The LaTeX.mk Makefile and related script tools*

Vincent Danjean Arnaud Legrand

2021/10/26

Abstract

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

Contents

*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 them 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) snipset

[1] 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 LaTeX-Make without installing it. However, note that:

- you wont beneficit of an update of LaTeX-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 LaTeX 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 bt `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 LaTeX documents to compile — `LU_MASTERS`

Example: `LU_MASTERS=figlatex texdepends latex-make`

This variable contains the basename of the LaTeX documents to compile.
If not set, `LaTeX.mk` looks for all `*.tex` files containing the `\documentclass` command.

Which LaTeX 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 see 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 see 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 LaTeX 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	`.tex` ⇒ `.dvi`
PS	DVI	DVIPS	`.dvi` ⇒ `.ps`
PDF		PDFLATEX	`.tex` ⇒ `.pdf`
LUALATEX		LUALATEX	`.tex` ⇒ `.pdf`
DVIPDF	DVI	DVIPDFM	`.dvi` ⇒ `.pdf`

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 see as a kind of document (postscript, PDF, DVI, etc.) and the way to create it. Several property are attached to each flavor. Currently, there exists 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-flavors into an other 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 flavors. Most are common, a few a specific to the kind of the flavor.

Name: the name of the flavor. It is used to declare dependencies between flavors (see below). It 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 have 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	`LU_`*target*`_`*NAME*	per document and per flavor value;
2	`TD_`*target*`_`*NAME*	per document and per flavor value filled by the `texdepends` LaTeX package;
3	`LU_`*master*`_`*NAME*	per document value;
4	*master*`_`*NAME*	per document value;
5	`LU_FLAVOR_`*flavor*`_`*NAME*	per flavor value;
6	`LU_`*NAME*	global value;
7	*NAME*	global value;
8	`_LU_...`*NAME*	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	`LU_`*target*`_`*NAME*	replacing per document and per flavor values;
2	*target*`_`*NAME*	cumulative per document and per flavor values;
3	`LU_`*master*`_`*NAME*	replacing per document values;
4	*master*`_`*NAME*	cumulative per document values;
5	`LU_FLAVOR_`*flavor*`_`*NAME*	replacing per flavor values;
6	`FLAVOR_`*flavor*`_`*NAME*	cumulative per flavor values;
7	`LU_`*NAME*	replacing global values;
8	*NAME*	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` LaTeX 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
progvarname	SET	Target	Program to launch for the corresponding flavor
progvarname`_OPTIONS`	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` LaTeX 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` LaTeX 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

```
⟨*makefile⟩

####[ Check Software ]##############################################

ifeq ($(filter else-if,$(.FEATURES)),)
$(error GNU Make 3.81 needed. Please, update your software.)
 exit 1
endif

# Some people want to call our Makefile snippet with
# make -f LaTeX.mk
# This should not work as $(MAKE) is call recursively and will not read
# LaTeX.mk again. We cannot just add LaTeX.mk to MAKEFILES as LaTeX.mk
# should be read AFTER a standard Makefile (if any) that can define some
# variables (LU_MASTERS, ...) that LaTeX.mk must see.
# So I introduce an HACK here that try to workaround the situation. Keep in
# mind that this hack is not perfect and does not handle all cases
# (for example, "make -f my_latex_config.mk -f LaTeX.mk" will not recurse
# correctly)
ifeq ($(foreach m,$(MAKEFILES), $(m)) $(lastword $(MAKEFILE_LIST)),$(MAKEFILE_LIST))
# We are the first file read after the ones from MAKEFILES
# So we assume we are read due to "-f LaTeX.mk"
LU_LaTeX.mk_NAME := $(lastword $(MAKEFILE_LIST))
# Is this Makefile correctly read for recursive calls ?
ifeq ($(findstring -f $(LU_LaTeX.mk_NAME),$(MAKE)),)
$(info ********************************************************************************)
$(info Warning: $(LU_LaTeX.mk_NAME) called directly. I suppose that you run:)
$(info Warning: $(MAKE) -f $(LU_LaTeX.mk_NAME) $(MAKECMDGOALS))
$(info Warning: or something similar that does not allow recursive invocation of make)
$(info Warning: )
$(info Warning: Trying to enable a workaround. This ACK will be disabled in a future)
$(info Warning: release. Consider using another syntax, for example:)
$(info Warning: $(MAKE) -f $(LU_LaTeX.mk_NAME) MAKE="$(MAKE) -f $(LU_LaTeX.mk_NAME)" $(MAKECMDGOALS))
$(info ********************************************************************************)
MAKE+= -f $(LU_LaTeX.mk_NAME)
endif
endif

####[ Configuration ]################################################

# list of messages categories to display
LU_SHOW ?= warning #info debug debug-vars

# Select GNU/BSD/MACOSX utils (cp, rm, mv, ...)
LU_UTILS ?= $(shell ( /bin/cp --heelp > /dev/null 2>&1 && echo GNU ) || echo BSD )
export LU_UTILS

####[ End of configuration ]#########################################
# Modifying the remaining of this document may endanger you life!!! ;)

#---------------------------------------------------------------------
# Controling verbosity
ifdef VERB
MAK_VERB :=  $(VERB)
else
```

```
#MAK_VERB :=  debug
#MAK_VERB :=  verbose
#MAK_VERB :=  normal
MAK_VERB :=  quiet
#MAK_VERB :=  silent
endif

#-----------------------------------------------------------------------
# MAK_VERB -> verbosity
ifeq ($(MAK_VERB),debug)
