Semantic Markup for Mathematical Statements*

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

November 19, 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 [**Kohlhase:ulsmf08**; **sTeX:online**] 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 [**Kohlhase:OMDoc1.2**] 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 [**MaySch:eltte09**] 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, the `statements` package passes the `showmeta` to the `metakeys` package. If this
showmeta   is set, then the metadata keys are shown (see [**Kohlhase:metakeys:ctan**] 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 [**Kohlhase:sref:ctan**].

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 [**MaySch:eltte09**] 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.

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

2.2.2 Symbols

symboldec
The `symboldec` environment can be used for declaring concepts and symbols. Note 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. SᴛEX 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 $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$ (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
\inlinetypedec
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
\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.

```
\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

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 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: $:=\succ$)

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 [**Kohlhase:smomdl:ctan**]: 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

7

\defiiis for that case: \defiis{simple}{group} is equivalent to much longer
\defiiis   \adefii{simple groups}{simple}{group}.

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

Note that the \definiendum, \defi, \defii, and \defiii macros can only be
used inside the definitional situation, i.e. in a definition or symboldec environ-
ment or a \inlinedef macro. If you find yourself in a situation where you want to
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 environ-
ment, 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.

8

\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

\termref phrases[1]. Therefore, the `\termref` can be used to make this information explicit. 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 [**KohAmb:smmssl:ctan**].

`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

---

[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`.

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 [**KohAmb:smmssl:ctan**]. 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` 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

`\symref` this the `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]

---

[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.

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` [**KohGin:smss:svn**] package not in `statements` itself.

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:github:on**].

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 -*-
```

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`.

```
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 DeclareOption('mh', sub {AssignValue('@statements' => 1);
18 PassOptions('omtext','sty',ToString(Digest(T_CS('\CurrentOption'))));
19 PassOptions('modules','sty',ToString(Digest(T_CS('\CurrentOption'))));});
20 DeclareOption('defindex', sub{Digest(T_CS('\@defindextrue')); });
21 DeclareOption(undef,sub{PassOptions('omtext','sty',ToString(Digest(T_CS('\CurrentOption'))));
22                                         PassOptions('modules','sty',ToString(Digest(T_CS('\Curr
23 ProcessOptions();
24 ⟨/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.

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

Now, we define an auxiliary function that lowercases strings

```
37 ⟨∗ltxml⟩
38 sub lowcase {my ($string) = @_; $string ? return lc(ToString($string)) : return('')}#$
```

```
39 sub dashed { join('-',map($_->toString,@_));}#$
```
40 ⟨/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]

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

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

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

---

[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.

13

```
80    "#$output"||’’;
81  }
82  ⟨/ltxml⟩
```

For the other languages, we set up triggers

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

5.2   Statements

\STpresent

```
86  ⟨*package⟩
87  \providecommand\STpresent[1]{#1}
88  ⟨/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.

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

assertion

```
103 ⟨*package⟩
104 \newenvironment{assertion}[1][]{\metasetkeys{omtext}{#1}\sref@target%
105 \@in@omtexttrue%
106 \ifx\omtext@display\st@flow\itshape\noindent\ignorespaces%
107 \else% display!=flow
108 \ifx\omtext@title\@empty\begin{ST\omtext@type AssEnv}%
109 \else\begin{ST\omtext@type AssEnv}[\omtext@title]\fi\fi%
110 \ifx\omtext@type\@empty\sref@label@id{here}\else%
111 \sref@label@id{\STpresent{\csname ST\omtext@type AssEnvKeyword\endcsname}~\@currentlabel}
112 \fi}%display=flow
113 {\ifx\omtext@display\st@flow\else\end{ST\omtext@type AssEnv}\@in@omtextfalse\fi}
```

14

```
114 ⟨/package⟩
115 ⟨∗ltxml⟩
116 DefStatement('{assertion} OptionalKeyVals:omtext',
117   "<omdoc:assertion "
118   .  "?&GetKeyVal(#1,'id')(xml:id='&GetKeyVal(#1,'id')')() "
119   .  "?&GetKeyVal(#1,'theory')(theory='&GetKeyVal(#1,'theory')')() "
120   .  "type='&lowcase(&GetKeyVal(#1,'type'))'>"
121   .  "?&GetKeyVal(#1,'title')(<dc:title>&GetKeyVal(#1,'title')</dc:title>)()"
122   .  "<omdoc:CMP>#body"
123   ."</omdoc:assertion>\n");
124 ⟨/ltxml⟩
```

