

# Semantic Markup for Mathematical Statements\*

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

November 4, 2010

## Abstract

The `statements` package is part of the  $\S\TeX$  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  $\S\TeX$  files. This structure can be used by MKM systems for added-value services, either directly from the  $\S\TeX$  sources, or after translation.

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>The User Interface</b>	<b>2</b>
2.1	Package Options . . . . .	2
2.2	Statements . . . . .	2
2.3	Cross-Referencing Symbols and Concepts . . . . .	8
<b>3</b>	<b>Configuration of the Presentation</b>	<b>9</b>
<b>4</b>	<b>Limitations</b>	<b>9</b>
<b>5</b>	<b>The Implementation</b>	<b>9</b>
5.1	Package Options . . . . .	10
5.2	Statements . . . . .	12
5.3	Cross-Referencing Symbols and Concepts . . . . .	22
5.4	Providing IDs for OMDoc Elements . . . . .	24
5.5	Finale . . . . .	24

---

\*Version v1.0 (last revised 2010/06/25)

# 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  $\mathcal{S}\text{T}_{\text{E}}\text{X}$  sources, or after translation. Even though it is part of the  $\mathcal{S}\text{T}_{\text{E}}\text{X}$  collection, it can be used independently, like its sister package `sproofs`.

$\mathcal{S}\text{T}_{\text{E}}\text{X}$  [Koh08; Ste] is a version of  $\text{T}_{\text{E}}\text{X}/\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  that allows to markup  $\text{T}_{\text{E}}\text{X}/\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  documents semantically without leaving the document format, essentially turning  $\text{T}_{\text{E}}\text{X}/\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  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.

`display=` Since they have the same semantic status, they must both be marked up, but styled differently. We distinguish between these two presentational forms with the `display` key, which is allowed on all statement environments. If it has the value `block` (the default), then the statement will be presented in a paragraph of its own, have explicit discourse markers for its begin and end, possibly numbering,

etc. If it has the value `flow`, then no extra presentation will be added the semantic information is invisible to the reader. Another key that is present on all statement environments in the `id` key it allows to identify the statement with a name and to reference it with the semantic referencing infrastructure provided by the `sref` package [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 assertion is given in the `type` key. So instead of `\begin{Lemma}` we have to write `\begin{assertion}[type=lemma]` (see Example 1 for an example).

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

will lead to the result

**Lemma 1**  $\sum_{i=1}^n 2i - 1 = n^2$

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

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

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

### 2.2.2 Symbols

`symboldec` The `symboldec` environment can be used for declaring concepts and symbols. Note 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
<b>theorem, proposition</b>	an important assertion with a proof
Note that the meaning of <b>theorem</b> (in this case the existence of a proof) is not enforced by OMDoc applications. It can be appropriate to give an assertion the <b>theorem</b> , if the author knows of a proof (e.g. in the literature), but has not formalized it in OMDoc yet.	
<b>lemma</b>	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.	
<b>corollary</b>	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.	
<b>postulate, conjecture</b>	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.	
<b>false-conjecture</b>	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.	
<b>obligation, assumption</b>	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).	
<b>observation</b>	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

`definition` The `definition` environment is used for marking up mathematical definitions. Its peculiarity is that it defines (i.e. gives a meaning to) new mathematical concepts or objects. These are identified by the `\definiendum` macro, which is used 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 `\defin{word}` macro combines the functionality of the `\definiendum` macro with index markup from the `omdoc` package [Koh10b]: use `\defin[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 `\twindex` and `\atwindex` for (adjectivized) two-word compounds. Finally, the variants `\twindexalt` and `\atwindexalt` have an additional first argument that allows to specify an alternative text; see Figure 5

`\twindex`  
`\atwindex`  
`\twindexalt`  
`\atwindexalt`

Note that the `\definiendum`, `\defin`, `\twindex`, and `\atwindex` 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 “...*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 $$ such $$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	result	index
<code>\defin{concept}</code>		
concept	concept	concept
<code>\defin[csymbol]{concept}</code>		
csymbol	concept	concept
<code>\definalt[csymbol]{concepts}{concept}</code>		
csymbol	concepts	concept
<code>\twiundef{concept}{group}</code>		
concept-group	concept group	concept group, group - , concept
<code>\atwiundef{small}{concept}{group}</code>		
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<sup>1</sup>. Therefore, the `\termref` can be used to make this information explicit. It takes the keys

`cdbase` to specify a URI (a path actually, since L<sup>A</sup>T<sub>E</sub>X 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 *text* with (if the `hyperref` package is loaded) a hyperlink to the definition in module *cd* that defines the concept *name*, i.e. that contains `\definiendum[name]{something}`.

