

# The L<sup>A</sup>T<sub>E</sub>X.mk Makefile and related script tools\*

Vincent DANJEAN

Arnaud LEGRAND

2026/06/13

## Abstract

This package allows to compile all kind and complex L<sup>A</sup>T<sub>E</sub>X documents with the help of a Makefile. Dependencies are automatically tracked with the help of the `texdepends.sty` package.

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Quick start</b>	<b>2</b>
2.1	First (and often last) step . . . . .	2
2.2	Customization . . . . .	3
	Which L <sup>A</sup> T <sub>E</sub> X documents to compile . . . . .	3
	Which L <sup>A</sup> T <sub>E</sub> X main source for a document . . . . .	4
	Which flavors must be compiled . . . . .	4
	Which programs are called and with which options . . . . .	4
	Per target programs and options . . . . .	4
	Global and per target dependencies . . . . .	4
<b>3</b>	<b>Reference manual</b>	<b>5</b>
3.1	Flavors . . . . .	5
	3.1.1 What is a flavor ? . . . . .	5
	3.1.2 Defining a new flavor . . . . .	5
3.2	Variables . . . . .	6
	3.2.1 Two kind of variables . . . . .	7
	3.2.2 List of used variables . . . . .	8
<b>4</b>	<b>FAQ</b>	<b>10</b>
4.1	No rule to make target ‘LU_WATCH_FILES_SAVE’ . . . . .	10
<b>5</b>	<b>Implementation</b>	<b>11</b>
5.1	LaTeX.mk . . . . .	11
5.2	figdepth . . . . .	28
5.3	gensubfig . . . . .	29
5.4	svg2dev . . . . .	30
5.5	latexfilter . . . . .	31
5.6	svgdepth . . . . .	33

---

\*This file has version number v2.4.4, last revised 2026/06/13.

# 1 Introduction

`latex-make` is a collection of  $\text{\LaTeX}$  packages, scripts and Makefile fragments that allows to easily compile  $\text{\LaTeX}$  documents. The best feature is that ***dependencies are automatically tracked***<sup>1</sup>.

These tools can be used to compile small  $\text{\LaTeX}$  documents as well as big ones (such as, for example, a thesis with summary, tables of contents, list of figures, list of tabulars, multiple indexes and multiple bibliographies).

## 2 Quick start

### 2.1 First (and often last) step

When you want to use `latex-make`, most of the time you have to create a **Makefile** with the only line:

```
include LaTeX.mk
```

Then, the following targets are available: `dvi`, `ps`, `pdf`, `file.dvi`, `file.ps`, `file.pdf`, etc., `clean` and `distclean`.

All  $\text{\LaTeX}$  documents of the current directory should be compilable with respect to their dependencies. If something fails, please, provide me the smallest example you can create to show me what is wrong.

**Tip:** If you change the dependencies inside your document (for example, if you change `\include{first}` into `\include{second}`), you may have to type `make distclean` before being able to recompile your document. Else, `make` can fail, trying to build or found the old `first.tex` file.

**Shared work** If you work with other people that do not have installed (and do not want to install)  $\text{\LaTeX}$ -Make, you can use the `LaTeX-Make-local-install` target in `LaTeX.mk` to install required files in a local TEXMF tree. You can then commit this tree into your control version system. Then, in your **Makefile**, replace the single line

```
include LaTeX.mk
```

with something like

```
export TEXMFHOME=$(CURDIR)/relpath/to/local/tree/texmf
include $(shell env TEXMFHOME=$(TEXMFHOME) \
    kpsewhich -format texmfscripts LaTeX.mk)
```

If you have a previous value for `TEXMFHOME` that you do not want to override, you can use the following (more complexe) snippet

---

<sup>1</sup>Dependencies are tracked with the help of the `texdepend.sty` package that is automatically loaded: no need to specify it with `\usepackage{}` in your documents.

```

# Adapt the following line to find the local texmf tree
LOCAL_TEXMF:=$(CURDIR)/$(firstword $(wildcard texmf \
    ../texmf ../../texmf ../../../texmf))
# Get the old TEXMFHOME value
TEXMFHOME:=$(shell kpsewhich -var-value TEXMFHOME)
# If the new tree is already in it, do nothing, else add it
ifeq ($(filter $(LOCAL_TEXMF)//,$(TEXMFHOME)),)
TEXMFHOME := $(LOCAL_TEXMF)$(addprefix :,$(TEXMFHOME))
# display info so that users know what to define in order to
# compile documents directly with (pdf)latex
$(warning export TEXMFHOME=$(TEXMFHOME))
export TEXMFHOME
endif

include $(shell env TEXMFHOME=$(TEXMFHOME) \
    kpsewhich -format texmfscripts LaTeX.mk)

```

Doing so, all co-authors will be able to use L<sup>A</sup>T<sub>E</sub>X-Make without installing it. However, note that:

- you won't benefit of an update of L<sup>A</sup>T<sub>E</sub>X-Make in your system (you will continue to use the locally installed files) ;
- there is no support for upgrading locally installed files (but reexecuting the installation should do a correct upgrade most of the time) ;
- if a user tries to compile the L<sup>A</sup>T<sub>E</sub>X source code directly with [pdf]latex, he must before either have LaTeX-Make installed or define and export TEXMFHOME.

Another possibility is to install package files (\*.sty) into a directory pointed by TEXINPUTS, scripts files (\*.py) into a directory pointed by TEXMFSCRIPTS, and to directly include LaTeX.mk. For example:

```

# use local files by default
# packages in sty/ subdir and scripts in bin/
TEXINPUTS:=sty$(addprefix :,$(TEXINPUTS))::
TEXMFSCRIPTS:=bin$(addprefix :,$(TEXMFSCRIPTS))::
export TEXINPUTS
export TEXMFSCRIPTS
# Force using local LaTeX.mk and not system-wide LaTeX.mk if available
include $(CURDIR)/LaTeX.mk

```

## 2.2 Customization

Of course, lots of things can be customized. Here are the most useful ones. Look at the section 3 for more detailed and complete possibilities.

Customization is done through variables in the Makefile set *before* including LaTeX.mk. Setting them after can sometimes work, but not always and it is not supported.

### Which L<sup>A</sup>T<sub>E</sub>X documents to compile

LU\_MASTERS

**Example:** LU\_MASTERS=figlatex texdepends latex-make

This variable contains the basename of the L<sup>A</sup>T<sub>E</sub>X documents to compile.

If not set, LaTeX.mk looks for all \*.tex files containing the \documentclass command.

### Which L<sup>A</sup>T<sub>E</sub>X main source for a document

*master\_MAIN*

**Example:** `figlatex_MAIN=figlatex.dtx`

There is one such variable per documents declared in `LU_MASTERS`. It contains the file against which the `latex` (or `pdflatex`, etc.) program must be run.

If not set, `master.tex` is used.

### Which flavors must be compiled

*LU\_FLAVORS*

**Example:** `LU_FLAVORS=DVI DVIPDF`

A flavor can be seen as a kind of document (postscript, PDF, DVI, etc.) and the way to create it. For example, a PDF document can be created directly from the `.tex` file (with `pdflatex`), from a `.dvi` file (with `dvipdfm`) or from a postscript file (with `ps2pdf`). This would be three different flavors.

Some flavors are already defined in `LaTeX.mk`. Other flavors can be defined by the user (see section 3.1.2). The list of predefined flavors can be seen in the table 1. A flavor can depend on another. For example, the flavor creating a postscript file from a DVI file depends on the flavor creating a DVI file from a L<sup>A</sup>T<sub>E</sub>X file. This is automatically handled.

If not set, PS and PDF are used (and DVI due to PS).

Flavor	dependency	program variable	Transformation
DVI		LATEX	<code>.tex</code> $\Rightarrow$ <code>.dvi</code>
PS	DVI	DVIPS	<code>.dvi</code> $\Rightarrow$ <code>.ps</code>
PDF		PDFLATEX	<code>.tex</code> $\Rightarrow$ <code>.pdf</code>
LUALATEX		LUALATEX	<code>.tex</code> $\Rightarrow$ <code>.pdf</code>
DVIPDF	DVI	DVIPDFM	<code>.dvi</code> $\Rightarrow$ <code>.pdf</code>

*For example, the DVI flavor transforms a \*.tex file into a \*.dvi file with the Makefile command `$(LATEX) $(LATEX_OPTIONS)`*

Table 1: Predefined flavors

### Which programs are called and with which options

*prog/prog\_OPTIONS*

**Example:** `DVIPS=dvips`  
`DVIPS_OPTIONS=-t a4`

Each flavor has a program variable name that is used by `LaTeX.mk` to run the program. Another variable with the suffix `\_OPTIONS` is also provided if needed. See the table 1 the look for the program variable name associated to the predefined flavors.

Other programs are also run in the same manner. For example, the `makeindex` program is run from `LaTeX.mk` with the help of the variables `MAKEINDEX` and `MAKEINDEX_OPTIONS`.

### Per target programs and options

*master\_prog/master\_prog\_OPTIONS*

**Example:** `figlatex_DVIPS=dvips`  
`figlatex_DVIPS_OPTIONS=-t a4`

Note that, if defined, *master\_prog* will **replace** *prog* whereas *master\_prog\_OPTIONS* will **be added to** *prog\_OPTIONS* (see section 3.2 for more details).

### Global and per target dependencies

*DEPENDS/master\_DEPENDS*

**Example:** `DEPENDS=texdepends.sty`  
`figlatex_DEPENDS=figlatex.tex`

All flavor targets will depend to theses files. This should not be used as dependencies are automatically tracked.

## 3 Reference manual

### 3.1 Flavors

#### 3.1.1 What is a flavor ?

A flavor can be seen as a kind of document (postscript, PDF, DVI, etc.) and the way to create it. Several properties are attached to each flavor. Currently, there exist two kinds of flavors:

**TEX-flavors:** these flavors are used to compile a `*.tex` file into a target. A  $\text{\LaTeX}$  compiler (`latex`, `pdflatex`, etc.) is used;

**DVI-flavors:** these flavors are used to compile a file produced by a TEX-flavor into another file. Examples of such flavors are all the ones converting a DVI file into another format (postscript, PDF, etc.).

Several properties are attached to each flavor. Most are common, a few are specific to the kind of the flavor.

**Name:** the name of the flavor. It is used to declare dependencies between flavors (see below). It is also used to tell which flavor should be compiled for each document (see the `FLAVORS` variables);

**Program variable name:** name of the variable that will be used to run the program of this flavor. This name is used for the program and also for the options (variable with the `_OPTIONS` suffix);

**Target extension:** extension of the target of the flavor. The dot must be added if wanted;

**Master target:** if not empty, all documents registered for the flavor will be built when this master target is called;

**XFig extensions to clean (*TEX-flavor only*):** files extensions of figures that will be cleaned for the `clean` target. Generally, there is `.pstex_t` `.pstex` when using `latex` and `.pdftex_t` `.pdftex` when using `pdflatex`;

**Dependency *DVI-flavor only*:** name of the TEX-flavor the one depends upon.

#### 3.1.2 Defining a new flavor

To define a new flavor named `NAME`, one just has to declare a `lu-define-flavor-NAME` that calls and evaluates the `lu-create-flavor` with the right parameters, ie:

- name of the flavor;
- kind of flavor (`tex` or `dvi`);
- program variable name;
- target extension;
- master target;
- XFig extensions to clean *or* TEX-flavor to depend upon.

For example, `LaTeX.mk` already defines:

### DVI flavor

```
define lu-define-flavor-DVI
  $$ (eval $$ (call lu-create-flavor,DVI,tex,LATEX,.dvi,dvi,\
    .pstex_t .pstex))
endef
```

**Tip:** the LATEX program variable name means that the program called will be the one in the LATEX variable and that options in the LATEX\_OPTIONS variable will be used.

