How to use package MultEdu

(How to prepare interesting and attractive teaching material)

János Végh

V0.5.5 @2016.08.19 (https://github.com/jvegh/MultEdu\V0.5.4)

Abstract

For teaching my own courses, I developed a set of macros, since to display the course material under different circumstances different forms of teaching materials are needed. On the lectures I present the theoretical material in form of slides, and the explanation referring to the slides (of course in somewhat compressed form) I offer for my students, in a booklet-like form. My students studies that material either from printed hard copies, or from screen, using a browser or sometimes on mobile devices. The field is in continuous development, so I need to develop my teaching material also continuously. Because of this, it is a must to develop those different forms of the material synchronously. The simplest way to do so, is to use the same source, with proper formatting instructions. It is a serious challange, to develop course material for the today's students, who are used to lessons with high resolution, attractive graphics, good computer background.

As a common base, I used LaTeX, from which I produce the slides from the lectures using package beamer, and the reading material using package 'memoir'. This latter one can even reach the "on demand printing" quality. The printed material attempts to catch the attention with attractive graphical appearance, using above the average amount of figures (of course on can make also 'book-like' book, too). The booklet-like version contains all figures from the lectures, and some comprehensive version of the text of the lecture. The same text appears in screen-oriented form in the WEB-book format, and in the eBook compatible (native PDF) format.In those two forms (mainly targeting the small-screen mobile devices) bigger fonts are used and one figure/screen is displayed.

To satisfy those, somewhat contradictional requirements, one must make bargains,and more time and care must be invented in the formatting. The macros allow to support also foreign languages, and even to prepare course in English and your own language side by side. Using the possibilities of LaTeX, animations, movies, web-pages, sound files, etc. can also be embedded, but one has to think about the equivalent appearance on hard copies.

First edition: August 2016

V0.5.5 @2016.08.19 (https://github.com/jvegh/MultEdu V0.5.4)
10 09 08 07 06 05 04 03 02 01 19 18 17 16 15 14 13

Contents

Chapter

1 General information

1.1 Introduction

For teaching my courses I needed to develop course material, in different forms of appearance; the present package is a by-product of this activity. Good course materials develop quickly, especially, if the field of science in question itself is renewed daily. In informatics, the technology, the statistics, the products, the tools, etc. change year by year, and this alone is a good reason to renew the teaching material for a new semester.

The todays education needs the course material in various forms: in the lecture room for the projected picture well organized text with many pictures are needed, which also serve as a good guide for the lecturer, too. To prepare for the exams, the explanation provided by the lecturer when projecting the slides is also needed. That means, the course material should be available in printable and browsable form, as well as for mobile devices. A frequent case is that the same material shall be provided in foreign language(s), for foreign students. In many cases one can rely to good textbooks, but for the more specialized courses this course material serves as a basic tool for preparing to the exams.

The present macro package is developed for my own purposes, which I attempted to develop in such a way, that when developing course material, one should not deal with the technology of displaying. In this way also others can use the package, when they follow the rules. The package is quite good in some fields, at some point I needed to make a bargain between the different needs, not perfect in some points, and of course in much more aspects I did not have the time to develop features.

The present document is a demo and test at the same time. It attempts to describe the many features, and also tests if the features really work. Because of the many features, and their interference, this job needs a lot of work and time, so the documentation does not always match the actual features, especially in this initial phase.

The macro package can be used at (at least) three different levels. Even the lowest level assumes some familiarity with LaTeX. At the very basic level, you might just take the package, replace and modify files in the distribution. At the advanced level (this assumes reading the User's manual ☺) the user learns the facilities provided in the package, and prepares his/her courses actively using those facilities. Power users might add their own macros (preferably uploaded to the distribution), i.e. take part in the development.

1.2 Installing and utilizing `MultEdu`

MultEdu, as any package based on LaTeX, assumes that the user has experiences with using LaTeX. I.e. some LaTeX distribution must already be installed on the system of the user. If you want to use the batch processing facility, the CMake system must also be installed.

For the simplicity of utilization and starting up, the best way is to create a main directory for your family of projects and a subdirectory for your first project, as described below. The quickest way is to copy `./Workstuff` (after deleting and renaming some files) and to prepare your own "Hello World" program. Making minor changes to that source you may experience some features of the package. Then, it is worth at least to skim the user's manual, to see what features you need. After that, you may start your own development. At the beginning text only, later you can learn the advanced possibilities. Do not forget: LaTeX is hard, it needs accurate coding, and so is MultEdu, too. Frequent saving and using versioning can help a lot.

