

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

Vincent DANJEAN

Arnaud LEGRAND

2012/03/17

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	2
	Which L ^A T _E X documents to compile	2
	Which L ^A T _E X main source for a document	2
	Which flavors must be compiled	3
	Which programs are called and with which options	3
	Per target programs and options	3
	Global and per target dependencies	3
3	Reference manual	4
3.1	Flavors	4
	3.1.1 What is a flavor ?	4
	3.1.2 Defining a new flavor	4
3.2	Variables	5
	3.2.1 Two kind of variables	5
	3.2.2 List of used variables	7
4	FAQ	9
4.1	No rule to make target ‘LU_WATCH_FILES_SAVE’	9
5	Implementation	10
5.1	LaTeX.mk	10
5.2	LaTeX.mk.conf	26
5.3	figdepth	26
5.4	gensubfig	27
5.5	svg2dev	28
5.6	latexfilter	29
5.7	svgdepth	30

*This file has version number v2.1.2, last revised 2012/03/17.

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.

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.

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.

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 `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	<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>
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
```

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 | <i>LU_target_NAME</i> | per document and per flavor value; |
| 2 | <i>TD_target_NAME</i> | per document and per flavor value filled by the <code>texdepends</code> L ^A T _E X package; |
| 3 | <i>LU_master_NAME</i> | per document value; |
| 4 | <i>master_NAME</i> | per document value; |
| 5 | <i>LU_FLAVOR_flavor_NAME</i> | per flavor value; |
| 6 | <i>LU_NAME</i> | global value; |
| 7 | <i>NAME</i> | global value; |
| 8 | <i>_LU...NAME</i> | internal L ^A T _E X.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 | <i>LU_target_NAME</i> | replacing per document and per flavor values; |
| 2 | <i>target_NAME</i> | cumulative per document and per flavor values; |
| 3 | <i>LU_master_NAME</i> | replacing per document values; |
| 4 | <i>master_NAME</i> | cumulative per document values; |
| 5 | <i>LU_FLAVOR_flavor_NAME</i> | replacing per flavor values; |
| 6 | <i>FLAVOR_flavor_NAME</i> | cumulative per flavor values; |
| 7 | <i>LU_NAME</i> | replacing global values; |
| 8 | <i>NAME</i> | cumulative global values; |

Tip: if not defined, *LU_variable* defaults to “\$(*variable*) \$(*_LU_variable*)” and *_LU_variable* contains default values managed by L^AT_EX.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
<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_ <i>kind</i>	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 # For global site options
42 -include LaTeX.mk.conf
43
44 # list of messages categories to display
45 LU_SHOW ?= warning #info debug debug-vars
46
47 # Select GNU/BSD/MACOSX utils (cp, rm, mv, ...)
48 LU_UTILS ?= GNU
49
50 #####[ End of configuration ]#####
51 # Modifying the remaining of this document may endanger you life!!! ;)
52
53 #-----
54 # Controlling verbosity
```

```

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

