

The L^AT_EX.mk Makefile and related script tools*

Vincent DANJEAN

Arnaud LEGRAND

2021/10/26

Abstract

This package allows to compile all kind and complex L^AT_EX documents with the help of a Makefile. Dependencies are automatically tracked with the help of the `texdepends.sty` package.

Contents

1	Introduction	2
2	Quick start	2
2.1	First (and often last) step	2
2.2	Customization	3
	Which L ^A T _E X documents to compile	3
	Which L ^A T _E 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
3	Reference manual	5
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
4	FAQ	10
4.1	No rule to make target ‘LU_WATCH_FILES_SAVE’	10
5	Implementation	11
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 v2.4.3, last revised 2021/10/26.

1 Introduction

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

These tools can be used to compile small \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 \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) \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

¹Dependencies 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^AT_EX-Make without installing it. However, note that:

- you won't benefit of an update of L^AT_EX-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^AT_EX 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^AT_EX documents to compile

LU_MASTERS

Example: LU_MASTERS=figl_AT_EX texdepends latex-make

This variable contains the basename of the L^AT_EX documents to compile.

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

Which L^AT_EX 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^AT_EX 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 \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	<code>LU_target_NAME</code>	per document and per flavor value;
2	<code>TD_target_NAME</code>	per document and per flavor value filled by the <code>texdepends</code> L ^A T _E X package;
3	<code>LU_master_NAME</code>	per document value;
4	<code>master_NAME</code>	per document value;
5	<code>LU_FLAVOR_flavor_NAME</code>	per flavor value;
6	<code>LU_NAME</code>	global value;
7	<code>NAME</code>	global value;
8	<code>_LU_...NAME</code>	internal LaTeX.mk 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	<code>LU_target_NAME</code>	replacing per document and per flavor values;
2	<code>target_NAME</code>	cumulative per document and per flavor values;
3	<code>LU_master_NAME</code>	replacing per document values;
4	<code>master_NAME</code>	cumulative per document values;
5	<code>LU_FLAVOR_flavor_NAME</code>	replacing per flavor values;
6	<code>FLAVOR_flavor_NAME</code>	cumulative per flavor values;
7	<code>LU_NAME</code>	replacing global values;
8	<code>NAME</code>	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^AT_EX 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` `LATEX` package and `LaTeX.mk`.