1.3 Structure of `MultEdu`

The MultEdu system is assumed to be used with the directory structure below. It comes with two main subdirectories: `./common` comprises all files of the MultEdu system, and `./Workstuff` models the users subdirectory structure.

```
.
|-- common
|-- WorkStuff
```

You may add your project groups stuff like

```
.
|-- Exams
|-- Labs
|-- Lectures
|-- Papers
```

which directories have a subdirectory structure similar to that of `|-- WorkStuff`

1.3.1 Subdirectory `common`

Subdirectory `./common` comprises some special subsubdirectories and general purpose macro files. MultEdu attempts to be as user-friendly as possible: it uses default settings, files, images, etc., to allow a quick start for a new development.

```
.
|-- common
| |-- defaults
| |-- formats
| |-- images
```

Subsubdirectory `./defaults` contains some default text, like copyright. In general, if the user does not provide its own elements, MultEdu uses the defaults instead (provided that using and presenting it is not disabled, see later.)

Subsubdirectory `./formats` contains the possible format specification macros, here you can add your own format macros.

Subsubdirectory `./images` contains some images, partly the ones which are used as defaults.

1.3.2 Subdirectory `Workstuff`

Subdirectory `./Workstuff` contains the files of the present demo, and serves as an example of using the system (a kind of User's Guide). It contains a sample project `./Workstuff/Demo`, which has three main files.

```
|-- WorkStuff
| |-- Demo
| . |-- CMakeLists.txt
| . |-- Demo.tex
| . |-- Main.tex
```

The real main source file is `Main.tex`, and `Demo.tex` is a lightweight envelope to it. (if you want to use UseLATEX, you need to use the file with name `Main.tex`, the envelop must be concerted with the CMakeLists.txt file)

Subdirectory `./Workstuff` has some subsubdirectories, for different goals.

```
|-- WorkStuff
| |-- Demo
| . |-- build
| . . . |-- build
| . |-- dat
| . |-- fig
| . |-- lst
| . |-- src
```

The file `Main.tex` inputs files in the sub-subdirectories.

Subsubdirectory
| . |-- `src` is the place for the user's source files,
| . |-- `fig` for the images.
| . |-- `lst` for the program source files,
| . |-- `dat` for the other data (like tabulated data for pgfplot or TikZ figures source code).

You may use (and handle!, especially in CMakeLists.txt) further subsubdirectories.

It is also possible to use CMake package UseLATEX for compiling your text to different formats and languages in batch mode; producing the documents in different languages and formats in one single step. File `CMakeLists.txt` serves for that goal.

Subsubdirectories
|-- `build` and
| . . |-- `build`
are only needed if using CMake; they contain temporary files created during processing. The system also makes its own copy of the subdirectory `common` in your project directory (corresponding to subdirectory `Demo`). Those files can be deleted any time: when compiling, CMake will regenerate them.

1.3.3 Generated files

During compilation, LaTeX generates a number of different working files. These will unfortunately pollute the project base directory. As shown in section 1.3.2, only 3 files are needed for the operation, the rest can be deleted any time.

Compilation in batch mode also prepares some `.tex` files, which can be removed also any time, or even can be compiled manually. Do not forget to edit file `src/Defines.tex` before compiling, if you use them for that goal.

1.4 The `MultEdu` distribution kit

The `MultEdu` package come with full source (and full faith). The author is rather power user than LaTeX expert. Many of the macros are adapted from ideas and solution on the Internet. The source contains references to the original publisher, but the users' guide does not waste space for acknowledgements. However, the author acknowledges the contribution of all respective authors both for the code and the support on different user communities.

The package contains also some `.pdf` files in different output formats and languages. The file name do not contain the version number (their title page does). The purpose of those files (in addition to serving as users' guide) to allow the potential users to decide at a glance, whether they like the provided features.

The package MultEdu is provided 'as is'. It is developed continuously and in a non-uniform way. I myself can develop course materials with it. Both macros and documention keep developing, but it requires (lot of) time. Reports on faults in operation or errors is documentation is evaluated as help in the development, even I might consider feature requests.

CHAPTER 2

Sectioning document

2.1 Document units

Basically, the document must be organized as 'beamer' needs it, but to print it in a book-like form, the sectioning must be changed, and also the package 'beamerarticle' must be used. In order to provide a uniform wrapper around sectioning, MultEdu introduces its own sectioning units. The source text itself comprises 'frames'.

2.1.1 Frames

These units actually correspond to the ones used in format 'book', and MultEdu transforms them properly when preparing slides.

