Slides and Course Notes for Jacobs University*

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

February 18, 2014

Abstract

We present a document class from which we can generate both course slides and course notes in a transparent way. Furthermore, we present a set of LaTeXML bindings for these, so that we can also generate OMDoc-based course materials, e.g. for inclusion in the ActiveMath system.

Contents

*Version ? (last revised ?)

1 Introduction

This Document class is derived from `beamer.cls` [Tana], specializes it with Jacobs stuff and adds a notes version that is more suited to printing than the one supplied by `beamer.cls`.

2 The User Interface

The `mikoslides` class takes the notion of a slide frame from Till Tantau's excellent `beamer` class and adapts its notion of frames for use in the STEXand OMDoc. To support semantic course notes, it extends the notion of mixing frames and explanatory text, but rather than treating the frames as images (or integrating their contents into the flowing text), the `mikoslides` package displays the slides as such in the course notes to give students a visual anchor into the slide presentation in the course (and to distinguish the different writing styles in slides and course notes).

In practice we want to generate two documents from the same source: the slides for presentation in the lecture and the course notes as a narrative document for home study. To achieve this, the `mikoslides` class has two modes: *slides mode* and *notes mode* which are determined by the package option.

2.1 Package Options

EdN:1 The `mikoslides` class takes a variety of class options:[1]

`slides` `a`
- The options `slides`nd notes`notes` switch between slides mode and notes mode (see Section 2.2).

`sectocframes`
- If the option `sectocframes` is given, then special frames with section table of contents are produced headers [2]

EdN:2

`showmeta`
- `showmeta`. If this is set, then the metadata keys are shown (see [Koh13] for details and customization options).

`frameimages`
- If the option `frameimages` is set, then slide mode also shows the `\frameimage`-generated frames.

2.2 Notes and Slides

`frame` `note`
Slides are represented with the `frame` just like in the `beamer` class, see [Tanb] for details. The `mikoslides` class adds the `note` environment for encapsulating the course note fragments.[1]


Note that it is essential to start and end the `notes` environment at the start of the line – in particular, there may not be leading blanks – else LaTeX becomes confused and throws error messages that are difficult to decipher.

[1] EdNote: leaving out noproblems for the moment until we decide what to do with it.
[2] EdNote: document the functionality
[1] MK: it would be very nice, if we did not need this environment, and this should be possible in principle, but not without intensive LaTeX trickery. Hints to the author are welcome.

```
\begin{note}
  We start this course with ...
\end{note}

\begin{frame}
  \frametitle{The first slide}
  ...
\end{frame}
\begin{note}
  ... and more explanatory text
\end{note}

\begin{frame}
  \frametitle{The second slide}
  ...
\end{frame}
...
```

Example 1: A typical Course Notes File

By interleaving the `frame` and `note` environments, we can build course notes as shown in Figure 1.

`\frameimage`

Sometimes, we want to integrate slides as images after all – e.g. because we already have a PowerPoint presentation, to which we want to add STEXnotes. In this case we can use `\frameimage[`⟨*opt*⟩`]{`⟨*path*⟩`}`, where ⟨*opt*⟩ are the options of `\includegraphics` from the `graphicx` package [CR99] and ⟨*path*⟩ is the file path (extension can be left off like in `\includegraphics`).

2.3 Header and Footer Lines

2.4 Colors and Highlighting

`\textwarning`

The `\textwarning` macro generates a warning sign:

2.5 Front Matter, Titles, etc

2.6 Miscellaneous

2.7 Support for MathHub

Much of the STEXcontent is hosed on MathHub (`http://MathHub.info`), a portal and archive for flexiformal mathematics. MathHub offers GIT repositories (public and private escrow) for mathematical documentation projects, online and offline authoring and document development infrastructure, and a rich, interactive reading interface. The `modules` package supports repository-sensitive operations on MathHub.

Note that MathHub has two-level repository names of the form ⟨*group*⟩/⟨*repo*⟩, where ⟨*group*⟩ is a MathHub-unique repository group and ⟨*repo*⟩ a repository name that is ⟨*group*⟩-unique. The file and directory structure of a repository is arbitrary – except that it starts with the directory `source` because they are Math Archives in the sense of [Hor+11]. But this structure can be hidden from the sTEXauthor with MathHub-enabled versions of the `modules` macros.