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

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

Then we configure the presentation of the theorem environments

```
137 \theorembodyfont{\itshape}
138 \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

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

EdN:5   example   [5]

---

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

```
151 ⟨*package⟩
152 \let\usevocab=\usemodule
153 \def\st@example@initialize{}\def\st@example@terminate{}
154 \define@statement@env{example}
155 \def\st@example@kw{Example}
156 \theorembodyfont{\upshape}
157 \newtheorem{STexampleEnv}[STtheoremAssEnv]{\st@example@kw}
158 ⟨/package⟩
159 ⟨*ltxml⟩
160 DefMacro('\usevocab','\usemodule');
161 DefStatement('{example} OptionalKeyVals:omtext',
162       "<omdoc:example "
163       . "?&GetKeyVal(#1,'id')(xml:id='&GetKeyVal(#1,'id')')() "
164       . "?&GetKeyVal(#1,'for')(for='&hash_wrapper(&GetKeyVal(#1,'for'))')()>"
165       . "?&GetKeyVal(#1,'title')(<dc:title>&GetKeyVal(#1,'title')</dc:title>)()"
166      . "#body"
167      . "</omdoc:example>\n");
168 ⟨/ltxml⟩
```

axiom

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

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

```
184 ⟨*package⟩
185 \srefaddidkey{symboldec}
186 \addmetakey{symboldec}{functions}
187 \addmetakey{symboldec}{role}
188 \addmetakey*{symboldec}{title}
189 \addmetakey*{symboldec}{name}
190 \addmetakey{symboldec}{subject}
191 \addmetakey*{symboldec}{display}
192 \newenvironment{symboldec}[1][]{\metasetkeys{symboldec}{#1}\sref@target\st@indeftrue%
193 \ifx\symboldec@display\st@flow\else{\noindent\stDMemph{\symdef@type} \symboldec@name:}\fi%
194 \ifx\symboldec@title\@empty~\else~(\stDMemph{\symboldec@title})\par\fi}{}
195 ⟨/package⟩
196 ⟨*ltxml⟩
```

16

```
197 DefStatement('{symboldec} OptionalKeyVals:symboldec',
198      "<omdoc:symbol "
199    . "?&GetKeyVal(#1,'id')(xml:id='&GetKeyVal(#1,'id')')"
200    .                  "(xml:id='&makeNCName(&GetKeyVal(#1,'name')).def.sym')"
201    .            "name='&GetKeyVal(#1,'name')'>"
202    . "?&GetKeyVal(#1,'title')(<dc:title>&GetKeyVal(#1,'title')</dc:title>)()"
203    . "<dc:description>#body"
204    ."</omdoc:symbol>\n");
```
205 ⟨/ltxml⟩

5.2.1 Types

\symtype   [6]

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

\inlinetypedec

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

typedec   We first define a theorem environment
234 ⟨∗package⟩
235 \def\st@typedec@kw{Type Declaration}
236 \theorembodyfont{\upshape}

---

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

17

237 \newtheorem{STtypedecEnv}[STtheoremAssEnv]{\st@typedec@kw}

and then the environment itself.

238 \newenvironment{typedec}[2][]{\metasetkeys{omtext}{#1}\sref@target%
239 \def\thedectype{#2}%
240 \ifx\omtext@display\st@flow\else%
241 \ifx\omtext@title\@empty\begin{STtypedecEnv}\else\begin{STtypedecEnv}[\omtext@title]\fi%
242 \ifx\sref@id\@empty\else\label{typedec.\sref@id}\fi
243 \ifx\sref@id\@empty\sref@label@id{here}\else%
244 \sref@label@id{\STpresent{\csname STtypedecEnvKeyword\endcsname}~\@currentlabel}\fi%
245 \ignorespaces}
246 {\ifx\omtext@display\st@flow\else\end{STtypedecEnv}\fi\omtext@post@skip}
247 ⟨/package⟩
248 ⟨∗ltxml⟩
249 DefStatement('{typedec} OptionalKeyVals:omtext {}',
250   "<omdoc:type for='&GetKeyVal(#1,'for')'"
251 .   "?&GetKeyVal(#1,'id')(xml:id='&GetKeyVal(#1,'id').not')()"
252 .   "?&GetKeyVal(#1,'system')(xml:id='&GetKeyVal(#1,'system')')()>"
253 .   "?&GetKeyVal(#1,'title')(<dc:title>&GetKeyVal(#1,'title')</dc:title>)()"
254 .   "<ltx:Math><ltx:XMath>#2</ltx:XMath></ltx:Math>"
255 .   "<omdoc:CMP>#body"
256 ."</omdoc:type>");
257 ⟨/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.

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