COMMON_PREFIX  =  echo "         ======> building " $@ "<======" ; \
 printf "%s $(@F) due to:$(foreach file,$?,\n      * $(file))\n" $1; set -x;
#
COMMON_HIDE   := set -x;
COMMON_CLEAN  := set -x;
SHOW_LATEX:=true
else
ifeq ($(MAK_VERB),verbose)
COMMON_PREFIX  =  echo "         ======> building " $@ "<======" ; \
 printf "%s $(@F) due to:$(foreach file,$?,\n      * $(file))\n" $1;
#
COMMON_HIDE   :=#
COMMON_CLEAN  :=#
SHOW_LATEX:=true
else
ifeq ($(MAK_VERB),normal)
COMMON_PREFIX  =#
COMMON_HIDE   :=  @
COMMON_CLEAN  :=#
SHOW_LATEX:=true
else
ifeq ($(MAK_VERB),quiet)
COMMON_PREFIX  =  @ echo "         ======> building " $@ "<======" ;
#  echo "due to $?" ;
COMMON_HIDE   :=  @
COMMON_CLEAN  :=#
SHOW_LATEX:=
else  # silent
COMMON_PREFIX  =  @
COMMON_HIDE   :=  @
COMMON_CLEAN  :=  @
SHOW_LATEX:=
endif
endif
endif
endif

#-----------------------------------------------------------------------
# Old LaTeX have limitations
_LU_PDFTEX_EXT ?= pdftex

###############################################################################
# Utilities
LU_CP=$(LU_CP_$(LU_UTILS))
LU_MV=$(LU_MV_$(LU_UTILS))
LU_RM=$(LU_RM_$(LU_UTILS))
LU_CP_GNU ?= cp -a --
LU_MV_GNU ?= mv --
```

```
LU_RM_GNU ?= rm -f --
LU_CP_BSD ?= cp -p
LU_MV_BSD ?= mv
LU_RM_BSD ?= rm -f
LU_CP_MACOSX ?= /bin/cp -p
LU_MV_MACOSX ?= /bin/mv
LU_RM_MACOSX ?= /bin/rm -f

lu-show=\
$(if $(filter $(LU_SHOW),$(1)), \
 $(if $(2), \
  $(if $(filter-out $(2),$(MAKELEVEL)),,$(3)), \
  $(3)))
lu-show-infos=\
$(if $(filter $(LU_SHOW),$(1)), \
 $(if $(2), \
  $(if $(filter-out $(2),$(MAKELEVEL)),,$(warning $(3))), \
  $(warning $(3))))
lu-show-rules=$(call lu-show-infos,info,0,$(1))
lu-show-flavors=$(call lu-show-infos,info,0,$(1))
lu-show-var=$(call lu-show-infos,debug-vars,,  * Set $(1)=$($(1)))
lu-show-read-var=$(eval $(call lu-show-infos,debug-vars,,  Reading $(1) in $(2) ctx: $(3)))$(3)
lu-show-readone-var=$(eval $(call lu-show-infos,debug-vars,,  Reading $(1) for $(2) [one value]: $(3)))$
lu-show-set-var=$(call lu-show-infos,debug-vars,,  * Setting $(1) for $(2) to value: $(3))
lu-show-add-var=$(call lu-show-infos,debug-vars,,  * Adding to $(1) for $(2) values: $(value 3))
lu-show-add-var2=$(call lu-show-infos,warning,,  * Adding to $(1) for $(2) values: $(value 3))

lu-save-file=$(call lu-show,debug,,echo "saving $1" ;) \
 if [ -f "$1" ];then $(LU_CP) "$1" "$2" ;else $(LU_RM) "$2" ;fi
lu-cmprestaure-file=\
 if cmp -s "$1" "$2"; then \
  $(LU_MV) "$2" "$1" ; \
  $(call lu-show,debug,,echo "$1" not modified ;) \
 else \
  $(call lu-show,debug,,echo "$1" modified ;) \
  if [ -f "$2" -o -f "$1" ]; then \
   $(RM) -- "$2" ; \
   $3 \
  fi ; \
 fi

lu-clean=$(if $(strip $(1)),$(RM) $(1))

define lu-bug # description
  $$(warning Internal error: $(1))
  $$(error You probably found a bug. Please, report it.)
endef

###############################################################################
###############################################################################
###############################################################################
###############################################################################
##################                                  #########################
##################          Variables               #########################
##################                                  #########################
###############################################################################
###############################################################################
###############################################################################
```

```
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 inheritage)
192 $(if $(filter-out undefined,$(origin LU_$2$1)),$(LU_$2$1),$($2$1) $(_LU_$2$1_MK) $(TD_$2$1))
193 endef
194 define lu-define-addvar # 1:suffix_fnname 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,$$(value $4))
198   endef
199   define lu-def-addvar-inherited-ctx$1 # 1:VAR 2:...
200     $6
201     _LU_$2$$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   endef
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,$$(ctx)))))
214   endef
215 endef
216
217 # Global variable
218 # VAR (DEPENDS)
219 $(eval $(call lu-define-addvar,-global,,global,2))
220
221 # Per flavor variable
222 # FLAVOR_$2_VAR (FLAVOR_DVI_DEPENDS)
223 # 2: flavor name
224 # Inherit from VAR (DEPENDS)
225 $(eval $(call lu-define-addvar,-flavor,FLAVOR_$$2_,flavor $$2,3,\
226   GLOBAL,\
227   $$(eval $$(call lu-def-addvar-inherited-ctx-global,$$1)) \
228 ))
229
```

```
# Per master variable
# $2_VAR (source_DEPENDS)
# 2: master name
# Inherit from VAR (DEPENDS)
$(eval $(call lu-define-addvar,-master,$$2_,master $$2,3,\
  GLOBAL,\
  $$(eval $$(call lu-def-addvar-inherited-ctx-global,$$1)) \
))

# Per target variable
# $2$(EXT of $3)_VAR (source.dvi_DEPENDS)
# 2: master name
# 3: flavor name
# Inherit from $2_VAR FLAVOR_$3_VAR (source_DEPENDS FLAVOR_DVI_DEPENDS)
$(eval $(call lu-define-addvar,,$$2$$(call lu-getvalue-flavor,EXT,$$3)_,target $$2$$(call lu-getvalue-fl
  $$2_ FLAVOR_$$3_,\
  $$(eval $$(call lu-def-addvar-inherited-ctx-master,$$1,$$2)) \
  $$(eval $$(call lu-def-addvar-inherited-ctx-flavor,$$1,$$3)) \
))

# Per index/glossary variable
# $(2)_$(3)_VAR (INDEX_source_DEPENDS)
# 2: type (INDEX, GLOSS, ...)
# 3: index name
# Inherit from VAR (DEPENDS)
$(eval $(call lu-define-addvar,-global-index,$$2_$$3_,index $$3[$$2],4,\
  GLOBAL,\
  $$(eval $$(call lu-def-addvar-inherited-ctx-global,$$1)) \
))

# Per master and per index/glossary variable
# $(2)_$(3)_$(4)_VAR (source_INDEX_source_DEPENDS)
# 2: master name
# 3: type (INDEX, GLOSS, ...)
# 4: index name
# Inherit from $2_VAR $3_$4_VAR (source_DEPENDS INDEX_source_DEPENDS)
$(eval $(call lu-define-addvar,-master-index,$$2_$$3_$$4_,index $$2/$$4[$$3],5,\