`\mhframeimage` The `\mhframeimage` macro is a variant of `\frameimage` with repository support. Instead of writing

```
\defpath{MathHub}{/user/foo/lmh/MathHub}
\frameimage{\MathHub{fooMH/bar/source/baz/foobar}}
```

we can simply write (assuming that `\MathHub` is defined as above)

```
\mhframeimage[fooMH/bar]{baz/foobar}
```

Note that the `\mhframeimage` form is more semantic, which allows more advanced document management features in MathHub.

If `baz/foobar` is the "current module", i.e. if we are on the MathHub path ...`MathHub/fooMH/bar`..., then stating the repository in the first optional argument is redundant, so we can just use

```
\mhframeimage{baz/foobar}
```

Of course, neither LaTeX nor LaTeXML know about the repositories when they are called from a file system, so we can use the `\mhcurrentrepos` macro from the `modules` package to tell them. But this is only needed to initialize the infrastructure in the driver file. In particular, we do not need to set it in in each module, since the `\importmhmodule` macro sets the current repository automatically.

Caveat if you want to use the MathHub support macros (let's call them mh-variants), then every time a module is imported or a document fragment is included from another repos, the mh-variant `\importmhmodule` must be used, so that the "current repository" is set accordingly. To be exact, we only need to use mh-variants, if the imported module or included document fragment use mh-variants.

3 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 sTEXTRAC [sTeX].

1. the class should be divided into concerns. [sTeX], issue 1684
2. when option `book` or `report` is given together with `sectocframes` chapter-level omgroups generate a spurious slide with a bare heading. This has something to do with the fact that beamer does not support `\chapter`

4 The Implementation

The `mikoslides` 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.

4.1 Initialization and Class Options

For the LaTeXML bindings, we make sure the right perl packages are loaded.

```
⟨*ltxml⟩
# -*- CPERL -*-
package LaTeXML::Package::Pool;
use strict;
use LaTeXML::Package;
⟨/ltxml⟩
```

For LaTeX we define some Package Options and switches for the `mikoslides` class and activate them by passing them on to `beamer.cls` the appropriate packages.

```
⟨*cls⟩
\DeclareOption{showmeta}{\PassOptionsToPackage{\CurrentOption}{metakeys}}
\newif\ifnotes\notesfalse
\newif\ifsectocframes\sectocframesfalse
\newif\ifframeimages\frameimagesfalse
\newif\ifproblems\problemstrue
\DeclareOption{notes}{\notestrue}
\DeclareOption{slides}{\notesfalse}
\DeclareOption{noproblems}{\problemsfalse}
\DeclareOption{sectocframes}{\sectocframestrue}
\DeclareOption{frameimages}{\frameimagestrue}
```

the next two define the `frontmatter` environment so that the later `\renewcommand` does not lead to trouble.

```
\newif\if@part\@partfalse
\DeclareOption{report}{\@parttrue\PassOptionsToClass{\CurrentOption}{omdoc}}
\DeclareOption{book}{\@parttrue\PassOptionsToClass{\CurrentOption}{omdoc}}
\DeclareOption*{\PassOptionsToClass{\CurrentOption}{omdoc}
                \PassOptionsToClass{\CurrentOption}{beamer}}
\ProcessOptions
⟨/cls⟩
⟨*ltxml⟩
RawTeX('\newif\ifnotes\notesfalse');
RawTeX('\newif\ifproblems\problemsfalse');
⟨/ltxml⟩
```

