Semantic Markup for Mathematical Statements*

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

January 28, 2012

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 v1.1 (last revised 2012/01/28)

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 [Koh08; Ste] 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

`showmeta` The `statements` package takes a single option: `showmeta`. If this is set, then the metadata keys are shown (see [Koh10a] 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 distinguis 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 [Koh10c].

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 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 ⟨*type*⟩ the `assertion` environment calls the `ST`⟨*type*⟩`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`⟨*type*⟩`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 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`,

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).	
observation	if everything else fails
This type is the catch-all if none of the others applies.	

Example 2: Types of Mathematical Assertions

`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 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. 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 `\term`, 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.

`\definiendum`

`defin` The `\defi{`⟨*word*⟩`}` macro combines the functionality of the `\definiendum` macro with index markup from the `omdoc` package [Koh10b]: use `\defi[`⟨*name*⟩`]{`⟨*word*⟩`}` to markup a definiendum ⟨*word*⟩ with system name ⟨*name*⟩ that appear in the index — in other words in almost all definitions of single-word concepts. We also have the variants `\defii` and `\defiii` for (adjectivized) two-word compounds. Finally, the varaiants `\adefi`, `\adefii`,and `\adefiii` have an additional first argument that allows to specify an alternative text; see Figure 5

`\defii` `\defiii` `\adefi` `\adefii` `\adefiii`

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 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 "$\ldots s(o)$ 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.

2.2.4 Examples

`example` The `example` environment is a generic statement environment, except that the

```
\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

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 (P1) 0 is a natural number.

...

Axiom 6 (P5) Any property P such $P(0)$ and $P(s(k))$ whenever $P(k)$ holds for all n in $\mathbb{N}$

Example 3: Semantic Markup for the Peano Axioms

```
\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 7 1 is the successor of 0 (formally: $1 := s(0)$)

Example 4: A Definition based on Figure 3

source system name	result	index
`\defin{concept}`		
`concept`	concept	concept
`\defin[csymbol]{concept}`		
`csymbol`	concept	concept
`\definalt[csymbol]{concepts}{concept}`		
`csymbol`	concepts	concept
`\twindef{concept}{group}`		
`concept-group`	concept group	concept group, group - , concept
`\atwindef{small}{concept}{group}`		
`small-concept-group`	small concept group	small concept group, concept group - , small

Example 5: Some definienda with Index

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.

2.3 Cross-Referencing Symbols and Concepts

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

\termref

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 [KGA10].

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. Finally, 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*⟩}.

\trefi \trefii \trefiii \atref*

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.

[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.

Note that the \termref treatment above is natural for "concepts" declared by the \termdef macro from the modules package [KGA10]. 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 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.)

\symref

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 preset this in the class file. [1]

EdNote:1

Some authors like to lowercase the semantic references, i.e. use "axiom 2.6" instead of the default "Axiom 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.[2]

\STpresent

EdNote:2

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.[3].

EdNote:3

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 TRAC [Ste].

1. none reported yet

[1] EdNote: function declarations

[2] EdNote: this does not quite work as yet, since **STpresent** is applied when the label is written. But we would really like to have it applied when the reference is constructed. But for that we need to split the label into keyword and number in package sref.

[3] EdNote: we might want to develop an extension statements-babel in the future.

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
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

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.

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

```
1 ⟨*package⟩
2 \DeclareOption{showmeta}{\PassOptionsToPackage{\CurrentOption}{metakeys}}
3 \DeclareOption*{\PassOptionsToPackage{\CurrentOption}{omdoc}}
```

Finally, we need to declare the end of the option declaration section to LaTeX.

```
4 \ProcessOptions
5 ⟨/package⟩
```

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.

```
6 ⟨*package⟩
7 \RequirePackage{omtext}
8 \RequirePackage{modules}
9 \RequirePackage[hyperref]{ntheorem}
```

```
\theoremstyle{plain}
```
⟨/package⟩
⟨*ltxml⟩
```
# -*- CPERL -*-
package LaTeXML::Package::Pool;
use strict;
use LaTeXML::Package;
RequirePackage('omtext');
RequirePackage('modules');
```
⟨/ltxml⟩

Now, we define an auxiliary function that lowercases strings

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

EdNote:4

⟨*ltxml⟩
```
sub makeNCName {
  my ($name) = @_;
  my $ncname=$name;
  $ncname=~s/\s/_/g;  #Spaces to underscores
  $ncname="_$ncname" if $ncname!~/^(\w|_)/;  #Ensure start with letter or underscore
  ##More to come...
  $ncname;
}
```
⟨/ltxml⟩