  $$2_ $$3_$$4_,\
  $$(eval $$(call lu-def-addvar-inherited-ctx-master,$$1,$$2)) \
  $$(eval $$(call lu-def-addvar-inherited-ctx-global-index,$$1,$$3,$$4)) \
))

# Per target and per index/glossary variable
# $(2)$(EXT of $3)_$(4)_$(5)_VAR (source.dvi_INDEX_source_DEPENDS)
# 2: master name
# 3: flavor name
# 4: type (INDEX, GLOSS, ...)
# 5: index name
# Inherit from $2$(EXT of $3)_VAR $(2)_$(3)_$(4)_VAR
# (source.dvi_DEPENDS source_INDEX_source_DEPENDS)
$(eval $(call lu-define-addvar,-index,$$2$$(call lu-getvalue-flavor,EXT,$$3)_$$4_$$5_,index $$2$$(call l
  $$2$$(call lu-getvalue-flavor,EXT,$$3)_ $$2_$$4_$$5_,\
  $$(eval $$(call lu-def-addvar-inherited-ctx,$$1,$$2,$$3)) \
  $$(eval $$(call lu-def-addvar-inherited-ctx-master-index,$$1,$$2,$$4,$$5)) \
))
```

```
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 ))
```

```
endef

define lu-getvalue-index # 1:name 2:master 3:flavor 4:type 5:indexname
$(call lu-show-readone-var,$(1),master/flavor/index $(2)/$(3)/[$(4)]$(5),$(or \
 $(LU_$(2)$(call lu-getvalue-flavor,EXT,$(3))_$(4)_$(5)_$(1)), \
 $(LU_$(2)_$(4)_$(5)_$(1)), \
 $(TD_$(2)$(call lu-getvalue-flavor,EXT,$(3))_$(4)_$(5)_$(1)), \
 $($(2)_$(4)_$(5)_$(1)), \
 $(LU_$(4)_$(5)_$(1)), \
 $($(4)_$(5)_$(1)), \
 $(LU_$(2)$(call lu-getvalue-flavor,EXT,$(3))_$(4)_$(1)), \
 $(LU_$(2)_$(4)_$(1)), \
 $($(2)_$(4)_$(1)), \
 $(LU_$(4)_$(1)), \
 $($(4)_$(1)), \
 $(LU_$(2)_$(1)), \
 $($(2)_$(1)), \
 $(LU_FLAVOR_$(3)_$(1)), \
 $(LU_$(1)), \
 $($(1)), \
 $(_LU_$(2)$(call lu-getvalue-flavor,EXT,$(3))_$(4)_$(5)_$(1)), \
 $(_LU_$(2)_$(4)_$(5)_$(1)), \
 $(_LU_$(4)_$(5)_$(1)), \
 $(_LU_$(2)$(call lu-getvalue-flavor,EXT,$(3))_$(4)_$(1)), \
 $(_LU_$(2)_$(4)_$(1)), \
 $(_LU_FLAVOR_$(3)_$(4)_$(1)), \
 $(_LU_$(4)_$(1)), \
 $(_LU_$(2)$(call lu-getvalue-flavor,EXT,$(3))_$(1)), \
 $(_LU_$(2)_$(1)), \
 $(_LU_FLAVOR_$(3)_$(1)), \
 $(_LU_$(1))\
))
endef

define lu-call-prog # 1:varname 2:master 3:flavor [4:index]
$(call lu-getvalue,$(1),$(2),$(3)) $(call lu-getvalues,$(1)_OPTIONS,$(2),$(3))
endef

define lu-call-prog-index # 1:varname 2:master 3:flavor 4:type 5:indexname
$(call lu-getvalue$(if $(4),-index),$(1),$(2),$(3),$(4),$(5)) \
 $(call lu-getvalues$(if $(4),-index),$(1)_OPTIONS,$(2),$(3),$(4),$(5))
endef

define lu-call-prog-flavor # 1:master 2:flavor
$(call lu-call-prog,$(call lu-getvalue,VARPROG,$(1),$(2)),$(1),$(2))
endef

################################################################################
################################################################################
################################################################################
################################################################################
##################                                ##########################
##################     Global variables           ##########################
##################                                ##########################
################################################################################
################################################################################
################################################################################
################################################################################
```

```
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
```

```
#REBUILD_RULES ?= latex texdepends bibtopic bibtopic_undefined_references
$(eval $(call lu-addtovar-global,REBUILD_RULES,latex texdepends))

# Default maximum recursion level
$(eval $(call lu-setvar-global,MAX_REC,6))

##############################################################################
##############################################################################
##############################################################################
##############################################################################
##################                                  ########################
##################          Flavors                 ########################
##################                                  ########################
##############################################################################
##############################################################################
##############################################################################
##############################################################################
##############################################################################

define lu-create-texflavor # 1:name 2:tex_prog 3:file_ext
      # 4:master_cible 5:fig_extention_to_clean
  _LU_FLAVORS_DEFINED_TEX += $(1)
  $(eval $(call lu-setvar-flavor,VARPROG,$(1),$(2)))
  $(eval $(call lu-setvar-flavor,EXT,$(1),$(3)))
  $(eval $(call lu-setvar-flavor,TARGETNAME,$(1),$(4)))
  $(eval $(call lu-addtovar-flavor,CLEANFIGEXT,$(1),$(5)))
  $(eval $(call lu-addtovar-flavor,CLEANSVGEXT,$(1),$(5)))
endef

define lu-create-dviflavor # 1:name 2:dvi_prog 3:file_ext
      # 4:master_cible 5:tex_flavor_depend
  $$(eval $$(call lu-define-flavor,$(5)))
  _LU_FLAVORS_DEFINED_DVI += $(1)
  $(eval $(call lu-setvar-flavor,VARPROG,$(1),$(2)))
  $(eval $(call lu-setvar-flavor,EXT,$(1),$(3)))
  $(eval $(call lu-setvar-flavor,TARGETNAME,$(1),$(4)))
  $(eval $(call lu-setvar-flavor,DEPFLAVOR,$(1),$(5)))
endef

define lu-create-flavor # 1:name 2:type 3..7:options
  $$(if $$(filter $(1),$(_LU_FLAVORS_DEFINED)), \
 $$(call lu-show-flavors,Flavor $(1) already defined), \
 $$(call lu-show-flavors,Creating flavor $(1) ($(2))) \
 $$(eval $$(call lu-create-$(2)flavor,$(1),$(3),$(4),$(5),$(6),$(7))))
endef

define lu-define-flavor # 1:name
  $$(eval $$(call lu-define-flavor-$(1)))
endef

define lu-flavor-rules # 1:name
 $$(call lu-show-flavors,Defining rules for flavor $(1))