### PDF flavor

```
define lu-define-flavor-PDF
  $$ (eval $$ (call lu-create-flavor,PDF,tex,PDFLATEX,.pdf,pdf,\
    .pdftex_t .$$(_LU_PDFTEX_EXT)))
endef
```

### LuaLaTeX flavor

```
define lu-define-flavor-LUALATEX
  $$ (eval $$ (call lu-create-flavor,LUALATEX,tex,LUALATEX,.pdf,pdf,\
    .pdftex_t .$$(_LU_PDFTEX_EXT)))
endef
```

### PS flavor

```
define lu-define-flavor-PS
  $$ (eval $$ (call lu-create-flavor,PS,dvi,DVIPS,.ps,ps,DVI))
endef
```

**Tip:** for DVI-flavors, the program will be invoked with with the option `-o target` and with the name of the file source in argument.

### DVIPDF flavor

```
define lu-define-flavor-DVIPDF
  $$ (eval $$ (call lu-create-flavor,DVIPDF,dvi,DVIPDFM,.pdf,pdf,DVI))
endef
```

## 3.2 Variables

LaTeX.mk use a generic mechanism to manage variables, so that lots of thing can easily be customized per document and/or per flavor.

### 3.2.1 Two kind of variables

**LaTeX.mk** distinguish two kind of variables. The first one (called SET-variable) is for variables where only *one* value can be set. For example, this is the case for a variable that contain the name of a program to launch. The second one (called ADD-variable) is for variables where values can be cumulative. For example, this will be the case for the options of a program.

For each variable used by **LaTeX.mk**, there exists several variables that can be set in the Makefile so that the value will be used for all documents, only for one document, only for one flavor, etc.

**SET-variable.** For each SET-variable *NAME*, we can find in the Makfile:

1	<b>LU_target_</b> <i>NAME</i>	per document and per flavor value;
2	<b>TD_target_</b> <i>NAME</i>	per document and per flavor value filled by the <b>texdepends</b> L <sup>A</sup> T <sub>E</sub> X package;
3	<b>LU_master_</b> <i>NAME</i>	per document value;
4	<i>master_</i> <i>NAME</i>	per document value;
5	<b>LU_FLAVOR_</b> <i>flavor_</i> <i>NAME</i>	per flavor value;
6	<b>LU_</b> <i>NAME</i>	global value;
7	<i>NAME</i>	global value;
8	<b>_LU_...</b> <i>NAME</i>	internal <b>LaTeX.mk</b> default values.

The first set variable will be used.

**Tip:** in case of flavor context or document context, only relevant variables will be checked. For example, the SET-variable **MAIN** that give the main source of the document will be evaluated in document context, so only 4, 5, 6, 7 and 8 will be used (and I cannot see any real interest in using 6 or 7 for this variable).

**Tip2:** in case of context of index (when building indexes or glossary), there exists several other variables per index to add to this list (mainly ending with *\_kind\_indexname\_**NAME* or *\_kind\_**NAME*). Refer to the sources if you really need them.

**ADD-variable.** An ADD-variable is cumulative. The user can replace or add any values per document, per flavor, etc.

1	<b>LU_target_</b> <i>NAME</i>	replacing per document and per flavor values;
2	<i>target_</i> <i>NAME</i>	cumulative per document and per flavor values;
3	<b>LU_master_</b> <i>NAME</i>	replacing per document values;
4	<i>master_</i> <i>NAME</i>	cumulative per document values;
5	<b>LU_FLAVOR_</b> <i>flavor_</i> <i>NAME</i>	replacing per flavor values;
6	<b>FLAVOR_</b> <i>flavor_</i> <i>NAME</i>	cumulative per flavor values;
7	<b>LU_</b> <i>NAME</i>	replacing global values;
8	<i>NAME</i>	cumulative global values;

**Tip:** if not defined, *LU\_variable* defaults to “\$(*variable*) \$(\_LU\_variable)” and *\_LU\_variable* contains default values managed by **LaTeX.mk** and the **texdepends** L<sup>A</sup>T<sub>E</sub>X package.

**Example:** the ADD-variable **FLAVORS** is invoked in document context to know which flavors needs to be build for each document. This means that *LU\_master\_FLAVORS* will be used.

```

# We override default value for MASTERS
LU_MASTERS=foo bar baz
# By default, only the DVIPDF flavor will be build
FLAVORS=DVIPDF

bar_FLAVORS=PS
LU_baz_FLAVORS=PDF
# there will be rules to build
# * foo.dvi and foo.pdf
#   (the DVIPDF flavor depends on the DVI flavor)
# * bar.dvi, bar.pdf and bar.ps
#   (the PS flavor is added to global flavors)
# * baz.pdf
#   (the PDF flavor will be the only one for baz) include
LaTeX.mk

```

### 3.2.2 List of used variables

Here are most of the variables used by `LaTeX.mk`. Users should only have to sometimes managed the first ones. The latter are described here for information only (and are subject to modifications). Please, report a bug if some of them are not correctly pickup by the `texdepends` L<sup>A</sup>T<sub>E</sub>X package and `LaTeX.mk`.



Name	Kind	Context of use	Description
<b>MASTERS</b>	ADD	Global	List of documents to compile. These values will be used as jobname. <b>Default:</b> basename of *.tex files containing the \documentclass pattern
<b>FLAVORS</b>	ADD	Document	List of flavors for each document. <b>Default:</b> PS PDF
<b>MAIN</b>	SET	Document	Master tex source file <b>Default:</b> master.tex
<b>DEPENDS</b>	ADD	Target	List of dependencies
<b>DEPENDS_EXCLUDE</b>	ADD	Target	Dependencies to forget. Useful when LaTeX Make wrongly auto-detect false dependencies
<i>progvarname</i>	SET	Target	Program to launch for the corresponding flavor
<i>progvarname_OPTIONS</i>	ADD	Target	Options to use when building the target
<b>STYLE</b>	SET	Index	Name of the index/glossary style file to use (.ist, etc.)
<b>TARGET</b>	SET	Index	Name of the index/glossary file to produce (.ind, .gls, etc.)
<b>SRC</b>	SET	Index	Name of the index/glossary file source (.idx, .glo, etc.)
<b>FIGURES</b>	ADD	Target	Lists of figures included
<b>BIBFILES</b>	ADD	Target	Lists of bibliography files used (.bib)
<b>BIBSTYLES</b>	ADD	Target	Lists of bibliography style files used (.bst)
<b>BBLFILES</b>	ADD	Target	Lists of built bibliography files (.bbl)
<b>INPUT</b>	ADD	Target	Lists of input files (.cls, .sty, .tex, etc.)
<b>OUTPUTS</b>	ADD	Target	Lists of output files (.aux, etc.)
<b>GRAPHICSPATH</b>	ADD	Target	\graphicspath{} arguments
<b>GPATH</b>	ADD	Target	List of directories from GRAPHICSPATH without { and }, separated by spaces
<b>INDEXES</b>	ADD	Target	Kinds of index (INDEX, GLOSS, etc.)
<b>INDEXES_kind</b>	ADD	Target	List of indexes or glossaries
<b>WATCHFILES</b>	ADD	Target	List of files that trigger a rebuild if modified (.aux, etc.)
<b>REQUIRED</b>	ADD	Target	List of new dependencies found by the texdepends L <sup>A</sup> T <sub>E</sub> X package
<b>MAX_REC</b>	SET	Target	Maximum level of recursion authorized
<b>REBUILD_RULES</b>	ADD	Target	List of rebuild rules to use (can be modified by the texdepends L <sup>A</sup> T <sub>E</sub> X package)
<b>EXT</b>	SET	Flavor	Target file extension of the flavor
<b>DEPFLAVOR</b>	SET	Flavor	TEX-flavor a DVI-flavor depend upon
<b>CLEANFIGEXT</b>	ADD	Flavor	Extensions of figure files to remove on clean

## 4 FAQ

### 4.1 No rule to make target ‘LU\_WATCH\_FILES\_SAVE’

⇒ *When using LaTeX.mk, I got the error:*

```
make[1]: *** No rule to make target ‘LU_WATCH_FILES_SAVE’. Stop.
```

`make` is called in such a way that does not allow correct recursive calls. As one can not know by advance how many times `LATEX`, `bibTEX`, etc. will need to be run, `latex-make` use recursive invocations of `make`. This means that in the `LaTeX.mk` makefile, there exist rules such as:

```
$(MAKE) INTERNAL_VARIABLE=value internal_target
```

In order `latex-make` to work, this invocation of `make` must read the same rules and variable definitions as the main one. This means that calling "`make -f LaTeX.mk foo.pdf`" in a directory with only `foo.tex` will not work. Recursive invocations of `make` will not load `LaTeX.mk`, will search for a `Makefile` in the current directory and will complain about being unable to build the `LU_WATCH_FILES_SAVE` internal target.

The solution is to call `make` so that recursive invocations will read the same variables and rules. For example:

```
make -f LaTeX.mk MAKE="make -f LaTeX.mk" foo.pdf
```

or (if there is no `Makefile` in the directory):

```
env MAKEFILES=LaTeX.mk make foo.pdf
```