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

⟨*ltxml⟩
```
sub simple_wrapper {
  #Deref if array reference
  my @input;
  foreach (@_) {
  if (ref $_ && $_ =~ /ARRAY/ && $_ !~ /LaTeXML/) {
      @input=(@input,@$_);
    } else
      { push (@input,$_); }
  }
  return '' if (!@input);
  @input = map(split(/\s*,\s*/,ToString($_)),@input);
  my $output=join(" ",@input);
  $output=~s/(^ )|[{}]//g; #remove leading space and list separator brackets
  $output||'';
```

[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.

```
}
sub hash_wrapper{
  #Deref if array reference
  my @input;
  foreach (@_) {
  if (ref $_ && $_ =~ /ARRAY/ && $_ !~ /LaTeXML/) {
      @input=(@input,@$_);
    } else
      { push (@input,$_); }
  }
  return '' if (!@input);
  @input = map(split(/\s*,\s*/,ToString($_)),@input);
  my $output=join(".sym #",@input);
  $output=~s/(^\.sym )|[{}]//g; #remove leading space and list separator brackets
  "#$output"||'';
}
⟨/ltxml⟩
```

5.2 Statements

\STpresent

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

```
⟨*package⟩
\def\define@statement@env#1{%
\newenvironment{#1}[1][]{\metasetkeys{omtext}{##1}\sref@target%
\ifx\omtext@display\st@flow\else%
\ifx\omtext@title\@empty\begin{ST#1Env}\else\begin{ST#1Env}[\omtext@title]\fi%
\ifx\sref@id\@empty\else\label{#1.\sref@id}\fi
\csname st@#1@initialize\endcsname\fi
\ifx\sref@id\@empty\sref@label@id{here}\else%
\sref@label@id{\STpresent{\csname ST#1EnvKeyword\endcsname}~\@currentlabel}\fi}
{\csname st@#1@terminate\endcsname\ifx\omtext@display\st@flow\else\end{ST#1Env}\fi}}
⟨/package⟩
```

assertion

```
⟨*package⟩
\newenvironment{assertion}[1][]{\metasetkeys{omtext}{#1}\sref@target%
\ifx\omtext@display\st@flow\else%
\ifx\omtext@title\@empty\begin{ST\omtext@type AssEnv}%
\else\begin{ST\omtext@type AssEnv}[\omtext@title]\fi\fi%
```

```
\ifx\omtext@type\@empty\sref@label@id{here}\else%
\sref@label@id{\STpresent{\csname ST\omtext@type AssEnvKeyword\endcsname}~\@currentlabel}\fi}
{\ifx\omtext@display\st@flow\else\end{ST\omtext@type AssEnv}\fi}
⟨/package⟩
⟨*ltxml⟩
DefEnvironment('{assertion} OptionalKeyVals:omtext',
  "<omdoc:assertion "
  .   "?&KeyVal(#1,'id')(xml:id='&KeyVal(#1,'id')')() "
  .   "?&KeyVal(#1,'theory')(theory='&KeyVal(#1,'theory')')() "
  .   "type='&lowcase(&KeyVal(#1,'type'))'>"
  .   "?&KeyVal(#1,'title')(<dc:title>&KeyVal(#1,'title')</dc:title>)()"
 .  "<omdoc:CMP>#body"
  ."</omdoc:assertion>\n");
⟨/ltxml⟩
```

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

```
⟨*package⟩
\def\st@theorem@kw{Theorem}
\def\st@lemma@kw{Lemma}
\def\st@proposition@kw{Proposition}
\def\st@corollary@kw{Corollary}
\def\st@conjecture@kw{Conjecture}
\def\st@falseconjecture@kw{Conjecture (false)}
\def\st@postulate@kw{Postulate}
\def\st@obligation@kw{Obligation}
\def\st@assumption@kw{Assumption}
\def\st@observation@kw{Observation}
```

Then we configure the presentation of the theorem environments

```
\theorembodyfont{\itshape}
\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`

```
\newtheorem{STtheoremAssEnv}{\st@theorem@kw}
\newtheorem{STlemmaAssEnv}[STtheoremAssEnv]{\st@lemma@kw}
\newtheorem{STpropositionAssEnv}[STtheoremAssEnv]{\st@proposition@kw}
\newtheorem{STcorollaryAssEnv}[STtheoremAssEnv]{\st@corollary@kw}
\newtheorem{STconjectureAssEnv}[STtheoremAssEnv]{\st@conjecture@kw}
\newtheorem{STfalseconjectureAssEnv}[STtheoremAssEnv]{\st@falseconjecture@kw}
\newtheorem{STpostulateAssEnv}[STtheoremAssEnv]{\st@postulate@kw}
\newtheorem{STobligationAssEnv}[STtheoremAssEnv]{\st@obligation@kw}
\newtheorem{STassumptionAssEnv}[STtheoremAssEnv]{\st@assumption@kw}
\newtheorem{STobservationAssEnv}[STtheoremAssEnv]{\st@observation@kw}
⟨/package⟩
```

`example`