 $$(if $$(call lu-getvalue-flavor,TARGETNAME,$(1)), \
 $$(call lu-getvalue-flavor,TARGETNAME,$(1)): \
 $$(call lu-getvalues-flavor,TARGETS,$(1)))
 $$(if $$(call lu-getvalue-flavor,TARGETNAME,$(1)), \
 .PHONY: $$(call lu-getvalue-flavor,TARGETNAME,$(1)))
endef
```

```
520
521 define lu-define-flavor-DVI #
522   $$(eval $$(call lu-create-flavor,DVI,tex,LATEX,.dvi,dvi,\
523  .pstex_t .pstex))
524 endef
525
526 define lu-define-flavor-PDF #
527   $$(eval $$(call lu-create-flavor,PDF,tex,PDFLATEX,.pdf,pdf,\
528  .pdftex_t .$$(_LU_PDFTEX_EXT)))
529 endef
530
531 define lu-define-flavor-LUALATEX #
532   $$(eval $$(call lu-create-flavor,LUALATEX,tex,LUALATEX,.pdf,pdf,\
533  .pdftex_t .$$(_LU_PDFTEX_EXT)))
534 endef
535
536 define lu-define-flavor-PS #
537   $$(eval $$(call lu-create-flavor,PS,dvi,DVIPS,.ps,ps,DVI))
538 endef
539
540 define lu-define-flavor-DVIPDF #
541   $$(eval $$(call lu-create-flavor,DVIPDF,dvi,DVIPDFM,.pdf,pdf,DVI))
542 endef
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_$
587  "$(1).log" >> "$(1)$(4).mk" ;\
588 if [ -f "$(1)$(4)_FAILED" ]; then \
589  echo "************************************" ;\
590  echo "Building $(1)$(4) fails" ;\
591  echo "************************************" ;\
592  echo "Here are the last lines of the log file" ;\
593  echo "If this is not enought, try to" ;\
594  echo "call 'make' with 'VERB=verbose' option" ;\
595  echo "************************************" ;\
596  echo "==> Last lines in $(1).log <==" ; \
597  sed -e '/^[?] X$$/,$$d' \
598      -e '/^Here is how much of TeX'"'"'s memory you used:$$/,$$d' \
599   < "$(1).log" | tail -n 20; \
600  return 1; \
601 fi; \
602 if [ "$$NO_TEXDEPENDS_FILE" = 1 ]; then \
603  echo "************************************" ;\
604  echo "texdepends does not seems be loaded" ;\
605  echo "Either your (La)TeX installation is wrong or you found a bug." ;\
606  echo "If so, please, report it (with the result of shell command 'kpsepath tex')";\
607  echo "Aborting compilation" ;\
608  echo "************************************" ;\
609  touch "$(1)$(4)_FAILED" ; \
610  return 1 ;\
611 fi ;\
612    }; doit
613 endef
614
615 .PHONY: clean-build-fig
616
617 ############################################################
618 define lu-master-texflavor-index-vars # MASTER FLAVOR TYPE INDEX ext(.dvi/.pdf)
619  $$(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_$$(call lu-getvalue-index
629    $$(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   $$(call lu-getvalue-index,SRC,$(1),$(2),$(3),$(4)))))
644 endef ##################################################
645 define lu-master-texflavor-index # MASTER FLAVOR INDEX ext(.dvi/.pdf)
646 $$(eval $$(call lu-master-texflavor-index-vars,$(1),$(2),$(3),$(4)))
647 $$(eval $$(call lu-master-texflavor-index-rules,$(1),$(2),$(3),$(4)))
648 endef
649 ##########################################################
650
651 ##########################################################
652 define lu-master-texflavor-vars # MASTER FLAVOR ext(.dvi/.pdf)
653 $$(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),$$(index),$(3)))))
679 endef ##################################################
680 define lu-master-texflavor-rules # MASTER FLAVOR ext(.dvi/.pdf)
681 $$(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 **********************************) \
```

```
  $$(warning **********************************) \
  $$(warning **********************************) \
  $$(warning Stopping generation of $$@) \
  $$(warning I got max recursion level $$(call lu-getvalue,MAX_REC,$(1),$(2))) \
  $$(warning Set LU_$(1)_$(2)_MAX_REC, LU_MAX_REC_$(1) or LU_MAX_REC if you need it) \
  $$(warning **********************************) \
  $$(warning **********************************) \
  $$(warning **********************************) \
  $$(error Aborting generation of $$@))
 $$(MAKE) LU_REC_MASTER="$(1)" LU_REC_FLAVOR="$(2)" LU_REC_TARGET="$$@"\
  LU_WATCH_FILES_SAVE
 $$(COMMON_PREFIX)$$(call _lu-do-latex\
  ,$(1),$(2),$$(call lu-getvalue-master,MAIN,$(1)),$(3))
 $$(MAKE) LU_REC_MASTER="$(1)" LU_REC_FLAVOR="$(2)" LU_REC_TARGET="$$@"\
  LU_WATCH_FILES_RESTORE
 $$(MAKE) LU_REC_MASTER="$(1)" LU_REC_FLAVOR="$(2)" LU_REC_TARGET="$$@"\
  $(1)$(3)_NEED_REBUILD
ifneq ($(LU_REC_TARGET),)
 $(1)$(3)_NEED_REBUILD_IN_PROGRESS:
 $$(COMMON_HIDE)touch $(1)$(3)_NEED_REBUILD_IN_PROGRESS
 $$(addprefix LU_rebuild_,$$(call lu-getvalues,REBUILD_RULES,$(1),$(2))): \
 $(1)$(3)_NEED_REBUILD_IN_PROGRESS
.PHONY: $(1)$(3)_NEED_REBUILD
 $(1)$(3)_NEED_REBUILD: \
    $(1)$(3)_NEED_REBUILD_IN_PROGRESS \
    $$(addprefix LU_rebuild_,$$(call lu-getvalues,REBUILD_RULES,$(1),$(2)))
 $$(COMMON_HIDE)$(RM) $(1)$(3)_NEED_REBUILD_IN_PROGRESS
 $$(COMMON_HIDE)if [ -f "$(1)$(3)_NEED_REBUILD" ];then\
  echo "*******************************************" ;\
  echo "*********** New build needed **************" ;\
  echo "*******************************************" ;\
  cat "$(1)$(3)_NEED_REBUILD" ; \
  echo "*******************************************" ;\
 fi
 $$(MAKE) LU_REC_LEVEL=$$(shell expr $$(LU_REC_LEVEL) + 1) \
  $$(LU_REC_TARGET)
endif
 clean-build-fig::
 $$(call lu-clean,$$(foreach fig, \
    $$(basename $$(wildcard $$(filter %.fig, \
   $$(call lu-getvalues,FIGURES,$(1),$(2))))), \
    $$(addprefix $$(fig),$$(call lu-getvalues-flavor,CLEANFIGEXT,$(2)))))
 $$(call lu-clean,$$(foreach svg, \
    $$(basename $$(wildcard $$(filter %.svg, \
   $$(call lu-getvalues,FIGURES,$(1),$(2))))), \
    $$(addprefix $$(svg),$$(call lu-getvalues-flavor,CLEANSVGEXT,$(2)))))
 clean:: clean-build-fig
 $$(call lu-clean,$$(call lu-getvalues,OUTPUTS,$(1),$(2)) \
  $$(call lu-getvalues,BBLFILES,$(1),$(2)) \
  $$(addsuffix .mk,$$(call lu-getvalues,SUBFIGS,$(1),$(2))) \
     $$(patsubst %.bbl,%.blg,$$(call lu-getvalues,BBLFILES,$(1),$(2))))