```

```

113
114 lu-show=\
115 $(if $(filter $(LU_SHOW),$(1)), \
116 $(if $(2), \
117 $(if $(filter-out $(2),$(MAKELEVEL)),,$(3)), \
118 $(3)))
119 lu-show-infos=\
120 $(if $(filter $(LU_SHOW),$(1)), \
121 $(if $(2), \
122 $(if $(filter-out $(2),$(MAKELEVEL)),,$(warning $(3))), \
123 $(warning $(3))))
124 lu-show-rules=$(call lu-show-infos,info,0,$(1))
125 lu-show-flavors=$(call lu-show-infos,info,0,$(1))
126 lu-show-var=$(call lu-show-infos,debug-vars,, * Set $(1)=$( $(1)))
127 lu-show-read-var=$(eval $(call lu-show-infos,debug-vars,, Read-
    ing $(1) in $(2) ctx: $(3)))$(3)
128 lu-show-readone-var=$(eval $(call lu-show-infos,debug-vars,, Read-
    ing $(1) for $(2) [one value]: $(3)))$(3)
129 lu-show-set-var=$(call lu-show-infos,debug-vars,, * Setting $(1) for $(2) to value: $(3))
130 lu-show-add-var=$(call lu-show-infos,debug-vars,, * Adding to $(1) for $(2) val-
    ues: $(value 3))
131 lu-show-add-var2=$(call lu-show-infos,warning,, * Adding to $(1) for $(2) val-
    ues: $(value 3))
132
133 lu-save-file=$(call lu-show,debug,,echo "saving $1" ;) \
134 if [ -f "$1" ];then $(LU_CP) "$1" "$2" ;else $(LU_RM) "$2" ;fi
135 lu-cmprestaure-file=\
136 if cmp -s "$1" "$2"; then \
137 $(LU_MV) "$2" "$1" ; \
138 $(call lu-show,debug,,echo "$1" not modified ;) \
139 else \
140 $(call lu-show,debug,,echo "$1" modified ;) \
141 if [ -f "$2" -o -f "$1" ]; then \
142 $(RM) -- "$2" ; \
143 $3 \
144 fi ; \
145 fi
146
147 lu-clean=$(if $(strip $(1)),$(RM) $(1))
148
149 define lu-bug # description
150   $$ (warning Internal error: $(1))
151   $$ (error You probably found a bug. Please, report it.)
152 endef
153
154 #####
155 #####
156 #####
157 #####
158 #####
159 #####          Variables          #####
160 #####
161 #####
162 #####
163 #####
164 #####
165 #####
166 #

```

```

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

```

```

224 # 2: master name
225 # Inherit from VAR (DEPENDS)
226 $(eval $(call lu-define-addvar,-master,$$2_,master $$2,3,\
227   GLOBAL,\
228   $$$(eval $$$(call lu-def-addvar-inherited-ctx-global,$$1)) \
229 ))
230
231 # Per target variable
232 # $$$(EXT of $3)_VAR (source.dvi_DEPENDS)
233 # 2: master name
234 # 3: flavor name
235 # Inherit from $2_VAR FLAVOR_$3_VAR (source_DEPENDS FLAVOR_DVI_DEPENDS)
236 $(eval $(call lu-define-addvar,,$$2$$$(call lu-getvalue-flavor,EXT,$$3)_ ,target $$2$$$(call lu-
  getvalue-flavor,EXT,$$3),4,\
237   $$2_ FLAVOR_$$3_,\
238   $$$(eval $$$(call lu-def-addvar-inherited-ctx-master,$$1,$$2)) \
239   $$$(eval $$$(call lu-def-addvar-inherited-ctx-flavor,$$1,$$3)) \
240 ))
241
242 # Per index/glossary variable
243 # $(2)_$(3)_VAR (INDEX_source_DEPENDS)
244 # 2: type (INDEX, GLOSS, ...)
245 # 3: index name
246 # Inherit from VAR (DEPENDS)
247 $(eval $(call lu-define-addvar,-global-index,$$2_$$3_,index $$3[$$2],4,\
248   GLOBAL,\
249   $$$(eval $$$(call lu-def-addvar-inherited-ctx-global,$$1)) \
250 ))
251
252 # Per master and per index/glossary variable
253 # $(2)_$(3)_$(4)_VAR (source_INDEX_source_DEPENDS)
254 # 2: master name
255 # 3: type (INDEX, GLOSS, ...)
256 # 4: index name
257 # Inherit from $2_VAR $3_$4_VAR (source_DEPENDS INDEX_source_DEPENDS)
258 $(eval $(call lu-define-addvar,-master-index,$$2_$$3_$$4_,index $$2/$$4[$$3],5,\
259   $$2_ $$3_$$4_,\
260   $$$(eval $$$(call lu-def-addvar-inherited-ctx-master,$$1,$$2)) \
261   $$$(eval $$$(call lu-def-addvar-inherited-ctx-global-index,$$1,$$3,$$4)) \
262 ))
263
264 # Per target and per index/glossary variable
265 # $(2)$(EXT of $3)_$(4)_$(5)_VAR (source.dvi_INDEX_source_DEPENDS)
266 # 2: master name
267 # 3: flavor name
268 # 4: type (INDEX, GLOSS, ...)
269 # 5: index name
270 # Inherit from $2$(EXT of $3)_VAR $(2)_$(3)_$(4)_VAR
271 # (source.dvi_DEPENDS source_INDEX_source_DEPENDS)
272 $(eval $(call lu-define-addvar,-index,$$2$$$(call lu-getvalue-
  flavor,EXT,$$3)_$$4_$$5_,index $$2$$$(call lu-getvalue-flavor,EXT,$$3)/$$5[$$4],6,\
273   $$2$$$(call lu-getvalue-flavor,EXT,$$3)_ $$2_$$4_$$5_,\
274   $$$(eval $$$(call lu-def-addvar-inherited-ctx,$$1,$$2,$$3)) \
275   $$$(eval $$$(call lu-def-addvar-inherited-ctx-master-index,$$1,$$2,$$4,$$5)) \
276 ))
277
278
279