Depending on the options, we either load the `article`-based `omdoc` or the `beamer` class. In the first case, we also have to make the `beamer`-specific things

available to `article` via the `beamerarticle` package. We use options to avoid loading theorem-like environments, since we want to use our own from the sTEX packages.

```
⟨*cls⟩
\ifnotes
\LoadClass{omdoc}
\RequirePackage{a4wide}
\RequirePackage{marginnote}
\RequirePackage{mdframed}
\RequirePackage[notheorems,noamsthm,noxcolor]{beamerarticle}
\else
\LoadClass[notheorems,noamsthm,10pt]{beamer}
\newcounter{Item}
\newcounter{paragraph}
\newcounter{subparagraph}
\newcounter{Hfootnote}
\usetheme{Jacobs}
\fi
⟨/cls⟩
⟨*ltxml⟩
LoadClass('omdoc');
RequirePackage('tikzinput');
DefConstructor('\usetheme{}','');
⟨/ltxml⟩
```

EdN:3 now, we load the remaining packages for both versions. [3]

```
⟨*cls⟩
\RequirePackage{tikzinput}
\RequirePackage{stex}
\RequirePackage{latexml}
\RequirePackage{amssymb}
\RequirePackage{tikz}
\usepgflibrary{shapes}
\usetikzlibrary{arrows}
\usetikzlibrary{positioning}
\usetikzlibrary{tikzmark}%experimental/beta but very useful
\usetikzlibrary{fit}
\RequirePackage{url}
\RequirePackage{amsmath}
\RequirePackage{comment}
\RequirePackage{standalone}
\RequirePackage{textcomp}
⟨/cls⟩
⟨*ltxml⟩
RequirePackage('stex');
RequirePackage('latexml');
RequirePackage('amssymb');
```

[3] EdNote: MK: eventually (when tikz support is fully realized in LaTeXML) get rid of the standalone package

```
RequirePackage('graphicx');
RequirePackage('tikz');
RequirePackage('url');
 RequirePackage('amsmath');
⟨/ltxml⟩
```

4.2 Notes and Slides

We define the sizes of slides in the notes. Somehow, we cannot get by with the same here.

```
⟨*cls⟩
\newcounter{slide}
\newlength{\slidewidth}\setlength{\slidewidth}{12.3cm}
\newlength{\slideheight}\setlength{\slideheight}{9cm}
⟨/cls⟩
⟨*ltxml⟩
DefRegister('\slidewidth'     => Dimension('13.5cm'));
DefRegister('\slideheight'     => Dimension('9cm'));
⟨/ltxml⟩
```

`note` The `note` environment is used to leave out text in the `slides` mode. It does not have a counterpart in OMDoc. So for course notes, we define the `note` environment to be a no-operation otherwise we declare the `note` environment as a comment via the `comment` package.

```
⟨*cls⟩
\ifnotes\renewenvironment{note}{\ignorespaces}{}\else\excludecomment{note}\fi
⟨/cls⟩
⟨*ltxml⟩
DefEnvironment('{note}','#body');
⟨/ltxml⟩
```

We start by giving the LaTeXML binding for the `frame` environment from the `beamer` class. We first set up the slide boxes in `article` mode. We set up sizes and provide a box register for the frames and a counter for the slides.

```
⟨*cls⟩
\ifnotes
\newlength{\slideframewidth}\setlength{\slideframewidth}{1.5pt}
```

`frame` We first define the keys.

```
\addmetakey{frame}{label}
\addmetakey[yes]{frame}{allowframebreaks}
\addmetakey{frame}{allowdisplaybreaks}
\addmetakey[yes]{frame}{fragile}
\addmetakey[yes]{frame}{shrink}
\addmetakey[yes]{frame}{squeeze}
```

We redefine the `itemize` environment so that it looks more like the one in `beamer` with `Jacobs` theme. We create the box with the `mdframed` environment from the

equinymous package. Then we define the environment, read them, and construct the slide number and label.

```
\renewenvironment{frame}[1][]%
{\metasetkeys{frame}{#1}%
\stepcounter{slide}\def\@currentlabel{\theslide}%
\ifx\frame@label\@empty\else\label{\frame@label}\fi
```