## 5 Implementation

### 5.1 LaTeX.mk

```
1 <*makefile>
2
3 #####[ Check Software ]#####
4
5 ifeq ($(filter else-if,$(.FEATURES)),)
6 $(error GNU Make 3.81 needed. Please, update your software.)
7 exit 1
8 endif
9
10 # Some people want to call our Makefile snippet with
11 # make -f LaTeX.mk
12 # This should not work as $(MAKE) is call recursively and will not read
13 # LaTeX.mk again. We cannot just add LaTeX.mk to MAKEFILES as LaTeX.mk
14 # should be read AFTER a standard Makefile (if any) that can define some
15 # variables (LU_MASTERS, ...) that LaTeX.mk must see.
16 # So I introduce an HACK here that try to workaround the situation. Keep in
17 # mind that this hack is not perfect and does not handle all cases
18 # (for example, "make -f my_latex_config.mk -f LaTeX.mk" will not recurse
19 # correctly)
20 ifeq ($(foreach m,$(MAKEFILES), $(m)) $(lastword $(MAKEFILE_LIST)),$(MAKEFILE_LIST))
21 # We are the first file read after the ones from MAKEFILES
22 # So we assume we are read due to "-f LaTeX.mk"
23 LU_LaTeX.mk_NAME := $(lastword $(MAKEFILE_LIST))
24 # Is this Makefile correctly read for recursive calls ?
25 ifeq ($(findstring -f $(LU_LaTeX.mk_NAME),$(MAKE)),)
26 $(info #####)
27 $(info Warning: $(LU_LaTeX.mk_NAME) called directly. I suppose that you run:)
28 $(info Warning: $(MAKE) -f $(LU_LaTeX.mk_NAME) $(MAKECMDGOALS))
29 $(info Warning: or something similar that does not allow recursive invocation of make)
30 $(info Warning: )
31 $(info Warning: Trying to enable a workaround. This ACK will be disabled in a future)
32 $(info Warning: release. Consider using another syntax, for example:)
33 $(info Warning: $(MAKE) -f $(LU_LaTeX.mk_NAME) MAKE="$(MAKE) -f $(LU_LaTeX.mk_NAME)" $(MAKECMDGOALS))
34 $(info #####)
35 MAKE+= -f $(LU_LaTeX.mk_NAME)
36 endif
37 endif
38
39 #####[ Configuration ]#####
40
41 # list of messages categories to display
42 LU_SHOW ?= warning #info debug debug-vars
43
44 # Select GNU/BSD/MACOSX utils (cp, rm, mv, ...)
45 LU_UTILS ?= $(shell ( /bin/cp --heelp > /dev/null 2>&1 && echo GNU ) || echo BSD )
46 export LU_UTILS
47
48 #####[ End of configuration ]#####
49 # Modifying the remaining of this document may endanger you life!!! ;)
50
51 #-----
52 # Controlling verbosity
53 ifdef VERB
54 MAK_VERB := $(VERB)
55 else
```

```

56 #MAK_VERB := debug
57 #MAK_VERB := verbose
58 #MAK_VERB := normal
59 MAK_VERB := quiet
60 #MAK_VERB := silent
61 endif
62
63 #-----
64 # MAK_VERB -> verbosity
65 ifeq ($(MAK_VERB),debug)
66 COMMON_PREFIX = echo "          =====> building " "$@" "<=====" ; \
67 printf "%s $(@F) due to:$(foreach file,$?,\n          * $(file))\n" $1; set -x;
68 #
69 COMMON_HIDE := set -x;
70 COMMON_CLEAN := set -x;
71 SHOW_LATEX:=true
72 else
73 ifeq ($(MAK_VERB),verbose)
74 COMMON_PREFIX = echo "          =====> building " "$@" "<=====" ; \
75 printf "%s $(@F) due to:$(foreach file,$?,\n          * $(file))\n" $1;
76 #
77 COMMON_HIDE :=#
78 COMMON_CLEAN :=#
79 SHOW_LATEX:=true
80 else
81 ifeq ($(MAK_VERB),normal)
82 COMMON_PREFIX =#
83 COMMON_HIDE := @
84 COMMON_CLEAN :=#
85 SHOW_LATEX:=true
86 else
87 ifeq ($(MAK_VERB),quiet)
88 COMMON_PREFIX = @ echo "          =====> building " "$@" "<=====" ;
89 # echo "due to $?" ;
90 COMMON_HIDE := @
91 COMMON_CLEAN :=#
92 SHOW_LATEX:=
93 else # silent
94 COMMON_PREFIX = @
95 COMMON_HIDE := @
96 COMMON_CLEAN := @
97 SHOW_LATEX:=
98 endif
99 endif
100 endif
101 endif
102
103 #-----
104 # Old LaTeX have limitations
105 _LU_PDFTEX_EXT ?= pdftex
106
107 #####
108 # Utilities
109 LU_CP=$(LU_CP_$(LU_UTILS))
110 LU_MV=$(LU_MV_$(LU_UTILS))
111 LU_RM=$(LU_RM_$(LU_UTILS))
112 LU_CP_GNU ?= cp -a --
113 LU_MV_GNU ?= mv --

```

```

114 LU_RM_GNU ?= rm -f --
115 LU_CP_BSD ?= cp -p
116 LU_MV_BSD ?= mv
117 LU_RM_BSD ?= rm -f
118 LU_CP_MACOSX ?= /bin/cp -p
119 LU_MV_MACOSX ?= /bin/mv
120 LU_RM_MACOSX ?= /bin/rm -f
121
122 lu-show=\
123 $(if $(filter $(LU_SHOW),$(1)), \
124 $(if $(2), \
125 $(if $(filter-out $(2),$(MAKELEVEL)),,$(3)), \
126 $(3)))
127 lu-show-infos=\
128 $(if $(filter $(LU_SHOW),$(1)), \
129 $(if $(2), \
130 $(if $(filter-out $(2),$(MAKELEVEL)),,$(warning $(3))), \
131 $(warning $(3))))
132 lu-show-rules=$(call lu-show-infos,info,0,$(1))
133 lu-show-flavors=$(call lu-show-infos,info,0,$(1))
134 lu-show-var=$(call lu-show-infos,debug-vars,, * Set $(1)=$( $(1)))
135 lu-show-read-var=$(eval $(call lu-show-infos,debug-vars,, Reading $(1) in $(2) ctx: $(3)))$(3)
136 lu-show-readone-var=$(eval $(call lu-show-infos,debug-vars,, Reading $(1) for $(2) [one value]: $(3)))
137 lu-show-set-var=$(call lu-show-infos,debug-vars,, * Setting $(1) for $(2) to value: $(3))
138 lu-show-add-var=$(call lu-show-infos,debug-vars,, * Adding to $(1) for $(2) values: $(value 3))
139 lu-show-add-var2=$(call lu-show-infos,warning,, * Adding to $(1) for $(2) values: $(value 3))
140
141 lu-save-file=$(call lu-show,debug,,echo "saving $1" ;) \
142 if [ -f "$1" ];then $(LU_CP) "$1" "$2" ;else $(LU_RM) "$2" ;fi
143 lu-cmprestaure-file=\
144 if cmp -s "$1" "$2"; then \
145 $(LU_MV) "$2" "$1" ; \
146 $(call lu-show,debug,,echo "$1" not modified ;) \
147 else \
148 $(call lu-show,debug,,echo "$1" modified ;) \
149 if [ -f "$2" -o -f "$1" ]; then \
150 $(RM) -- "$2" ; \
151 $3 \
152 fi ; \
153 fi
154
155 lu-clean=$(if $(strip $(1)),$(RM) $(1))
156
157 define lu-bug # description
158 $$ (warning Internal error: $(1))
159 $$ (error You probably found a bug. Please, report it.)
160 endef
161
162 #####
163 #####
164 #####
165 #####
166 #####
167 ##### Variables #####
168 #####
169 #####
170 #####
171 #####

```

```

172 #####
173 #####
174 #
175 # _LU_FLAVORS_DEFINED : list of available flavors
176 # _LU_FLAV_*_'flavname' : per flavor variables
177 #   where * can be :
178 #   PROGNAME : variable name for programme (and .._OPTIONS for options)
179 #   EXT : extension of created file
180 #   TARGETNAME : global target
181 #   DEPFLAVOR : flavor to depend upon
182 #   CLEANFIGEXT : extensions to clean for fig figures
183 _LU_FLAVORS_DEFINED = $_LU_FLAVORS_DEFINED_TEX) $_LU_FLAVORS_DEFINED_DVI)
184
185 # INDEXES_TYPES = GLOSS INDEX
186 # INDEXES_INDEX = name1 ...
187 # INDEXES_GLOSS = name2 ...
188 # INDEX_name1_SRC
189 # GLOSS_name2_SRC
190
191 define _lu-getvalues# 1:VAR 2:CTX (no inheritance)
192 $(if $(filter-out undefined,$(origin LU_$2$1)),$(LU_$2$1),$(2$1) $_LU_$2$1_MK) $(TD_$2$1))
193 endif
194 define lu-define-addvar # 1:suffix_fname 2:CTX 3:disp-debug 4:nb_args 5:inherited_ctx 6:ctx-build-deper
195   define lu-addtovar$1 # 1:VAR 2:... $4: value
196     _LU_$2$1_MK+=$(4)
197     $(call lu-show-add-var,$$1,$3,$$(value $4))
198   endif
199   define lu-def-addvar-inherited-ctx$1 # 1:VAR 2:...
200     $6
201     _LU_$2$1_INHERITED_CTX=$(sort \
202       $(foreach ctx,$5,$$(ctx) $$$(if $(filter-out undefined,$$(origin \
203         LU_$(ctx)$1)),,\
204         $$(_LU_$(ctx)$1_INHERITED_CTX)))
205     $$$$(call lu-show-var,_LU_$2$1_INHERITED_CTX)
206   endif
207   define lu-getvalues$1# 1:VAR 2:...
208   $$$(if $(filter-out undefined,$$(origin _LU_$2$1_INHERITED_CTX)),,$$(eval \
209     $(call lu-def-addvar-inherited-ctx$1,$$1,$2,$3,$4,$5,$6)\
210   ))$(call lu-show-read-var,$$1,$3,$$(foreach ctx,\
211     $(if $2,$2,GLOBAL) $(if $(filter-out undefined,$$(origin LU_$2$1)),,\
212     $$(_LU_$2$1_INHERITED_CTX))\
213     ,$(call _lu-getvalues,$$1,$$(filter-out GLOBAL,$$(ctx))))
214   endif
215 endif
216
217 # Global variable
218 # VAR (DEPENDS)
219 $(eval $(call lu-define-addvar,-global,,global,2))
220
221 # Per flavor variable
222 # FLAVOR_$2_VAR (FLAVOR_DVI_DEPENDS)
223 # 2: flavor name
224 # Inherit from VAR (DEPENDS)
225 $(eval $(call lu-define-addvar,-flavor,FLAVOR_$2_,flavor $$2,3,\
226   GLOBAL,\
227   $(eval $(call lu-def-addvar-inherited-ctx-global,$$1)) \
228 ))
229

```

```

230 # Per master variable
231 # $2_VAR (source_DEPENDS)
232 # 2: master name
233 # Inherit from VAR (DEPENDS)
234 $(eval $(call lu-define-addvar,-master,$$2_,master $$2,3,\
235 GLOBAL,\
236 $$$(eval $$$(call lu-def-addvar-inherited-ctx-global,$$1)) \
237 ))
238
239 # Per target variable
240 # $$$(EXT of $3)_VAR (source.dvi_DEPENDS)
241 # 2: master name
242 # 3: flavor name
243 # Inherit from $2_VAR FLAVOR_$3_VAR (source_DEPENDS FLAVOR_DVI_DEPENDS)
244 $(eval $(call lu-define-addvar,,$$2$$$(call lu-getvalue-flavor,EXT,$$3)_ ,target $$2$$$(call lu-getvalue-f
245 $$2_ FLAVOR_$$3_,\
246 $$$(eval $$$(call lu-def-addvar-inherited-ctx-master,$$1,$$2)) \
247 $$$(eval $$$(call lu-def-addvar-inherited-ctx-flavor,$$1,$$3)) \
248 ))
249
250 # Per index/glossary variable
251 # $(2)_$(3)_VAR (INDEX_source_DEPENDS)
252 # 2: type (INDEX, GLOSS, ...)
253 # 3: index name
254 # Inherit from VAR (DEPENDS)
255 $(eval $(call lu-define-addvar,-global-index,$$2_$$3_,index $$3[$$2],4,\
256 GLOBAL,\
257 $$$(eval $$$(call lu-def-addvar-inherited-ctx-global,$$1)) \
258 ))
259
260 # Per master and per index/glossary variable
261 # $(2)_$(3)_$(4)_VAR (source_INDEX_source_DEPENDS)
262 # 2: master name
263 # 3: type (INDEX, GLOSS, ...)
264 # 4: index name
265 # Inherit from $2_VAR $3_$4_VAR (source_DEPENDS INDEX_source_DEPENDS)
266 $(eval $(call lu-define-addvar,-master-index,$$2_$$3_$$4_,index $$2/$$4[$$3],5,\
267 $$2_ $$3_$$4_,\
268 $$$(eval $$$(call lu-def-addvar-inherited-ctx-master,$$1,$$2)) \
269 $$$(eval $$$(call lu-def-addvar-inherited-ctx-global-index,$$1,$$3,$$4)) \
270 ))
271
272 # Per target and per index/glossary variable
273 # $(2)$(EXT of $3)_$(4)_$(5)_VAR (source.dvi_INDEX_source_DEPENDS)
274 # 2: master name
275 # 3: flavor name
276 # 4: type (INDEX, GLOSS, ...)
277 # 5: index name
278 # Inherit from $$$(EXT of $3)_VAR $(2)_$(3)_$(4)_VAR
279 # (source.dvi_DEPENDS source_INDEX_source_DEPENDS)
280 $(eval $(call lu-define-addvar,-index,$$2$$$(call lu-getvalue-flavor,EXT,$$3)_$$4_$$5_,index $$2$$$(call
281 $$2$$$(call lu-getvalue-flavor,EXT,$$3)_ $$2_$$4_$$5_,\
282 $$$(eval $$$(call lu-def-addvar-inherited-ctx,$$1,$$2,$$3)) \
283 $$$(eval $$$(call lu-def-addvar-inherited-ctx-master-index,$$1,$$2,$$4,$$5)) \
284 ))
285
286
287