```

```

280
281
282
283 define lu-setvar-global # 1:name 2:value
284   _LU_$ (1) ?= $(2)
285   $$ (eval $$ (call lu-show-set-var,$(1),global,$(2)))
286 endef
287
288 define lu-setvar-flavor # 1:name 2:flavor 3:value
289   _LU_FLAVOR_$ (2)_$(1) ?= $(3)
290   $$ (eval $$ (call lu-show-set-var,$(1),flavor $(2),$(3)))
291 endef
292
293 define lu-setvar-master # 1:name 2:master 3:value
294   _LU_$ (2)_$(1) ?= $(3)
295   $$ (eval $$ (call lu-show-set-var,$(1),master $(2),$(3)))
296 endef
297
298 define lu-setvar # 1:name 2:master 3:flavor 4:value
299   _LU_$ (2)$$ (call lu-getvalue-flavor,EXT,$(3))_$(1)=$(4)
300   $$ (eval $$ (call lu-show-set-var,$(1),master/flavor $(2)/$(3),$(4)))
301 endef
302
303 define lu-getvalue # 1:name 2:master 3:flavor
304 $(call lu-show-readone-var,$(1),master/flavor $(2)/$(3),$(or \
305 $(LU_$ (2))$(call lu-getvalue-flavor,EXT,$(3))_$(1)), \
306 $(TD_$ (2))$(call lu-getvalue-flavor,EXT,$(3))_$(1)), \
307 $(LU_$ (2)_$(1)), \
308 $( $(2)_$(1)), \
309 $(LU_FLAVOR_$ (3)_$(1)), \
310 $(LU_$ (1)), \
311 $( $(1)), \
312 $(LU_$ (2))$(call lu-getvalue-flavor,EXT,$(3))_$(1)), \
313 $(LU_$ (2)_$(1)), \
314 $(LU_FLAVOR_$ (3)_$(1)), \
315 $(LU_$ (1))\
316 ))
317 endef
318
319 define lu-getvalue-flavor # 1:name 2:flavor
320 $(call lu-show-readone-var,$(1),flavor $(2),$(or \
321 $(LU_FLAVOR_$ (2)_$(1)), \
322 $(LU_$ (1)), \
323 $( $(1)), \
324 $(LU_FLAVOR_$ (2)_$(1)), \
325 $(LU_$ (1))\
326 ))
327 endef
328
329 define lu-getvalue-master # 1:name 2:master
330 $(call lu-show-readone-var,$(1),master $(2),$(or \
331 $(LU_$ (2)_$(1)), \
332 $( $(2)_$(1)), \
333 $(LU_$ (1)), \
334 $( $(1)), \
335 $(LU_$ (2)_$(1)), \
336 $(LU_$ (1))\
337 ))