We redefine the `itemize` environment so that it looks more like the one in `beamer` with `Jacobs` theme.

```
\def\itemize@level{outer}%
\def\itemize@outer{outer}%
\def\itemize@inner{inner}%
\renewcommand\newpage{}%
\renewcommand\metakeys@show@keys[2]{\marginnote{{\scriptsize ##2}}}%
\renewenvironment{itemize}%
{\ifx\itemize@level\itemize@outer\def\itemize@label{$\rhd$}\fi%
 \ifx\itemize@level\itemize@inner\def\itemize@label{$\scriptstyle\rhd$}\fi%
  \begin{list}%
    {\itemize@label}%
    {\setlength{\labelsep}{.3em}\setlength{\labelwidth}{.5em}\setlength{\leftmargin}{1.5em}}%
  \edef\itemize@level{\itemize@inner}}%
{\end{list}}
```

We create the box with the `mdframed` environment from the equinymous package.

```
\begin{mdframed}[linewidth=\slideframewidth,skipabove=1ex,skipbelow=1ex,
userdefinedwidth=\slidewidth,align=center]\sf}
{\medskip\miko@slidelabel\end{mdframed}}
⟨/cls⟩
⟨*ltxml⟩
DefEnvironment('{frame}[]',
   "<omdoc:omgroup layout='slide'>"
  .  "#body\n"
  ."</omdoc:omgroup>\n\n",
 afterDigestBegin=>sub {
   $_[1]->setProperty(theory=>LookupValue('current_module')); });
⟨/ltxml⟩#$
```

Now, we need to redefine the frametitle (we are still in course notes mode).

`\frametitle`

```
⟨*cls⟩
\renewcommand{\frametitle}[1]{{\Large\bf\sf\color{blue}{#1}}\medskip}
\fi
⟨/cls⟩
⟨*ltxml⟩
DefConstructor('\frametitle{}',
  "\n<omdoc:metadata><dc:title>#1</dc:title></omdoc:metadata>");
⟨/ltxml⟩
```

\frameimage We have to make sure that the width is overwritten, for that we check the
EdN:4 \Gin@ewidth macro from the graphicx package[4]

```
⟨*cls⟩
\newcommand\frameimage[2][]{\stepcounter{slide}%
\iframeimages%
\def\Gin@ewidth{}\setkeys{Gin}{#1}%
\ifnotes\else\vfill\fi%
\ifx\Gin@ewidth\@empty\mycgraphics[width=\slidewidth,#1]{#2}\else\mycgraphics[#1]{#2}\fi
\par\strut\hfill{\footnotesize Slide \arabic{slide}}%
\ifnotes\else\vfill\fi%
\fi}%iframeimages
⟨/cls⟩
⟨*ltxml⟩
DefMacro('\frameimage[]{}','\@frameimage{\includegrahics[#1,width=\slidewidth]{#2}}');
DefConstructor('\@frameimage{}',"<omdoc:omgroup layout='slide'>#1</omdoc:omgroup>\n");
⟨/ltxml⟩
```

4.3 Header and Footer Lines

Now, we set up the infrastructure for the footer line of the slides, we use boxes for the logos, so that they are only loaded once, that considerably speeds up processing.

```
⟨*cls⟩
\newlength{\slidelogoheight}
\ifnotes\setlength{\slidelogoheight}{.4cm}\else\setlength{\slidelogoheight}{1cm}\fi
\newsavebox{\slidelogo}\sbox{\slidelogo}{\includegraphics[height=\slidelogoheight]{jacobs-logo}
```

Now, we set up the copyright and licensing, the copyright remains with the author, but we use the Creative Commons Attribuition-ShareAlike license to strengthen den public domain. Here the problem is that we want a hyperref on the CC logo, if hyperref is loaded, and otherwise not. As hyperref is always loaded, we have to find out at the beginning of the document whether it is, set up a switch, and later in the footer line decide what to do.

```
\def\source{Michael Kohlhase}% customize locally
\def\copyrightnotice{\footnotesize\copyright:\hspace{.3ex}{\source}}
\newsavebox{\cclogo}\sbox{\cclogo}{\includegraphics[height=\slidelogoheight]{cc_somerights}}
\newif\ifcchref\cchreffalse
\AtBeginDocument{\@ifpackageloaded{hyperref}{\cchreftrue}{\cchreffalse}}
\def\licensing{\ifcchref\href{http://creativecommons.org/licenses/by-sa/2.5/}{\usebox{\cclogo}}
```