```
⟨*package⟩
```

```
\def\st@example@initialize{}\def\st@example@terminate{}
\define@statement@env{example}
\def\st@example@kw{Example}
\theorembodyfont{\upshape}
\newtheorem{STexampleEnv}[STtheoremAssEnv]{\st@example@kw}
⟨/package⟩
⟨*ltxml⟩
DefEnvironment('{example} OptionalKeyVals:omtext',
       "<omdoc:example "
       . "?&KeyVal(#1,'id')(xml:id='&KeyVal(#1,'id')')() "
       . "?&KeyVal(#1,'for')(for='&hash_wrapper(&KeyVal(#1,'for'))')()>"
       . "?&KeyVal(#1,'title')(<dc:title>&KeyVal(#1,'title')</dc:title>)()"
      . "<omdoc:CMP>#body"
     . "</omdoc:example>\n");
⟨/ltxml⟩
```

axiom

```
⟨*package⟩
\def\st@axiom@initialize{}\def\st@axiom@terminate{}
\define@statement@env{axiom}
\def\st@axiom@kw{Axiom}
\theorembodyfont{\upshape}
\newtheorem{STaxiomEnv}[STtheoremAssEnv]{\st@axiom@kw}
⟨/package⟩
⟨*ltxml⟩
DefEnvironment('{axiom} OptionalKeyVals:omtext',
  "<omdoc:axiom "
  .   "?&KeyVal(#1,'id')(xml:id='&KeyVal(#1,'id')')()>"
  .   "?&KeyVal(#1,'title')(<dc:title>&KeyVal(#1,'title')</dc:title>)()"
 . "<omdoc:CMP>#body"
  . "</omdoc:axiom>\n");
⟨/ltxml⟩
```

symboldec

```
⟨*package⟩
\srefaddidkey{symboldec}
\addmetakey{symboldec}{functions}
\addmetakey{symboldec}{role}
\addmetakey*{symboldec}{title}
\addmetakey{symboldec}{name}
\addmetakey{symboldec}{subject}
\addmetakey*{symboldec}{display}
\def\symboldec@type{Symbol}
\newenvironment{symboldec}[1][]{\metasetkeys{symboldec}{#1}\sref@target\st@indeftrue%
\ifx\symboldec@display\st@flow\else{\stDMemph{\symboldec@type} \symboldec@name:}\fi%
\ifx\symboldec@title\@empty~\else~(\stDMemph{\symboldec@title})\par\fi}{}
⟨/package⟩
⟨*ltxml⟩
DefEnvironment('{symboldec} OptionalKeyVals:symboldec',
       "<omdoc:symbol "
```

```
      .  "?&KeyVal(#1,'id')(xml:id='&KeyVal(#1,'id')')"
      .                    "(xml:id='&makeNCName(&KeyVal(#1,'name')).def.sym')"
      .                "name='&KeyVal(#1,'name')'>"
      .  "?&KeyVal(#1,'title')(<dc:title>&KeyVal(#1,'title')</dc:title>)()"
      .  "<dc:description>#body"
      ."</omdoc:symbol>\n");
⟨/ltxml⟩
```

\symtype