Just as the `\definiendum` macro has the variants `\twindex` and `\atwindex` for composita, the `\termref` has variants `\twinref` and `\atwindex` that take two and three arguments for the parts of the compositum. Generally, concepts that are marked up by `\definiendum{name}` or `\defin\meta{name}` in the definition are referenced by `\termref\meta{name}`, concepts defined via `\twindex{first}{second}` with `\twinref{first}{second}` and analogously for `\atwindex` and `\atwindex`.

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

`\symref` `statements` package supplies the `\symref` macro. Like `\termref`, and invocation

<sup>1</sup>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`.



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.)

### 3 Configuration of the Presentation

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

`\termemph` The `\termemph` macro does the same for the style for `\termin`, 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.<sup>1</sup>

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.<sup>2</sup>

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.<sup>3</sup>

### 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

### 5 The Implementation

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

<sup>1</sup>EDNOTE: function declarations

<sup>2</sup>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`.

<sup>3</sup>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

## 5.1 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 L<sup>A</sup>T<sub>E</sub>X.

```
4 \ProcessOptions
5 </package>
```

The next measure is to ensure that some S<sup>T</sup>E<sub>X</sub> 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 L<sup>A</sup>-T<sub>E</sub>X<sub>M</sub>L, 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}
10 \theoremstyle{plain}
11 </package>
12 <*ltxml>
13 # -*- PERL -*-
14 package LaTeXML::Package::Pool;
15 use strict;
16 use LaTeXML::Package;
17 RequirePackage('omtext');
```

```
18 RequirePackage('modules');
19 </!xml>
```

Now, we define an auxiliary function that lowercases strings

```
20 <!--xml-->
21 sub lowercase {my ($string) = @_; $string ? return lc(ToString($string)) : return('')}##$
22 sub dashed { join('-',map($_->toString,@_));}##$
23 </!xml-->
```

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.<sup>4</sup>

```
24 <!--xml-->
25 sub makeNCName {
26   my ($name) = @_;
27   my $ncname=$name;
28   $ncname=~s/\s/_/g; #Spaces to underscores
29   $ncname="_$ncname" if $ncname!~/^\w|_/; #Ensure start with letter or underscore
30   ##More to come...
31   $ncname;
32 }
33 </!xml-->
```

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

```
34 <!--xml-->
35 sub simple_wrapper {
36   #Deref if array reference
37   my @input;
38   foreach (@_) {
39     if (ref $_ && $_ =~ /ARRAY/ && $_ !~ /LaTeXML/) {
40       @input=(@input,@$_);
41     } else
42       { push (@input,$_); }
43   }
44   return '' if (!@input);
45   @input = map(split(/\s*,\s*/,ToString($_)),@input);
46   my $output=join(" ",@input);
47   $output=~s/^(^ )|[\{\}]/g; #remove leading space and list separator brackets
48   $output||'';
49 }
50 sub hash_wrapper{
51   #Deref if array reference
52   my @input;
53   foreach (@_) {
54     if (ref $_ && $_ =~ /ARRAY/ && $_ !~ /LaTeXML/) {
55       @input=(@input,@$_);
56     } else
```

---

<sup>4</sup>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.

```

57     { push (@input,$_); }
58   }
59   return '' if (!@input);
60   @input = map(split(/\s*,\s*/,ToString($_)),@input);
61   my $output=join(".sym #",@input);
62   $output=~s/(\.sym )|[\{\}]/g; #remove leading space and list separator brackets
63   "#$output"||'';
64 }
65 </ltxml>

```

## 5.2 Statements

`\STpresent`

```

66 <*package>
67 \providecommand\STpresent[1]{#1}
68 </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.

```

69 <*package>
70 \def\define@statement@env#1{%
71 \newenvironment{#1}[1][\metasetkeys{omtext}{#1}\sref@target%
72 \ifx\omtext@display\st@flow\else%
73 \ifx\omtext@title\@empty\begin{ST#1Env}\else\begin{ST#1Env}[\omtext@title]\fi%
74 \ifx\sref@id\@empty\else\label{#1.\sref@id}\fi
75 \csname st@#1@initialize\endcsname\fi
76 \ifx\sref@id\@empty\sref@label@id{here}\else%
77 \sref@label@id{\STpresent{\csname ST#1EnvKeyword\endcsname}~\@currentlabel}\fi}
78 {\csname st@#1@terminate\endcsname\ifx\omtext@display\st@flow\else\end{ST#1Env}\fi}}
79 </package>