EdN:5 Now, we set up the slide label for the article mode[5]

\slidelabel

```
\newcommand\miko@slidelabel%
{\vbox to \slidelogoheight{\vss\hbox to \slidewidth%
{\licensing\hfill\copyrightnotice\hfill\arabic{slide}\hfill\usebox{\slidelogo}}}}
```

[4] EdNote: MK@DG; we need to do that in the LaTeXML binding as well!
[5] EdNote: see that we can use the themes for the slides some day. This is all fake.

4.4 Colors and Highlighting

Now, we set up an infrastructure for highlighting phrases in slides. Note that we use content-oriented macros for highlighting rather than directly using color markup. The first thing to to is to adapt the green so that it is dark enough for most beamers

```
\AtBeginDocument{\definecolor{green}{rgb}{0,.5,0}\definecolor{purple}{cmyk}{.3,1,0,.17}}
```

We customize the `\defemph`, `\notemph`, and `\stDMemph` macros with colors for the use in the `statements` package. Furthermore we customize the `\@@lec` macro for the appearance of line end comments in `\lec`.

```
% \def\STpresent#1{\textcolor{blue}{#1}}
\def\defemph#1{{\textcolor{magenta}{#1}}}
\def\notemph#1{{\textcolor{magenta}{#1}}}
\def\stDMemph#1{{\textcolor{blue}{#1}}}
\def\@@lec#1{(\textcolor{green}{#1})}
⟨/cls⟩
⟨*ltxml⟩
#DefMacro('\defemph{}','{\textcolor{magenta}{#1}}');
#DefMacro('\notemph{}','{\textcolor{magenta}{#1}}');
⟨/ltxml⟩
```

I like to use the dangerous bend symbol for warnings, so we provide it here.

`\textwarning` as the macro can be used quite often we put it into a box register, so that it is only loaded once.

```
⟨*cls⟩
\pgfdeclareimage[width=.9em]{miko@small@dbend}{dangerous-bend}
\def\smalltextwarning{\pgfuseimage{miko@small@dbend}\xspace}
\pgfdeclareimage[width=1.5em]{miko@dbend}{dangerous-bend}
\def\textwarning{\raisebox{-.05cm}{\pgfuseimage{miko@dbend}}\xspace}
\pgfdeclareimage[width=2.5em]{miko@big@dbend}{dangerous-bend}
\def\bigtextwarning{\raisebox{-.05cm}{\pgfuseimage{miko@big@dbend}}\xspace}
⟨/cls⟩
⟨*ltxml⟩
DefMacro('\textwarning','\@textwarning\xspace');
DefConstructor('\@textwarning',"");
⟨/ltxml⟩
```

4.5 Front Matter, Titles, etc

We need to redefine the frontmatter macros inherited from the `beamer` class for LaTeXML, since there they take an optional argument.

```
⟨*ltxml⟩
DefMacro('\title[]{}', '\@add@frontmatter{ltx:title}{#1}');
DefMacro('\date[]{}', '\@add@frontmatter{ltx:date}[role=creation]{#1}');
DefMacro('\author[]{}', sub { andSplit(T_CS('\@author'),$_[1]); });#$
⟨/ltxml⟩
```