```

```

338 endif
339
340 define lu-getvalue-index # 1:name 2:master 3:flavor 4:type 5:indexname
341 $(call lu-show-readone-var,$(1),master/flavor/index $(2)/$(3)/[$(4)]$(5),$(or \
342 $(LU_$(2))$(call lu-getvalue-flavor,EXT,$(3))_$(4)_$(5)_$(1)), \
343 $(LU_$(2)_$(4)_$(5)_$(1)), \
344 $(TD_$(2))$(call lu-getvalue-flavor,EXT,$(3))_$(4)_$(5)_$(1)), \
345 $($ (2)_$(4)_$(5)_$(1)), \
346 $(LU_$(4)_$(5)_$(1)), \
347 $($ (4)_$(5)_$(1)), \
348 $(LU_$(2))$(call lu-getvalue-flavor,EXT,$(3))_$(4)_$(1)), \
349 $(LU_$(2)_$(4)_$(1)), \
350 $($ (2)_$(4)_$(1)), \
351 $(LU_$(4)_$(1)), \
352 $($ (4)_$(1)), \
353 $(LU_$(2)_$(1)), \
354 $($ (2)_$(1)), \
355 $(LU_FLAVOR_$(3)_$(1)), \
356 $(LU_$(1)), \
357 $($ (1)), \
358 $(LU_$(2))$(call lu-getvalue-flavor,EXT,$(3))_$(4)_$(5)_$(1)), \
359 $(LU_$(2)_$(4)_$(5)_$(1)), \
360 $(LU_$(4)_$(5)_$(1)), \
361 $(LU_$(2))$(call lu-getvalue-flavor,EXT,$(3))_$(4)_$(1)), \
362 $(LU_$(2)_$(4)_$(1)), \
363 $(LU_FLAVOR_$(3)_$(4)_$(1)), \
364 $(LU_$(4)_$(1)), \
365 $(LU_$(2))$(call lu-getvalue-flavor,EXT,$(3))_$(1)), \
366 $(LU_$(2)_$(1)), \
367 $(LU_FLAVOR_$(3)_$(1)), \
368 $(LU_$(1))\
369 ))
370 endif
371
372 define lu-call-prog # 1:varname 2:master 3:flavor [4:index]
373 $(call lu-getvalue,$(1),$(2),$(3)) $(call lu-getvalues,$(1)_OPTIONS,$(2),$(3))
374 endif
375
376 define lu-call-prog-index # 1:varname 2:master 3:flavor 4:type 5:indexname
377 $(call lu-getvalue$(if $(4),-index),$(1),$(2),$(3),$(4),$(5)) \
378 $(call lu-getvalues$(if $(4),-index),$(1)_OPTIONS,$(2),$(3),$(4),$(5))
379 endif
380
381 define lu-call-prog-flavor # 1:master 2:flavor
382 $(call lu-call-prog,$(call lu-getvalue,VARPROG,$(1),$(2)),$(1),$(2))
383 endif
384
385 #####
386 #####
387 #####
388 #####
389 #####
390 ##### Global variables #####
391 #####
392 #####
393 #####
394 #####
395 #####

```

```

396 #####
397
398 # Globals variables
399 $(eval $(call lu-setvar-global,LATEX,latex))
400 $(eval $(call lu-setvar-global,PDFLATEX,pdflatex))
401 $(eval $(call lu-setvar-global,DVIPS,dvips))
402 $(eval $(call lu-setvar-global,DVIPDFM,dvipdfm))
403 $(eval $(call lu-setvar-global,BIBTEX,bibtex))
404 #$(eval $(call lu-setvar-global,MPOST,TEX="$(LATEX)" mpost))
405 $(eval $(call lu-setvar-global,FIG2DEV,fig2dev))
406 #$(eval $(call lu-setvar-global,SVG2DEV,svg2dev))
407 $(eval $(call lu-setvar-global,EPSTOPDF,epstopdf))
408 $(eval $(call lu-setvar-global,MAKEINDEX,makeindex))
409
410 # Look for local version, then texmfscript, then in PATH of our program
411 # At each location, we prefer with suffix than without
412 define _lu_which # VARNAME progname
413 ifeq ($(origin _LU_$(1)_DEFAULT), undefined)
414 _LU_$(1)_DEFAULT := $(firstword $(wildcard \
415     $(addprefix bin/, $(2)) $(basename $(2))) \
416     $(addprefix ./, $(2)) $(basename $(2))) \
417     $(shell kpsewhich -format texmfscripts $(2)) \
418     $(shell kpsewhich -format texmfscripts $(basename $(2))) \
419     $(foreach dir, $(subst :, , $(PATH)), \
420     $(dir)/$(2) $(dir)/$(basename $(2))) \
421 ) $(2))
422 export _LU_$(1)_DEFAULT
423 endif
424 $$$(eval $(call lu-setvar-global,$(1),$_LU_$(1)_DEFAULT))
425 endef
426
427 $(eval $(call _lu_which,GENSUBFIG,gensubfig.py))
428 $(eval $(call _lu_which,FIGDEPTH,figdepth.py))
429 $(eval $(call _lu_which,GENSUBSVG,gensubfig.py))
430 $(eval $(call _lu_which,SVGDEPTH,svgdepth.py))
431 $(eval $(call _lu_which,SVG2DEV,svg2dev.py))
432 $(eval $(call _lu_which,LATEXFILTER,latexfilter.py))
433
434 # Rules to use to check if the build document (dvi or pdf) is up-to-date
435 # This can be overruled per document manually and/or automatically
436 #REBUILD_RULES ?= latex texdepends bibtopic bibtopic_undefined_references
437 $(eval $(call lu-addtovar-global,REBUILD_RULES,latex texdepends))
438
439 # Default maximum recursion level
440 $(eval $(call lu-setvar-global,MAX_REC,6))
441
442 #####
443 #####
444 #####
445 #####
446 #####
447 ##### Flavors #####
448 #####
449 #####
450 #####
451 #####
452 #####
453 #####