```

`assertion`

```

80 <*package>
81 \newenvironment{assertion}[1][\metasetkeys{omtext}{#1}\sref@target%
82 \ifx\omtext@display\st@flow\else%
83 \ifx\omtext@title\@empty\begin{ST\omtext@type AssEnv}%
84 \else\begin{ST\omtext@type AssEnv}[\omtext@title]\fi\fi%
85 \ifx\omtext@type\@empty\sref@label@id{here}\else%
86 \sref@label@id{\STpresent{\csname ST\omtext@type AssEnvKeyword\endcsname}~\@currentlabel}\fi}
87 {\ifx\omtext@display\st@flow\else\end{ST\omtext@type AssEnv}\fi}
88 </package>
89 <*ltxml>
90 DefCMPEnvironment(' {assertion} OptionalKeyVals:omtext',
91   "<omdoc:assertion "
92   . " ?&KeyVal{#1,'id'}(xml:id='&KeyVal{#1,'id}')")() "

```

```

93 .   "?&KeyVal(#1,'theory')(theory='&KeyVal(#1,'theory'))()"
94 .   "type='&lowercase(&KeyVal(#1,'type'))'"
95 .   "?&KeyVal(#1,'title')(<dc:title>&KeyVal(#1,'title')</dc:title>)"
96 .   "<omdoc:CMPE><omdoc:p>#body"
97 .   "</omdoc:assertion>\n");
98 </ltxml>

```

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

```

99 <*package>
100 \def\st@theorem@kw{Theorem}
101 \def\st@lemma@kw{Lemma}
102 \def\st@proposition@kw{Proposition}
103 \def\st@corollary@kw{Corollary}
104 \def\st@conjecture@kw{Conjecture}
105 \def\st@falseconjecture@kw{Conjecture (false)}
106 \def\st@postulate@kw{Postulate}
107 \def\st@obligation@kw{Obligation}
108 \def\st@assumption@kw{Assumption}
109 \def\st@observation@kw{Observation}

```

Then we configure the presentation of the theorem environments

```

110 \theorembodyfont{\itshape}
111 \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`

```

112 \newtheorem{STtheoremAssEnv}{\st@theorem@kw}
113 \newtheorem{STlemmaAssEnv}[STtheoremAssEnv]{\st@lemma@kw}
114 \newtheorem{STpropositionAssEnv}[STtheoremAssEnv]{\st@proposition@kw}
115 \newtheorem{STcorollaryAssEnv}[STtheoremAssEnv]{\st@corollary@kw}
116 \newtheorem{STconjectureAssEnv}[STtheoremAssEnv]{\st@conjecture@kw}
117 \newtheorem{STfalseconjectureAssEnv}[STtheoremAssEnv]{\st@falseconjecture@kw}
118 \newtheorem{STpostulateAssEnv}[STtheoremAssEnv]{\st@postulate@kw}
119 \newtheorem{STobligationAssEnv}[STtheoremAssEnv]{\st@obligation@kw}
120 \newtheorem{STassumptionAssEnv}[STtheoremAssEnv]{\st@assumption@kw}
121 \newtheorem{STobservationAssEnv}[STtheoremAssEnv]{\st@observation@kw}
122 </package>

```

`example`

```

123 <*package>
124 \def\st@example@initialize{}\def\st@example@terminate{}
125 \define@statement@env{example}
126 \def\st@example@kw{Example}
127 \theorembodyfont{\upshape}
128 \newtheorem{STexampleEnv}[STtheoremAssEnv]{\st@example@kw}
129 </package>
130 <*ltxml>
131 DefCMPEnvironment('{example} OptionalKeyVals:omtext',

```

```

132     "<omdoc:example "
133     . "?&KeyVal(#1,'id')(xml:id='&KeyVal(#1,'id')')() "
134     . "?&KeyVal(#1,'for')(for='&hash_wrapper(&KeyVal(#1,'for'))')()>"
135     . "?&KeyVal(#1,'title')(<dc:title>&KeyVal(#1,'title')</dc:title>())"
136     . "<omdoc:CMP><omdoc:p>#body"
137     . "</omdoc:example>\n";
138 </ltxml>

```

axiom

```

139 <*package>
140 \def\st@axiom@initialize{}\def\st@axiom@terminate{}
141 \define@statement@env{axiom}
142 \def\st@axiom@kw{Axiom}
143 \theorembodyfont{\upshape}
144 \newtheorem{STaxiomEnv}[STtheoremAssEnv]{\st@axiom@kw}
145 </package>
146 <*ltxml>
147 DefCMPEnvironment('{axiom} OptionalKeyVals:omtext',
148   "<omdoc:axiom "
149   . "?&KeyVal(#1,'id')(xml:id='&KeyVal(#1,'id')')()>"
150   . "?&KeyVal(#1,'title')(<dc:title>&KeyVal(#1,'title')</dc:title>())"
151   . "<omdoc:CMP><omdoc:p>#body"
152   . "</omdoc:axiom>\n");
153 </ltxml>

```

symboldec