```

```

288
289
290
291 define lu-setvar-global # 1:name 2:value
292   _LU_$ (1) ?= $(2)
293   $$ (eval $$ (call lu-show-set-var,$(1),global,$(2)))
294 endif
295
296 define lu-setvar-flavor # 1:name 2:flavor 3:value
297   _LU_FLAVOR_$ (2)_$(1) ?= $(3)
298   $$ (eval $$ (call lu-show-set-var,$(1),flavor $(2),$(3)))
299 endif
300
301 define lu-setvar-master # 1:name 2:master 3:value
302   _LU_$ (2)_$(1) ?= $(3)
303   $$ (eval $$ (call lu-show-set-var,$(1),master $(2),$(3)))
304 endif
305
306 define lu-setvar # 1:name 2:master 3:flavor 4:value
307   _LU_$ (2)$$ (call lu-getvalue-flavor,EXT,$(3))_$(1)=$(4)
308   $$ (eval $$ (call lu-show-set-var,$(1),master/flavor $(2)/$(3),$(4)))
309 endif
310
311 define lu-getvalue # 1:name 2:master 3:flavor
312 $(call lu-show-readone-var,$(1),master/flavor $(2)/$(3),$(or \
313 $(LU_$ (2))$(call lu-getvalue-flavor,EXT,$(3))_$(1)), \
314 $(TD_$ (2))$(call lu-getvalue-flavor,EXT,$(3))_$(1)), \
315 $(LU_$ (2)_$(1)), \
316 $($ (2)_$(1)), \
317 $(LU_FLAVOR_$ (3)_$(1)), \
318 $(LU_$ (1)), \
319 $($ (1)), \
320 $_LU_$ (2)$(call lu-getvalue-flavor,EXT,$(3))_$(1)), \
321 $_LU_$ (2)_$(1)), \
322 $_LU_FLAVOR_$ (3)_$(1)), \
323 $_LU_$ (1))\
324 ))
325 endif
326
327 define lu-getvalue-flavor # 1:name 2:flavor
328 $(call lu-show-readone-var,$(1),flavor $(2),$(or \
329 $(LU_FLAVOR_$ (2)_$(1)), \
330 $(LU_$ (1)), \
331 $($ (1)), \
332 $_LU_FLAVOR_$ (2)_$(1)), \
333 $_LU_$ (1))\
334 ))
335 endif
336
337 define lu-getvalue-master # 1:name 2:master
338 $(call lu-show-readone-var,$(1),master $(2),$(or \
339 $(LU_$ (2)_$(1)), \
340 $($ (2)_$(1)), \
341 $(LU_$ (1)), \
342 $($ (1)), \
343 $_LU_$ (2)_$(1)), \
344 $_LU_$ (1))\
345 ))

```



```

346 endif
347
348 define lu-getvalue-index # 1:name 2:master 3:flavor 4:type 5:indexname
349 $(call lu-show-readone-var,$(1),master/flavor/index $(2)/$(3)/[$(4)]$(5),$(or \
350 $(LU_$(2))$(call lu-getvalue-flavor,EXT,$(3))_$(4)_$(5)_$(1)), \
351 $(LU_$(2)_$(4)_$(5)_$(1)), \
352 $(TD_$(2))$(call lu-getvalue-flavor,EXT,$(3))_$(4)_$(5)_$(1)), \
353 $($ (2)_$(4)_$(5)_$(1)), \
354 $(LU_$(4)_$(5)_$(1)), \
355 $($ (4)_$(5)_$(1)), \
356 $(LU_$(2))$(call lu-getvalue-flavor,EXT,$(3))_$(4)_$(1)), \
357 $(LU_$(2)_$(4)_$(1)), \
358 $($ (2)_$(4)_$(1)), \
359 $(LU_$(4)_$(1)), \
360 $($ (4)_$(1)), \
361 $(LU_$(2)_$(1)), \
362 $($ (2)_$(1)), \
363 $(LU_FLAVOR_$(3)_$(1)), \
364 $(LU_$(1)), \
365 $($ (1)), \
366 $(_LU_$(2))$(call lu-getvalue-flavor,EXT,$(3))_$(4)_$(5)_$(1)), \
367 $(_LU_$(2)_$(4)_$(5)_$(1)), \
368 $(_LU_$(4)_$(5)_$(1)), \
369 $(_LU_$(2))$(call lu-getvalue-flavor,EXT,$(3))_$(4)_$(1)), \
370 $(_LU_$(2)_$(4)_$(1)), \
371 $(_LU_FLAVOR_$(3)_$(4)_$(1)), \
372 $(_LU_$(4)_$(1)), \
373 $(_LU_$(2))$(call lu-getvalue-flavor,EXT,$(3))_$(1)), \
374 $(_LU_$(2)_$(1)), \
375 $(_LU_FLAVOR_$(3)_$(1)), \
376 $(_LU_$(1))\
377 ))
378 endif
379
380 define lu-call-prog # 1:varname 2:master 3:flavor [4:index]
381 $(call lu-getvalue,$(1),$(2),$(3)) $(call lu-getvalues,$(1)_OPTIONS,$(2),$(3))
382 endif
383
384 define lu-call-prog-index # 1:varname 2:master 3:flavor 4:type 5:indexname
385 $(call lu-getvalue$(if $(4),-index),$(1),$(2),$(3),$(4),$(5)) \
386 $(call lu-getvalues$(if $(4),-index),$(1)_OPTIONS,$(2),$(3),$(4),$(5))
387 endif
388
389 define lu-call-prog-flavor # 1:master 2:flavor
390 $(call lu-call-prog,$(call lu-getvalue,VARPROG,$(1),$(2)),$(1),$(2))
391 endif
392
393 #####
394 #####
395 #####
396 #####
397 #####
398 ##### Global variables #####
399 #####
400 #####
401 #####
402 #####
403 #####

```

```

404 #####
405
406 # Globals variables
407 $(eval $(call lu-setvar-global,LATEX,latex))
408 $(eval $(call lu-setvar-global,PDFLATEX,pdflatex))
409 $(eval $(call lu-setvar-global,LUALATEX,lualatex))
410 $(eval $(call lu-setvar-global,DVIPS,dvips))
411 $(eval $(call lu-setvar-global,DVIPDFM,dvipdfm))
412 $(eval $(call lu-setvar-global,BIBTEX,bibtex))
413 ##$(eval $(call lu-setvar-global,MPOST,TEX="$(LATEX)" mpost))
414 $(eval $(call lu-setvar-global,FIG2DEV,fig2dev))
415 ##$(eval $(call lu-setvar-global,SVG2DEV,svg2dev))
416 $(eval $(call lu-setvar-global,EPSTOPDF,epstopdf))
417 $(eval $(call lu-setvar-global,MAKEINDEX,makeindex))
418 $(eval $(call lu-setvar-global,MAKEGLOSSARIES,makeglossaries))
419
420 # workaround the fact that $(shell ...) ignore locally exported variables
421 # get only the variables with plain names
422 _LU_MAKE_ENV := $(shell echo '$(VARIABLES)' | awk -v RS=' ' '/^[a-zA-Z0-9]+$$/' )
423 _LU_SHELL_EXPORT := $(foreach v,$(_LU_MAKE_ENV),$(v)='$(v)')
424 _lu_run_kpsewhich=$(shell $(LU_SHELL_EXPORT) kpsewhich -format $1 $2)
425
426 # Look first into the TDS (texmfscripts), then in PATH for our program
427 # At each location, we prefer with suffix than without
428 define _lu_which # VARNAME progname
429 ifeq ($(origin _LU_$(1)_DEFAULT), undefined)
430 _LU_$(1)_DEFAULT := $$($(firstword $$$(wildcard \
431 $$$(call _lu_run_kpsewhich,texmfscripts,$(2)) \
432 $$$(call _lu_run_kpsewhich,texmfscripts,$$(basename $(2))) \
433 $$$(foreach dir,$$(subst :, ,$(PATH)), \
434 $$$(dir)/$(2) $$$(dir)/$$$(basename $(2))) \
435 ) $(2))
436 export _LU_$(1)_DEFAULT
437 _LU_$(1)_DEFAULT_OLD := $$($(firstword $$$(wildcard \
438 $$$(addprefix bin/, $(2) $$$(basename $(2))) \
439 $$$(addprefix ./, $(2) $$$(basename $(2))))))
440 $$$(if $$$(filter-out $$(_LU_$(1)_DEFAULT), $$(_LU_$(1)_DEFAULT_OLD)), \
441 $$$(if $$(_lu_scripts_warnings),, \
442 $$$(eval _lu_scripts_warnings:=done) \
443 $$$(warning By default, this version of LaTeX-Make do not use \
444 scripts in $$$(dir $$(_LU_$(1)_DEFAULT_OLD)) anymore.) \
445 $$$(warning For example $$(_LU_$(1)_DEFAULT) is used instead of $$(_LU_$(1)_DEFAULT_OLD)) \
446 $$$(warning If you want to keep the old behavior, add into your \
447 Makefile something like:)\
448 $$$(warning export TEXMFSCRIPTS:=$(dir $$(_LU_$(1)_DEFAULT_OLD))$$$$$(addprefix :,$$$$$(TEXMFSCRIPTS))
449 #$$$$(warning _LU_$(1)_DEFAULT=$$_LU_$(1)_DEFAULT))
450 endif
451 $$$(eval $(call lu-setvar-global,$(1),$$(_LU_$(1)_DEFAULT)))
452 endef
453
454 $(eval $(call _lu_which,GENSUBFIG,gensubfig.py))
455 $(eval $(call _lu_which,FIGDEPTH,figdepth.py))
456 $(eval $(call _lu_which,GENSUBSVG,gensubfig.py))
457 $(eval $(call _lu_which,SVGDEPTH,svgdepth.py))
458 $(eval $(call _lu_which,SVG2DEV,svg2dev.py))
459 $(eval $(call _lu_which,LATEXFILTER,latexfilter.py))
460
461 # Rules to use to check if the build document (dvi or pdf) is up-to-date