```
⟨*package⟩
\newcommand{\symtype}[2]{Type (#1): $#2$}
⟨/package⟩
⟨*ltxml⟩
DefConstructor('\symtype{}{}',
  "<omdoc:type system='#1'><ltx:Math><ltx:XMath>#2</ltx:XMath></ltx:Math></omdoc:type>");
⟨/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`.

```
⟨*package⟩
\newif\ifst@indef\st@indeffalse
\newenvironment{definition}[1][]{\metasetkeys{omtext}{#1}\sref@target\st@indeftrue%
\ifx\omtext@display\st@flow\else%
\ifx\omtext@title\@empty\begin{STdefinitionEnv}\else\begin{STdefinitionEnv}[\omtext@title]\fi\f
\ifx\sref@id\@empty\sref@label@id{here}\else%
\sref@label@id{\STpresent{\csname STdefinitionEnvKeyword\endcsname}~\@currentlabel}\fi}
{\ifx\omtext@display\st@flow\else\end{STdefinitionEnv}\fi}
\def\st@definition@kw{Definition}
\theorembodyfont{\upshape}
\newtheorem{STdefinitionEnv}[STtheoremAssEnv]{\st@definition@kw}
⟨/package⟩
⟨*ltxml⟩
sub definitionBody {
    my ($doc, $keyvals, %props) = @_;
    my $for = $keyvals->getValue('for') if $keyvals;
    my $type = $keyvals->getValue('type') if $keyvals;
   my %for_attr=();
    if (ToString($for)) {
      $for = ToString($for);
      $for =~ s/^{(.+)}$/$1/eg;
      foreach (split(/,\s*/,$for)) {
        $for_attr{$_}=1;
     }}
   if ($props{theory}) {
    my @symbols = @{$props{defs} || []};
    foreach my $symb(@symbols) {
      next if $for_attr{$symb};
      $for_attr{$symb}=1;
      $doc->insertElement('omdoc:symbol', undef, (name=>$symb, "xml:id"=>makeNCName("$symb.def.
    }
```

```
  }
   my %attrs = ();
   $for = join(" ",(keys %for_attr));
   $attrs{'for'} = $for if $for;
   my $id = $keyvals->getValue('id') if $keyvals;
   $attrs{'xml:id'} = $id if $id;
   $attrs{'type'} = $type if $type;
   if ($props{theory}) {
     $doc->openElement('omdoc:definition', %attrs);
   } else {
     $attrs{'type'}='definition';
     $doc->openElement('omdoc:omtext', %attrs);
   }
   my $title = $keyvals->getValue('title') if $keyvals;
   if ($title) {
     $doc->openElement('omdoc:metadata');
     $doc->openElement('dc:title');
     $doc->absorb($title);
     $doc->closeElement('dc:title');}
   $doc->openElement('omdoc:CMP');
  $doc->absorb($props{body}) if $props{body};
  $doc->maybeCloseElement('omdoc:CMP');
   if ($props{theory}) {
     $doc->closeElement('omdoc:definition');
   } else {
     $doc->closeElement('omdoc:omtext');
   }
   return; }
DefEnvironment('{definition} OptionalKeyVals:omtext', sub{definitionBody(@_)},
  afterDigestBegin=>sub {
    my ($stomach, $whatsit) = @_;
    my @symbols = ();
    $whatsit->setProperty(theory=>LookupValue('current_module'));
    $whatsit->setProperty(defs=>\@symbols);
    AssignValue('defs', \@symbols);  return; },
  afterDigest => sub { AssignValue('defs', undef); return; });#$
⟨/ltxml⟩
```

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

```
⟨*package⟩
\def\notemph#1{{\bf{#1}}}
\def\st@notation@terminate{}
\def\st@notation@initialize{}
\define@statement@env{notation}
\def\st@notation@kw{Notation}
\theorembodyfont{\upshape}
\newtheorem{STnotationEnv}[STtheoremAssEnv]{\st@notation@kw}
⟨/package⟩
⟨*ltxml⟩
```

```
DefEnvironment('{notation} OptionalKeyVals:omtext',
  "<omdoc:definition "
 .   "?&KeyVal(#1,'id')(xml:id='&KeyVal(#1,'id').not')()"
 .   "?&KeyVal(#1,'for')(for='&simple_wrapper(&KeyVal(#1,'for'))')()>"
 . "?&KeyVal(#1,'title')(<dc:title>&KeyVal(#1,'title')</dc:title>)()"
 . "<omdoc:CMP>#body"
  . "</omdoc:definition>\n");
DefConstructor('\notatiendum OptionalKeyVals:notation {}',
               "<ltx:text class='notatiendum'>#2</ltx:text>");
```
⟨/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.

```
⟨*package⟩
\def\st@def@target#1#2{\def\@test{#1}%
\ifst@indef% if we are in a definition or such
\ifx\omtext@theory\@empty% if there is no theory attribute
\@ifundefined{mod@id}% if we are not in a module
{\PackageWarning{statements}{definiendum in unidentified module\MessageBreak
\protect\definiendum, \protect\defi,
\protect\defii, \protect\defiii\MessageBreak
can only be referenced when called in a module with id key}}%
{\ifx\@test\@empty%
\expandafter\sref@target@ifh{sref@#2@\mod@id @target}{}\else%
\expandafter\sref@target@ifh{sref@#1@\mod@id @target}{}\fi}%
\else\expandafter\sref@target@ifh{sref@#1@\omtext@theory @target}{}\fi%
\else\PackageError{statements}%
{definiendum outside definition context\MessageBreak
\protect\definiendum, \protect\defi,
\protect\defii, \protect\defiii\MessageBreak
do not make sense semantically outside a definition.\MessageBreak
Consider wrapping the defining phrase in a \protect\inlinedef}%
\fi}
⟨/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.