Usage:

```
\MEframe[keys]{subtitle}{content}
```

Legal keys are

`shrink=true|false` and `plain=true|false`

By default, both are false.

2.1.2 Chapter

Correspondingly, the biggest unit is the 'chapter'. (As mentioned, for slides it is transformed to 'section'.) Usage:

```
\MEchapter[short title]{long title}
```

When preparing slides, it is transformed to `\section`

2.1.3 Section and below

The next, smaller unit is the 'section'. (As mentioned, for slides it is transformed to 'subsection'.) Usage:

```
\MEsection[short title]{long title}
```

In a similar way, there exists

`\MEsubsection [short title] {long title}` and

`\MEsubsubsection [short title] {long title}`

the latter one is transformed for slides to `\paragraph`.

2.2 Dual language sources

It happens, that I teach the same course in my mother tongue for my domestic students, and in English, for foreign students. The course material is the same, and it must be developed in parallel. Obviously it is advantageous, if they are located in the same source file, side by side; so they can be developed in the same action. The `\UseSecondLanguage` macro supports this method.

The macros introduced above have a version with prefix 'MED' rather than 'ME' only, which takes double argument sets (arguments for both languages). Depending on whether `\UseSecondLanguage` is defined, the first or the second argument set is used.

2.2.1 Switching between languages

Usage:

```
\UseSecondLanguage{YES}
```

where the argument in `{}` is not relevant, only if this macro is defined or not.

The two kinds of macros can be mixed, but only the 'D' macros are sensitive to changing the language.

2.2.2 Frames

In dual language documents, usually

```
\MEDframe[keys]{subtitle, first language}
{content, first language } {subtitle, second
language} {content, second language}
```

is used. I.e. the user provides titles and contents in both languages, and for preparing the output, selects one of them with `\UseSecondLanguage`.

Although that point of view has less importance, when presenting a conference talk, it is very important to properly utilize the available time. MultEdu can support this through displaying a chrono time on the slides. An example:

```
\MEframe{Frame title
\ifx\EnableTimer\undefined \else
\initclock\fi}
```

The MultEdu also warns the with changing the color of the time value, if we are approaching the end of the lecture. The maximum time can be set using instruction `\def\LectureTime{minutes}`, the default value is 15. The timer starts when the second slide is projected, and is reset, when the lecturer returns to that slide.

2.2.3 Chapter

Correspondingly, the biggest unit in a dual language document is the 'Dchapter'. (As mentioned, for slides it is transformed to 'Dsection'.) Usage:

`\MEDchapter[short title1]{long title1}{short title2}{long title2}`

which is transformed to

`\MEchapter[short title1]{long title1}` or

`\MEchapter[short title2]{long title2}` calls,

depending on whether `\UseSecondLanguage` is or is not defined.

The usage of the lower units is absolutely analogous.

2.3 Chapter illustration

Some book styles also allow presenting some illustration at the beginning of the chapters.

Usage:

`\MEchapterillustration{file}`

For slides, the illustration appears in a 'plain' style style. For books, the picture is placed at the beginning of the chapter. If the file name is empty, a `fig/DefaultIllustration.png` file is searched. If the file not found, no illustration generated.

If macro `\DisableChapterIllustration` is defined, no picture generated.

2.4 Concerting text on slides and printed output

The printed outputs usually contain much more text, than the slides. This extra text can be placed in the source file inside an `\ao{text}` (article only) macro, where the extra text appears inside the macro. That text appears only in the printed output, and is not visible on the slides. Take case, the text must be reasonable in both version; especially if used within a sentence.

2.5 Floating objects

LaTeX might handle objects like figures, tables, program listings, etc. as "floating objects, i.e. they might appear at a place, where LaTeX thinks to be optimal. This place is not necessarily the place in the printed materials, what you expect based on the referece point in the source but they do on the slides. Because of this, do not refer to the listings like 'In the following listing'. Instead, using something like '`In listing \ref{lst:hello.cpp}`' is suggested.

In contrast, on the slides the lobject appears in the right place, but has no number. Because of this the really good method of referencing is something like '`In listing \ao{\ref{lst:hello.cpp}}`' is the really good one. Take care of the meaning in the sentence, both on slides and printed output.

Chapter 3

Preparing program listings

When teaching programming, it is a frequent need to display program listings. Through using package 'listings', MultEdu can implement this in very good quality. For details not described here see documentation of package 'listings'.

Notice that here the ratio of the listings within the text is unusually high, so it is very hard for the compiler to find good positioning. In the case of real texts, the page is much more aesthetic.