```
282      }}
283      if ($props{theory}) {
284        my @symbols = @{$props{defs} || []};
285        my $signature = $props{signature};
286        foreach my $symb(@symbols) {
287          next if $for_attr{$symb};
288          my $qualified_symbol = $signature ? "$signature?$symb" : $symb;
289          $for_attr{$qualified_symbol}=1;
290          if (!$props{multiling}) {
291            $doc->insertElement('omdoc:symbol', undef, (name=>$symb, "xml:id"=>makeNCName("$symb.
292        }
293      }
294      my %attrs = ();
295      $for = join(" ",(sort keys %for_attr));
296      $attrs{'for'} = $for if $for;
297      my $id = $keyvals->getValue('id') if $keyvals;
298      $attrs{'xml:id'} = $id if $id;
299      $attrs{'type'} = $type if $type;
300      if ($props{theory}) {
301        $doc->openElement('omdoc:definition', %attrs);
302      } else {
303        $attrs{'type'}='definition';
304        $doc->openElement('omdoc:omtext', %attrs);
305      }
306      my $title = $keyvals->getValue('title') if $keyvals;
307      if ($title) {
308        $doc->openElement('omdoc:metadata');
309        $doc->openElement('dc:title');
310        $doc->absorb($title);
311        $doc->closeElement('dc:title');}
312    $doc->openElement('omdoc:CMP');
313    $doc->absorb($props{body}) if $props{body};
314    $doc->maybeCloseElement('omdoc:CMP');
315      if ($props{theory}) {
316        $doc->closeElement('omdoc:definition');
317      } else {
318        $doc->closeElement('omdoc:omtext');
319      }
320      return; }
321 # We use the standard DefEnvironment here, since
322 # afterDigestBegins would collide otherwise
323 DefEnvironment('{definition} OptionalKeyVals:omtext', \&definitionBody,
324   afterDigestBegin=>sub {
325     my ($stomach, $whatsit) = @_;
326     my @symbols = ();
327     $whatsit->setProperty(multiling=>LookupValue('multiling'));
328     $whatsit->setProperty(theory=>(LookupValue('modnl_signature') || LookupValue('current_modul
329     $whatsit->setProperty(defs=>\@symbols);
330     $whatsit->setProperty(signature=>LookupValue('modnl_signature'));
331     AssignValue('defs', \@symbols);
```

```
332     declareFunctions($stomach,$whatsit);
333     return; },
334   afterDigest => sub { AssignValue('defs', undef); return; });
335 ⟨/ltxml⟩%$
```

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

```
336 ⟨*package⟩
337 \def\notemph#1{#1}
338 \def\st@notation@terminate{}
339 \def\st@notation@initialize{}
340 \define@statement@env{notation}
341 \def\st@notation@kw{Notation}
342 \theorembodyfont{\upshape}
343 \newtheorem{STnotationEnv}[STtheoremAssEnv]{\st@notation@kw}
344 ⟨/package⟩
345 ⟨*ltxml⟩
346 DefStatement('{notation} OptionalKeyVals:omtext',
347   "<omdoc:definition "
348 .   "?&GetKeyVal(#1,'id')(xml:id='&GetKeyVal(#1,'id').not')()"
349 .   "?&GetKeyVal(#1,'for')(for='&simple_wrapper(&GetKeyVal(#1,'for'))')()>"
350 . "?&GetKeyVal(#1,'title')(<dc:title>&GetKeyVal(#1,'title')</dc:title>)()"
351 . "<omdoc:CMP>#body"
352   . "</omdoc:definition>\n");
353 DefConstructor('\notatiendum OptionalKeyVals:notation {}',
354                "<ltx:text class='notatiendum'>#2</ltx:text>");
355 ⟨/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.