```

```

462 # This can be overruled per document manually and/or automatically
463 #REBUILD_RULES ?= latex texdepends bibtopic bibtopic_undefined_references
464 $(eval $(call lu-addtovar-global,REBUILD_RULES,latex texdepends))
465
466 # Default maximum recursion level
467 $(eval $(call lu-setvar-global,MAX_REC,6))
468
469 #####
470 #####
471 #####
472 #####
473 #####
474 #####          Flavors          #####
475 #####          #####
476 #####
477 #####
478 #####
479 #####
480 #####
481
482 define lu-create-texflavor # 1:name 2:tex_prog 3:file_ext
483     # 4:master_cible 5:fig_extention_to_clean
484     _LU_FLAVORS_DEFINED_TEX += $(1)
485     $(eval $(call lu-setvar-flavor,VARPROG,$(1),$(2)))
486     $(eval $(call lu-setvar-flavor,EXT,$(1),$(3)))
487     $(eval $(call lu-setvar-flavor,TARGETNAME,$(1),$(4)))
488     $(eval $(call lu-addtovar-flavor,CLEANFIGEXT,$(1),$(5)))
489     $(eval $(call lu-addtovar-flavor,CLEANSVGEXT,$(1),$(5)))
490 endef
491
492 define lu-create-dviflavor # 1:name 2:dvi_prog 3:file_ext
493     # 4:master_cible 5:tex_flavor_depend
494     $$$(eval $$$(call lu-define-flavor,$(5)))
495     _LU_FLAVORS_DEFINED_DVI += $(1)
496     $(eval $(call lu-setvar-flavor,VARPROG,$(1),$(2)))
497     $(eval $(call lu-setvar-flavor,EXT,$(1),$(3)))
498     $(eval $(call lu-setvar-flavor,TARGETNAME,$(1),$(4)))
499     $(eval $(call lu-setvar-flavor,DEPFLAVOR,$(1),$(5)))
500 endef
501
502 define lu-create-flavor # 1:name 2:type 3..7:options
503     $$$(if $$$(filter $(1),$_LU_FLAVORS_DEFINED)), \
504     $$$(call lu-show-flavors,Flavor $(1) already defined), \
505     $$$(call lu-show-flavors,Creating flavor $(1) ($(2))) \
506     $$$(eval $(call lu-create-$(2)flavor,$(1),$(3),$(4),$(5),$(6),$(7)))
507 endef
508
509 define lu-define-flavor # 1:name
510     $$$(eval $(call lu-define-flavor-$(1)))
511 endef
512
513 define lu-flavor-rules # 1:name
514     $$$(call lu-show-flavors,Defining rules for flavor $(1))
515     $$$(if $$$(call lu-getvalue-flavor,TARGETNAME,$(1)), \
516     $$$(call lu-getvalue-flavor,TARGETNAME,$(1)): \
517     $$$(call lu-getvalues-flavor,TARGETS,$(1)))
518     $$$(if $$$(call lu-getvalue-flavor,TARGETNAME,$(1)), \
519     .PHONY: $$$(call lu-getvalue-flavor,TARGETNAME,$(1)))

```

```

520 endif
521
522 define lu-define-flavor-DVI #
523   $$ (eval $$ (call lu-create-flavor,DVI,tex,LATEX,.dvi,dvi,\
524     .pstex_t .pstex))
525 endif
526
527 define lu-define-flavor-PDF #
528   $$ (eval $$ (call lu-create-flavor,PDF,tex,PDFLATEX,.pdf,pdf,\
529     .pdftex_t .$$(_LU_PDFTEX_EXT)))
530 endif
531
532 define lu-define-flavor-LUALATEX #
533   $$ (eval $$ (call lu-create-flavor,LUALATEX,tex,LUALATEX,.pdf,pdf,\
534     .pdftex_t .$$(_LU_PDFTEX_EXT)))
535 endif
536
537 define lu-define-flavor-PS #
538   $$ (eval $$ (call lu-create-flavor,PS,dvi,DVIPS,.ps,ps,DVI))
539 endif
540
541 define lu-define-flavor-DVIPDF #
542   $$ (eval $$ (call lu-create-flavor,DVIPDF,dvi,DVIPDFM,.pdf,pdf,DVI))
543 endif
544
545 $(eval $(call lu-addtovar-global,FLAVORS,PDF PS))
546
547 #####
548 #####
549 #####
550 #####
551 #####
552 ##### Masters #####
553 #####
554 #####
555 #####
556 #####
557 #####
558 #####
559
560 define _lu-do-latex # 1:master 2:flavor 3:source.tex 4:ext(.dvi/.pdf)
561   exec 3>&1; \
562   run() { \
563     printf "Running:" 1>&3 ; \
564     for arg; do \
565       printf "%s" " '$$arg'" 1>&3 ; \
566     done ; echo 1>&3 ; \
567     "$$@" ; \
568   }; \
569   doit() { \
570     $(RM) -v "$(1)$(4)_FAILED" \
571     "$(1)$(4)_NEED_REBUILD" \
572     "$(1)$(4).mk" ; \
573     ( echo X | \
574       run $(call lu-call-prog-flavor,$(1),$(2)) \
575       --interaction errorstopmode \
576       --jobname "$(1)" \
577       '\RequirePackage[extension="$(4)"]{texdepends}\input' "${(3)}" || \

```

```

578 touch "${1}${4}_FAILED" ; \
579 if grep -sq '^! LaTeX Error:' "${1}.log" ; then \
580 touch "${1}${4}_FAILED" ; \
581 fi \
582 ) | $(call lu-call-prog,LATEXFILTER,$(1),$(2)) ; \
583 NO_TEXDEPENDS_FILE=0 ;\
584 if [ ! -f "${1}${4}.mk" ]; then \
585 NO_TEXDEPENDS_FILE=1 ;\
586 fi ;\
587 sed -e 's,\\openout[0-9]* = \([^\']*.*\),TD_${1}${4}_OUTPUTS += \1,p;s,\\openout[0-9]* = \'(.*)\'',TD_${1}${4}_OUTPUTS += \1,p' \
588 "${1}.log" >> "${1}${4}.mk" ;\
589 if [ -f "${1}${4}_FAILED" ]; then \
590 echo "*****" ;\
591 echo "Building ${1}${4} fails" ;\
592 echo "*****" ;\
593 echo "Here are the last lines of the log file" ;\
594 echo "If this is not enough, try to" ;\
595 echo "call 'make' with 'VERB=verbose' option" ;\
596 echo "*****" ;\
597 echo "==> Last lines in ${1}.log <==" ; \
598 sed -e '/^[?] X$$/,,$$d' \
599 -e '/^Here is how much of TeX'"'"'s memory you used:$$/,,$$d' \
600 < "${1}.log" | tail -n 20; \
601 return 1; \
602 fi; \
603 if [ "$$NO_TEXDEPENDS_FILE" = 1 ]; then \
604 echo "*****" ;\
605 echo "texdepends does not seems be loaded" ;\
606 echo "Either your (La)TeX installation is wrong or you found a bug." ;\
607 echo "If so, please, report it (with the result of shell command 'kpsepath tex')";\
608 echo "Aborting compilation" ;\
609 echo "*****" ;\
610 touch "${1}${4}_FAILED" ; \
611 return 1 ;\
612 fi ;\
613 }; doit
614 endif
615
616 .PHONY: clean-build-fig
617
618 #####
619 define lu-master-texflavor-index-vars # MASTER FLAVOR TYPE INDEX ext(.dvi/.pdf)
620 $$($(call lu-show-rules,Setting flavor index vars for $(1)/$(2)/[$(3)]$(4))
621 $$$(eval $$$(call lu-addtovar,DEPENDS,$(1),$(2), \
622 $$$(call lu-getvalue-index,TARGET,$(1),$(2),$(3),$(4))))
623 $$$(eval $$$(call lu-addtovar,WATCHFILES,$(1),$(2), \
624 $$$(call lu-getvalue-index,SRC,$(1),$(2),$(3),$(4))))
625 endef #####
626 define lu-master-texflavor-index-rules # MASTER FLAVOR TYPE INDEX ext(.dvi/.pdf)
627 $$($(call lu-show-rules,Setting flavor index rules for $(1)/$(2)/[$(3)]$(4))
628 $$$(if $$(_LU_DEF_IND_$$$(call lu-getvalue-index,TARGET,$(1),$(2),$(3),$(4))), \
629 $$$(call lu-show-rules,=> Skipping: already defined in flavor $$(_LU_DEF_IND_$$$(call lu-getvalue-index,TARGET,$(1),$(2),$(3),$(4))))
630 $$$(eval $$$(call _lu-master-texflavor-index-rules\
631 ,$(1),$(2),$(3),$(4),$(5),$$$(call lu-getvalue-index,TARGET,$(1),$(2),$(3),$(4))))
632 endef
633 define _lu-master-texflavor-index-rules # MASTER FLAVOR TYPE INDEX ext TARGET
634 $(6): \
635 $$$(call lu-getvalue-index,SRC,$(1),$(2),$(3),$(4)) \

```

```

636    $$ (wildcard $$ (call lu-getvalue-index, STYLE, $(1), $(2), $(3), $(4)))
637    $$ (COMMON_PREFIX) $$ (call lu-call-prog-index, $$ (firstword $$ (call lu-getvalue-index, PROGVAR, $(1), $(2), $(3), $(4))),
638    $$ (addprefix -s, $$ (call lu-getvalue-index, STYLE, $(1), $(2), $(3), $(4))) \
639    -o $$@ $$<
640    _LU_DEF_IND_$(6)=$(2)
641    clean::
642    $$ (call lu-clean, $$ (call lu-getvalue-index, TARGET, $(1), $(2), $(3), $(4)) \
643    $$ (addsuffix .ilg, $(basename \
644    $$ (call lu-getvalue-index, SRC, $(1), $(2), $(3), $(4))))
645    endif #####
646    define lu-master-texflavor-index # MASTER FLAVOR INDEX ext(.dvi/.pdf)
647    $$ (eval $$ (call lu-master-texflavor-index-vars, $(1), $(2), $(3), $(4)))
648    $$ (eval $$ (call lu-master-texflavor-index-rules, $(1), $(2), $(3), $(4)))
649    endif
650    #####
651
652    #####
653    define lu-master-texflavor-vars # MASTER FLAVOR ext(.dvi/.pdf)
654    $$ (call lu-show-rules, Setting flavor vars for $(1)/$(2))
655    -include $(1)$(3).mk
656    $$ (eval $$ (call lu-addtovar, DEPENDS, $(1), $(2), \
657    $$ (call lu-getvalues, FIGURES, $(1), $(2)) \
658    $$ (call lu-getvalues, BIBFILES, $(1), $(2)) \
659    $$ (wildcard $$ (call lu-getvalues, INPUTS, $(1), $(2)) \
660    $$ (wildcard $$ (call lu-getvalues, BIBSTYLES, $(1), $(2)) \
661    $$ (call lu-getvalues, BBLFILES, $(1), $(2)) \
662    ))
663
664    $$ (eval $$ (call lu-addtovar-flavor, TARGETS, $(2), $(1)$(3)))
665
666    $$ (eval $$ (call lu-addtovar, GPATH, $(1), $(2), \
667    $$ (subst },, $(subst {,, $(subst }{, , \
668    $$ (call lu-getvalue, GRAPHICSPATH, $(1), $(2))))))
669
670    $$ (if $$ (sort $$ (call lu-getvalues, SUBFIGS, $(1), $(2))), \
671    $$ (eval include $$ (addsuffix .mk, $(sort \
672    $$ (call lu-getvalues, SUBFIGS, $(1), $(2))))))
673
674    $$ (eval $$ (call lu-addtovar, WATCHFILES, $(1), $(2), \
675    $$ (filter %.aux, $$ (call lu-getvalues, OUTPUTS, $(1), $(2))))))
676
677    $$ (foreach type, $$ (call lu-getvalues, INDEXES, $(1), $(2)), \
678    $$ (foreach index, $$ (call lu-getvalues, INDEXES_$(type), $(1), $(2)), \
679    $$ (eval $$ (call lu-master-texflavor-index-vars, $(1), $(2), $(type), $(index), $(3))))))
680    endif #####
681    define lu-master-texflavor-rules # MASTER FLAVOR ext(.dvi/.pdf)
682    $$ (call lu-show-rules, Defining flavor rules for $(1)/$(2))
683    $$ (call lu-getvalues, BBLFILES, $(1), $(2)): \
684    $$ (sort          $$ (call lu-getvalues, BIBFILES, $(1), $(2)) \
685    $$ (wildcard $$ (call lu-getvalues, BIBSTYLES, $(1), $(2))))
686    $(1)$(3): %$(3): \
687    $$ (filter-out $$ (call lu-getvalues, DEPENDS_EXCLUDE, $(1), $(2)), \
688    $$ (call lu-getvalues, DEPENDS, $(1), $(2)) \
689    $$ (call lu-getvalues, REQUIRED, $(1), $(2))) \
690    $$ (if $$ (wildcard $(1)$(3)_FAILED), LU_FORCE,) \
691    $$ (if $$ (wildcard $(1)$(3)_NEED_REBUILD), LU_FORCE,) \
692    $$ (if $$ (wildcard $(1)$(3)_NEED_REBUILD_IN_PROGRESS), LU_FORCE,)
693    $$ (if $$ (filter-out $$ (LU_REC_LEVEL), $$ (call lu-getvalue, MAX_REC, $(1), $(2))), , \