 $$(call lu-clean,$$(wildcard $(1).log))
 distclean::
 $$(call lu-clean,$$(wildcard $(1)$(3) $(1)$(3)_FAILED \
  $(1)$(3)_NEED_REBUILD $(1)$(3)_NEED_REBUILD_IN_PROGRESS))
 $$(foreach type,$$(call lu-getvalues,INDEXES,$(1),$(2)), \
   $$(foreach index,$$(call lu-getvalues,INDEXES_$$(type),$(1),$(2)), \
    $$(eval $$(call lu-master-texflavor-index-rules,$(1),$(2),$$(type),$$(index),$(3)))))
```

```
endef ######################################################
define lu-master-texflavor # MASTER FLAVOR ext(.dvi/.pdf)
 $$(eval $$(call lu-master-texflavor-vars,$(1),$(2),$(3)))
 $$(eval $$(call lu-master-texflavor-rules,$(1),$(2),$(3)))
endef
############################################################

############################################################
define lu-master-dviflavor-vars # MASTER FLAVOR ext(.ps)
 $$(call lu-show-rules,Setting flavor vars for \
 $(1)/$(2)/$$(call lu-getvalue-flavor,DEPFLAVOR,$(2)))
# $$(eval $$(call lu-addvar,VARPROG,$(1),$(2)))
# $$(eval $$(call lu-addvar,$$(call lu-getvalue,VARPROG,$(1),$(2)),$(1),$(2)))
 $$(eval $$(call lu-addtovar-flavor,TARGETS,$(2),$(1)$(3)))
endef ######################################################
define lu-master-dviflavor-rules # MASTER FLAVOR ext(.ps)
 $$(call lu-show-rules,Defining flavor rules for \
 $(1)/$(2)/$$(call lu-getvalue-flavor,DEPFLAVOR,$(2)))
 $(1)$(3): %$(3): %$$(call lu-getvalue-flavor,EXT,$$(call lu-getvalue-flavor,DEPFLAVOR,$(2)))
 $$(call lu-call-prog-flavor,$(1),$(2)) -o $$@ $$<
 distclean::
 $$(call lu-clean,$$(wildcard $(1)$(3)))
endef ######################################################
define lu-master-dviflavor # MASTER FLAVOR ext(.ps)
 $$(eval $$(call lu-master-dviflavor-vars,$(1),$(2),$(3)))
 $$(eval $$(call lu-master-dviflavor-rules,$(1),$(2),$(3)))
endef
############################################################

############################################################
define lu-master-vars # MASTER
 $$(call lu-show-rules,Setting vars for $(1))
 $$(eval $$(call lu-setvar-master,MAIN,$(1),$(1).tex))
 $$(eval $$(call lu-addtovar-master,DEPENDS,$(1),\
 $$(call lu-getvalue-master,MAIN,$(1))))
 _LU_$(1)_DVI_FLAVORS=$$(filter $$(_LU_FLAVORS_DEFINED_DVI),\
 $$(sort $$(call lu-getvalues-master,FLAVORS,$(1))))
 _LU_$(1)_TEX_FLAVORS=$$(filter $$(_LU_FLAVORS_DEFINED_TEX),\
 $$(sort $$(call lu-getvalues-master,FLAVORS,$(1)) \
  $$(LU_REC_FLAVOR) \
 $$(foreach dvi,$$(call lu-getvalues-master,FLAVORS,$(1)), \
  $$(call lu-getvalue-flavor,DEPFLAVOR,$$(dvi)))))
 $$(foreach flav,$$(_LU_$(1)_TEX_FLAVORS), $$(eval $$(call \
 lu-master-texflavor-vars,$(1),$$(flav),$$(call lu-getvalue-flavor,EXT,$$(flav)))))
 $$(foreach flav,$$(_LU_$(1)_DVI_FLAVORS), $$(eval $$(call \
 lu-master-dviflavor-vars,$(1),$$(flav),$$(call lu-getvalue-flavor,EXT,$$(flav)))))
endef ######################################################
define lu-master-rules # MASTER
 $$(call lu-show-rules,Defining rules for $(1))
 $$(foreach flav,$$(_LU_$(1)_TEX_FLAVORS), $$(eval $$(call \
 lu-master-texflavor-rules,$(1),$$(flav),$$(call lu-getvalue-flavor,EXT,$$(flav)))))
 $$(foreach flav,$$(_LU_$(1)_DVI_FLAVORS), $$(eval $$(call \
 lu-master-dviflavor-rules,$(1),$$(flav),$$(call lu-getvalue-flavor,EXT,$$(flav)))))
endef ######################################################
define lu-master # MASTER
 $$(eval $$(call lu-master-vars,$(1)))
 $$(eval $$(call lu-master-rules,$(1)))
endef
```

```
################################################################
#$(warning $(call LU_RULES,example))
$(eval $(call lu-addtovar-global,MASTERS,\
 $$(shell grep -l '\\documentclass' *.tex 2>/dev/null | sed -e 's/\.tex$$$$//')))
ifneq ($(LU_REC_TARGET),)
_LU_DEF_MASTERS = $(LU_REC_MASTER)
_LU_DEF_FLAVORS = $(LU_REC_FLAVOR) $(FLAV_DEPFLAVOR_$(LU_REC_FLAVOR))
else
_LU_DEF_MASTERS = $(call lu-getvalues-global,MASTERS)
_LU_DEF_FLAVORS = $(sort $(foreach master,$(_LU_DEF_MASTERS),\
 $(call lu-getvalues-master,FLAVORS,$(master))))
endif

$(foreach flav, $(_LU_DEF_FLAVORS), $(eval $(call lu-define-flavor,$(flav))))
$(foreach master, $(_LU_DEF_MASTERS), $(eval $(call lu-master-vars,$(master))))
$(foreach flav, $(_LU_FLAVORS_DEFINED), $(eval $(call lu-flavor-rules,$(flav))))
$(foreach master, $(_LU_DEF_MASTERS), $(eval $(call lu-master-rules,$(master))))

#####################################################################""
# Gestion des subfigs

%<<MAKEFILE
%.subfig.mk: %.subfig %.fig
 $(COMMON_PREFIX)$(call lu-call-prog,GENSUBFIG) \
  -p '$$(COMMON_PREFIX)$(call lu-call-prog,FIGDEPTH) < $$< > $$@' \
  -s $*.subfig $*.fig < $^ > $@
%MAKEFILE

%<<MAKEFILE
%.subfig.mk: %.subfig %.svg
 $(COMMON_PREFIX)$(call lu-call-prog,GENSUBSVG) \
  -p '$$(COMMON_PREFIX)$(call lu-call-prog,SVGDEPTH) < $$< > $$@' \
  -s $*.subfig $*.svg < $^ > $@
%MAKEFILE

clean::
 $(call lu-clean,$(FIGS2CREATE_LIST))
 $(call lu-clean,$(FIGS2CREATE_LIST:%.fig=%.pstex))
 $(call lu-clean,$(FIGS2CREATE_LIST:%.fig=%.pstex_t))
 $(call lu-clean,$(FIGS2CREATE_LIST:%.fig=%.$(_LU_PDFTEX_EXT)))
 $(call lu-clean,$(FIGS2CREATE_LIST:%.fig=%.pdftex_t))
 $(call lu-clean,$(FIGS2CREATE_LIST:%.svg=%.pstex))
 $(call lu-clean,$(FIGS2CREATE_LIST:%.svg=%.pstex_t))
 $(call lu-clean,$(FIGS2CREATE_LIST:%.svg=%.$(_LU_PDFTEX_EXT)))
 $(call lu-clean,$(FIGS2CREATE_LIST:%.svg=%.pdftex_t))

.PHONY: LU_FORCE clean distclean
LU_FORCE:
 @echo "Previous compilation failed. Rerun needed"

#$(warning $(MAKEFILE))

distclean:: clean

%<<MAKEFILE
%.eps: %.fig
 $(COMMON_PREFIX)$(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
```

```

LU_rebuild_texdepends:
 $(call lu-rebuild-head)
 $(COMMON_HIDE)if grep -sq '^Package texdepends Warning: .* Check dependencies again.$$'\
  "$(LU_REC_LOGFILE)" ; then \
  $(call lu-rebuild-needed,"$@: new depends required") \
 fi

LU_rebuild_bibtopic:
 $(call lu-rebuild-head)
⟨/makefile⟩
```