3.1 Setting appearance

Package 'listings' allows to set up the style of displaying program listings according to our taste (and the requirements). MultEdu pre-sets some style and allows to modify it as much as you like.

Macro

`\MESetStandardListingFormat` sets up a default appearance, and no programming language. Macro

`\MESetListingFormat[options]{language}`

sets the language, the same appearance as macro

`\MESetStandardListingFormat`

and also allows to overwrite parameters of 'listings' through 'options'.

3.2 Displaying inline fragments

A tipical task is to display a shorter fragment, like a line or a keyword. It is possible using `\lstinline|code|`.

The LaTeX commands appearing in this documentation are produced in such a way that at the beginning of the chapter commands

`\MESetListingFormat{TeX}`

`\lstset{basicstyle= \ttfamily\color{black}\normalsize}`

or

`\MESetListingFormat[basicstyle= \ttfamily\color{black}\normalsize]{TeX}`

are issued (otherwise the character size of the program text will be too small).

3.3 Displaying program listings

Program listings can be displayed using macro `\MESourceFile[keys] {filename} {caption} {label}{scale}`. Possible keys: `wide[=false],decorations[={}]`.

Listing 3.1: "Hello World" – a C++ way

```
#include <iostream>
using namespace std;
int
main ( int argc, char ** argv )
{
  // print welcome message
  cout << "Hello World" << endl;
  return 0;
}
```

The command used to display Listing 3.1 was

`\MESourceFile[language={[ISO]C++}] {lst/HelloWorld.cpp} {A "Hello World"- C++ program} {lst:hello.cpp}{}`

Many times one needs wider program listings. In the case of the two-column printing, the listing shall fill the width of the two columns. In the case of one-column printing, the narrow list extend to 70% of the text width, while the wide lists span the width of both columns.

The wide listings can be placed even hardly on the printed page (the first proper place, relative to the appearance of the macro is the top of the next page), and in addition, the orders of normal and wide listings cannot be changed. Because of this, the place where the listing appears, might be relatively far from the place of referencing it.

The command used to display Listing 3.2 :

`\MESourceFile[language={[ISO]C++},wide] {lst/HelloWorld.cpp} {A "Hello World"- C++ program, wide} {lst:Whello.cpp}{}`

3.4 Decorations on listings

Different decorations can be placed on top of listings. To do so, one has to use the keyword `decorations`, and to insert as arguments the macros presented in this section.

Listing 3.2: A "Hello World"- C++ program, wide

```
#include <iostream>
using namespace std;
int
main ( int argc, char ** argv )
{
  // print welcome message
  cout << "Hello World" << endl;
  return 0;
}
```

The general form:

`\MESourceFile[options, decorations={ list of decorations } ] {source file} {caption} {label}{}`

where the list of decorations may contain any of the decoration macros presented in the section.[1] In `options` any option, used by package 'listings' applies.

3.4.1 Highlighting lines

To highlight a program body in listing 3.3 the macro

`\MESourceFile[language={[ISO]C++}, decorations={ \MESourcelinesHighlight {HelloBalloon} {lst:HLhello.cpp} {6}{8} } ] {lst/HelloWorld.cpp} {"Hello World" -- a C++ way, kijel˜Olt} {lst:HLhello.cpp}{}`

shall be used.

Listing 3.3: "Hello World" – a C++ way, highlighted

```
#include <iostream>
using namespace std;
int
main ( int argc, char ** argv )
{
  // print welcome message
  cout << "Hello World" << endl;
  return 0;
}
```

3.4.2 Commenting highlighted lines

The higlighting box can also be commented. Using macro

`\MESourceBalloonComment[keys]{BallonName} {ShiftPosition} {Comment} {CommentShape}`

allows to comment the balloon created previously. Here `BallonName` is the first argument of `\MESourcelinesHighlight`, `ShiftPosition` is the shift of the comment box, `Comment` is the comment text. Possible keys, with defaults are:

`width[=3cm]` and `color[=deeppeach]`.