```

```

694  $$ (warning *****) \
695  $$ (warning *****) \
696  $$ (warning *****) \
697  $$ (warning Stopping generation of $$@) \
698  $$ (warning I got max recursion level $$ (call lu-getvalue,MAX_REC,$(1),$(2))) \
699  $$ (warning Set LU_$(1)_$(2)_MAX_REC, LU_MAX_REC_$(1) or LU_MAX_REC if you need it) \
700  $$ (warning *****) \
701  $$ (warning *****) \
702  $$ (warning *****) \
703  $$ (error Aborting generation of $$@)
704  $$ (MAKE) LU_REC_MASTER="$(1)" LU_REC_FLAVOR="$(2)" LU_REC_TARGET="$$@" \
705  LU_WATCH_FILES_SAVE
706  $$ (COMMON_PREFIX) $$ (call _lu-do-latex\
707  ,$(1),$(2),$$ (call lu-getvalue-master,MAIN,$(1)),$(3))
708  $$ (MAKE) LU_REC_MASTER="$(1)" LU_REC_FLAVOR="$(2)" LU_REC_TARGET="$$@" \
709  LU_WATCH_FILES_RESTORE
710  $$ (MAKE) LU_REC_MASTER="$(1)" LU_REC_FLAVOR="$(2)" LU_REC_TARGET="$$@" \
711  $(1)$(3)_NEED_REBUILD
712  ifneq ($(LU_REC_TARGET),)
713  $(1)$(3)_NEED_REBUILD_IN_PROGRESS:
714  $$ (COMMON_HIDE) touch $(1)$(3)_NEED_REBUILD_IN_PROGRESS
715  $$ (addprefix LU_rebuild_,$$ (call lu-getvalues,REBUILD_RULES,$(1),$(2))): \
716  $(1)$(3)_NEED_REBUILD_IN_PROGRESS
717  .PHONY: $(1)$(3)_NEED_REBUILD
718  $(1)$(3)_NEED_REBUILD: \
719  $(1)$(3)_NEED_REBUILD_IN_PROGRESS \
720  $$ (addprefix LU_rebuild_,$$ (call lu-getvalues,REBUILD_RULES,$(1),$(2)))
721  $$ (COMMON_HIDE) $(RM) $(1)$(3)_NEED_REBUILD_IN_PROGRESS
722  $$ (COMMON_HIDE) if [ -f "$(1)$(3)_NEED_REBUILD" ]; then \
723  echo "*****" ; \
724  echo "***** New build needed *****" ; \
725  echo "*****" ; \
726  cat "$(1)$(3)_NEED_REBUILD" ; \
727  echo "*****" ; \
728  fi
729  $$ (MAKE) LU_REC_LEVEL=$$ (shell expr $$ (LU_REC_LEVEL) + 1) \
730  $$ (LU_REC_TARGET)
731  endif
732  clean-build-fig::
733  $$ (call lu-clean,$$ (foreach fig, \
734  $$ (basename $$ (wildcard $$ (filter %.fig, \
735  $$ (call lu-getvalues,FIGURES,$(1),$(2))))), \
736  $$ (addprefix $(fig),$$ (call lu-getvalues-flavor,CLEANFIGEXT,$(2))))))
737  $$ (call lu-clean,$$ (foreach svg, \
738  $$ (basename $$ (wildcard $$ (filter %.svg, \
739  $$ (call lu-getvalues,FIGURES,$(1),$(2))))), \
740  $$ (addprefix $(svg),$$ (call lu-getvalues-flavor,CLEANSVGEXT,$(2))))))
741  clean:: clean-build-fig
742  $$ (call lu-clean,$$ (call lu-getvalues,OUTPUTS,$(1),$(2)) \
743  $$ (call lu-getvalues,BBLFILES,$(1),$(2)) \
744  $$ (addsuffix .mk,$$ (call lu-getvalues,SUBFIGS,$(1),$(2)) \
745  $$ (patsubst %.bbl,%.blg,$$ (call lu-getvalues,BBLFILES,$(1),$(2))))))
746  $$ (call lu-clean,$$ (wildcard $(1).log))
747  distclean::
748  $$ (call lu-clean,$$ (wildcard $(1)$(3) $(1)$(3)_FAILED \
749  $(1)$(3)_NEED_REBUILD $(1)$(3)_NEED_REBUILD_IN_PROGRESS))
750  $$ (foreach type,$$ (call lu-getvalues,INDEXES,$(1),$(2)), \
751  $$ (foreach index,$$ (call lu-getvalues,INDEXES_$(type),$(1),$(2)), \

```

```

752    $$$(eval $$$(call lu-master-texflavor-index-rules,$(1),$(2),$(type),$(index),$(3))))
753 endif #####
754 define lu-master-texflavor # MASTER FLAVOR ext(.dvi/.pdf)
755    $$$(eval $$$(call lu-master-texflavor-vars,$(1),$(2),$(3)))
756    $$$(eval $$$(call lu-master-texflavor-rules,$(1),$(2),$(3)))
757 endif
758 #####
759
760 #####
761 define lu-master-dviflavor-vars # MASTER FLAVOR ext(.ps)
762    $$$(call lu-show-rules,Setting flavor vars for \
763    $(1)/$(2)/$$$(call lu-getvalue-flavor,DEPFLAVOR,$(2)))
764 # $$$(eval $$$(call lu-addvar,VARPROG,$(1),$(2)))
765 # $$$(eval $$$(call lu-addvar,$$(call lu-getvalue,VARPROG,$(1),$(2)),$(1),$(2)))
766    $$$(eval $$$(call lu-addtovar-flavor,TARGETS,$(2),$(1)$(3)))
767 endif #####
768 define lu-master-dviflavor-rules # MASTER FLAVOR ext(.ps)
769    $$$(call lu-show-rules,Defining flavor rules for \
770    $(1)/$(2)/$$$(call lu-getvalue-flavor,DEPFLAVOR,$(2)))
771    $(1)$(3): %$(3): %$$$(call lu-getvalue-flavor,EXT,$$(call lu-getvalue-flavor,DEPFLAVOR,$(2)))
772    $$$(call lu-call-prog-flavor,$(1),$(2)) -o $$@ $$<
773 distclean::
774    $$$(call lu-clean,$$(wildcard $(1)$(3)))
775 endif #####
776 define lu-master-dviflavor # MASTER FLAVOR ext(.ps)
777    $$$(eval $$$(call lu-master-dviflavor-vars,$(1),$(2),$(3)))
778    $$$(eval $$$(call lu-master-dviflavor-rules,$(1),$(2),$(3)))
779 endif
780 #####
781
782 #####
783 define lu-master-vars # MASTER
784    $$$(call lu-show-rules,Setting vars for $(1))
785    $$$(eval $$$(call lu-setvar-master,MAIN,$(1),$(1).tex))
786    $$$(eval $$$(call lu-addtovar-master,DEPENDS,$(1),\
787    $$$(call lu-getvalue-master,MAIN,$(1))))
788    _LU_$(1)_DVI_FLAVORS=$$(filter $$(_LU_FLAVORS_DEFINED_DVI),\
789    $$$(sort $$$(call lu-getvalues-master,FLAVORS,$(1))))
790    _LU_$(1)_TEX_FLAVORS=$$(filter $$(_LU_FLAVORS_DEFINED_TEX),\
791    $$$(sort $$$(call lu-getvalues-master,FLAVORS,$(1)) \
792    $$(_LU_REC_FLAVOR) \
793    $$$(foreach dvi,$$(call lu-getvalues-master,FLAVORS,$(1)), \
794    $$$(call lu-getvalue-flavor,DEPFLAVOR,$$(dvi))))
795    $$$(foreach flav,$$( _LU_$(1)_TEX_FLAVORS), $$$(eval $$$(call \
796    lu-master-texflavor-vars,$(1),$(flav),$$$(call lu-getvalue-flavor,EXT,$$(flav))))
797    $$$(foreach flav,$$( _LU_$(1)_DVI_FLAVORS), $$$(eval $$$(call \
798    lu-master-dviflavor-vars,$(1),$(flav),$$$(call lu-getvalue-flavor,EXT,$$(flav))))
799 endif #####
800 define lu-master-rules # MASTER
801    $$$(call lu-show-rules,Defining rules for $(1))
802    $$$(foreach flav,$$( _LU_$(1)_TEX_FLAVORS), $$$(eval $$$(call \
803    lu-master-texflavor-rules,$(1),$(flav),$$$(call lu-getvalue-flavor,EXT,$$(flav))))
804    $$$(foreach flav,$$( _LU_$(1)_DVI_FLAVORS), $$$(eval $$$(call \
805    lu-master-dviflavor-rules,$(1),$(flav),$$$(call lu-getvalue-flavor,EXT,$$(flav))))
806 endif #####
807 define lu-master # MASTER
808    $$$(eval $$$(call lu-master-vars,$(1)))
809    $$$(eval $$$(call lu-master-rules,$(1)))

```



```

810 endif
811 #####
812
813 #$(warning $(call LU_RULES,example))
814 $(eval $(call lu-addtovar-global,MASTERS,\
815   $$$(shell grep -l '\documentclass' *.tex 2>/dev/null | sed -e 's/\.tex$$$$//'))
816 ifneq ($(LU_REC_TARGET),)
817   _LU_DEF_MASTERS = $(LU_REC_MASTER)
818   _LU_DEF_FLAVORS = $(LU_REC_FLAVOR) $(FLAV_DEPFLAVOR_$(LU_REC_FLAVOR))
819 else
820   _LU_DEF_MASTERS = $(call lu-getvalues-global,MASTERS)
821   _LU_DEF_FLAVORS = $(sort $(foreach master,$(_LU_DEF_MASTERS),\
822     $(call lu-getvalues-master,FLAVORS,$(master))))
823 endif
824
825 $(foreach flav, $_LU_DEF_FLAVORS), $(eval $(call lu-define-flavor,$(flav)))
826 $(foreach master, $_LU_DEF_MASTERS), $(eval $(call lu-master-vars,$(master)))
827 $(foreach flav, $_LU_FLAVORS_DEFINED), $(eval $(call lu-flavor-rules,$(flav)))
828 $(foreach master, $_LU_DEF_MASTERS), $(eval $(call lu-master-rules,$(master)))
829
830 #####
831 # Gestion des subfigs
832
833 %<<MAKEFILE
834 %.subfig.mk: %.subfig %.fig
835   $(COMMON_PREFIX)$(call lu-call-prog,GENSUBFIG) \
836   -p '$$(COMMON_PREFIX)$(call lu-call-prog,FIGDEPTH) < $$< > $$$@' \
837   -s $*.subfig $*.fig < $^ > $@
838 %MAKEFILE
839
840 %<<MAKEFILE
841 %.subfig.mk: %.subfig %.svg
842   $(COMMON_PREFIX)$(call lu-call-prog,GENSUBSVG) \
843   -p '$$(COMMON_PREFIX)$(call lu-call-prog,SVGDEPTH) < $$< > $$$@' \
844   -s $*.subfig $*.svg < $^ > $@
845 %MAKEFILE
846
847 clean::
848   $(call lu-clean,$(FIGS2CREATE_LIST))
849   $(call lu-clean,$(FIGS2CREATE_LIST:%.fig=%.pstex))
850   $(call lu-clean,$(FIGS2CREATE_LIST:%.fig=%.pstex_t))
851   $(call lu-clean,$(FIGS2CREATE_LIST:%.fig=%.($_LU_PDFTEX_EXT)))
852   $(call lu-clean,$(FIGS2CREATE_LIST:%.fig=%.pdftex_t))
853   $(call lu-clean,$(FIGS2CREATE_LIST:%.svg=%.pstex))
854   $(call lu-clean,$(FIGS2CREATE_LIST:%.svg=%.pstex_t))
855   $(call lu-clean,$(FIGS2CREATE_LIST:%.svg=%.($_LU_PDFTEX_EXT)))
856   $(call lu-clean,$(FIGS2CREATE_LIST:%.svg=%.pdftex_t))
857
858 .PHONY: LU_FORCE clean distclean
859 LU_FORCE:
860   @echo "Previous compilation failed. Rerun needed"
861
862 #$(warning $(MAKEFILE))
863
864 distclean:: clean
865
866 %<<MAKEFILE
867 %.eps: %.fig