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

```
371 \protect\definiendum, \protect\defi,
372 \protect\defii, \protect\defiii\MessageBreak
373 do not make sense semantically outside a definition.\MessageBreak
374 Consider wrapping the defining phrase in a \protect\inlinedef}%
375 \fi}
376 ⟨/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.

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

`\definiendum`

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

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

```
391 ⟨*package⟩
392 \newcommand\notatiendum[2][]{\notemph{#2}}
393 ⟨/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.

```
394 ⟨*package⟩
395 \newcommand\defi[2][]{\st@def@target{#1}{#2}\defemph{#2}\def\@phrase{#2}\@defi}
396 \newcommand\@defi[1][]{\ifdef@index\omdoc@index[#1]{\@phrase}\fi\xspace}
397 \newcommand\defis[2][]{\st@def@target{#1}{#2}\defemph{#2s}\def\@phrase{#2}\@defi}
398 ⟨/package⟩
```

```
399 ⟨*ltxml⟩
400 DefConstructor('\defi[]{} OptionalKeyVals:DEF',
401   "?#defindex(<omdoc:idx><omdoc:idt>)"
402   .    "<omdoc:term role='definiendum' name='?#1(#1)(#2)' cd='#theory'>#2</omdoc:term>"
403   ."?#defindex(</omdoc:idt><omdoc:ide index='default'><omdoc:idp>#2</omdoc:idp></omdoc:ide></omd
404   afterDigest => sub { defHelper(@_, 'defi'); },
405   alias=>'\defi');
406 DefConstructor('\defis[]{} OptionalKeyVals:DEF',
407   "?#defindex(<omdoc:idx><omdoc:idt>)"
408   .    "<omdoc:term role='definiendum' name='?#1(#1)(#2)' cd='#theory'>#2s</omdoc:term>"
409   ."?#defindex(</omdoc:idt><omdoc:ide index='default'><omdoc:idp>#2</omdoc:idp></omdoc:ide></omd
410   afterDigest => sub { defHelper(@_, 'defi'); },
411   alias=>'\defi');
412 ⟨/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.

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

\defii

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

```
444        afterDigest => sub { defHelper(@_, 'defii'); },
445        alias=>'\defii');
446 DefConstructor('\defiis[]{}{} OptionalKeyVals:DEF',
447    "?#defindex(<omdoc:idx><omdoc:idt>)"
448 .    "<omdoc:term role='definiendum' name='?#1(#1)(&dashed(#2,#3))' cd='#theory'>"
449 .       "#2 #3s"
450 .    "</omdoc:term>"
451 ."?#defindex(</omdoc:idt><omdoc:ide index='default'><omdoc:idp>#2</omdoc:idp><omdoc:idp>#3</om
452        afterDigest => sub { defHelper(@_, 'defii'); },
453        alias=>'\defii');
454 ⟨/ltxml⟩
```

\adefii   analogous to \adefi

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

\defiii   similar to \defii

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

```
490            . "<omdoc:term role='definiendum' cd='#theory' name='?#1(#1)(&dashed(#2,#3,#4))'>#2 #3
491 ."?#defindex(</omdoc:idt><omdoc:ide index='default'><omdoc:idp>#2</omdoc:idp><omdoc:idp>#3</om
492        afterDigest => sub { defHelper(@_, 'defiiii'); },
493        alias=>'\defiiii');
494 ⟨/ltxml⟩
```

\adefiii

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

\inlineex

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

\inlineass

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

\inlinedef

```
531 ⟨∗package⟩
532 \newcommand\inlinedef[2][]{\metasetkeys{omtext}{#1}%
533 \if@in@omtext\else% we are not in an omtext or statement
534 \PackageError{modules}{\protect\inlinedef\space outside a statement!}%
535 {Try wrapping the paragraph in a\MessageBreak
536 \protect\begin{omtext}, \protect\begin{assertion}, \protect\begin{axiom}, ... \MessageBreak
537 whatever is suitable semantically}\fi%
538 \sref@target\sref@label@id{here}\st@indeftrue #2}
539 ⟨/package⟩
540 ⟨∗ltxml⟩
541 DefConstructor('\inlinedef OptionalKeyVals:omtext {}', sub {
542   my ($document, $keyvals, $body, %props) = @_;
543   my $for = $keyvals->getValue('for') if $keyvals;
544   my %for_attr=();
545   if (ToString($for)) {
546     $for = ToString($for);
547     $for =~ s/^{(.+)}$/$1/eg;
548     foreach (split(/,\s*/,$for)) {
549       $for_attr{$_}=1;
550     }}
551   my @symbols = @{$props{defs} || []};
552   #Prepare for symbol insertion -insert before the parent of the closest ancestor CMP element
553   my $original_node = $document->getNode;
554   my $statement_ancestor = $document->findnode('./ancestor::omdoc:CMP/..', $original_node);
555   foreach my $symb(@symbols) {
556     next if $for_attr{$symb};
557     $for_attr{$symb}=1;
558     my $symbolnode = XML::LibXML::Element->new('symbol');
559     $symbolnode->setAttribute(name=>$symb);
560     $symbolnode->setAttribute("xml:id"=>makeNCName("$symb.def.sym"));
561   if ($statement_ancestor) {
562     $statement_ancestor->parentNode->insertBefore($symbolnode,$statement_ancestor);
563   } else {
564     Error('malformed', $statement_ancestor, $original_node, "\\inlinedef outside a statement!
565 Try wrapping the paragraph in a begin{omtext}, \\begin{assertion}, \\begin{axiom}...\nwhatever
566   #Restore the insertion point
567   $document->setNode($original_node);
568   my %attrs = ();
569   $for = join(" ",(sort keys %for_attr));
570   $attrs{'for'} = $for if $for;
571   my $id = $keyvals->getValue('id') if $keyvals;
572   $attrs{'xml:id'} = $id if $id;
573   $attrs{'class'} = 'inlinedef';
574   $document->openElement('ltx:text',%attrs);
575   $document->absorb($body);
576   $document->closeElement('ltx:text'); },
577   #Prepare 'defs' hooks for \defi and \definiendum symbol names
578   beforeDigest=>sub {
579     my @symbols = ();
580     AssignValue('defs', \@symbols); return; },
```

