasse 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 Kohlhasse, Deyan Ginev, and Rares Ambrus. *modules.sty: Semantic Macros and Module Scoping in sTeX*. Tech. rep. Comprehensive T_EX Archive Network (CTAN), 2015. URL: <http://www.ctan.org/get/macros/latex/contrib/stex/modules/modules.pdf>.
- [Koh06] Michael Kohlhasse. 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 Kohlhasse. “Using L^AT_EX 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 Kohlhasse. *metakeys.sty: A generic framework for extensible Metadata in L^AT_EX*. Tech. rep. Comprehensive T_EX Archive Network (CTAN), 2015. URL: <http://www.ctan.org/tex-archive/macros/latex/contrib/stex/metakeys/metakeys.pdf>.
- [Koh15b] Michael Kohlhasse. *omdoc.sty/cls: Semantic Markup for Open Mathematical Documents in L^AT_EX*. Tech. rep. Comprehensive T_EX Archive Network (CTAN), 2015. URL: <http://www.ctan.org/tex-archive/macros/latex/contrib/stex/omdoc/omdoc.pdf>.
- [Koh15c] Michael Kohlhasse. *sref.sty: Semantic Crossreferencing in L^AT_EX*. Tech. rep. Comprehensive T_EX 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 L^AT_EX-Theorem Environment*. Self-documenting L^AT_EX 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).