# Structural Markup for Proofs*

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

May 7, 2008

**Abstract**

The sproof packge 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 supplies macros and environment that allow to annotate the structure of mathematical proofs in sTeX files. This structure can be used by MKM systems for added-value services, either directly from the sTeX sources, or after translation.

## 1 Introduction

The sproof (semantic proofs) package supplies macros and environment that allow to annotate the structure of mathematical proofs in sTeX files. 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 statements.

sTeX 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).

We will go over the general intuition by way of our running example (see Figure 1 for the source and Figure 2 for the formatted result).[1]

EdNote(1)

## 2 The User Interface

### 2.1 Proofs and Proof steps

sproof

The proof environment is the main container for proofs. It takes an optional

*Version ? (last revised ?)

[1] EdNote: talk a bit more about proofs and their structure,... maybe copy from OMDoc spec.

```
% \begin{sproof}[id=simple-proof,for=sum-over-odds]
%    {We prove that $\sum_{i=1}^n{2i-1}=n^{2}$ by induction over $n$}
%   \begin{spfcases}{For the induction we have to consider the following cases:}
%    \begin{spfcase}{$n=1$}
%     \begin{spfstep}[display=flow] then we compute $1=1^2$\end{spfstep}
%    \end{spfcase}
%    \begin{spfcase}{$n=2$}
%       \begin{sproofcomment}[display=flow]
%         This case is not really necessary, but we do it for the
%         fun of it (and to get more intuition).
%       \end{sproofcomment}
%       \begin{spfstep}[display=flow] We compute $1+3=2^{2}=4$.\end{spfstep}
%    \end{spfcase}
%    \begin{spfcase}{$n>1$}
%       \begin{spfstep}[type=assumption,id=ind-hyp]
%         Now, we assume that the assertion is true for a certain $k\geq 1$,
%         i.e. $\sum_{i=1}^k{(2i-1)}=k^{2}$.
%       \end{spfstep}
%       \begin{sproofcomment}
%         We have to show that we can derive the assertion for $n=k+1$ from
%         this assumption, i.e. $\sum_{i=1}^{k+1}{(2i-1)}=(k+1)^{2}$.
%       \end{sproofcomment}
%       \begin{spfstep}
%         We obtain $\sum_{i=1}^{k+1}{2i-1}=\sum_{i=1}^k{2i-1}+2(k+1)-1$
%         \begin{justification}[method=arith:split-sum]
%           by splitting the sum.
%         \end{justification}
%       \end{spfstep}
%       \begin{spfstep}
%         Thus we have $\sum_{i=1}^{k+1}{(2i-1)}=k^2+2k+1$
%         \begin{justification}[method=fertilize] by inductive hypothesis.\end{justification}
%       \end{spfstep}
%       \begin{spfstep}[type=conclusion]
%         We can \begin{justification}[method=simplify]simplify\end{justification}
%         the right-hand side to ${k+1}^2$, which proves the assertion.
%       \end{spfstep}
%    \end{spfcase}
%     \begin{spfstep}[type=conclusion]
%       We have considered all the cases, so we have proven the assertion.
%     \end{spfstep}
%   \end{spfcases}
% \end{sproof}
%
```

**Example 1:** A very explicit proof, marked up semantically

KeyVal argument that allows to specify the id (identifier) and for (for which assertion is this a proof) keys. The regular argument of the proof environment contains an introductory comment, that may be used to announce the proof style. The proof environment contains a sequence of \step, proofcomment, and pfcases environments that are used to markup the proof steps. The proof environment has a variant Proof, which does not use the proof end marker. This is convenient, if a proof ends in a case distinction, which brings it's own proof end marker with it. The Proof environment is a variant of proof that does not mark the end of a proof with a little box; presumably, since one of the subproofs already has one and then a box supplied by the outer proof would generate an otherwise empty line. The \sproofidea macro allows to give a one-paragraph description of the proof idea.

sProof

\sproofidea

spfstep

Regular proof steps are marked up with the step environment, which takes an optional KeyVal argument for annotations. A proof step usually contains a local assertion (the text of the step) together with some kind of evidence that this can be derived from already established assertions.