```

```

454
455 define lu-create-texflavor # 1:name 2:tex_prog 3:file_ext
456   # 4:master_cible 5:fig_extention_to_clean
457   _LU_FLAVORS_DEFINED_TEX += $(1)
458   $(eval $(call lu-setvar-flavor,VARPROG,$(1),$(2)))
459   $(eval $(call lu-setvar-flavor,EXT,$(1),$(3)))
460   $(eval $(call lu-setvar-flavor,TARGETNAME,$(1),$(4)))
461   $(eval $(call lu-addtovar-flavor,CLEANFIGEXT,$(1),$(5)))
462   $(eval $(call lu-addtovar-flavor,CLEANSVGEXT,$(1),$(5)))
463 endef
464
465 define lu-create-dviflavor # 1:name 2:dvi_prog 3:file_ext
466   # 4:master_cible 5:tex_flavor_depend
467   $$$(eval $$$(call lu-define-flavor,$(5)))
468   _LU_FLAVORS_DEFINED_DVI += $(1)
469   $(eval $(call lu-setvar-flavor,VARPROG,$(1),$(2)))
470   $(eval $(call lu-setvar-flavor,EXT,$(1),$(3)))
471   $(eval $(call lu-setvar-flavor,TARGETNAME,$(1),$(4)))
472   $(eval $(call lu-setvar-flavor,DEPFLAVOR,$(1),$(5)))
473 endef
474
475 define lu-create-flavor # 1:name 2:type 3..7:options
476   $$$(if $$$(filter $(1),$_LU_FLAVORS_DEFINED)), \
477   $$$(call lu-show-flavors,Flavor $(1) already defined), \
478   $$$(call lu-show-flavors,Creating flavor $(1) ($(2))) \
479   $$$(eval $$$(call lu-create-$(2)flavor,$(1),$(3),$(4),$(5),$(6),$(7))))
480 endef
481
482 define lu-define-flavor # 1:name
483   $$$(eval $$$(call lu-define-flavor-$(1)))
484 endef
485
486 define lu-flavor-rules # 1:name
487   $$$(call lu-show-flavors,Defining rules for flavor $(1))
488   $$$(if $$$(call lu-getvalue-flavor,TARGETNAME,$(1)), \
489   $$$(call lu-getvalue-flavor,TARGETNAME,$(1)): \
490   $$$(call lu-getvalues-flavor,TARGETS,$(1)))
491   $$$(if $$$(call lu-getvalue-flavor,TARGETNAME,$(1)), \
492   .PHONY: $$$(call lu-getvalue-flavor,TARGETNAME,$(1)))
493 endef
494
495 define lu-define-flavor-DVI #
496   $$$(eval $$$(call lu-create-flavor,DVI,tex,LATEX,.dvi,dvi,\
497   .pstex_t .pstex))
498 endef
499
500 define lu-define-flavor-PDF #
501   $$$(eval $$$(call lu-create-flavor,PDF,tex,PDFLATEX,.pdf,pdf,\
502   .pdftex_t .$$$_LU_PDFTEX_EXT)))
503 endef
504
505 define lu-define-flavor-PS #
506   $$$(eval $$$(call lu-create-flavor,PS,dvi,DVIPS,.ps,ps,DVI))
507 endef
508
509 define lu-define-flavor-DVIPDF #
510   $$$(eval $$$(call lu-create-flavor,DVIPDF,dvi,DVIPDFM,.pdf,pdf,DVI))
511 endef