```

```

868 $(COMMON_PREFIX)$(call lu-call-prog,FIG2DEV) -L eps $< $@
869
870 %.pdf: %.fig
871 $(COMMON_PREFIX)$(call lu-call-prog,FIG2DEV) -L pdf $< $@
872
873 %.pstex: %.fig
874 $(COMMON_PREFIX)$(call lu-call-prog,FIG2DEV) -L pstex $< $@
875
876 %.pstex: %.svg
877 $(COMMON_PREFIX)$(call lu-call-prog,SVG2DEV) -L pstex $< $@
878
879
880 .PRECIOUS: %.pstex
881 %.pstex_t: %.fig %.pstex
882 $(COMMON_PREFIX)$(call lu-call-prog,FIG2DEV) -L pstex_t -p $*.pstex $< $@
883
884 %.pstex_t: %.svg %.pstex
885 $(COMMON_PREFIX)$(call lu-call-prog,SVG2DEV) -L pstex_t -p $*.pstex $< $@
886
887
888 %.$(_LU_PDFTEX_EXT): %.fig
889 $(COMMON_PREFIX)$(call lu-call-prog,FIG2DEV) -L pdftex $< $@
890
891 %.$(_LU_PDFTEX_EXT): %.svg
892 $(COMMON_PREFIX)$(call lu-call-prog,SVG2DEV) -L pdftex $< $@
893
894 .PRECIOUS: %.$(_LU_PDFTEX_EXT)
895 %.pdftex_t: %.fig %.$(_LU_PDFTEX_EXT)
896 $(COMMON_PREFIX)$(call lu-call-prog,FIG2DEV) -L pdftex_t -p $*.$(_LU_PDFTEX_EXT) $< $@
897
898 %.pdftex_t: %.svg %.$(_LU_PDFTEX_EXT)
899 $(COMMON_PREFIX)$(call lu-call-prog,SVG2DEV) -L pdftex_t -p $*.$(_LU_PDFTEX_EXT) $< $@
900
901 %.pdf: %.eps
902 $(COMMON_PREFIX)$(call lu-call-prog,EPSTOPDF) --filter < $< > $@
903 %MAKEFILE
904
905 #####
906 # Les flavors
907 LU_REC_LEVEL ?= 1
908 ifneq ($(LU_REC_TARGET),)
909 export LU_REC_FLAVOR
910 export LU_REC_MASTER
911 export LU_REC_TARGET
912 export LU_REC_LEVEL
913 LU_REC_LOGFILE=$(LU_REC_MASTER).log
914 LU_REC_GENFILE=$(LU_REC_MASTER)$(call lu-getvalue-flavor,EXT,$(LU_REC_FLAVOR))
915
916 lu-rebuild-head=$(info *** Checking rebuild with rule '$(subst LU_rebuild_,,$@)')
917 lu-rebuild-needed=echo $(1) >> "$(LU_REC_GENFILE)_NEED_REBUILD" ;
918
919 .PHONY: $(addprefix LU_rebuild_,latex texdepends bibtex)
920 LU_rebuild_latex:
921 $(call lu-rebuild-head)
922 $(COMMON_HIDE)if grep -sq 'Rerun to get'\
923 "$$(LU_REC_LOGFILE)" ; then \
924 $(call lu-rebuild-needed\
925 ,"$@: new run needed (LaTeX message 'Rerun to get...')") \

```

```

926 fi
927
928 LU_rebuild_texdepends:
929 $(call lu-rebuild-head)
930 $(COMMON_HIDE)if grep -sq '^Package texdepends Warning: .* Check dependencies again.$$\''\
931 "$$(LU_REC_LOGFILE)" ; then \
932 $(call lu-rebuild-needed,"$$: new depends required") \
933 fi
934
935 LU_rebuild_bibtopic:
936 $(call lu-rebuild-head)
937 </makefile>

```

This part is not needed: already checked with the `lu_rebuild_latex` rule

```

938 <*notused>
939 $(COMMON_HIDE)if grep -sq 'Rerun to get indentation of bibitems right'\
940 "$$(LU_REC_LOGFILE)" ; then \
941 $(call lu-rebuild-needed,"$$: new run needed") \
942 fi
943 $(COMMON_HIDE)if grep -sq 'Rerun to get cross-references right'\
944 "$$(LU_REC_LOGFILE)" ; then \
945 $(call lu-rebuild-needed,"$$: new run needed") \
946 fi
947 </notused>
948 <*makefile>
949 $(COMMON_HIDE)sed -e '/^Package bibtopic Warning: Please (re)run BibTeX on the file(s):$$/,/^(bibtopic:
950 "$$(LU_REC_LOGFILE)" | while read file ; do \
951 touch $$file.aux ; \
952 $(call lu-rebuild-needed,"bibtopic: $$file.bbl outdated") \
953 done
954
955 LU_rebuild_bibtopic_undefined_references:
956 $(call lu-rebuild-head)
957 $(COMMON_HIDE)if grep -sq 'There were undefined references'\
958 "$$(MASTER_$(LU_REC_MASTER)).log" ; then \
959 $(call lu-rebuild-needed,"$$: new run needed") \
960 fi
961
962 .PHONY: LU_WATCH_FILES_SAVE LU_WATCH_FILES_RESTORE
963 LU_WATCH_FILES_SAVE:
964 $(COMMON_HIDE)$(foreach file, $(sort \
965 $(call lu-getvalues,WATCHFILES,$(LU_REC_MASTER),$(LU_REC_FLAVOR))), \
966 $(call lu-save-file,$(file),$(file).orig);)
967
968 LU_WATCH_FILES_RESTORE:
969 $(COMMON_HIDE)$(foreach file, $(sort \
970 $(call lu-getvalues,WATCHFILES,$(LU_REC_MASTER),$(LU_REC_FLAVOR))), \
971 $(call lu-cmprestaure-file,"$(file)","$(file).orig",\
972 echo "New $(file) file" >> $(LU_REC_GENFILE)_NEED_REBUILD;\
973 );)
974
975 endif
976
977 %<<MAKEFILE
978 %.bbl: %.aux
979 $(COMMON_PREFIX)$(call lu-call-prog,BIBTEX) $*
980 %MAKEFILE
981
982 _LaTeX_Make_GROUPS=texmfscripts tex

```

```

983 _LaTeX_Make_texmfscripts = LaTeX.mk figdepth.py gensubfig.py svg2dev.py svgdepth.py latexfilter.py
984 _LaTeX_Make_texmfscripts_DIR = scripts/latex-make
985 _LaTeX_Make_tex = figlatex.sty pdfswitch.sty texdepends.sty texgraphicx.sty
986 _LaTeX_Make_tex_DIR = tex/latex/latex-make
987
988 .PHONY: LaTeX-Make-local-install LaTeX-Make-local-uninstall
989
990 LaTeX-Make-local-uninstall::
991 $(if $(TEXMF_INSTALL_ROOT_DIR),,\
992 $(error TEXMF_INSTALL_ROOT_DIR must be set when calling LaTeX-Make-local-uninstall))
993 $(foreach g,$(_LaTeX_Make_GROUPS),\
994   $(foreach f,$(_LaTeX_Make_$(g)), \
995     $(LU_RM) $(TEXMF_INSTALL_ROOT_DIR)/$_LaTeX_Make_$(g)_DIR/$f && \
996   ) (rmdir $(TEXMF_INSTALL_ROOT_DIR)/$_LaTeX_Make_$(g)_DIR || true) && \
997   ) $(LU_RM) LaTeX.mk
998
999 LU_INSTALL_PKSEWHICH?=env -u TEXMFHOME kpsewhich
1000 LaTeX-Make-local-install::
1001 $(if $(TEXMF_INSTALL_ROOT_DIR),,\
1002 $(error TEXMF_INSTALL_ROOT_DIR must be set when calling LaTeX-Make-local-install))
1003 $(if $(filter texmf,$(notdir $(TEXMF_INSTALL_ROOT_DIR))),,\
1004 $(if $(FORCE),,\
1005 $(warning TEXMF_INSTALL_ROOT_DIR does not end with 'texmf')\
1006 $(error Use FORCE=1 if you really want to use this value of TEXMF_INSTALL_ROOT_DIR)))
1007 $(foreach g,$(_LaTeX_Make_GROUPS),\
1008   mkdir -p $(TEXMF_INSTALL_ROOT_DIR)/$_LaTeX_Make_$(g)_DIR && \
1009   $(foreach f,$(_LaTeX_Make_$(g)), \
1010     $(LU_CP) -v $$($(LU_INSTALL_PKSEWHICH) -format $g $f) $(TEXMF_INSTALL_ROOT_DIR)/$_LaTeX_Make_$(g)_DIR/$f && \
1011   )) echo "Installation into $(TEXMF_INSTALL_ROOT_DIR) done."
1012 @echo "==> You must ensure your TEXMFHOME contains this path <=="
1013
1014 </makefile>

```

## 5.2 figdepth

```

1015 <*figdepth>
1016 #!/usr/bin/python3
1017 #coding=utf8
1018
1019 """
1020
1021 stdin : the original xfig file
1022 stdout : the output xfig file
1023 args : all depths we want to keep
1024
1025 """
1026
1027 from __future__ import print_function
1028 import optparse
1029 import os.path
1030 import sys
1031
1032 def main():
1033     parser = optparse.OptionParser()
1034     (options, args) = parser.parse_args()
1035
1036     depths_to_keep = set()
1037     for arg in args:
1038         depths_to_keep.add(arg)

```

```

1039
1040     comment = ''
1041     display = True
1042     def show(depth, line):
1043         if depth in depths_to_keep:
1044             print(comment+line, end='')
1045             return True
1046         else:
1047             return False
1048     for line in sys.stdin:
1049         if line[0] == '#':
1050             comment += line
1051             continue
1052         if line[0] in "\t ":
1053             if display:
1054                 print(line, end='')
1055         else:
1056             Fld = line.split(' ', 9999)
1057             if not Fld[0] or Fld[0] not in ('1', '2', '3', '4', '5'):
1058                 print(comment+line, end='')
1059                 display = True
1060             elif Fld[0] == '4':
1061                 display = show(Fld[3], line)
1062             else:
1063                 display = show(Fld[6], line)
1064             comment = ''
1065
1066 if __name__ == "__main__":
1067     main()
1068 </figdepth>