```

154 <*package>
155 \srefaddidkey{symboldec}
156 \addmetakey{symboldec}{functions}
157 \addmetakey{symboldec}{role}
158 \addmetakey*{symboldec}{title}
159 \addmetakey{symboldec}{name}
160 \addmetakey*{symboldec}{display}
161 \def\symboldec@type{Symbol}
162 \newenvironment{symboldec}[1][\metasetkeys{symboldec}{#1}\sref@target\st@indeftrue%
163 \ifx\symboldec@display\st@flow\else\stDMemph{\symboldec@type} \symboldec@name:}\fi%
164 \ifx\symboldec@title\@empty~\else~(\stDMemph{\symboldec@title})\par\fi}{}
165 </package>
166 <*ltxml>
167 DefEnvironment('{symboldec} OptionalKeyVals:symboldec',
168   "<omdoc:symbol "
169   . " ?&KeyVal(#1,'id')(xml:id='&KeyVal(#1,'id')')"
170   . " (xml:id='&makeNCName(&KeyVal(#1,'name'))).def.sym'"
171   . " name='&KeyVal(#1,'name')'>"
172   . "<dc:description>#body</dc:description>"
173   . "</omdoc:symbol>\n");
174 </ltxml>

```

\symtype

```

175 <*package>
176 \newcommand{\syntype}[2]{Type (#1): #2}
177 </package>
178 <*txml>
179 DefConstructor('\syntype{}-{}',
180 "<omdoc:type system='#1'>#2</omdoc:type>");
181 </txml>

```

**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`.

```

182 <*package>
183 \newif\ifst@indef\st@indeffalse
184 \newenvironment{definition}[1][\metasetkeys{omtext}{#1}\sref@target\st@indeftrue%
185 \ifx\omtext@display\st@flow\else%
186 \ifx\omtext@title\@empty\begin{STdefinitionEnv}\else\begin{STdefinitionEnv}[\omtext@title]\fi%
187 \ifx\sref@id\@empty\sref@label@id{here}\else%
188 \sref@label@id{\STpresent{\csname STdefinitionEnvKeyword\endcsname}~\@currentlabel}\fi}
189 {\ifx\omtext@display\st@flow\else\end{STdefinitionEnv}\fi}
190 \def\st@definition@kw{Definition}
191 \theorembodyfont{\upshape}
192 \newtheorem{STdefinitionEnv}[STtheoremAssEnv]{\st@definition@kw}
193 </package>
194 <*txml>
195 sub definitionBody {
196   my ($doc, $keyvals, %props) = @_;
197   my $for = $keyvals->getValue('for') if $keyvals;
198   my $type = $keyvals->getValue('type') if $keyvals;
199   my %for_attr=();
200   if (ToString($for)) {
201     $for = ToString($for);
202     $for =~ s/^(.+)$/1/eg;
203     foreach (split(/, \s*/, $for)) {
204       $for_attr{$_}=1;
205     }
206   }
207   my @symbols = @{$props{defs} || []};
208   foreach my $symb(@symbols) {
209     next if $for_attr{$symb};
210     $for_attr{$symb}=1;
211   }
212   my %attrs = ();
213   $for = join(" ", (keys %for_attr));
214   $attrs{'for'} = $for if $for;
215   my $id = $keyvals->getValue('id') if $keyvals;
216   $attrs{'xml:id'} = $id if $id;
217   $attrs{'type'} = $type if $type;
218   $doc->openElement('omdoc:definition', %attrs);
219   my $title = $keyvals->getValue('title') if $keyvals;
220   if ($title) {
221     $doc->openElement('omdoc:metadata');

```

```

222     $doc->openElement('dc:title');
223     $doc->absorb($title);
224     $doc->closeElement('dc:title');}
225 $doc->openElement('omdoc:CMP');
226 $doc->openElement('omdoc:p');
227 $doc->absorb($props{body}) if $props{body};
228 $doc->maybeCloseElement('omdoc:p');
229 $doc->maybeCloseElement('omdoc:CMP');
230 $doc->closeElement('omdoc:definition');
231 return; }
232 DefCMPEnvironment('{definition} OptionalKeyVals:omtext', sub{definitionBody(@_)},
233 afterDigestBegin=>sub {
234   my ($stomach, $whatsit) = @_;
235   my @symbols = ();
236   $whatsit->setProperty(defs=>\@symbols);
237   AssignValue('defs', \@symbols); return; },
238 afterDigest => sub { AssignValue('defs', undef); return; });
239 </ltxml>

```

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