```

```

512
513 $(eval $(call lu-addtovar-global,FLAVORS,PDF PS))
514
515 #####
516 #####
517 #####
518 #####
519 #####
520 ##### Masters #####
521 #####
522 #####
523 #####
524 #####
525 #####
526 #####
527
528 define _lu-do-latex # 1:master 2:flavor 3:source.tex 4:ext(.dvi/.pdf)
529   exec 3>&1; \
530   run() { \
531     echo -n "Running:" 1>&3 ; \
532     for arg; do \
533       echo -n " '$$arg'" 1>&3 ; \
534     done ; echo 1>&3 ; \
535     "$$@" ; \
536   }; \
537   doit() { \
538     $(RM) -v "$(1)$(4)_FAILED" \
539     "$(1)$(4)_NEED_REBUILD" \
540     "$(1)$(4).mk" ;\
541     ( echo X | \
542     run $(call lu-call-prog-flavor,$(1),$(2)) \
543     --interaction errorstopmode \
544     --jobname "$(1)" \
545     '\RequirePackage[extension="$(4)"]{texdepends}\input'"{$(3)}" || \
546     touch "$(1)$(4)_FAILED" ; \
547     if grep -sq '^! LaTeX Error:' "$(1).log" ; then \
548       touch "$(1)$(4)_FAILED" ; \
549     fi \
550   ) | $(call lu-call-prog,LATEXFILTER,$(1),$(2)) ; \
551   NO_TEXDEPENDS_FILE=0 ;\
552   if [ ! -f "$(1)$(4).mk" ]; then \
553     NO_TEXDEPENDS_FILE=1 ;\
554   fi ;\
555   sed -e 's,\\openout[0-9]* = '\\(.*)',,TD_$(1)$(4)_OUTPUTS += \\1,p;d' \
556   "$(1).log" >> "$(1)$(4).mk" ;\
557   if [ -f "$(1)$(4)_FAILED" ]; then \
558     echo "*****" ;\
559     echo "Building $(1)$(4) fails" ;\
560     echo "*****" ;\
561     echo "Here are the last lines of the log file" ;\
562     echo "If this is not enough, try to" ;\
563     echo "call 'make' with 'VERB=verbose' option" ;\
564     echo "*****" ;\
565     echo "==> Last lines in $(1).log <==" ; \
566     sed -e '/^[?] X$$/,,$$d' \
567     -e '/^Here is how much of TeX''''s memory you used:$$/,,$$d' \
568     < "$(1).log" | tail -n 20; \
569   return 1; \