```

### 5.3 gensubfig

```

1069 <*gensubfig>
1070 #!/usr/bin/python3
1071 #coding=utf8
1072
1073 """
1074
1075 Arguments passes :
1076     - fichier image (image.fig ou image.svg)
1077     - -s fichier subfig (image.subfig)
1078     - -p chemin du script pour generer les sous-images (svgdepth.py ou figdepth.py)
1079
1080 Sortie standard :
1081     - makefile pour creer les sous-images (au format .fig ou .svg), et pour les supprimer
1082
1083 """
1084
1085 from __future__ import print_function
1086 from optparse import OptionParser
1087 import os.path
1088
1089 def main():
1090     parser = OptionParser(usage='usage: %prog [options] svg file', description='Creates a\
1091 Makefile generating subfigures using figdepth.py or svgdepth.py')
1092     parser.add_option("-s", "--subfig", dest="subfig", help="subfig file")
1093     parser.add_option("-p", "--depth", dest="depth", help="full path of depth script")
1094     (options, args) = parser.parse_args()

```

```

1095     if len(args) < 1:
1096         parser.error("incorrect number of arguments")
1097     if not options.subfig:
1098         parser.error("no subfig file specified")
1099     if not options.depth:
1100         parser.error("no depth script specified")
1101
1102     (root, ext) = os.path.splitext(args[0])
1103     sf_name = options.subfig
1104     ds_name = options.depth
1105     varname = '%s_FIGS' % root.upper()
1106
1107     subfigs = []
1108     for line in open(options.subfig, 'r'):
1109         t = line.find('#') # looking for comments
1110         if t > -1: line = line[0:t] # remove comments...
1111         line = line.strip() #remove blank chars
1112         if line == '': continue
1113         subfigs.append(line)
1114
1115     count = 1
1116     for subfig in subfigs:
1117         print("%s_%d%s: %s%s %s" % (root, count, ext, root, ext, sf_name))
1118         print("\t%s %s" % (ds_name, subfig))
1119         print("")
1120         count += 1
1121     print("%s := $(foreach n, " % varname, end='')
1122     count = 1
1123     for subfig in subfigs:
1124         print('%d ' % count, end='')
1125         count += 1
1126     print(", %s_$(n)%s)" % (root, ext))
1127     print("FILES_TO_DISTCLEAN += $(%s)" % varname)
1128     print("FIGS2CREATE_LIST += $(%s)" % varname)
1129     print("$(TEMPORAIRE): $(%s)" % varname)
1130
1131 if __name__ == "__main__":
1132     main()
1133 </gensubfig>

```

## 5.4 svg2dev

```

1134 <*svg2dev>
1135 #!/usr/bin/python3
1136 #coding=utf8
1137
1138 from optparse import OptionParser
1139 import shutil
1140 import os
1141 import subprocess
1142
1143 svg2eps = 'inkscape %s -C --export-filename=%s.eps --export-type=eps --export-latex'
1144 svg2pdf = 'inkscape %s -C --export-filename=%s.pdf --export-type=pdf --export-latex'
1145
1146 def create_image(input_filename, output_filename, mode, ext):
1147     subprocess.Popen(mode % (input_filename, output_filename),
1148         stdout=subprocess.PIPE, shell=True).communicate()[0]
1149
1150     o_ext = output_filename + '.' + ext

```

```

1151 o = output_filename
1152 o_ext_tex = output_filename + '.' + ext + '_tex'
1153 o_t = output_filename + '_t'
1154
1155 shutil.move(o_ext, o)
1156
1157 fin = open(o_ext_tex, 'r')
1158 fout = open(o_t, 'w')
1159
1160 #\includegraphics[width=\unitlength,page=1]{logo.pdf}tex}
1161 for line in fin:
1162     # FIXME: be more conservative in the replacement
1163     # (in case '{'+o_ext+'}' appears somewhere else)
1164     out = line.replace('{'+os.path.basename(o_ext)+'}', '{'+os.path.basename(o)+'}')
1165     fout.write(out)
1166
1167 fin.close()
1168 fout.close()
1169 os.remove(o_ext_tex)
1170
1171 def main():
1172     parser = OptionParser()
1173     parser.add_option("-L", "--format", dest="outputFormat",
1174         metavar="FORMAT", help="output format", default="spstex")
1175     parser.add_option("-p", "--portrait", dest="portrait", help="dummy arg")
1176     (options, args) = parser.parse_args()
1177     if len(args) != 2: return
1178     (input_filename, output_filename) = args
1179     fmt = options.outputFormat
1180     portrait = options.portrait
1181
1182     if fmt == 'eps':
1183         create_image(input_filename, output_filename, svg2eps, 'eps')
1184     elif fmt == 'spstex' or fmt == 'pstex':
1185         create_image(input_filename, output_filename, svg2eps, 'eps')
1186     elif fmt == 'spstex_t' or fmt == 'pstex_t':
1187         pass
1188     elif fmt == 'spdfTEX' or fmt == 'pdfTEX':
1189         create_image(input_filename, output_filename, svg2pdf, 'pdf')
1190     elif fmt == 'spdfTEX_t' or fmt == 'pdfTEX_t':
1191         pass
1192
1193 if __name__ == "__main__":
1194     main()
1195
1196 </svg2dev>

```

## 5.5 latexfilter

`latexfilter.py` is a small python program that hides most of the output of  $\text{\TeX}$ / $\text{\LaTeX}$  output. It only display info, warnings, errors and underfull/overfull hbox/vbox.

```

1197 <*/latexfilter>
1198 #!/usr/bin/python3
1199 #coding=utf8
1200
1201 """
1202
1203 stdin : the original LaTeX log file
1204 stdout : the output filtered log file

```

```

1205
1206 """
1207
1208 from __future__ import print_function
1209 import optparse
1210 import os.path
1211 import re
1212 import sys
1213 import io
1214
1215 def main():
1216     parser = optparse.OptionParser()
1217     (options, args) = parser.parse_args()
1218
1219     display = 0
1220     in_display = 0
1221     start_line = ''
1222     warnerror_re = re.compile(r"^(LaTeX|Package|Class)( (.*)? (Warning:|Error:))")
1223     fullbox_re = re.compile(r"^(Underfull|Overfull) \\[hv]box")
1224     accu = ''
1225     # PDFLaTeX log file is not really in latin-1 (in T1 more exactly)
1226     # but all bytes are corrects in latin-1, so python won't stop
1227     # while parsing log.
1228     # Without specifying this encoding (ie using default utf-8), we
1229     # can get decode errors (UnicodeDecodeError: 'utf-8' codec can't decode byte...)
1230     with io.open(sys.stdin.fileno(), 'r', encoding='latin-1') as sin:
1231         for line in sin:
1232             if display > 0:
1233                 display -= 1
1234             if line[0:4].lower() in ('info', 'warn') or line[0:5].lower() == 'error':
1235                 display = 0
1236             line_groups = warnerror_re.match(line)
1237             if line_groups:
1238                 start_line = line_groups.group(3)
1239                 if not start_line:
1240                     start_line = ''
1241                 if line_groups.group(2):
1242                     start_line = "(" + start_line + ")"
1243                 display = 1
1244                 in_display = 1
1245             elif (start_line != '') and (line[0:len(start_line)] == start_line):
1246                 display = 1
1247             elif line == "\n":
1248                 in_display = 0
1249             elif line[0:4] == 'Chap':
1250                 display = 1
1251             elif fullbox_re.match(line):
1252                 display = 2
1253             if display:
1254                 print(accu, end="")
1255                 accu = line
1256             elif in_display:
1257                 print(accu[0:-1], end="")
1258                 accu = line
1259
1260 if __name__ == "__main__":
1261     main()
1262

```



1263 `</latexfilter>`

## 5.6 `svgdepth`

```
1264 <svgdepth>
1265 #!/usr/bin/python3
1266 #coding=utf8
1267
1268 import sys
1269 import xml.parsers.expat
1270
1271
1272 layers = []
1273 for arg in sys.argv:
1274     layers.append(arg)
1275
1276 parser = xml.parsers.expat.ParserCreate()
1277 class XmlParser(object):
1278     def __init__(self, layers):
1279         self.state_stack = [True]
1280         self.last_state = True
1281         self.layers = layers
1282     def XmlDeclHandler(self, version, encoding, standalone):
1283         sys.stdout.write("<?xml version='%s' encoding='%s'?>\n" % (version, encoding))
1284     def StartDoctypeDeclHandler(self, doctypeName, systemId, publicId, has_internal_subset):
1285         if publicId != None: sys.stdout.write("<!DOCTYPE %s PUBLIC \"%s\" \"%s\">\n" % \
1286             (doctypeName, publicId, systemId))
1287         else: sys.stdout.write("<!DOCTYPE %s \"%s\">\n" % (doctypeName, systemId))
1288     def StartElementHandler(self, name, attributes):
1289         if name.lower() == 'g':
1290             r = self.last_state and ('id' not in attributes or \
1291                 attributes['id'] in self.layers)
1292             self.last_state = r
1293             self.state_stack.append(r)
1294         if not self.last_state: return
1295         s = ""
1296         for k, v in attributes.items(): s += ' %s="%s"' % (k, v)
1297         sys.stdout.write("<%s%s>" % (name, s))
1298     def EndElementHandler(self, name):
1299         r = self.last_state
1300         if name.lower() == 'g':
1301             self.state_stack = self.state_stack[0:-1]
1302             self.last_state = self.state_stack[-1]
1303         if not r: return
1304         sys.stdout.write("</%s>" % (name))
1305     def CharacterDataHandler(self, data):
1306         if not self.last_state: return
1307         sys.stdout.write(data)
1308
1309 my_parser = XmlParser(layers)
1310
1311 parser.XmlDeclHandler = my_parser.XmlDeclHandler
1312 parser.StartDoctypeDeclHandler = my_parser.StartDoctypeDeclHandler
1313 parser.StartElementHandler = my_parser.StartElementHandler
1314 parser.EndElementHandler = my_parser.EndElementHandler
1315 parser.CharacterDataHandler = my_parser.CharacterDataHandler
1316
1317 for line in sys.stdin:
```

```

1318     parser.Parse(line, False)
1319 parser.Parse('', True)
1320
1321
1322 </svgdepth>

```

## 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.

### Symbols

`\"` ..... [1285](#), [1287](#)

### I

`\includegraphics` ..... [1160](#)

### U

`\unitlength` ..... [1160](#)

## Change History

v2.0.0	install ..... <a href="#">1</a>
General: First autocommented version ... <a href="#">1</a>	
v2.1.0	v2.2.5
General: That's the question ..... <a href="#">1</a>	General: fix output format of figdepth.py . <a href="#">1</a>
v2.1.1	v2.3.0
General: Improve error message ..... <a href="#">1</a>	General: Add DEPENDS-EXCLUDE, add doc and support for local texmf tree . <a href="#">1</a>
v2.1.2	v2.4.1
General: Switch from perl to python ..... <a href="#">1</a>	General: Fix encoding problem with latexfilter.pl ..... <a href="#">1</a>
v2.2.0	v2.4.2
General: Support to install LaTeX-Make locally ..... <a href="#">1</a>	General: No changes in latex-make.dtx ... <a href="#">1</a>
v2.2.1	v2.4.3
General: Improve configure ..... <a href="#">1</a>	General: Switch to T1 font encoding and lmodern font ..... <a href="#">1</a>
v2.2.2	Use python3 instead of python ..... <a href="#">1</a>
General: Fix bugs ..... <a href="#">1</a>	v2.4.4
v2.2.3	General: Add support for glossaries package ..... <a href="#">1</a>
General: Add LuaLaTeX support ..... <a href="#">1</a>	
v2.2.4	
General: Fix directory permissions on	