Listing 3.4 is produced using macro

`\MESourceFile[language={[ISO]C++},wide, decorations={ \MESourcelinesHighlight {HelloBalloon} {lst:HLChello.cpp} {6}{8} \MESourceBalloonComment{HelloCBalloon} {0cm,0cm} {This is the body} {CommentShape} } ] {lst/HelloWorld.cpp} {"Hello World" -- a C++ way, commenting highlighted} {lst:HLhello.cpp}{}`

3.4.3 Commenting source lines

The individual source lines can also be commented, see Listing 3.5. To produce it, the command was:

`\MESourceFile[language={[ISO]C++}, decorations={ \MESourcelineComment{lst:Chello.cpp} {6} {-1cm,0cm} {This is a comment} {CommentShape} } ]{lst/HelloWorld.cpp} {"Hello World" -- a C++ way, commenting source lines} {lst:Chello.cpp}{}`

3.4.4 Numbered balls to listing

On the program listing numbered balls can also be located, for referencing the lines from the text. This can be done using macro

`\MESourcelineListBalls[keys]{ListingLabel}{List of lines}`

which puts a numbered ball at the end of the listed lines. Here `ListingLabel` is the label of the listing, `List of lines` is the list of sequence numbers of the lines to be marked. Possible key, with defaults:

`color[=orange]` and `number[=1]`.

Notes:

- When making slides, the balls will be put to separated slides.
- The positioning using geometrical positions, does not consider 'firstline'.

The marked lines can then be referenced through the balls like '(Listing 3.6 ②) is the return instruction'. It can be produced using

[1] The compiler prepares in the first pass the program listing, the decorations follow in the next pass.

Listing 3.4: "Hello World" – a C++ way, remark to the highlighing

```
#include <iostream>
using namespace std;
int
main ( int argc, char ** argv )
{
  // print welcome message
  cout << "Hello World" << endl;   This is the body
  return 0;
}
```

Listing 3.5: "Hello World" – a C++ way, commenting source lines

```
#include <iostream>
using namespace std;
int
main ( int argc, char ** argv )
{
  // print welcome message   This is a comment
  cout << "Hello World" << endl;
  return 0;
}
```

`\MEBall{Listing~\ref{lst:LBhello.cpp}}{2}`

To produce Listing 3.6, the macro

```
\MESourceFile[language={[ISO]C++},
decorations={
\MESourcelineListBalls{lst:LBhello.cpp}{3,8,5}
} ] {lst/HelloWorld.cpp} {"Hello World" -- a
C++ way, with balls} {lst:LBhello.cpp}{}
```

has been used

Listing 3.6: "Hello World" – a C++ way, with balls

```
#include <iostream>
using namespace std;
int 1
main ( int argc, char ** argv )
{ 3
  // print welcome message
  cout << "Hello World" << endl;
  return 0; 2
}
```

3.4.5 Figure to listing

Sometimes one might need to insert figures into the listing. The macro is

`\MESourcelineFigure[keys] {SourceLabel} {LineNo} {ShiftPosition} {GraphicsFile}`.

Possible key is `width[=3cm]`

To produce Listing 3.7, macro

```
\MESourceFile[language={Verilog},wide,
decorations={
\MESourcelineFigure[width=5.2cm]
{lst:forloops.v}{8} {3.0,-.3} {fig/forloops}
} ] {lst/forloops.v} {Implementing
\ctext{for} loop with repeating HW}
{lst:forloops.v}{}
```

was used.

3.5 Other related macros

3.5.1 Comparing source files

Sometimes it is worth to compare source files, side by side. The macro for this is

```
\MESourceFileCompare[keys]{source file1}
{source file2} {caption} {label}
```

The command to produce Listing 3.9 is

```
\MESourceFileCompare[language={[ANSI]C}]
{lst/lower1.c} {lst/lower2.c} {Comparing two
routines for converting string to lower case}
{lst:lower12.c}
```

The macro does not touch the source files. In the figure, the empty lines, allowing to compare the source files with easy, were inserted manually.

3.5.2 Source with output

It is also useful sometimes to show the source file with its output. The macro

```
\MESourceFileWithResult} [keys]{source file}
{result file} {caption} {label}
```

allows to do that.

For producing Listing 3.10 the command

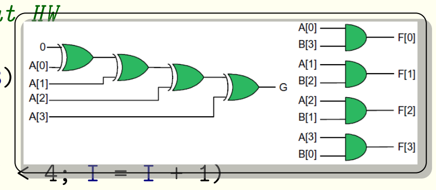
```
\MESourceFileWithResult [language=C++,wide,
decorations={ \MESourcelineListBalls
{lst:calculatorwithresult} {13,14,16,18,19}
}] {lst/expensive_calculator.cpp}
{lst/calculatorresult.txt} {The calculator
program with its result}
{lst:calculatorwithresult}
```

was used.