Note that both \premise and \justarg can be used with an empty second argument to mark up premises and arguments that are not explicitly mentioned in the text.

**Proof**: We prove that $\sum_{i=1}^{n} 2i-1 = n^2$ by induction over $n$

**P.1** For the induction we have to consider the following cases:

**P.1.1** $n = 1$: then we compute $1 = 1^2$ □

**P.1.2** $n = 2$: This case is not really necessary, but we do it for the fun of it (and to get more intuition). We compute $1 + 3 = 2^2 = 4$ □

**P.1.3** $n > 1$:

**P.1.3.1** Now, we assume that the assertion is true for a certain $k \geq 1$, i.e. $\sum_{i=1}^{k} (2i-1) = k^2$.

**P.1.3.2** We have to show that we can derive the assertion for $n = k+1$ from this assumption, i.e. $\sum_{i=1}^{k+1} (2i-1) = (k+1)^2$.

**P.1.3.3** We obtain $\sum_{i=1}^{k+1} (2i-1) = \sum_{i=1}^{k} (2i-1) + 2(k+1) - 1$ by splitting the sum

**P.1.3.4** Thus we have $\sum_{i=1}^{k+1} (2i-1) = k^2 + 2k + 1$ by inductive hypothesis.

**P.1.3.5** We can simplify the right-hand side to $k + 1^2$, which proves the assertion. □

**P.1.4** We have considered all the cases, so we have proven the assertion. □

**Example 2:** The formatted result of the proof in Figure 1

## 2.2 Justifications

`justification` This evidence is marked up with the `justification` environment in the `sproof` package. This environment totally invisible to the formatted result; it wraps the text in the proof step that corresponds to the evidence. The environment takes an optional `KeyVal` argument, which can have the `method` key, whose value is the name of a proof method (this will only need to mean something to the application that consumes the semantic annotations). Furthermore, the justification can contain "premises" (specifications to assertions that were used justify the step) and "arguments" (other information taken into account by the proof method).

`\premise` The `\premise` macro allows to mark up part of the text as reference to an assertion that is used in the argumentation. In the example in Figure 1 we have used the `\premise` macro to identify the inductive hypothesis.

`\justarg` The `\justarg` macro is very similar to `\premise` with the difference that it is used to mark up arguments to the proof method. Therefore the content of the first argument is interpreted as a mathematical object rather than as an identifier as in the case of `\premise`. In our example, we specified that the simplification should take place on the right hand side of the equation. Other examples include proof methods that instantiate. Here we would indicate the substituted object in a `\justarg` macro.

## 2.3 Proof Structure

`spfcases` The `pfcases` environment is used to mark up a proof by cases. This environment takes an optional `KeyVal` argument for semantic annotations and a second argument that allows to specify an introductory comment (just like in the `proof` environment).

`spfcase` The content of a `pfcases` environment are a sequence of case proofs marked up in the `pfcase` environment, which takes an optional `KeyVal` argument for semantic annotations. The second argument is used to specify the the description of the case under considertation. The content of a `pfcase` environment is the same as that of a `proof`, i.e. `step`s, `proofcomment`s, and `pfcases` environments.

`sproofcomment` The `proofcomment` environment is much like a `step`, only that it does not have an obejct-level assertion of its own. Rather than asserting some fact that is relevant for the proof, it is used to explain where the proof is going, what we are attempting to to, or what we have achieved so far. As such, it cannot be the target of a `\premise`.

## 2.4 Proof End Markers

Traditionally, the end of a mathematical proof is marked with a little box at the end of the last line of the proof (if there is space and on the end of the next line if there isn't), like so: □

`\sproofend` The `sproof` package provides the `\sproofend` macro for this. If a different symbol for the proof end is to be used (e.g. *q.e.d*), then this can be obtained by

`\sProofEndSymbol` specifying it using the `\sProofEndSymbol` configuration macro (e.g. by specifying

`\sProofEndSymbol{q.e.d}`).