```

240 <*package>
241 \def\notemph#1{{\bf{#1}}}
242 \def\st@notation@terminate{}
243 \def\st@notation@initialize{}
244 \define@statement@env{notation}
245 \def\st@notation@kw{Notation}
246 \theorembodyfont{\upshape}
247 \newtheorem{STnotationEnv}[STtheoremAssEnv]{\st@notation@kw}
248 </package>
249 <*ltxml>
250 DefCMPEnvironment('{notation} OptionalKeyVals:omtext',
251 "<omdoc:definition "
252 . " ?&KeyVal{#1,'id'}(xml:id='&KeyVal{#1,'id'}.not')()"
253 . " ?&KeyVal{#1,'for'}(for='&simple_wrapper(&KeyVal{#1,'for'})')()>"
254 . " ?&KeyVal{#1,'title'}(<dc:title>&KeyVal{#1,'title'}</dc:title>)"
255 . "<omdoc:CMP><omdoc:p>#body"
256 . "</omdoc:definition>\n");
257 DefConstructor('\notatiendum OptionalKeyVals:notation {}',
258 "<omdoc:phrase type='notation'>#2</omdoc:phrase>");
259 </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`, `\defin`, `\twindef`, and `\atwindef` 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.

```

260 <*package>

```



```

261 \def\st@def@target#1#2{\def\@test{#1}%
262 \ifst@indef% if we are in a definition or such
263 \ifx\omtext@theory\@empty% if there is no theory attribute
264 \@ifundefined{mod@id}% if we are not in a module
265 {\PackageWarning{statements}{definiendum in unidentified module\MessageBreak
266 \protect\definiendum, \protect\defin,
267 \protect\twindex, \protect\atwindex\MessageBreak
268 can only be referenced when called in a module with id key}}%
269 {\ifx\@test\@empty%
270 \expandafter\sref@target@ifh{sref@#2@\mod@id @target}{}\else%
271 \expandafter\sref@target@ifh{sref@#1@\mod@id @target}{}\fi}%
272 \else\expandafter\sref@target@ifh{sref@#1@\omtext@theory @target}{}\fi%
273 \else\PackageError{statements}%
274 {definiendum outside definition context\MessageBreak
275 \protect\definiendum, \protect\defin,
276 \protect\twindex, \protect\atwindex\MessageBreak
277 do not make sense semantically outside a definition.\MessageBreak
278 Consider wrapping the defining phrase in a \protect\inlinedef}%
279 \fi}
280 \end{package}

```

`\definiendum` The `\definiendum` and `\notatiendum` macros are very simple, but instead of defining them directly, we guard in the `\st@definition@initialize` macro so that they are local to the definition environment.

```

281 \end{package}
282 \newcommand{\definiendum}[2][\st@def@target{#1}{#2}\defemph{#2}]
283 \end{package}
284 \end{xml}
285 DefConstructor('\definiendum [] {}',
286   "<omdoc:term role='definiendum' name='#name' cd='#theory'>#2</omdoc:term>",
287   afterDigest => sub {
288     my ($stomach, $whatsit) = @_;
289     my $addr = LookupValue('defs');
290     my $name = $whatsit->getArg(1);
291     $name = $whatsit->getArg(2) unless $name;
292     $whatsit->setProperty(name=>$name->toString);
293     push(@$addr, $name->toString) if ($addr and $name);
294     $whatsit->setProperty(theory=>LookupValue('current_module'));
295     return; };#$
296 \end{xml}

```

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

```

297 \end{package}
298 \newcommand{\notatiendum}[2][\notemph{#2}]
299 \end{package}

```

We expand the LATEXML bindings for `\defin`, `\twindex` and `\atwindex` into two instances one will be used for the definition and the other for indexing.

```

\defin
300 <*package>
301 \newcommand{\defin}[2] [] {\definiendum[#1]{#2}\omdoc@index[#1]{#2}}
302 </package>
303 <*xml>
304 DefConstructor('\defin[]{}',
305     "<omdoc:idx>"
306     . "<omdoc:idt>"
307     . "<omdoc:term role='definiendum' name='?#1(#1)(#2)' cd='#theory'>#2</omdoc:term>"
308     . "</omdoc:idt>"
309     . "<omdoc:ide index='default'><omdoc:idp>#2</omdoc:idp></omdoc:ide>"
310     . "</omdoc:idx>",
311     afterDigest => sub {
312 my ($stomach, $whatsit) = @_;
313 my $addr = LookupValue('defs');
314 my $name = $whatsit->getArg(1);
315 $name = $whatsit->getArg(2) unless $name;
316 push(@$addr, $name->toString) if ($addr and $name);
317 $whatsit->setProperty(theory=>LookupValue('current_module'));#$
318 return; },
319     alias=>'defin');
320 </xml>