```
⟨*package⟩
\newcommand\@termdef[2][]{\def\@test{#1}%
```

```
\@ifundefined{mod@id}{}{\ifx\@test\@empty\def\@@name{#2}\else\def\@@name{#1}\fi%
\termdef{\mod@id @\@@name}{#2}}}
⟨/package⟩
```

\definiendum

```
⟨*package⟩
%\newcommand\definiendum[2][]{\st@def@target{#1}{#2}\@termdef[#1]{#2}\defemph{#2}}
\newcommand\definiendum[2][]{\st@def@target{#1}{#2}\defemph{#2}}
⟨/package⟩
⟨*ltxml⟩
DefConstructor('\definiendum [] {}',
       "<omdoc:term role='definiendum' name='#name' cd='#theory'>#2</omdoc:term>",
       afterDigest => sub {
 my ($stomach, $whatsit) = @_;
 my $addr = LookupValue('defs');
 my $name = $whatsit->getArg(1);
 $name = $whatsit->getArg(2) unless $name;
 $whatsit->setProperty(name=>$name->toString);
 push(@$addr, $name->toString) if ($addr and $name);
 $whatsit->setProperty(theory=>LookupValue('current_module'));
 return; });#$
⟨/ltxml⟩
```

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

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

```
⟨*package⟩
\newcommand{\defi}[2][]{\definiendum[#1]{#2}\omdoc@index[#1]{#2}}
⟨/package⟩
⟨*ltxml⟩
DefConstructor('\defi[]{}',
       "<omdoc:idx>"
      .  "<omdoc:idt>"
      .     "<omdoc:term role='definiendum' name='?#1(#1)(#2)' cd='#theory'>#2</omdoc:term>"
      .  "</omdoc:idt>"
      .  "<omdoc:ide index='default'><omdoc:idp>#2</omdoc:idp></omdoc:ide>"
      ."</omdoc:idx>",
       afterDigest => sub {
 my ($stomach, $whatsit) = @_;
 my $addr = LookupValue('defs');
 my $name = $whatsit->getArg(1);
 $name = $whatsit->getArg(2) unless $name;
```

```
 push(@$addr, $name->toString) if ($addr and $name);
 $whatsit->setProperty(theory=>LookupValue('current_module'));#$
 return; },
       alias=>'\defi');
⟨/ltxml⟩
```

\adefi

```
⟨*package⟩
\newcommand{\adefi}[3][]{\def\@test{#1}%
\ifx\@test\@empty\definiendum[#3]{#2}%
\else\definiendum[#1]{#2}\omdoc@index[#1]{#3}\fi}
⟨/package⟩
⟨*ltxml⟩
DefConstructor('\adefi[]{}{}',
       "<omdoc:idx>"
      .  "<omdoc:idt>"
      .     "<omdoc:term role='definiendum' name='?#1(#1)(#3)' cd='#theory'>#2</omdoc:term>"
      .  "</omdoc:idt>"
      .  "<omdoc:ide index='default'><omdoc:idp>#3</omdoc:idp></omdoc:ide>"
      ."</omdoc:idx>",
       afterDigest => sub {
 my ($stomach, $whatsit) = @_;
 my $addr = LookupValue('defs');
 my $name = $whatsit->getArg(1);
 $name = $whatsit->getArg(3) unless $name;
 push(@$addr, $name->toString) if ($addr and $name);
 $whatsit->setProperty(theory=>LookupValue('current_module'));#$
 return; },
      alias=>'\adefi');
⟨/ltxml⟩
```

\defii

```
⟨*package⟩
\newcommand{\defii}[3][]{\st@def@target{#1}{#2-#3}\defemph{#2 #3}\@twin[#1]{#2}{#3}}
⟨/package⟩
⟨*ltxml⟩
DefConstructor('\defii[]{}{}',
       "<omdoc:idx>"
      .  "<omdoc:idt>"
      .    "<omdoc:term role='definiendum' name='?#1(#1)(&dashed(#2,#3))' cd='#theory'>"
      .      "#2 #3"
      .    "</omdoc:term>"
      .  "</omdoc:idt>"
      .  "<omdoc:ide index='default'>"
      .    "<omdoc:idp>#2</omdoc:idp>"
      .    "<omdoc:idp>#3</omdoc:idp>"
      .  "</omdoc:ide>"
      ."</omdoc:idx>",
       afterDigest => sub {
 my ($stomach, $whatsit) = @_;
```

```
my $addr = LookupValue('defs');
 my $name = $whatsit->getArg(1);
 $name = $name->toString if $name;
 $name = $whatsit->getArg(2)->toString.'-'.$whatsit->getArg(3)->toString unless $name;
 push(@$addr, $name) if ($addr and $name);
 $whatsit->setProperty(theory=>LookupValue('current_module'));
 return; },
       alias=>'\defii');#$
⟨/ltxml⟩
```

\adefii