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

```
⟨*notused⟩
 $(COMMON_HIDE)if grep -sq 'Rerun to get indentation of bibitems right'\
  "$(LU_REC_LOGFILE)" ; then \
  $(call lu-rebuild-needed,"$@: new run needed") \
 fi
 $(COMMON_HIDE)if grep -sq 'Rerun to get cross-references right'\
  "$(LU_REC_LOGFILE)" ; then \
  $(call lu-rebuild-needed,"$@: new run needed") \
 fi
⟨/notused⟩
⟨*makefile⟩
 $(COMMON_HIDE)sed -e '/^Package bibtopic Warning: Please (re)run BibTeX on the file(s):$$/,/^(bibtopic)
    "$(LU_REC_LOGFILE)" | while read file ; do \
  touch $$file.aux ; \
  $(call lu-rebuild-needed,"bibtopic: $$file.bbl outdated") \
 done

LU_rebuild_bibtopic_undefined_references:
 $(call lu-rebuild-head)
 $(COMMON_HIDE)if grep -sq 'There were undefined references'\
  "$(MASTER_$(LU_REC_MASTER)).log" ; then \
  $(call lu-rebuild-needed,"$@: new run needed") \
 fi

.PHONY: LU_WATCH_FILES_SAVE LU_WATCH_FILES_RESTORE
LU_WATCH_FILES_SAVE:
 $(COMMON_HIDE)$(foreach file, $(sort \
  $(call lu-getvalues,WATCHFILES,$(LU_REC_MASTER),$(LU_REC_FLAVOR))), \
     $(call lu-save-file,$(file),$(file).orig);)

LU_WATCH_FILES_RESTORE:
 $(COMMON_HIDE)$(foreach file, $(sort \
  $(call lu-getvalues,WATCHFILES,$(LU_REC_MASTER),$(LU_REC_FLAVOR))), \
     $(call lu-cmprestaure-file,"$(file)","$(file).orig",\
  echo "New $(file) file" >> $(LU_REC_GENFILE)_NEED_REBUILD;\
  );)

endif

%<<MAKEFILE
%.bbl: %.aux
 $(COMMON_PREFIX)$(call lu-call-prog,BIBTEX) $*
%MAKEFILE

_LaTeX_Make_GROUPS=texmfscripts tex
_LaTeX_Make_texmfscripts = LaTeX.mk figdepth.py gensubfig.py svg2dev.py svgdepth.py latexfilter.py
```

```
_LaTeX_Make_texmfscripts_DIR = scripts/latex-make
_LaTeX_Make_tex = figlatex.sty pdfswitch.sty texdepends.sty texgraphicx.sty
_LaTeX_Make_tex_DIR = tex/latex/latex-make

.PHONY: LaTeX-Make-local-install LaTeX-Make-local-uninstall

LaTeX-Make-local-uninstall::
 $(if $(TEXMF_INSTALL_ROOT_DIR),,\
 $(error TEXMF_INSTALL_ROOT_DIR must be set when calling LaTeX-Make-local-uninstall))
 $(foreach g,$(_LaTeX_Make_GROUPS),\
  $(foreach f,$(_LaTeX_Make_$(g)), \
   $(LU_RM) $(TEXMF_INSTALL_ROOT_DIR)/$(_LaTeX_Make_$(g)_DIR)/$f && \
  ) (rmdir $(TEXMF_INSTALL_ROOT_DIR)/$(_LaTeX_Make_$(g)_DIR) || true) && \
 ) $(LU_RM) LaTeX.mk

LU_INSTALL_PKSEWHICH?=env -u TEXMFHOME kpsewhich
LaTeX-Make-local-install::
 $(if $(TEXMF_INSTALL_ROOT_DIR),,\
 $(error TEXMF_INSTALL_ROOT_DIR must be set when calling LaTeX-Make-local-install))
 $(if $(filter texmf,$(notdir $(TEXMF_INSTALL_ROOT_DIR))),,\
 $(if $(FORCE),,\
 $(warning TEXMF_INSTALL_ROOT_DIR does not end with 'texmf')\
 $(error Use FORCE=1 if you really want to use this value of TEXMF_INSTALL_ROOT_DIR)))
 $(foreach g,$(_LaTeX_Make_GROUPS),\
  mkdir -p $(TEXMF_INSTALL_ROOT_DIR)/$(_LaTeX_Make_$(g)_DIR) && \
  $(foreach f,$(_LaTeX_Make_$(g)), \
   $(LU_CP) -v $$($(LU_INSTALL_PKSEWHICH) -format $g $f)  $(TEXMF_INSTALL_ROOT_DIR)/$(_LaTeX_Make_$(g)_I
 )) echo "Installation into $(TEXMF_INSTALL_ROOT_DIR) done."
 @echo "==> You must ensure your TEXMFHOME contains this path <=="
```