Listing 3.7: Implementing `for` loop with repeating HW

```
// for == repeat HW

always @(A or B)
begin
  G = 0;
  for (I = 0; I < 4; I = I + 1)
  begin
    F[I] = A[I] & B[3-I];
    G = G ^ A[I];
  end
end
```



Listing 3.8: Comparing two routines for converting string to lower case

```
/* Convert string to lowercase: slow */
void lower1(char *s)
{
   int i;

   for (i = 0; i < strlen(s); i++)
   if (s[i] >= 'A' && s[i] <= 'Z')
    s[i] -= ('A' - 'a');
}
```

Listing 3.9: Comparing two routines for converting string to lower case

```
/* Convert string to lowercase: faster */
void lower2(char *s)
{
  int i;
  int len = strlen(s);
  for (i = 0; i < len; i++)

    if (s[i] >= 'A' && s[i] <= 'Z')
      s[i] -= ('A' - 'a');
}
```

Listing 3.10: The calculator program with its result

```
// Expensive Calculator
// Demonstrates built-in arithmetic operators

#include <iostream>
using namespace std;

int main()
{
  cout << "7 + 3 = " << 7 + 3 << endl;
  cout << "7 - 3 = " << 7 - 3 << endl;
  cout << "7 * 3 = " << 7 * 3 << endl;

  cout << "7 / 3 = " << 7 / 3 << endl;      1
  cout << "7.0 / 3.0 = " << 7.0 / 3.0 << endl;   2

  cout << "7 % 3 = " << 7 % 3 << endl;      3

  cout << "7 + 3 * 5 = " << 7 + 3 * 5 << endl;   4
  cout << "(7 + 3) * 5 = " << (7 + 3) * 5 << endl;   5

  return 0;
}
```

Listing 3.11: The calculator program with its result

```
7 + 3 = 10
7 - 3 = 4
7 * 3 = 21
7 / 3 = 2
7.0 / 3.0 = 2.33333
7 % 3 = 1
7 + 3 * 5 = 22
(7 + 3) * 5 = 50
```

3.6 Extra program languages

For my own goals, in addition to the programming languages defined in package 'listings', some further languages have been defined:

- diff
- [DIY]Assembler
- [ARM]Assembler
- [x64]Assembler
- [y86]Assembler

Chapter

4 Inserting figures

4.1 Traditional figures

Traditional figures can be displayed using macro

`\MEfigure[keys]{image file} {caption} {label} {copyright} {ScaleFactor}`.

Possible keys: `wide`.

©2011 http://pinterest.com

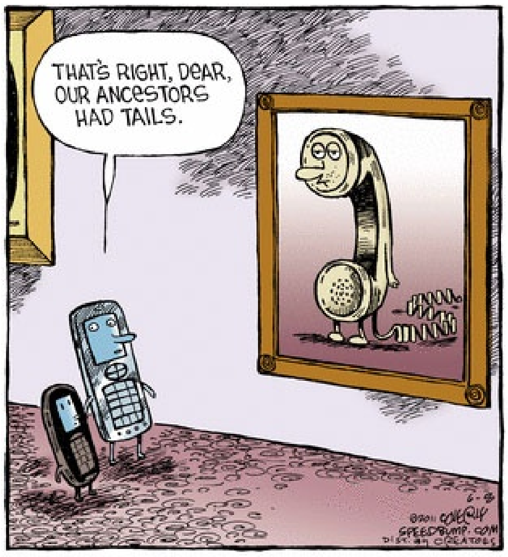

THAT'S RIGHT, DEAR, OUR ANCESTORS HAD TAILS.
©2011 COVERLY SPEEDBUMP.COM DIST. BY CREATORS

Figure 4.1: When new and old phones meet

On slides, the single-width figures are placed in 'columns'

The command used to display Figure 4.1 was

```
\MEfigure{fig/phone_anchestors} {When new and
old phones meet} {fig:phonenachestors} {2011
http://pinterest.com}{.8}
```

CHAPTER

5

Customizing document

The MultEdu system works perfectly with its default settings, but it cannot read your mind. The settings can be changed using definitions of form `\def{\xxx}`. The place where the settings can be changed, depends on the compilation mode, as described in chapter 6. The default values of the settings is given at the individual settings. The sections in this chapter provide a detailed description of the possible settings.

5.1 Default settings

The MultEdu system can interpret as an intention to change the default behavior either the presence of a file at a predefined file with a predefined name, or the thesence of definition of form `\def{Option{Value}}`. In the absence of such occurrences, MultEdu uses the default settings when generating the output file.