```
⟨*package⟩
\newcommand{\adefii}[4][]{\def\@test{#1}%
\ifx\@test\@empty\definiendum[#3-#4]{#2}%
\else\definiendum[#1]{#2}\@twin[#1]{#3}{#4}\fi}
⟨/package⟩
⟨*ltxml⟩
DefConstructor('\adefii[]{}{}{}',
       "<omdoc:idx>"
      .  "<omdoc:idt>"
      .    "<omdoc:term role='definiendum' name='?#1(#1)(&dashed(#3,#4))' cd='#theory'>"
      .       "#2"
      .    "</omdoc:term>"
      .  "</omdoc:idt>"
      .  "<omdoc:ide index='default'>"
      .    "<omdoc:idp>#3</omdoc:idp>"
      .    "<omdoc:idp>#4</omdoc:idp>"
      .  "</omdoc:ide>"
      ."</omdoc:idx>",
       afterDigest => sub {
 my ($stomach, $whatsit) = @_;
 my $addr = LookupValue('defs');
 my $name = $whatsit->getArg(1);
 $name = $name->toString if $name;
 $name = $whatsit->getArg(3)->toString.'-'.$whatsit->getArg(4)->toString unless $name;
 push(@$addr, $name) if ($addr and $name);
 $whatsit->setProperty(theory=>LookupValue('current_module'));
 return; },
       alias=>'\defii');#$
⟨/ltxml⟩
```

\defiii

```
⟨*package⟩
\newcommand{\defiii}[4][]{\st@def@target{#1}{#2-#3-#4}\defemph{#2 #3 #4}\@atwin[#1]{#2}{#3}{#4}
⟨/package⟩
⟨*ltxml⟩
DefConstructor('\defiii[]{}{}{}',
       "<omdoc:idx>"
       . "<omdoc:idt>"
         . "<omdoc:term role='definiendum' cd='#theory' name='?#1(#1)(&dashed(#2,#3,#4))'>#2 #3
```

```
       . "</omdoc:idt>"
       . "<omdoc:ide index='default'>"
         . "<omdoc:idp>#2</omdoc:idp>"
         . "<omdoc:idp>#3</omdoc:idp>"
         . "<omdoc:idp>#4</omdoc:idp>"
       . "</omdoc:ide>"
     . "</omdoc:idx>",
       afterDigest => sub {
 my ($stomach, $whatsit) = @_;
 my $addr = LookupValue('defs');
 my $name = $whatsit->getArg(1);
 $name = $name->toString if $name;
 $name = $whatsit->getArg(2)->toString.'-'.$whatsit->getArg(3)->toString.'-'.$whatsit->getArg(4
 push(@$addr, $name) if ($addr and $name);
 $whatsit->setProperty(theory=>LookupValue('current_module'));
 return; },
       alias=>'\defiii');
⟨/ltxml⟩
```

\adefiii

```
⟨*package⟩
\newcommand{\adefiii}[5][]{\def\@test{#1}%
\ifx\@test\@empty\definiendum[#3-#4-#5]{#2}%
\else\definiendum[#1]{#2}\@atwin[#1]{#3}{#4}{#5}\fi}
⟨/package⟩
⟨*ltxml⟩
DefConstructor('\adefiii[]{}{}{}{}',
       "<omdoc:idx>"
       . "<omdoc:idt>"
       . "<omdoc:term role='definiendum' cd='#theory' name='?#1(#1)(&dashed(#3,#4,#5))'>#2</omd
       . "</omdoc:idt>"
       . "<omdoc:ide index='default'>"
         . "<omdoc:idp>#3</omdoc:idp>"
         . "<omdoc:idp>#4</omdoc:idp>"
         . "<omdoc:idp>#5</omdoc:idp>"
       . "</omdoc:ide>"
     . "</omdoc:idx>",
       afterDigest => sub {
 my ($stomach, $whatsit) = @_;
 my $addr = LookupValue('defs');
 my $name = $whatsit->getArg(1);
 $name = $name->toString if $name;
 $name = $whatsit->getArg(3)->toString.'-'.$whatsit->getArg(4)->toString.'-'.$whatsit->getArg(5
 push(@$addr, $name) if ($addr and $name);
 $whatsit->setProperty(theory=>LookupValue('current_module'));
 return; },
       alias=>'\defiii');
⟨/ltxml⟩
```

\inlineex

```
⟨*package⟩
\newcommand{\inlineex}[2][]{\metasetkeys{omtext}{#1}\sref@target\sref@label@id{here}#2}
⟨/package⟩
⟨*ltxml⟩
DefConstructor('\inlineex OptionalKeyVals:omtext {}',
               "<ltx:text class='example'>#2</ltx:text>");
⟨/ltxml⟩
```

\inlinedef