Name	Kind	Context of use	Description
MASTERS	ADD	Global	List of documents to compile. These values will be used as jobname. Default: basename of *.tex files containing the \documentclass pattern
FLAVORS	ADD	Document	List of flavors for each document. Default: PS PDF
MAIN	SET	Document	Master tex source file Default: master.tex
DEPENDS	ADD	Target	List of dependencies
DEPENDS_EXCLUDE	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
STYLE	SET	Index	Name of the index/glossary style file to use (.ist, etc.)
TARGET	SET	Index	Name of the index/glossary file to produce (.ind, .gls, etc.)
SRC	SET	Index	Name of the index/glossary file source (.idx, .glo, etc.)
FIGURES	ADD	Target	Lists of figures included
BIBFILES	ADD	Target	Lists of bibliography files used (.bib)
BIBSTYLES	ADD	Target	Lists of bibliography style files used (.bst)
BBLFILES	ADD	Target	Lists of built bibliography files (.bbl)
INPUT	ADD	Target	Lists of input files (.cls, .sty, .tex, etc.)
OUTPUTS	ADD	Target	Lists of output files (.aux, etc.)
GRAPHICSPATH	ADD	Target	\graphicspath{} arguments
GPATH	ADD	Target	List of directories from GRAPHICSPATH without { and }, separated by spaces
INDEXES	ADD	Target	Kinds of index (INDEX, GLOSS, etc.)
INDEXES_kind	ADD	Target	List of indexes or glossaries
WATCHFILES	ADD	Target	List of files that trigger a rebuild if modified (.aux, etc.)
REQUIRED	ADD	Target	List of new dependencies found by the texdepends L ^A T _E X package
MAX_REC	SET	Target	Maximum level of recursion authorized
REBUILD_RULES	ADD	Target	List of rebuild rules to use (can be modified by the texdepends L ^A T _E X package)
EXT	SET	Flavor	Target file extension of the flavor
DEPFLAVOR	SET	Flavor	TEX-flavor a DVI-flavor depend upon
CLEANFIGEXT	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 endef
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 endef
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 endef
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 endef
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 endef
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 endef
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
419 # workaround the fact that $(shell ...) ignore locally exported variables
420 # get only the variables with plain names
421 _LU_MAKE_ENV := $(shell echo '$(VARIABLES)' | awk -v RS=' ' '/^[a-zA-Z0-9]+$$/' )
422 _LU_SHELL_EXPORT := $(foreach v,$(_LU_MAKE_ENV),$(v)='$(v)')
423 _lu_run_kpsewhich=$(shell $(LU_SHELL_EXPORT) kpsewhich -format $1 $2)
424
425 # Look first into the TDS (texmfscripts), then in PATH for our program
426 # At each location, we prefer with suffix than without
427 define _lu_which # VARNAME progname
428 ifeq ($(origin _LU_$1_DEFAULT), undefined)
429 _LU_$1_DEFAULT := $$($(firstword $$$(wildcard \
430 $$$(call _lu_run_kpsewhich,texmfscripts,$(2)) \
431 $$$(call _lu_run_kpsewhich,texmfscripts,$$(basename $(2))) \
432 $$$(foreach dir,$$(subst :, ,$(PATH)), \
433 $$$(dir)/$(2) $$$(dir)/$$$(basename $(2))) \
434 ) $(2))
435 export _LU_$1_DEFAULT
436 _LU_$1_DEFAULT_OLD := $$($(firstword $$$(wildcard \
437 $$$(addprefix bin/, $(2) $$$(basename $(2))) \
438 $$$(addprefix ./, $(2) $$$(basename $(2)))) \
439 $$$(if $$$(filter-out $$(_LU_$1_DEFAULT), $$(_LU_$1_DEFAULT_OLD)), \
440 $$$(if $$(_lu_scripts_warnings),, \
441 $$$(eval _lu_scripts_warnings:=done) \
442 $$$(warning By default, this version of LaTeX-Make do not use \
443 scripts in $$$(dir $$(_LU_$1_DEFAULT_OLD)) anymore.) \
444 $$$(warning For example $$(_LU_$1_DEFAULT) is used instead of $$(_LU_$1_DEFAULT_OLD)) \
445 $$$(warning If you want to keep the old behavior, add into your \
446 Makefile something like:)\
447 $$$(warning export TEXMFSCRIPTS=$$(dir $$(_LU_$1_DEFAULT_OLD))$$$$$(addprefix :,$$$$$(TEXMFSCRIPTS)
448 #$$$(warning _LU_$1_DEFAULT=$$_LU_$1_DEFAULT))
449 endif
450 $$$(eval $(call lu-setvar-global,$(1),$_LU_$1_DEFAULT)))
451 endef
452
453 $(eval $(call _lu_which,GENSUBFIG,gensubfig.py))
454 $(eval $(call _lu_which,FIGDEPTH,figdepth.py))
455 $(eval $(call _lu_which,GENSUBSVG,gensubfig.py))
456 $(eval $(call _lu_which,SVGDEPTH,svgdepth.py))
457 $(eval $(call _lu_which,SVG2DEV,svg2dev.py))
458 $(eval $(call _lu_which,LATEXFILTER,latexfilter.py))
459
460 # Rules to use to check if the build document (dvi or pdf) is up-to-date
461 # This can be overruled per document manually and/or automatically