⟨/makefile⟩

5.2 figdepth

⟨∗figdepth⟩

```
#!/usr/bin/python3
#coding=utf8

"""

stdin : the original xfig file
stdout : the output xfig file
args : all depths we want to keep

"""

from __future__ import print_function
import optparse
import os.path
import sys

def main():
    parser = optparse.OptionParser()
    (options, args) = parser.parse_args()

    depths_to_keep = set()
    for arg in args:
        depths_to_keep.add(arg)

```

```
    comment = ''
    display = True
    def show(depth, line):
        if depth in depths_to_keep:
            print(comment+line, end='')
            return True
        else:
            return False
    for line in sys.stdin:
        if line[0] == '#':
            comment += line
            continue
        if line[0] in "\t ":
            if display:
                print(line, end='')
        else:
            Fld = line.split(' ', 9999)
            if not Fld[0] or Fld[0] not in ('1', '2', '3', '4', '5'):
                print(comment+line, end='')
                display = True
            elif Fld[0] == '4':
                display = show(Fld[3], line)
            else:
                display = show(Fld[6], line)
            comment = ''

if __name__ == "__main__":
    main()
```
⟨/figdepth⟩

5.3 gensubfig

⟨∗gensubfig⟩
```
#!/usr/bin/python3
#coding=utf8

"""

Arguments passes :
    - fichier image (image.fig ou image.svg)
    - -s fichier subfig (image.subfig)
    - -p chemin du script pour generer les sous-images (svgdepth.py ou figdepth.py)

Sortie standard :
    - makefile pour creer les sous-images (au format .fig ou .svg), et pour les supprimer

"""

from __future__ import print_function
from optparse import OptionParser
import os.path

def main():
    parser = OptionParser(usage='usage: %prog [options] svg file', description='Creates a\
Makefile generating subfigures using figdepth.py or svgdepth.py')
    parser.add_option("-s", "--subfig", dest="subfig", help="subfig file")
    parser.add_option("-p", "--depth", dest="depth", help="full path of depth script")
    (options, args) = parser.parse_args()
    if len(args) < 1:
```

```
        parser.error("incorrect number of arguments")
    if not options.subfig:
        parser.error("no subfig file specified")
    if not options.depth:
        parser.error("no depth script specified")

    (root, ext) = os.path.splitext(args[0])
    sf_name = options.subfig
    ds_name = options.depth
    varname = '%s_FIGS' % root.upper()

    subfigs = []
    for line in open(options.subfig, 'r'):
        t = line.find('#') # looking for comments
        if t > -1: line = line[0:t] # remove comments...
        line = line.strip() #remove blank chars
        if line == '': continue
        subfigs.append(line)

    count = 1
    for subfig in subfigs:
        print("%s_%d%s: %s%s %s" % (root, count, ext, root, ext, sf_name))
        print("\t%s %s" % (ds_name, subfig))
        print("")
        count += 1
    print("%s := $(foreach n, " % varname, end='')
    count = 1
    for subfig in subfigs:
        print('%d ' % count, end='')
        count += 1
    print(", %s_$(n)%s)" % (root, ext))
    print("FILES_TO_DISTCLEAN += $(%s)" % varname)
    print("FIGS2CREATE_LIST += $(%s)" % varname)
    print("$(TEMPORAIRE): $(%s)" % varname)

if __name__ == "__main__":
    main()
```
⟨/gensubfig⟩