Some of the proof structuring macros above will insert proof end symbols for sub-proofs, in most cases, this is desirable to make the proof structure explicit, but sometimes this wastes space (especially, if a proof ends in a case analysis which will supply its own proof end marker). Therefore, all proof environments have the `noproofend` keyword that suppresses the proof end markers for this element. It can be specified on its own, and does not need a value (if one is specified, that is completely ignored).

noproofend

# 3 The Implementation

We first set up the Perl Packages for LaTeXML

```
1 ⟨*ltxml⟩
2 # -*- CPERL -*-
3 package LaTeXML::Package::Pool;
4 use strict;
5 use LaTeXML::Package;
6 RequirePackage('omdoc');
7 ⟨/ltxml⟩
8 % Then we make sure that the {\stex} |omdoc| package is loaded.
9 %    \begin{macrocode}
10 ⟨*package⟩
11 \RequirePackage{omdoc}[2007/09/09]
12 ⟨/package⟩
```

Then we define the pacakge options and what they do.

```
13 ⟨*package⟩
14 \newif\ifjust@method\just@methodfalse
15 \DeclareOption{method}{\just@methodtrue}
16 ⟨/package⟩
```

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). Firpf we have the general options

```
17 ⟨*package⟩
18 \newif\ifspf@env\spf@envfalse
19 \newif\ifspf@id\spf@idfalse
20 \newif\ifspf@display\spf@displayfalse
21 \DeclareOption{id}{\spf@idtrue\spf@envtrue}
22 \DeclareOption{env}{\spf@envtrue}
23 \DeclareOption{display}{\spf@displaytrue\spf@envtrue}
```

And then the options that are specific to the `sproof` package.

```
24 \newif\ifspf@for\spf@forfalse
25 \newif\ifspf@from\spf@forfalse
26 \newif\ifspf@type\spf@typefalse
27 \newif\ifspf@title\spf@titlefalse
28 \newif\ifspf@proofend\spf@proofendtrue
29 \newif\ifspf@continues\spf@continuesfalse
```

```
\DeclareOption{for}{\spf@fortrue\spf@envtrue}
\DeclareOption{from}{\spf@fromtrue\spf@envtrue}
\DeclareOption{type}{\spf@typetrue\spf@envtrue}
\DeclareOption{title}{\spf@titletrue\spf@envtrue}
\DeclareOption{continues}{\spf@continuepfrue\spf@envtrue}
\DeclareOption{noproofend}{\spf@proofendfalse\spf@envtrue}
```

`\spftrue` For convenience, we collect the switches into one.

```
\def\spftrue{\spf@fortrue\spf@fromtrue\spf@typetrue\spf@continuepfrue}
```

Now, we define a set of collective options and tell LaTeX about the end of the declaration section.

```
\DeclareOption{draft}{\spf@envtrue\just@methodtrue}
\DeclareOption{all}{\spf@envtrue\just@medhodtrue}
\ProcessOptions
⟨/package⟩
```

## 3.1 Proofs

```
⟨*package⟩
\define@key{spf}{id}{\def\spf@id{#1}}
\define@key{spf}{display}{\def\spf@display{#1}}
\define@key{spf}{for}{\def\spf@for{#1}}
\define@key{spf}{from}{\def\spf@from{#1}}
\define@key{spf}{sproofend}{\spf@proofendtrue\def\sproof@box{#1}}
\define@key{spf}{noproofend}[imeanit]{\spf@proofendfalse}
\define@key{spf}{type}{\def\spf@type{#1}}
\define@key{spf}{title}{\def\spf@title{#1}}
\define@key{spf}{continues}{\def\spf@continues{#1}}
⟨/package⟩
⟨*ltxml⟩
DefKeyVal('pf','id','Semiverbatim');
DefKeyVal('pf','display','Semiverbatim'); # not used at the moment
DefKeyVal('pf','for','Semiverbatim');
DefKeyVal('pf','from','Semiverbatim');
DefKeyVal('pf','proofend','Semiverbatim');
DefKeyVal('pf','noproofend','Semiverbatim');
DefKeyVal('pf','type','Semiverbatim');
DefKeyVal('pf','title','Semiverbatim');
DefKeyVal('pf','continues','Semiverbatim');
⟨/ltxml⟩
```