```
⟨*package⟩
\newcommand{\inlinedef}[2][]{\metasetkeys{omtext}{#1}\sref@target\sref@label@id{here}\st@indeft
⟨/package⟩
⟨*ltxml⟩
DefConstructor('\inlinedef OptionalKeyVals:omtext {}', sub {
 my ($document, $keyvals, $body, %props) = @_;
 my $for = $keyvals->getValue('for') if $keyvals;
 my %for_attr=();
 if (ToString($for)) {
   $for = ToString($for);
   $for =~ s/^{(.+)}$/$1/eg;
   foreach (split(/,\s*/,$for)) {
     $for_attr{$_}=1;
   }}
 my @symbols = @{$props{defs} || []};
 #Prepare for symbol insertion -insert before the parent of the closest ancestor CMP element
  my $original_node = $document->getNode;
 my $xc = XML::LibXML::XPathContext->new( $original_node );
 $xc->registerNs('omdoc', 'http://omdoc.org/ns');
 my ($statement_ancestor) = $xc->findnodes('./ancestor::omdoc:CMP/..');
 foreach my $symb(@symbols) {
   next if $for_attr{$symb};
   $for_attr{$symb}=1;
   my $symbolnode = XML::LibXML::Element->new('symbol');
   $symbolnode->setAttribute(name=>$symb);
   $symbolnode->setAttribute("xml:id"=>makeNCName("$symb.def.sym"));
   $statement_ancestor->parentNode->insertBefore($symbolnode,$statement_ancestor);
 }
 #Restore the insertion point
 $document->setNode($original_node);
 my %attrs = ();
 $for = join(" ",(keys %for_attr));
 $attrs{'for'} = $for if $for;
 my $id = $keyvals->getValue('id') if $keyvals;
 $attrs{'xml:id'} = $id if $id;
$attrs{'class'} = 'inlinedef';
 $document->openElement('ltx:text',%attrs);
 $document->absorb($body);
$document->closeElement('ltx:text'); },
 #Prepare 'defs' hooks for \defi and \definiendum symbol names
  beforeDigest=>sub {
```

```
    my @symbols = ();
    AssignValue('defs', \@symbols); return; },
 #Adopt collected names as 'defs' property, remove hooks
  afterDigest=>sub {
    my ($stomach, $whatsit) = @_;
    my $defsref = LookupValue('defs');
    my @defs = @$defsref;
    $whatsit->setProperty('defs',\@defs);
    AssignValue('defs',undef);
 return; });
⟨/ltxml⟩
```

5.3 Cross-Referencing Symbols and Concepts

\termref@set The `term` macro uses the `cd` and `name` keys for hyperlinking to create hyper-refs, if the `hyperref` package is loaded: We first see if the `cd` key was given, if not we define it as the local module identifier.

```
⟨*package⟩
\addmetakey[\mod@id]{termref}{cd}
\addmetakey{termref}{cdbase}
\addmetakey{termref}{name}
\addmetakey{termref}{role}
\def\termref@set#1#2{\def\termref@name{#2}\metasetkeys{termref}{#1}}
```

\termref

```
\newcommand{\termref}[2][]{\metasetkeys{termref}{#1}\st@termref{#2}}
⟨/package⟩
⟨*ltxml⟩
DefConstructor('\termref OptionalKeyVals:termref {}',
                "<omdoc:term "
               .  "?&KeyVal(#1,'cdbase')(cdbase='&KeyVal(#1,'cdbase')')() "
              .  "cd='?&KeyVal(#1,'cd')(&KeyVal(#1,'cd'))(#module)' "
               .  "name='&KeyVal(#1,'name')'>"
               .  "#2"
               ."</omdoc:term>",
      afterDigest=>sub{$_[1]->setProperty(module=>LookupValue('current_module'))});
⟨/ltxml⟩%$
```

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.

```
⟨*package⟩
\def\st@termref#1{\ifx\termref@name\@empty\def\termref@name{#1}\fi%
\ifx\termref@cdbase\@empty\mod@termref\termref@cd\termref@name{#1}%
\else\sref@href@ifh\termref@cdbase{#1}\fi}
⟨/package⟩
```

\tref*

```
⟨ltxml⟩RawTeX('
⟨*package | ltxml⟩
\newcommand\atrefi[3][]{\def\@test{#1}\ifx\@test\@empty\termref[name=#3]{#2}\else\termref[cd=#1
\newcommand\atrefii[4][]{\atrefi[#1]{#2}{#3-#4}}
\newcommand\atrefiii[5][]{\atrefi[#1]{#2}{#3-#4-#5}}
```

\tref*