```

```

\definalt
321 <*package>
322 \newcommand{\definalt}[3] [] {\definiendum[#1]{#2}\omdoc@index[#1]{#3}}
323 </package>
324 <*xml>
325 DefConstructor('\definalt[]{}{}',
326     "<omdoc:idx>"
327     . "<omdoc:idt>"
328     . "<omdoc:term role='definiendum' name='?#1(#1)(#3)' cd='#theory'>#2</omdoc:term>"
329     . "</omdoc:idt>"
330     . "<omdoc:ide index='default'><omdoc:idp>#3</omdoc:idp></omdoc:ide>"
331     . "</omdoc:idx>",
332     afterDigest => sub {
333 my ($stomach, $whatsit) = @_;
334 my $addr = LookupValue('defs');
335 my $name = $whatsit->getArg(1);
336 $name = $whatsit->getArg(3) unless $name;
337 push(@$addr, $name->toString) if ($addr and $name);
338 $whatsit->setProperty(theory=>LookupValue('current_module'));#$
339 return; },
340     alias=>'definalt');
341 </xml>

```

```

\twindef
342 <*package>
343 \newcommand{\twindef}[3] [] {\st@def@target[#1]{#2-#3}\defemph[#2 #3]\@twin[#1]{#2}{#3}}

```

```

344 </package>
345 <*ltxml>
346 DefConstructor('\twindex []{}{}',
347     "<omdoc:idx>"
348     . "<omdoc:idt>"
349     . "<omdoc:term role='definiendum' name='?#1(#1)&dashed(#2,#3)'' cd='#theory'>"
350     . "#2 #3"
351     . "</omdoc:term>"
352     . "</omdoc:idt>"
353     . "<omdoc:ide index='default'>"
354     . "<omdoc:idp>#2</omdoc:idp>"
355     . "<omdoc:idp>#3</omdoc:idp>"
356     . "</omdoc:ide>"
357     . "</omdoc:idx>",
358     afterDigest => sub {
359 my ($stomach, $whatsit) = @_;
360 my $addr = LookupValue('defs');
361 my $name = $whatsit->getArg(1);
362 $name = $name->toString if $name;
363 $name = $whatsit->getArg(2)->toString.'-'. $whatsit->getArg(3)->toString unless $name;
364 push(@$addr, $name) if ($addr and $name);
365 $whatsit->setProperty(theory=>LookupValue('current_module'));
366 return; },
367     alias=>'\twindex');#$
368 </ltxml>

```

\twindexdefault

```

369 <*package>
370 \newcommand{\twindexdefault}[4][[]]{\definiendum[#1]{#2}\twin[#1]{#3}{#4}}
371 </package>
372 <*ltxml>
373 DefConstructor('\twindexdefault []{}{}{}',
374     "<omdoc:idx>"
375     . "<omdoc:idt>"
376     . "<omdoc:term role='definiendum' name='?#1(#1)&dashed(#3,#4)'' cd='#theory'>"
377     . "#2"
378     . "</omdoc:term>"
379     . "</omdoc:idt>"
380     . "<omdoc:ide index='default'>"
381     . "<omdoc:idp>#3</omdoc:idp>"
382     . "<omdoc:idp>#4</omdoc:idp>"
383     . "</omdoc:ide>"
384     . "</omdoc:idx>",
385     afterDigest => sub {
386 my ($stomach, $whatsit) = @_;
387 my $addr = LookupValue('defs');
388 my $name = $whatsit->getArg(1);
389 $name = $name->toString if $name;
390 $name = $whatsit->getArg(3)->toString.'-'. $whatsit->getArg(4)->toString unless $name;
391 push(@$addr, $name) if ($addr and $name);

```



```

438 my ($stomach, $whatsit) = @_;
439 my $addr = LookupValue('defs');
440 my $name = $whatsit->getArg(1);
441 $name = $name->toString if $name;
442 $name = $whatsit->getArg(3)->toString.'-'. $whatsit->getArg(4)->toString.'-'. $whatsit->getArg(5);
443 push(@$addr, $name) if ($addr and $name);
444 $whatsit->setProperty(theory=>LookupValue('current_module'));
445 return; },
446     alias=>'atwindef');
447 </ltxml>