5.2 Options for using package `MultEdu`

5.2.1 Options for Beamer-based formats

MultEdu allows to utilize two popular screen widths. The default is the spreading format with aspect ratio 16:9. To set ratio 4:3, use
`{\def\DisableWideScreen{YES}}`

Sometimes (mainly in the case of short presentations) the table of contents is not necessary at all. It can be disabled through defining
`{\def\DisableTOC{YES}}`

It might also happen, that chapter-level TOC is still needed, but the section level not. This can be reached through defining
`{\def\DisableSectionTOC{YES}}`

5.3 Files for package `MultEdu`

The files affecting the appearance of your documents must fit the overall structure of files, as described in section 1.3.2. It is a good policy to change files only in your project subdirectory, since the commonly used files of the package are overwritten when using batch compile.

5.3.1 `src/Heading`

Some kind of heading usually belongs to the document. As an example see file `src/Heading.tex` of this user's guide.

Line `\def\LectureAuthor{J\'anos V\'egh}` defines the author, lines `\def\LectureTitle{How to use package MultEdu}` and `\def\LectureSubtitle{(How to prepare interesting and attractive teaching material)}` the main title and its subtitle. Also a university name or conference name can be defined in `\def\LecturePublisher{University or conference}` line. It is good practice to define `\def\LectureRevision{V\Version\ \at year.mm.dd}`, too.

When
using dual-language source files, one has to prepare the source in a form which allows to select source lines depending on the language. To prepare dual-language documents, the definitions should be put in frame like `\ifthenelse{\equal{\LectureLanguage}{english}}`

```
{% in English
}% true
{% NOT english
}
```

Also here you can give e-mail address

`\def\LectureEmail{Janos.Vegh\at unideb.hu}`

Furthermore, one can provide `BibTeX`, even conditionally, depending on the language or the presence of some files

```
\IfFileExists{src/Bibliographyhu}
{\def\LectureBibliography{src/Bibliography
,src/Bibliographyhu}}
{\def\LectureBibliography{src/Bibliography}}
```

Chapter 6

Compiling document

6.1 Manual mode compiling

File `Main.tex` is the common part of the dual compilation system. This contains the real source code. Any setting in this file (as well as in the included files) overwrites the settings, in both the manual and the batch mode, so it is better not to use any settings here. The best policy is to collect all the settings in a separate file, which is then included in the envelope file.

Developing course materials is best to do using an editor, integrated into an IDE. You need to read the envelope file (corresponding to `Demo.tex`) into the editor and mark it as your main document. In the file `Main.tex` you should insert references to the chapters of your course material. Those chapter files should be placed in subdirectory `src`, following the structure of the demonstrational material.

The settings file should be placed in subdirectory `src`, its reasonable name can be `Defines.tex`. The task of the wrapper file `Demo.tex` is only to input the setting file and the main file.

The batch compilation generates a file `Defines.tex`, which goes into subdirectory `build/build/src`. (You may use it to 'cheat', what settings and how should be utilized.) The batch compilation also generates a template file `Defines.tex.in` in subdirectory `src`. The content of this file corresponds to the last pass of the batch compilation.

6.2 Batch mode compiling

Batch processing serves (mainly) the goal to generate the output from the common source in the different formats and languages.

From technical reasons, MultEdu prepares a private copy from the MultEdu files, in the subdirectory `common` of the project. You may safely experiment with this copy or also delete it; the next batch compile will recreate it. (I.e. one should save the valuable developments; possibly in subdirectory `../../common` if you want to use it also by the other project groups.)

The compilation comproses three stages

- in the project directory in file `CMakeLists.txt` edit settings for the actual compilation
- change to subdirectory `build/build` and give command `cmake ../..`. In response MultEdu prepares the respective configuration files and source files (and also displays on the screen which files will be prepared)
- in the same directory give command `make`, which actually starts compiling

The batch compiling by default is longer than the manual one. It considers the worst case: recompiles everything, even when it would not strictly needed. Thsi time should be multiplied by the numebr of languages and number of formats, i.e. for longer documents it might require several minutes.

However, since that compilation works independently, and MultEdu prepares its own copy used in the compilation, you may continue editing and/or compiling your files, even the ones used in the batch compilation. The handling of the source and result files is independent as well. MultEdu prepares a separated source file for its own use and compiles it to its own output file. The file names of the own copies comprise codes of the language, format and version.

6.3 Changing default settings

Settings of MultEdu can be defined using `\def{OptionName}` macros. If the compiler does not find the corresponding macro, the default setting will be used. The settings differ in the cases of manual and batch compiling. During batch processing the compiler uses settings from file `build/build/src/Defines.tex`, which is newly created based on the settings in `CMakeFiles.txt`. During manual compilation, the settings from fájl `src/Defines.tex` are used. These two setting files should have the same (or at least similar) content, but the latter one is only handled by the user.