`\show@st@keys@aux` We now define a macro that shows the `st` keys, if in draft mode, they annotate the document with key/value pairs.

```
⟨*package⟩
\def\show@st@keys@aux{%
\@ifundefined{spf@id}{}{\ifspf@id{id=\spf@id},\fi}%
\@ifundefined{spf@display}{}{\ifspf@display{display=\spf@display}\fi}}
\def\clear@pf@keys{\let\spf@id=\relax\let\spf@display=\relax}
⟨/package⟩
```

`\show@pf@keys@aux` we do the same for the `pf` keys.

```
⟨*package⟩
\def\show@pf@keys@aux{%
\@ifundefined{spf@for}{}{\ifspf@for{for=\spf@for},\fi}%
\@ifundefined{spf@from}{}{\ifspf@from{from=\spf@from},\fi}%
\@ifundefined{spf@prefix}{}{\ifspf@prefix{prefix=\spf@prefix},\fi}%
\@ifundefined{spf@type}{}{\ifspf@type{type=\spf@type},\fi}%
\@ifundefined{spf@title}{}{\ifspf@title{title=\spf@title},\fi}%
\@ifundefined{spf@continues}{}{\ifspf@continues{continues=\spf@continues}\fi}}
⟨/package⟩
```

`\show@pf@keys` and combine them, so that the code is more readable.

```
⟨*package⟩
\def\show@pf@keys#1{\footnote{#1[\show@pf@keys@aux\show@pf@keys@aux]}%
\clear@pf@keys\clear@pf@keys}
\def\clear@pf@keys{\let\spf@for=\relax\let\spf@from=\relax\let\spf@prefix=\relax%
\let\spf@type=\relax\let\spf@title=\relax\let\spf@continues=\relax}
⟨/package⟩
```

`\spf@flow` We define this macro, so that we can test whether the `display` key has the value `flow`

```
⟨package⟩\def\spf@flow{flow}
```

For proofs, we will have to have deeply nested structures of enumerated list-like environments. However, LaTeX only allows `enumerate` environments up to nesing depth 4 and general list environments up to listing depth 6. This is not enough for us. Therefore we have decided to go along the route proposed by Leslie Lamport to use a single top-level list with dotted sequences of numbers to identify the position in the proof tree. Unfortunately, we could not use his `pf.sty` package directly, since it does not do automatic numbering, and we have to add keyword arguments all over the place, to accomodate semantic information.

EdNote(2) `pst@with@label` This environment manages[2] the path labeling of the proof steps in the description envionment of the outermost `proof` environment. The argument is the label prefix up to now; which we cache in `\pst@label` (we need evaluate it first, since are in the right place now!). Then we increment the proof depth which is stored in `\count10` (lower counters are used by TeX for page numbering) and initialize the next level counter `\count\count10` with 1. In the end call for this environment, we just decrease the proof depth counter by 1 again.

```
⟨*package⟩
\newenvironment{pst@with@label}[1]{\edef\pst@label{#1}\advance\count10 by 1\count\count10=1}
{\advance\count10 by -1}
```

`\the@pst@label` `\the@pst@label` evaluates to the current step label.

```
\def\the@pst@label{\pst@label.\number\count\count10}
```

[2] EdNote: This gets the labeling right but only works 8 levels deep

`\next@pst@label` `\next@pst@label` increments the step label at the current level.

```
\def\next@pst@label{\global\advance\count\count10 by 1}
```