```

\inlineex

```

448 <*package>
449 \newcommand{\inlineex}[2] [] {\metasetkeys{omtext}{#1}\sref@target\sref@label{id{here}#2}
450 </package>
451 <*ltxml>
452 DefConstructor('\inlineex OptionalKeyVals:omtext {}',
453     "<omdoc:phrase type='example'>#2</omdoc:phrase>");
454 </ltxml>

```

\inlinedef

```

455 <*package>
456 \newcommand{\inlinedef}[2] [] {\metasetkeys{omtext}{#1}\sref@target\sref@label{id{here}}\st@indef
457 </package>
458 <*ltxml>
459 DefConstructor('\inlinedef OptionalKeyVals:omtext {}', sub {
460 my ($document, $keyvals, $body, %props) = @_;
461 my $for = $keyvals->getValue('for') if $keyvals;
462 my %for_attr=();
463 if (ToString($for)) {
464     $for = ToString($for);
465     $for =~ s/^(.+)$/$/eg;
466     foreach (split(/,\s*/,$for)) {
467         $for_attr{$_}=1;
468     }
469 my @symbols = @{$props{defs} || []};
470 #Prepare for symbol insertion -insert before the parent of the closest ancestor CMP element
471 my $original_node = $document->getNode;
472 my $xc = XML::LibXML::XPathContext->new( $original_node );
473 $xc->registerNs('omdoc', 'http://omdoc.org/ns');
474 my ($statement_ancestor) = $xc->findnodes('./ancestor::omdoc:CMP/..');
475 foreach my $symb(@symbols) {
476     next if $for_attr{$symb};
477     $for_attr{$symb}=1;
478     my $symbolnode = XML::LibXML::Element->new('symbol');
479     $symbolnode->setAttribute(name=>$symb);
480     $symbolnode->setAttribute("xml:id"=>makeNCName("$symb.def.sym"));
481     $statement_ancestor->parentNode->insertBefore($symbolnode,$statement_ancestor);
482 }
483 #Restore the insertion point

```

```

484 $document->setNode($original_node);
485 my %attrs = ();
486 $for = join(" ",(keys %for_attr));
487 $attrs{'for'} = $for if $for;
488 my $id = $keyvals->getValue('id') if $keyvals;
489 $attrs{'xml:id'} = $id if $id;
490 $attrs{'type'} = 'inlinedef';
491 $document->openElement('omdoc:phrase',%attrs);
492 $document->absorb($body);
493 $document->closeElement('omdoc:phrase'); },
494 #Prepare 'defs' hooks for \defin and \definiendum symbol names
495 beforeDigest=>sub {
496   my @symbols = ();
497   AssignValue('defs', \@symbols); return; },
498 #Adopt collected names as 'defs' property, remove hooks
499 afterDigest=>sub {
500   my ($stomach, $whatsit) = @_;
501   my $defsref = LookupValue('defs');
502   my @defs = @$defsref;
503   $whatsit->setProperty('defs',\@defs);
504   AssignValue('defs',undef);
505 return; });
506 </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.

```

507 <*package>
508 \addmetakey[\mod@id]{termref}{cd}
509 \addmetakey{termref}{cdbase}
510 \addmetakey{termref}{name}
511 \addmetakey{termref}{role}
512 \def\termref@set#1#2{\def\termref@name{#2}\metasetkeys{termref}{#1}}

```

`\termref`

```

513 \newcommand{\termref}[2] []{\metasetkeys{termref}{#1}\st@termref{#2}}
514 </package>
515 <*ltxml>
516 DefConstructor('\termref OptionalKeyVals:termref {}',
517   "<omdoc:term cd='&KeyVal(#1,'cd')' name='&KeyVal(#1,'name')'>"
518   . "#2"
519   . "</omdoc:term>");
520 </ltxml>

```

The next macro is where the actual work is done.

`\st@termref` We determine whether the macro `<module>@cd@file@base` is defined. If it is, we

make the prefix of a URI reference in the local macro \@uri, which we compose to the hyper-reference.