Now, we specialize the slide environment that we have implemented above or inherited from `seminar.cls` for some abbreviations, e.g. separator slides and title slides.

```
⟨*cls⟩
\ifnotes\newcommand\titleframe{\maketitle}\else
\newcommand\titleframe{\begin{frame}\titlepage\end{frame}}\fi
\newenvironment{titleframewith}{\begin{frame}\titlepage}{\end{frame}}
\newenvironment{ttitle}{\begin{center}\LARGE\begin{tabular}{|c|}\hline}%
 {\\\hline\end{tabular}\end{center}\vspace{1ex minus 1ex}}
\newenvironment{ttitlejoint}[1]%
{\newbox\boxwith\setbox\boxwith\hbox{\begin{tabular}{c}{\em joint work with}\\#1\end{tabular}}%
\begin{center}\LARGE\begin{tabular}{c}\color{red}}%
{\\\box\boxwith\end{tabular}\end{center}%
\vspace{1ex minus 1ex}}
⟨/cls⟩
⟨*ltxml⟩
DefConstructor('\titleframe',"<omdoc:ignore>titleframe elided here</omdoc:ignore>");
DefEnvironment('{titleframewith}',
               "<omdoc:ignore>begin elided titleframe</omdoc:ignore>"
               . "#body"
              ."<omdoc:ignore>end elided titleframe</omdoc:ignore>");
DefEnvironment('{titleslide}',"");
DefEnvironment('{titleslide}',"<omdoc:omgroup>#body</omdoc:omgroup>");
DefEnvironment('{ttitle}', "\n<dc:title>#body</dc:title>");
⟨/ltxml⟩
%      Must be first command on slide to make positioning work.
⟨*cls⟩
\newcommand\putgraphicsat[3]{%
 \begin{picture}(0,0)\put(#1){\includegraphics[#2]{#3}}\end{picture}}
\newcommand\putat[2]{\begin{picture}(0,0)\put(#1){#2}\end{picture}}
⟨/cls⟩
```

4.6 Sectioning

If the `sectocframes` option is set, then we make section frames.

```
⟨*cls⟩
\ifsectocframes
\if@part\newcounter{mpart}
\newcounter{mchapter}
\newcounter{msection}[mchapter]
\else
\newcounter{msection}
\fi
\newcounter{msubsection}[msection]
\newcounter{msubsubsection}[msubsection]
\newcounter{msubsubsubsection}[msubsubsection]
\ifnotes\else% only in slides
\renewcommand\at@begin@omgroup[3][]{\begin{frame}%
```

```
\vfill\Large\centering
\red{\ifcase\section@level\or
\stepcounter{mpart}Part \Roman{mpart}\or%
\stepcounter{mchapter}Chapter \arabic{mchapter}\or
\stepcounter{msection}\if@part\arabic{mchapter}.\fi\arabic{msection}\or
\stepcounter{msubsection}\if@part\arabic{mchapter}.\fi\arabic{msection}.\arabic{msubsection}\or
\stepcounter{msubsubsection}\if@part\arabic{mchapter}.\fi\arabic{msection}.\arabic{msubsection}
\stepcounter{msubsubsubsection}\if@part\arabic{mchapter}.\fi\arabic{msection}.\arabic{msubsecti
\quad #3}\vfill
\end{frame}}
\fi% ifnotes
\fi% ifsectocframes
⟨/cls⟩
```

4.7 Miscellaneous

The following fixes an error I do not understand, this has something to do with beamer compatibility, which has similar definitions but only up to 1.

```
⟨*cls⟩
\expandafter\def\csname Parent2\endcsname{}
%    \begin{macrocode}
%
% We need to disregard the columns macros introduced by the |beamer| class
%    \begin{macrocode}
\ifnotes
\renewenvironment{columns}%
{\par\noindent\begin{minipage}\slidewidth\centering\leavevmode}%
{\end{minipage}\par\noindent}
\newsavebox\columnbox
\renewenvironment{column}[1]%
{\begin{lrbox}{\columnbox}\begin{minipage}{#1}}%
{\end{minipage}\end{lrbox}\usebox\columnbox}
\fi
⟨/cls⟩
⟨*ltxml⟩
DefEnvironment('{columns}',"#body");
DefEnvironment('{column}{}',"#body");
```

We also need to deal with overlay specifications introduced by the `beamer` class.[6]

EdN:6
EdN:7

[7]

```
DefConstructor('\uncover','#1');
#Define a Beamer Overlay Parameter type
 DefParameterType('BeamerOverlay', sub {
   my ($gullet) = @_;
   my $tok = $gullet->readXToken;
```

[6] EdNote: this is just to keep latexml quiet, no real functionality here.