`\sproofend` This macro places a little box at the end of the line if there is space, or at the end of the next line if there isn't

```
\def\sproof@box{\hbox{\vrule\vbox{\hrule width 6 pt\vskip 6pt\hrule}\vrule}}
\def\sproofend{\ifspf@proofend\hfil\null\nobreak\hfill\sproof@box\par\smallskip\fi}
\def\sProofEndSymbol#1{\def\sproof@box{#1}}
⟨/package⟩
⟨ltxml⟩DefConstructor('\sproofend',"");
```

`sproof` In this environment, we initialize the proof depth counter `\count10` to 10, and set up the description environment that will take the proof steps. At the end of the proof, we position the proof end into the last line.

```
⟨*package⟩
\newenvironment{@proof}[2][]{\setkeys{spf}{#1}
\count10=10\ifx\spf@display\spf@flow\else{\stDMemph{Proof}:}\fi%
\ifspf@env\show@pf@keys{sproof}\fi{ #2}%
\def\pst@label{}\newcount\pst@count% initialize the labeling mechanism
\begin{description}\begin{pst@with@label}{P}}
{\end{pst@with@label}\end{description}}
\newenvironment{sproof}[2][]{\begin{@proof}[#1]{#2}}{\sproofend\end{@proof}}
⟨/package⟩
⟨*ltxml⟩
DefEnvironment('{sproof} OptionalKeyVals:pf{}',
       "<omdoc:proof ?&KeyVal(#1,'id')(xml:id='&KeyVal(#1,'id')')()>\n"
       . "?#2(<omdoc:omtext><omdoc:CMP><omdoc:p>#2</omdoc:p></omdoc:CMP></omdoc:omtext>\n)()"
              . "#body"
             . "</omdoc:proof>\n");
⟨/ltxml⟩
```

`sproofidea`

```
⟨package⟩\newcommand{\sproofidea}[2][]{{\stDMemph{Proof Idea}:} #2\sproofend}
⟨*ltxml⟩
DefEnvironment('{sproofidea} OptionalKeyVals:pf {}',
       "<omdoc:proof ?&KeyVal(#1,'id')(xml:id='&KeyVal(#1,'id')')()>\n"
       . "<omdoc:omtext><omdoc:CMP><omdoc:p>#2</omdoc:p></omdoc:CMP></omdoc:omtext>\n"
     . "</omdoc:proof>\n");
⟨/ltxml⟩
```

The next two environments (proof steps) and comments, are mostly semantical, they take `KeyVal` arguments that specify their semantic role. In draft mode, they read these values and show them. If the surrounging proof had `display=flow`, then no new `\item` is generated, otherwise it is. In any case, the proof step number (at the current level) is incremented.

`spfstep`

```
⟨*package⟩
\newenvironment{spfstep}[1][]{\setkeys{spf}{#1}
```

```
\ifx\spf@display\spf@flow\else\item[\the@pst@label]\fi
\@ifundefined{spf@title}{}{(\stDMemph{\spf@title})}%
\ifspf@env\show@pf@keys{spfstep}\fi}
{\next@pst@label}
⟨/package⟩
⟨*ltxml⟩
DefCMPEnvironment('{spfstep} OptionalKeyVals:pf',
             "<omdoc:derive ?&KeyVal(#1,'id')(xml:id='&KeyVal(#1,'id')')()>"
           . "<omdoc:CMP><omdoc:p>#body</omdoc:p></omdoc:CMP>"
         . "</omdoc:derive>\n");
⟨/ltxml⟩
```

sproofcomment

```
⟨*package⟩
\newenvironment{sproofcomment}[1][]{\setkeys{spf}{#1}
\ifx\spf@display\spf@flow\else\item[\the@pst@label]\fi
\ifspf@env\show@pf@keys{sproofcomment}\fi}
{\next@pst@label}
⟨/package⟩
⟨*ltxml⟩
DefCMPEnvironment('{sproofcomment} OptionalKeyVals:pf',
       "<omdoc:omtext ?&KeyVal(#1,'id')(xml:id='&KeyVal(#1,'id')')()>"
       . "<omdoc:CMP><omdoc:p>#body</omdoc:p></omdoc:CMP>"
     . "</omdoc:omtext>");
⟨/ltxml⟩
```