```
521 <*package>
522 \def\st@termref#1{\ifx\termref@name\empty\def\termref@name{#1}\fi%
523 \mod@termref\termref@cd\termref@name{#1}}
```

\twinref

```
524 \newcommand{\twinref}[3][\termref@set{#1}{#2-#3}\st@termref{#2 #3}]
525 </package>
526 <*lxml>
527 DefConstructor('\twinref OptionalKeyVals:termref {}{}',
528     "<omdoc:term cd='&KeyVal(#1,'cd')' name='&KeyVal(#1,'name')'>"
529     . "#2"
530     . "</omdoc:term>");
531 </lxml>
```

\atwinref

```
532 <*package>
533 \newcommand{\atwinref}[4][\termref@set{#1}{#2-#3-#4}\st@termref{#2 #3 #4}]
534 </package>
535 <*lxml>
536 DefConstructor('\atwinref OptionalKeyVals:termref {}{}{}',
537     "<omdoc:term cd='&KeyVal(#1,'cd')' name='&KeyVal(#1,'name')'>"
538     . "#2"
539     . "</omdoc:term>");
540 </lxml>
```

\termin The termin macro is very simple, it just adds an index.

```
541 <*package>
542 \newcommand{\termin}[2][\termref{#1}{#2}\index{#2}]
543 </package>
544 <*lxml>
545 DefConstructor('\termin OptionalKeyVals:term {}',
546     "<omdoc:idx>"
547     . " <omdoc:idt>"
548     . " <omdoc:term cd='&KeyVal(#1,'cd')' name='&KeyVal(#1,'name')'>#2</omdoc:term>"
549     . " </omdoc:idt>"
550     . " <omdoc:ide index='default'><omdoc:idp>#2</omdoc:idp></omdoc:ide>"
551     . "</omdoc:idx>");
552 </lxml>
```

Now we care about the configuration switches, they are set to sensible values, if they are not defined already. These are just configuration parameters, which should not appear in documents, therefore we do not provide L<sup>A</sup>T<sub>E</sub>X<sub>M</sub>L bindings for them.

\\*emph

```
553 <*package>
554 \providecommand{\termemph}[1]{#1}
```

```

555 \providecommand{\defemph}[1]{\textbf{#1}}
556 \providecommand{\stDMemph}[1]{\textbf{#1}}
557 \end{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>`.

```

558 \end{package}
559 \newcommand{\symref}[2]{\@nameuse{mod@symref@#1}#2}
560 \end{package}
561 \end{xml}
562 DefConstructor('symref{}')',
563     "<omdoc:term cd='&LookupValue('symdef.#1.cd')' name='&LookupValue('symdef.#1.nam
564     . "#2"
565     . "</omdoc:term>");
566 \end{xml}

```

## 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`.

```

567 \end{xml}
568 Tag('omdoc:assertion', afterOpen=>\&numberIt, afterClose=>\&locateIt);
569 Tag('omdoc:definition', afterOpen=>\&numberIt, afterClose=>\&locateIt);
570 Tag('omdoc:example', afterOpen=>\&numberIt, afterClose=>\&locateIt);
571 Tag('omdoc:equation', afterOpen=>\&numberIt, afterClose=>\&locateIt);
572 Tag('omdoc:axiom', afterOpen=>\&numberIt, afterClose=>\&locateIt);
573 Tag('omdoc:symbol', afterOpen=>\&numberIt, afterClose=>\&locateIt);
574 Tag('omdoc:type', afterOpen=>\&numberIt, afterClose=>\&locateIt);
575 Tag('omdoc:term', afterOpen=>\&numberIt, afterClose=>\&locateIt);
576 \end{xml}

```

## 5.5 Finale

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

```

577 \end{xml}1;

```



## Index

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

<i>*</i> ,	9	statement,	2	OPENMATH,	3,	5
block						
statement,	2	LATEXML, 9, 10, 17, 23	statement			
			block,			2
flow		OMDOC, 2-5, 24	flow,			2

## References

- [KGA10] Michael Kohlhase, Deyan Ginev, and Rares Ambrus. *modules.sty: Semantic Macros and Module Scoping in sTeX*. Self-documenting L<sup>A</sup>T<sub>E</sub>X package. Comprehensive T<sub>E</sub>X 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 L<sup>A</sup>T<sub>E</sub>X 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 L<sup>A</sup>T<sub>E</sub>X*. Self-documenting L<sup>A</sup>T<sub>E</sub>X package. Comprehensive T<sub>E</sub>X 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 L<sup>A</sup>T<sub>E</sub>X*. Self-documenting L<sup>A</sup>T<sub>E</sub>X package. Comprehensive T<sub>E</sub>X 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 L<sup>A</sup>T<sub>E</sub>X*. Self-documenting L<sup>A</sup>T<sub>E</sub>X package. Comprehensive T<sub>E</sub>X 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 L<sup>A</sup>T<sub>E</sub>X-Theorem Environment*. Self-documenting L<sup>A</sup>T<sub>E</sub>X 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 12/02/2009).