```
581  #Adopt collected names as 'defs' property, remove hooks
582   afterDigest=>sub {
583      my ($stomach, $whatsit) = @_;
584      my $defsref = LookupValue('defs');
585      my @defs = @$defsref;
586      $whatsit->setProperty('defs',\@defs);
587      AssignValue('defs',undef);
588   return; });#$
```
589 ⟨/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.

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

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.

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

\tref*

614 ⟨ltxml⟩RawTeX('
615 ⟨∗package | ltxml⟩
616 \newcommand\atrefi[3][]{\def\@test{#1}%

```
617 \ifx\@test\@empty\termref[name=#3]{#2}\else\termref[cd=#1,name=#3]{#2}\fi}
618 \newcommand\atrefii[4][]{\atrefi[#1]{#2}{#3-#4}}
619 \newcommand\atrefiii[5][]{\atrefi[#1]{#2}{#3-#4-#5}}
```

\tref*

```
620 \newcommand\trefi[2][]{\atrefi[#1]{#2}{#2}}
621 \newcommand\trefii[3][]{\atrefi[#1]{#2 #3}{#2-#3}}
622 \newcommand\trefiii[4][]{\atrefi[#1]{#2 #3 #4}{#2-#3-#4}}
623 \newcommand\trefis[2][]{\atrefi[#1]{#2s}{#2}}
624 \newcommand\trefiis[3][]{\atrefi[#1]{#2 #3s}{#2-#3}}
625 \newcommand\trefiiis[4][]{\atrefi[#1]{#2 #3 #4s}{#2-#3-#4}}
626 ⟨/package | ltxml⟩
627 ⟨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

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

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

```
633 ⟨*package⟩
634 \newcommand\term[2][]{\def\@test{#1}%
635 \ifx\@test\@empty\else
636 \@ifundefined{module@defs@#1}{\PackageWarning{statements}%
637 {{\protect\term} specifies module #1 which is not in
638   scope\MessageBreak import it via e.g. via \protect\importmhmodule}}{}
639 \fi%
640 \PackageWarning{statements}%
641 {Dangling link (\protect\term) for "#2" still needs to be specified}%
642 \textcolor{blue}{\underline{#2}}}
643 ⟨/package⟩
644 ⟨*ltxml⟩
645 DefConstructor('\term{}',"<omdoc:term class='dangling-term-link' ?#1(cd='#1')()>#1</omdoc:term>
646 ⟨/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⟩.

```
647 ⟨*package⟩
648 \newcommand\symref[2]{\@nameuse{mod@symref@#1}{#2}}
649 ⟨/package⟩
650 ⟨*ltxml⟩
```

---

[7]EDNOTE: MK: document above

```
651 DefConstructor('\symref{}{}',
652                 "<omdoc:term cd='&LookupValue('symdef.#1.cd')' name='&LookupValue('symdef.#1.nam
653                 .  "#2"
654                 ."</omdoc:term>");
```
655 ⟨/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`.

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

5.5 Auxiliary Functionality

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

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

5.6   Deprecated Functionality

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

def

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

def

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

5.7   Finale

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

730 ⟨ltxml⟩1;