The next two environments also take a `KeyVal` arguement, but also a regular one, which contains a start text. Both environments start a new numbered proof level.

spfcases

In the `pfcases` environment, the start text is displayed as the first comment of the proof.

```
⟨*package⟩
\newenvironment{spfcases}[2][]{\setkeys{spf}{#1}
\def\@test{#2}\ifx\@test\empty\else
\ifx\spf@display\spf@flow {#2}\else\item[\the@pst@label]{#2} \fi\fi
\ifspf@env\show@pf@keys{spfcases}\fi
\begin{pst@with@label}{\pst@label.\number\count\count10}}
{\end{pst@with@label}\next@pst@label}
⟨/package⟩
⟨*ltxml⟩
DefEnvironment('{spfcases} OptionalKeyVals:pf {}',
       "<omdoc:derive ?&KeyVal(#1,'id')(xml:id='&KeyVal(#1,'id')')()>\n"
       . "<omdoc:CMP><omdoc:p>#2</omdoc:p></omdoc:CMP>\n"
         . "<omdoc:method xref='#proof-by-cases'>"
           . "#body"
         . "</omdoc:method>"
     . "</omdoc:derive>\n");
⟨/ltxml⟩
```

`spfcase` In the `pfcase` environment, the start text is displayed specification of the case after the `\item`

```
⟨*package⟩
\newenvironment{spfcase}[2][]{\setkeys{spf}{#1}
\ifx\spf@display\spf@flow\else\item[\the@pst@label]\fi
\def\@test{#2}\ifx\@test\empty\else{\stDMemph{#2}:}\fi
\ifspf@env\show@pf@keys{spfcase}\fi
\begin{pst@with@label}{\pst@label.\number\count\count10}}
{\ifx\spf@display\spf@flow\else\sproofend\fi\end{pst@with@label}\next@pst@label}
⟨/package⟩
⟨*ltxml⟩
DefEnvironment('{spfcase} OptionalKeyVals:pf{}',
       "<omdoc:proof ?&KeyVal(#1,'id')(xml:id='&KeyVal(#1,'id')')()>\n"
       . "?#2(<omdoc:omtext><omdoc:CMP><omdoc:p>#2</omdoc:p></omdoc:CMP></omdoc:omtext>\n)()"
       . "#body"
     . "</omdoc:proof>\n");
⟨/ltxml⟩
```

EdNote(3) `subproof` In the `subproof` environment, a new (lower-level) proof environment is started.[3]

```
⟨*package⟩
\newenvironment{subproof}{\begin{pst@with@label}{\pst@label.\number\count\count10}}
{\ifx\spf@display\spf@flow\else\sproofend\fi\end{pst@with@label}}
⟨/package⟩
⟨*ltxml⟩
DefEnvironment('{subproof}',
       "<omdoc:proof>\n #body\n</omdoc:proof>\n");
⟨/ltxml⟩
```

## 3.2 Justifications

We define the actions that are undertaken, when the keys for justifications are encountered. Here this is very simple, we just define an internal macro with the value, so that we can use it later.[4]

EdNote(4)

```
⟨*package⟩
\define@key{just}{id}{\def\just@id{#1}}
\define@key{just}{method}{\def\just@method{#1}}
⟨/package⟩
⟨*ltxml⟩
DefKeyVal('just','id','Semiverbatim');
DefKeyVal('just','method','Semiverbatim');
DefKeyVal('just','premises','Semiverbatim');
DefKeyVal('just','args','Semiverbatim');
⟨/ltxml⟩
```

`\show@just@keys` This macro shows all the key/value pairs when in draft mode.

```
⟨*package⟩
```