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```



```

810 #####
811
812 #$(warning $(call LU_RULES,example))
813 $(eval $(call lu-addtovar-global,MASTERS,\
814 $$$(shell grep -l '\documentclass' *.tex 2>/dev/null | sed -e 's/\.tex$$$$//'))
815 ifneq ($(LU_REC_TARGET),)
816 _LU_DEF_MASTERS = $(LU_REC_MASTER)
817 _LU_DEF_FLAVORS = $(LU_REC_FLAVOR) $(FLAV_DEPFLAVOR_$(LU_REC_FLAVOR))
818 else
819 _LU_DEF_MASTERS = $(call lu-getvalues-global,MASTERS)
820 _LU_DEF_FLAVORS = $(sort $(foreach master,$(_LU_DEF_MASTERS),\
821 $(call lu-getvalues-master,FLAVORS,$(master))))
822 endif
823
824 $(foreach flav, $_LU_DEF_FLAVORS), $(eval $(call lu-define-flavor,$(flav)))
825 $(foreach master, $_LU_DEF_MASTERS), $(eval $(call lu-master-vars,$(master)))
826 $(foreach flav, $_LU_FLAVORS_DEFINED), $(eval $(call lu-flavor-rules,$(flav)))
827 $(foreach master, $_LU_DEF_MASTERS), $(eval $(call lu-master-rules,$(master)))
828
829 #####"
830 # Gestion des subfigs
831
832 %<<MAKEFILE
833 %.subfig.mk: %.subfig %.fig
834 $(COMMON_PREFIX)$(call lu-call-prog,GENSUBFIG) \
835 -p '$$(COMMON_PREFIX)$(call lu-call-prog,FIGDEPTH) < $$< > $$$@' \
836 -s $*.subfig $*.fig < $^ > $@
837 %MAKEFILE
838
839 %<<MAKEFILE
840 %.subfig.mk: %.subfig %.svg
841 $(COMMON_PREFIX)$(call lu-call-prog,GENSUBSVG) \
842 -p '$$(COMMON_PREFIX)$(call lu-call-prog,SVGDEPTH) < $$< > $$$@' \
843 -s $*.subfig $*.svg < $^ > $@
844 %MAKEFILE
845
846 clean::
847 $(call lu-clean,$(FIGS2CREATE_LIST))
848 $(call lu-clean,$(FIGS2CREATE_LIST:%.fig=%.pstex))
849 $(call lu-clean,$(FIGS2CREATE_LIST:%.fig=%.pstex_t))
850 $(call lu-clean,$(FIGS2CREATE_LIST:%.fig=%. $(LU_PDFTEX_EXT)))
851 $(call lu-clean,$(FIGS2CREATE_LIST:%.fig=%.pdftex_t))
852 $(call lu-clean,$(FIGS2CREATE_LIST:%.svg=%.pstex))
853 $(call lu-clean,$(FIGS2CREATE_LIST:%.svg=%.pstex_t))
854 $(call lu-clean,$(FIGS2CREATE_LIST:%.svg=%. $(LU_PDFTEX_EXT)))
855 $(call lu-clean,$(FIGS2CREATE_LIST:%.svg=%.pdftex_t))
856
857 .PHONY: LU_FORCE clean distclean
858 LU_FORCE:
859 @echo "Previous compilation failed. Rerun needed"
860
861 #$(warning $(MAKEFILE))
862
863 distclean:: clean
864
865 %<<MAKEFILE
866 %.eps: %.fig
867 $(COMMON_PREFIX)$(call lu-call-prog,FIG2DEV) -L eps $< $@

```

```

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

```

```

926
927 LU_rebuild_texdepends:
928 $(call lu-rebuild-head)
929 $(COMMON_HIDE)if grep -sq '^Package texdepends Warning: .* Check dependencies again.$$\'\
930 "$(LU_REC_LOGFILE)" ; then \
931 $(call lu-rebuild-needed,"$0: new depends required") \
932 fi
933
934 LU_rebuild_bibtopic:
935 $(call lu-rebuild-head)
936 </makefile>