5.4 svg2dev

⟨∗svg2dev⟩
```
#!/usr/bin/python3
#coding=utf8

from optparse import OptionParser
import shutil
import os
import subprocess

svg2eps = 'inkscape %s -C --export-filename=%s.eps --export-type=eps --export-latex'
svg2pdf = 'inkscape %s -C --export-filename=%s.pdf --export-type=pdf --export-latex'

def create_image(input_filename, output_filename, mode, ext):
    subprocess.Popen(mode % (input_filename, output_filename),
        stdout=subprocess.PIPE, shell=True).communicate()[0]

    o_ext = output_filename + '.' + ext
    o = output_filename
```

```
    o_ext_tex = output_filename + '.' + ext + '_tex'
    o_t = output_filename + '_t'

    shutil.move(o_ext, o)

    fin = open(o_ext_tex, 'r')
    fout = open(o_t, 'w')

    #\includegraphics[width=\unitlength,page=1]{logo.pdftex}
    for line in fin:
        # FIXME: be more conservative in the replacement
        # (in case '{'+o_ext+'}' appeares somewhere else)
        out = line.replace('{'+os.path.basename(o_ext)+'}', '{'+os.path.basename(o)+'}')
        fout.write(out)

    fin.close()
    fout.close()
    os.remove(o_ext_tex)

def main():
    parser = OptionParser()
    parser.add_option("-L", "--format", dest="outputFormat",
        metavar="FORMAT", help="output format", default="spstex")
    parser.add_option("-p", "--portrait", dest="portrait", help="dummy arg")
    (options, args) = parser.parse_args()
    if len(args) != 2: return
    (input_filename, output_filename) = args
    fmt = options.outputFormat
    portrait = options.portrait

    if fmt == 'eps':
        create_image(input_filename, output_filename, svg2eps, 'eps')
    elif fmt == 'spstex' or fmt == 'pstex':
        create_image(input_filename, output_filename, svg2eps, 'eps')
    elif fmt == 'spstex_t' or fmt == 'pstex_t':
        pass
    elif fmt == 'spdftex' or fmt == 'pdftex':
        create_image(input_filename, output_filename, svg2pdf, 'pdf')
    elif fmt == 'spdftex_t' or fmt == 'pdftex_t':
        pass

if __name__ == "__main__":
    main()

⟨/svg2dev⟩
```

5.5 latexfilter

`latexfilter.py` is a small python program that hides most of the output of TEX/LATEX output. It only display info, warnings, errors and underfull/overfull hbox/vbox.

```
⟨∗latexfilter⟩
#!/usr/bin/python3
#coding=utf8

"""

stdin : the original LaTeX log file
stdout : the output filtered log file

```

```
"""

from __future__ import print_function
import optparse
import os.path
import re
import sys
import io

def main():
    parser = optparse.OptionParser()
    (options, args) = parser.parse_args()

    display = 0
    in_display = 0
    start_line = ''
    warnerror_re = re.compile(r"^(LaTeX|Package|Class)( (.*))? (Warning:|Error:)")
    fullbox_re = re.compile(r"^(Underfull|Overfull) \\[hv]box")
    accu = ''
    # PDFLaTeX log file is not really in latin-1 (in T1 more exactly)
    # but all bytes are corrects in latin-1, so python won't stop
    # while parsing log.
    # Without specifying this encoding (ie using default utf-8), we
    # can get decode errors (UnicodeDecodeError: 'utf-8' codec can't decode byte...)
    with io.open(sys.stdin.fileno(),'r',encoding='latin-1') as sin:
        for line in sin:
            if display > 0:
                display -= 1
            if line[0:4].lower() in ('info', 'warn') or line[0:5].lower() == 'error':
                display = 0
            line_groups = warnerror_re.match(line)
            if line_groups:
                start_line = line_groups.group(3)
                if not start_line:
                    start_line = ''
                if line_groups.group(2):
                    start_line = "(" + start_line + ")"
                display = 1
                in_display = 1
            elif (start_line != '') and (line[0:len(start_line)] == start_line):
                display = 1
            elif line == "\n":
                in_display = 0
            elif line[0:4] == 'Chap':
                display = 1
            elif fullbox_re.match(line):
                display = 2
            if display:
                print(accu, end="")
                accu = line
            elif in_display:
                print(accu[0:-1], end="")
                accu = line

if __name__ == "__main__":
    main()

⟨/latexfilter⟩
```

5.6 svgdepth

```
⟨*svgdepth⟩
#!/usr/bin/python3
#coding=utf8

import sys
import xml.parsers.expat


layers = []
for arg in sys.argv:
  layers.append(arg)

parser = xml.parsers.expat.ParserCreate()
class XmlParser(object):
    def __init__(self, layers):
        self.state_stack = [True]
        self.last_state = True
        self.layers = layers
    def XmlDeclHandler(self, version, encoding, standalone):
        sys.stdout.write("<?xml version='%s' encoding='%s'?>\n" % (version, encoding))
    def StartDoctypeDeclHandler(self, doctypeName, systemId, publicId, has_internal_subset):
        if publicId != None: sys.stdout.write("<!DOCTYPE %s PUBLIC \"%s\" \"%s\">\n" %\
            (doctypeName, publicId, systemId))
        else: sys.stdout.write("<!DOCTYPE %s \"%s\">\n" % (doctypeName, systemId))
    def StartElementHandler(self, name, attributes):
        if name.lower() == 'g':
            r = self.last_state and ('id' not in attributes or \
                attributes['id'] in self.layers)
            self.last_state = r
            self.state_stack.append(r)
        if not self.last_state: return
        s = ""
        for k, v in attributes.items(): s += ' %s="%s"' % (k, v)
        sys.stdout.write("<%s%s>" % (name, s))
    def EndElementHandler(self, name):
        r = self.last_state
        if name.lower() == 'g':
            self.state_stack = self.state_stack[0:-1]
            self.last_state = self.state_stack[-1]
        if not r: return
        sys.stdout.write("</%s>" % (name))
    def CharacterDataHandler(self, data):
        if not self.last_state: return
        sys.stdout.write(data)

my_parser = XmlParser(layers)

parser.XmlDeclHandler = my_parser.XmlDeclHandler
parser.StartDoctypeDeclHandler = my_parser.StartDoctypeDeclHandler
parser.StartElementHandler = my_parser.StartElementHandler
parser.EndElementHandler = my_parser.EndElementHandler
parser.CharacterDataHandler = my_parser.CharacterDataHandler

for line in sys.stdin:
    parser.Parse(line, False)
parser.Parse('', True)

```

1320
1321 ⟨/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.

Change History