6.3.1 Versioning

Multedu uses three-level version numbering (major, minor and patch). The course materials prepared with MultEdu have two kinds of version numbers: the user maintains his/her own version numbers, and the developer maintains version of MultEdu.

Version number of MultEdu is located in file `../../common/MEMacros.tex`; better not to change it. The own course material version number is held in file `CMakeFiles.txt`, and that setting will be refreshed in the generated source files (through file `Defines.tex`) when batch compiling. The version number of the course material appears also in the name of the generated file, so it is worth to use it in a consequent way.

Usage:

```
\def\Version{major.minor.patch}
```

6.3.2 Languages

MultEdu can handle single- and dual-language documents. Different spelling, section name, captions belong to the different languages. In the settings file the language must be specified, like using setting `\LectureLanguage{english}` (this is the default). The name of the selected language appears also in the name of the result file.

In the dual-language documents, a first and second language co-exist, meaning in which order the texts in the different languages appear in the document. This allows to develop course material in both languages simultanously, one below the other. Selecting the proper language one can generate output in either language. If `\UseSecondLanguage{}` is defined, then the text appearing in the second position will be processed, using the language features defined by `\LectureLanguage{}`.

When using batch compilation, the options `FirstLanguage` and `SecondLanguage` must be provided (that defines the language found in the dual-language macros in the first and second position, respectively). If option `NEED_BOTH_LANGUAGES` is on, the output file will be produced in both languages. If it is switched off, option `USE_SECOND_LANGUAGE` decides which language to use.

Chapter

7 Utilizing supplements

7.1 Acronyms and glossary

Especially in the case of technical courses, frequently occur abbreviations, mosaic words, unique interpretations of a term, etc. MultEdu can help you with using the `glossaries` package, to provide your students with a hyperlinked facility, to use those terms consequently.

Such elements should be used in the text like `\gls{ref}`. Here `ref` is a reference label, and in the text the short name of the referenced item appears. In the case of acronyms, the expansion also appears at the first occurrence of that acronym. Some examples are given below; for more explanation see package `glossaries`.

Especially in the case of technical courses, frequently occur abbreviations, mosaic words, unique interpretations of a term, etc. MultEdu can help you with using the `glossaries` package, to provide your students with a hyperlinked facility, to use those terms consequently.

References to such elements should be used in the text as `\gls{ref}`. Here `ref` is a reference label, and in the text the short name of the referenced item appears at that place. In the case of acronyms, the expansion also appears at the first occurrence of that acronym. Some examples are given below; for more explanation see package `glossaries`.

7.1.1 How to use acronyms and glossary

When as a sample you use the term computer, where Central Processing Unit (CPU) és Direct Memory Access (DMA) also happens; in the text `When as a \gls{sampleone} you use the term, \gls{computer} where \gls{CPU} and \gls{DMA} also happens` should appear. MultEdu then appends chapters `Acronyms` and `Glossary` to the end of the document, and clicking on those hyperlinked words, you are taken to the explanation of the terms. When there, you migh click on the page number after the term, to go back.

MultEdu expects that (if you want to use this facility) your project contains a file `src/Glossary.tex`, where the expansion of the referred to items can be found. The entries corresponding to the items used in the sample can be coded like

7.1.2 How to define acronyms and glossary

```
\ifthenelse{\equal{\LectureLanguage}{english}}
{
\newglossaryentry{computer}
{
name={computer},
description={is a programmable machine that
receives input,
stores and manipulates data, and provides
output in a useful format}
}
\newglossaryentry{sampleone} {name={sample},
description={a little example}}
\newacronym{CPU}{CPU}{Central Processing Unit}
\newacronym{DMA}{DMA}{Direct Memory Access}
}
{}
```

7.1.3 How to utilize acronyms and glossary

These facilities can of course be only reasonably used in printable formats. Formats based on `beamer` do not generate such a list of terms, but the `\gls{ref}` are of course usable.

An excellent facility for having acronym extension, term explanation, etc. always at hand, but without breaking the continouous text.

7.2 Indices

7.3 Using bibliography

Index

Acronyms

C | **D**

C

CPU
: Central Processing Unit. 13

D

DMA
: Direct Memory Access. 13

Glossary

C | M | S

C

computer
: is a programmable machine that receives input, stores and manipulates data, and provides output in a useful format. 13

M

MultEdu
: MultEdu is a LaTeX-based macro package, which produces .pdf output files in different formats and languages, from the same source file. Primarily for educational purposes.. 1–3, 5, 10–13

S

sample
: a little example. 13

List of Figures

Listings

Back cover