[3] EdNote: document this above

[4] EdNote: why are there more in ltxml?, why st@id?

```
\def\show@just@keys#1{\footnote{#1[\show@pf@keys@aux\show@just@keys@aux]}}%
\def\show@just@keys@aux{%
\@ifundefined{just@method}{\relax}{\ifjust@method{method=\just@method},\fi}%
\clear@pf@keys\clear@just@keys}
\def\clear@just@keys{\let\just@method=\relax}
⟨/package⟩
```

The next three environments and macros are purely semantic, so we ignore the keyval arguments for now and only display the content.[5]

EdNote(5)

justification

```
⟨*package⟩
\newenvironment{justification}[1][]{\ifspf@env\setkeys{just}{#1}
                                      \show@just@keys{justification}\fi}{}
⟨/package⟩
⟨*ltxml⟩
sub extractBodyText {
  my ($box, $remove) = @_;
  my $str = '';
  my @boxes = $box->unlist;
  foreach my $b(@boxes) {
    my $s = '';
    if ($b =~ /LaTeXML::Whatsit/) {
      my $body = $b->getBody;
      $s = $body ? extractBodyText($body, $remove) : '';
    } elsif ($b =~ /LaTeXML::Box/) {
      $s = $b->toString || '';
      @{$b}[0] = '' if $remove; }
    $str .= $s; }
  $str =~ s/\s+/ /g;
  $str; }

DefEnvironment('{justification} OptionalKeyVals:just', sub {
 my ($doc, $keys, %props) = @_;
 my $text = extractBodyText($props{body}, 1);
 my $node = LookupValue('_LastSeenCMP');
 $node->appendText($text) if $node;
 my $method = $keys ? $keys->getValue('method') : undef;
 $doc->openElement("omdoc:method", $method ? (xref => $method) : ());
 $doc->absorb($props{body}) if $props{body};
 $doc->closeElement("omdoc:method");
 return; });
⟨/ltxml⟩
```

\premise

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

[5] EdNote: need to do something about the premise in draft mode.

```
⟨*ltxml⟩
DefMacro('\premise[]{}', sub {
   my ($xref, $text) = ($_[1], $_[2]);
   my @res = (T_CS('\premise@content'));
   push(@res, T_OTHER('['), $xref->unlist, T_OTHER(']')) if $xref;
   push(@res, T_SPACE, $text->unlist) if $text;
   @res; });
DefConstructor('\premise@content[]', "<omdoc:premise xref='#1'/>");
⟨/ltxml⟩
```

`\justarg` the `\justarg` macro is purely semantic, so we ignore the keyval arguments for now and only display the content.

```
⟨*package⟩
\newcommand{\justarg}[2][]{#2}
⟨/package⟩
⟨*ltxml⟩
DefMacro('\justarg[]{}', sub { (($_[1] ? $_[1]->unlist : ()),
T_SPACE, $_[2]->unlist, T_SPACE); });

Tag('omdoc:derive', afterClose=>sub {
      my ($doc, $node) = @_;
      my @children = grep($_->nodeType == XML_ELEMENT_NODE, $node->childNodes);
      my $firstCMP = undef;
      foreach my $child(@children) {
        next unless ($child->localname || '') eq 'CMP';
        if ($child->hasChildNodes()) {
          next unless $#{$child->childNodes} == 0;
          next unless $child->firstChild->nodeType == XML_TEXT_NODE; }

        if ($firstCMP) {
          $firstCMP->appendText($child->textContent);
          $node->removeChild($child);
        } else { $firstCMP = $child; }
      }
    });

⟨/ltxml⟩
```

## 3.3 Providing IDs for OMDoc Elements

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

```
⟨*ltxml⟩
Tag('omdoc:proof',afterOpen=>\&numberIt);
Tag('omdoc:derive',afterOpen=>\&numberIt);
Tag('omdoc:method',afterOpen=>\&numberIt);
⟨/ltxml⟩
```

# 4 Finale

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

274 ⟨ltxml⟩`1;`