```

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

```

937 <*notused>
938 $(COMMON_HIDE)if grep -sq 'Rerun to get indentation of bibitems right'\
939 "$(LU_REC_LOGFILE)" ; then \
940 $(call lu-rebuild-needed,"$0: new run needed") \
941 fi
942 $(COMMON_HIDE)if grep -sq 'Rerun to get cross-references right'\
943 "$(LU_REC_LOGFILE)" ; then \
944 $(call lu-rebuild-needed,"$0: new run needed") \
945 fi
946 </notused>
947 <*makefile>
948 $(COMMON_HIDE)sed -e '/^Package bibtopic Warning: Please (re)run BibTeX on the file(s):$$/,/^(bibtopic)
949 "$(LU_REC_LOGFILE)" | while read file ; do \
950 touch $$file.aux ; \
951 $(call lu-rebuild-needed,"bibtopic: $$file.bbl outdated") \
952 done
953
954 LU_rebuild_bibtopic_undefined_references:
955 $(call lu-rebuild-head)
956 $(COMMON_HIDE)if grep -sq 'There were undefined references'\
957 "$(MASTER_$(LU_REC_MASTER)).log" ; then \
958 $(call lu-rebuild-needed,"$0: new run needed") \
959 fi
960
961 .PHONY: LU_WATCH_FILES_SAVE LU_WATCH_FILES_RESTORE
962 LU_WATCH_FILES_SAVE:
963 $(COMMON_HIDE)$(foreach file, $(sort \
964 $(call lu-getvalues,WATCHFILES,$(LU_REC_MASTER),$(LU_REC_FLAVOR))), \
965 $(call lu-save-file,$(file),$(file).orig);)
966
967 LU_WATCH_FILES_RESTORE:
968 $(COMMON_HIDE)$(foreach file, $(sort \
969 $(call lu-getvalues,WATCHFILES,$(LU_REC_MASTER),$(LU_REC_FLAVOR))), \
970 $(call lu-cmprestaure-file,"$(file)","$(file).orig",\
971 echo "New $(file) file" >> $(LU_REC_GENFILE)_NEED_REBUILD;\
972 );)
973
974 endif
975
976 %<<MAKEFILE
977 %.bbl: %.aux
978 $(COMMON_PREFIX)$(call lu-call-prog,BIBTEX) $*
979 %MAKEFILE
980
981 _LaTeX_Make_GROUPS=texmfscripts tex
982 _LaTeX_Make_texmfscripts = LaTeX.mk figdepth.py gensubfig.py svg2dev.py svgdepth.py latexfilter.py

```

```

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

```

5.2 figdepth

```

1014 <*figdepth>
1015 #!/usr/bin/python3
1016 #coding=utf8
1017
1018 """
1019
1020 stdin : the original xfig file
1021 stdout : the output xfig file
1022 args : all depths we want to keep
1023
1024 """
1025
1026 from __future__ import print_function
1027 import optparse
1028 import os.path
1029 import sys
1030
1031 def main():
1032     parser = optparse.OptionParser()
1033     (options, args) = parser.parse_args()
1034
1035     depths_to_keep = set()
1036     for arg in args:
1037         depths_to_keep.add(arg)
1038

```

```

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

```

5.3 gensubfig

```

1068 <*gensubfig>
1069 #!/usr/bin/python3
1070 #coding=utf8
1071
1072 """
1073
1074 Arguments passes :
1075 - fichier image (image.fig ou image.svg)
1076 - -s fichier subfig (image.subfig)
1077 - -p chemin du script pour generer les sous-images (svgdepth.py ou figdepth.py)
1078
1079 Sortie standard :
1080 - makefile pour creer les sous-images (au format .fig ou .svg), et pour les supprimer
1081
1082 """
1083
1084 from __future__ import print_function
1085 from optparse import OptionParser
1086 import os.path
1087
1088 def main():
1089     parser = OptionParser(usage='usage: %prog [options] svg file', description='Creates a\
1090 Makefile generating subfigures using figdepth.py or svgdepth.py')
1091     parser.add_option("-s", "--subfig", dest="subfig", help="subfig file")
1092     parser.add_option("-p", "--depth", dest="depth", help="full path of depth script")
1093     (options, args) = parser.parse_args()
1094     if len(args) < 1:

```

```

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

```

5.4 svg2dev

```

1133 <*svg2dev>
1134 #!/usr/bin/python3
1135 #coding=utf8
1136
1137 from optparse import OptionParser
1138 import shutil
1139 import os
1140 import subprocess
1141
1142 svg2eps = 'inkscape %s -C --export-filename=%s.eps --export-type=eps --export-latex'
1143 svg2pdf = 'inkscape %s -C --export-filename=%s.pdf --export-type=pdf --export-latex'
1144
1145 def create_image(input_filename, output_filename, mode, ext):
1146     subprocess.Popen(mode % (input_filename, output_filename),
1147         stdout=subprocess.PIPE, shell=True).communicate()[0]
1148
1149     o_ext = output_filename + '.' + ext
1150     o = output_filename

```

```

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

```

5.5 latexfilter

`latexfilter.py` is a small python program that hides most of the output of $\text{T}_{\text{E}}\text{X}/\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ output. It only display info, warnings, errors and underfull/overfull hbox/vbox.

```

1196 <!*latexfilter>
1197 #!/usr/bin/python3
1198 #coding=utf8
1199
1200 """
1201
1202 stdin : the original LaTeX log file
1203 stdout : the output filtered log file
1204

```

```

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

```


5.6 svgdepth

```
1263 (*svgdepth)
1264 #!/usr/bin/python3
1265 #coding=utf8
1266
1267 import sys
1268 import xml.parsers.expat
1269
1270
1271 layers = []
1272 for arg in sys.argv:
1273     layers.append(arg)
1274
1275 parser = xml.parsers.expat.ParserCreate()
1276 class XmlParser(object):
1277     def __init__(self, layers):
1278         self.state_stack = [True]
1279         self.last_state = True
1280         self.layers = layers
1281     def XmlDeclHandler(self, version, encoding, standalone):
1282         sys.stdout.write("<?xml version='%s' encoding='%s'?>\n" % (version, encoding))
1283     def StartDoctypeDeclHandler(self, doctypeName, systemId, publicId, has_internal_subset):
1284         if publicId != None: sys.stdout.write("<!DOCTYPE %s PUBLIC \"%s\" \"%s\">\n" %
1285             (doctypeName, publicId, systemId))
1286         else: sys.stdout.write("<!DOCTYPE %s \"%s\">\n" % (doctypeName, systemId))
1287     def StartElementHandler(self, name, attributes):
1288         if name.lower() == 'g':
1289             r = self.last_state and ('id' not in attributes or \
1290                 attributes['id'] in self.layers)
1291             self.last_state = r
1292             self.state_stack.append(r)
1293             if not self.last_state: return
1294             s = ""
1295             for k, v in attributes.items(): s += ' %s="%s"' % (k, v)
1296             sys.stdout.write("<%s%s>" % (name, s))
1297     def EndElementHandler(self, name):
1298         r = self.last_state
1299         if name.lower() == 'g':
1300             self.state_stack = self.state_stack[0:-1]
1301             self.last_state = self.state_stack[-1]
1302             if not r: return
1303             sys.stdout.write("</%s>" % (name))
1304     def CharacterDataHandler(self, data):
1305         if not self.last_state: return
1306         sys.stdout.write(data)
1307
1308 my_parser = XmlParser(layers)
1309
1310 parser.XmlDeclHandler = my_parser.XmlDeclHandler
1311 parser.StartDoctypeDeclHandler = my_parser.StartDoctypeDeclHandler
1312 parser.StartElementHandler = my_parser.StartElementHandler
1313 parser.EndElementHandler = my_parser.EndElementHandler
1314 parser.CharacterDataHandler = my_parser.CharacterDataHandler
1315
1316 for line in sys.stdin:
1317     parser.Parse(line, False)
1318 parser.Parse('', True)
1319
```

1320
1321 $\langle \text{svgdepth} \rangle$

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

\backslash " 1284, 1286

I

\backslash includegraphics 1159

U

\backslash unitlength 1159

Change History

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