[7] EdNote: Deyan: We reuse the CMP itemizations defined in the omdoc.cls.ltxml binding, adjusting the parameters to be overlay-sensitive

```
  if (ref $tok && ToString($tok) eq '<') {
    $gullet->readUntil(T_OTHER('>'));
  } else {
    $gullet->unread($tok) if ref $tok;
    undef; }},
    reversion=> sub {
 (T_OTHER('<'), $_[0]->revert, T_OTHER('>'));
      });

#Take the "from" field of the overlay range
sub overlayFrom {
  return "" unless defined $_[0];
  my $overlay=ToString($_[0]); $overlay =~ /^(\d+)/; $1;}

#Reuse the CMP itemizations, only adjust the \item constructors.
DefMacro('\beamer@group@item[] OptionalBeamerOverlay IfBeginFollows', sub {
  my($gullet,$tag,$overlay,$needwrapper)=@_;
  $overlay=$overlay||T_OTHER("");
  ( T_CS('\group@item@maybe@unwrap'),
    ($needwrapper ? (Invocation(T_CS('\beamer@group@item@wrap'),$tag,$overlay)->unlist) : ()) )
DefConstructor('\beamer@group@item@wrap {} OptionalBeamerOverlay',
       "<omdoc:omtext ?#2(overlay='&overlayFrom(#2)')()>"
       . "?#1(<dc:title>#1</dc:title>)()"
            . "<omdoc:CMP>",
       beforeDigest=>sub {
 Let('\group@item@maybe@unwrap','\group@item@unwrap');
 #$_[0]->bgroup;
return; },
       properties=>sub{ RefStepItemCounter(); });
#DefConstructor('\beamer@itemize@item[] OptionalBeamerOverlay',
#       "<omdoc:li ?#2(overlay='&overlayFrom(#2)')() >"
#     . "?#1(<dc:title>#1</dc:title>)()",
#       properties=>sub{ RefStepItemCounter(); });
DefConstructor('\beamer@enumerate@item[] OptionalBeamerOverlay',
       "<omdoc:li ?#2(overlay='&overlayFrom(#2)')() >"
     . "?#1(<dc:title>#1</dc:title>)()",
       properties=>sub{ RefStepItemCounter(); });
DefConstructor('\beamer@description@item[] OptionalBeamerOverlay',
       "<omdoc:di ?#2(overlay='&overlayFrom(#2)')() >"
       . "?#1(<omdoc:dt>#1</omdoc:dt>)()<omdoc:dd>", # trust di and dt to autoclose
       properties=>sub{ RefStepItemCounter(); });
⟨/ltxml⟩#$
```

Now, some things that are imported from the pgf and beamer packages:

```
⟨*ltxml⟩
DefMacro('\putgraphicsat{}{}{}','\mygraphics[#2]{#3}');
DefMacro('\putat{}{}','#2');
⟨/ltxml⟩

⟨*cls⟩
\ifproblems\newenvironment{problems}{}{}\else\excludecomment{problems}\fi
```

```
⟨/cls⟩
⟨*ltxml⟩
DefEnvironment('{problems}','#body');
⟨/ltxml⟩
```

4.8 Support for MathHub

`\mhframeimage` Use the current value of `\mh@currentrepos` or the value of the `mhrepos` key if it is given in `\frameimage`.

```
⟨cls⟩\addmetakey{Gin}{mhrepos}
⟨ltxml⟩DefKeyVal('Gin','mhrepos','Semiverbatim');
⟨ltxml⟩RawTeX('
⟨*ltxml | cls⟩
\newcommand\mhframeimage[2][]{\metasetkeys{Gin}{#1}%
\edef\mh@@repos{\mh@currentrepos}%
\ifx\Gin@mhrepos\@empty\frameimage[#1]{\MathHub{\mh@currentrepos/source/#2}}%
\else\frameimage[#1]{\MathHub{\Gin@mhrepos/source/#2}}\fi}
⟨/ltxml | cls⟩
⟨ltxml⟩');
```

4.9 Finale

Finally, we set the slide body font to the sans serif, and we terminate the LaTeXML bindings file with a success mark for perl.

```
⟨cls⟩\ifnotes\else\sf\fi
⟨ltxml⟩1;
```