```
\newcommand{\trefi}[2][]{\atrefi[#1]{#2}{#2}}
\newcommand{\trefii}[3][]{\atrefi[#1]{#2 #3}{#2-#3}}
\newcommand{\trefiii}[4][]{\atrefi[#1]{#2 #3 #4}{#2-#3-#4}}
⟨/package | ltxml⟩
⟨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

```
⟨*package⟩
\providecommand{\termemph}[1]{#1}
\providecommand{\defemph}[1]{{\textbf{#1}}}
\providecommand{\stDMemph}[1]{{\textbf{#1}}}
⟨/package⟩
```

\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*⟩.

```
⟨*package⟩
\newcommand{\symref}[2]{\@nameuse{mod@symref@#1}{#2}}
⟨/package⟩
⟨*ltxml⟩
DefConstructor('\symref{}{}',
               "<omdoc:term cd='&LookupValue('symdef.#1.cd')' name='&LookupValue('symdef.#1.nam
              .  "#2"
              ."</omdoc:term>");
⟨/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.

```
⟨*ltxml⟩
Tag('omdoc:assertion',afterOpen=>\&numberIt,afterClose=>\&locateIt);
Tag('omdoc:definition',afterOpen=>\&numberIt,afterClose=>\&locateIt);
Tag('omdoc:example',afterOpen=>\&numberIt,afterClose=>\&locateIt);
Tag('omdoc:requation',afterOpen=>\&numberIt,afterClose=>\&locateIt);
Tag('omdoc:axiom',afterOpen=>\&numberIt,afterClose=>\&locateIt);
```

```
Tag('omdoc:symbol',afterOpen=>\&numberIt,afterClose=>\&locateIt);
Tag('omdoc:type',afterOpen=>\&numberIt,afterClose=>\&locateIt);
Tag('omdoc:term',afterOpen=>\&numberIt,afterClose=>\&locateIt);
⟨/ltxml⟩
```

5.5 Deprecated Functionality

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

def

```
⟨ltxml⟩RawTeX('
⟨*package | ltxml⟩
\newcommand\defin[2][]{\defi[#1]{#2}%
\PackageWarning{statements}{\protect\defin\space is deprecated, use \protect\defi\space instead
\newcommand\twindef[3][]{\defii[#1]{#2}{#3}%
\PackageWarning{statements}{\protect\twindef\space is deprecated, use \protect\defii\space inst
\newcommand\atwindef[4][]{\defiii[#1]{#2}{#3}{#4}%
\PackageWarning{statements}{\protect\atwindef\space is deprecated, use \protect\defiii\space in
\newcommand\definalt[3][]{\adefi[#1]{#2}{#3}%
\PackageWarning{statements}{\protect\definalt\space is deprecated, use \protect\adefi\space ins
\newcommand\twindefalt[4][]{\adefii[#1]{#2}{#3}{#4}%
\PackageWarning{statements}{\protect\twindefalt\space is deprecated, use \protect\adefii\space
\newcommand\atwindefalt[5][]{\adefiii[#1]{#2}{#3}{#4}{#5}%
\PackageWarning{statements}{\protect\atwindefalt\space is deprecated, use \protect\adefiii\spac
```

def

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

5.6 Finale

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

```
⟨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.

References

[KGA10] Michael Kohlhase, Deyan Ginev, and Rares Ambrus. *modules.sty: Semantic Macros and Module Scoping in sTeX*. Self-documenting LaTeX package. Comprehensive TeX Archive Network (CTAN), 2010. URL: http://www.ctan.org/get/macros/latex/contrib/stex/modules/modules.pdf.

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

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

[Koh10a] Michael Kohlhase. *metakeys.sty: A generic framework for extensible Metadata in LaTeX*. Self-documenting LaTeX package. Comprehensive TeX Archive Network (CTAN), 2010. URL: http://www.ctan.org/tex-archive/macros/latex/contrib/stex/metakeys/metakeys.pdf.

[Koh10b] Michael Kohlhase. *omdoc.sty/cls: Semantic Markup for Open Mathematical Documents in LaTeX*. Self-documenting LaTeX package. Comprehensive TeX Archive Network (CTAN), 2010. URL: http://www.ctan.org/tex-archive/macros/latex/contrib/stex/omdoc/omdoc.pdf.

[Koh10c] Michael Kohlhase. *sref.sty: Semantic Crossreferencing in LaTeX*. Self-documenting LaTeX package. Comprehensive TeX Archive Network (CTAN), 2010. URL: http://www.ctan.org/tex-archive/macros/latex/contrib/stex/sref/sref.pdf.

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

[Ste] *Semantic Markup for LaTeX*. Project Homepage. URL: http://trac.kwarc.info/sTeX/ (visited on 02/22/2011).