```

```

570 fi; \
571 if [ "$$NO_TEXDEPENDS_FILE" = 1 ]; then \
572 echo "*****" ;\
573 echo "texdepends does not seems be loaded" ;\
574 echo "Either your (La)TeX installation is wrong or you found a bug." ;\
575 echo "If so, please, report it (with the result of shell command 'kpsepath tex')";\
576 echo "Aborting compilation" ;\
577 echo "*****" ;\
578 touch "$ (1)$ (4)_FAILED" ; \
579 return 1 ;\
580 fi ;\
581 }; doit
582 endif
583
584 .PHONY: clean-build-fig
585
586 #####
587 define lu-master-texflavor-index-vars # MASTER FLAVOR TYPE INDEX ext(.dvi/.pdf)
588   $$ (call lu-show-rules,Setting flavor index vars for $ (1)/$ (2)/[$ (3)]$ (4))
589   $$ (eval $$ (call lu-addtovar,DEPENDS,$ (1),$ (2), \
590     $$ (call lu-getvalue-index,TARGET,$ (1),$ (2),$ (3),$ (4))))
591   $$ (eval $$ (call lu-addtovar,WATCHFILES,$ (1),$ (2), \
592     $$ (call lu-getvalue-index,SRC,$ (1),$ (2),$ (3),$ (4))))
593   endif #####
594 define lu-master-texflavor-index-rules # MASTER FLAVOR TYPE INDEX ext(.dvi/.pdf)
595   $$ (call lu-show-rules,Setting flavor index rules for $ (1)/$ (2)/[$ (3)]$ (4))
596   $$ (if $$ (_LU_DEF_IND_$$ (call lu-getvalue-index,TARGET,$ (1),$ (2),$ (3),$ (4))), \
597     $$ (call lu-show-rules,=> Skipping: already defined in fla-
598       vor $$ (_LU_DEF_IND_$$ (call lu-getvalue-index,TARGET,$ (1),$ (2),$ (3),$ (4)))), \
599     $$ (eval $$ (call _lu-master-texflavor-index-rules\
600       , $ (1),$ (2),$ (3),$ (4),$ (5)),$$ (call lu-getvalue-index,TARGET,$ (1),$ (2),$ (3),$ (4))))
601   endif
602 define _lu-master-texflavor-index-rules # MASTER FLAVOR TYPE INDEX ext TARGET
603   $ (6): \
604     $$ (call lu-getvalue-index,SRC,$ (1),$ (2),$ (3),$ (4)) \
605     $$ (wildcard $$ (call lu-getvalue-index,STYLE,$ (1),$ (2),$ (3),$ (4)))
606   $$ (COMMON_PREFIX)$$ (call lu-call-prog-index,MAKEINDEX,$ (1),$ (2),$ (3),$ (4)) \
607   $$ (addprefix -s ,$$ (call lu-getvalue-index,STYLE,$ (1),$ (2),$ (3),$ (4))) \
608   -o $$@ $$<
609   _LU_DEF_IND_ $ (6)=$ (2)
610   clean::
611   $$ (call lu-clean,$$ (call lu-getvalue-index,TARGET,$ (1),$ (2),$ (3),$ (4)) \
612     $$ (addsuffix .ilg,$$ (basename \
613       $$ (call lu-getvalue-index,SRC,$ (1),$ (2),$ (3),$ (4))))
614   endif #####
615 define lu-master-texflavor-index # MASTER FLAVOR INDEX ext(.dvi/.pdf)
616   $$ (eval $$ (call lu-master-texflavor-index-vars,$ (1),$ (2),$ (3),$ (4)))
617   $$ (eval $$ (call lu-master-texflavor-index-rules,$ (1),$ (2),$ (3),$ (4)))
618   endif
619 #####
620 #####
621 define lu-master-texflavor-vars # MASTER FLAVOR ext(.dvi/.pdf)
622   $$ (call lu-show-rules,Setting flavor vars for $ (1)/$ (2))
623   -include $ (1)$ (3).mk
624   $$ (eval $$ (call lu-addtovar,DEPENDS,$ (1),$ (2), \
625     $$ (call lu-getvalues,FIGURES,$ (1),$ (2)) \
626     $$ (call lu-getvalues,BIBFILES,$ (1),$ (2)) \

```

```

627  $$$(wildcard $$$(call lu-getvalues,INPUTS,$(1),$(2))) \
628  $$$(wildcard $$$(call lu-getvalues,BIBSTYLES,$(1),$(2))) \
629      $$$(call lu-getvalues,BBLFILES,$(1),$(2))\
630  ))
631
632  $$$(eval $$$(call lu-addtovar-flavor,TARGETS,$(2),$(1)$(3)))
633
634  $$$(eval $$$(call lu-addtovar,GPATH,$(1),$(2), \
635      $$$(subst },,$$(subst {,,,$$(subst }{, \
636      $$$(call lu-getvalue,GRAPHICSPATH,$(1),$(2))))))
637
638  $$$(if $$$(sort $$$(call lu-getvalues,SUBFIGS,$(1),$(2))), \
639  $$$(eval include $$$(addsuffix .mk,$$(sort \
640  $$$(call lu-getvalues,SUBFIGS,$(1),$(2))))))
641
642  $$$(eval $$$(call lu-addtovar,WATCHFILES,$(1),$(2), \
643  $$$(filter %.aux, $$$(call lu-getvalues,OUTPUTS,$(1),$(2))))))
644
645  $$$(foreach type,$$(call lu-getvalues,INDEXES,$(1),$(2)), \
646      $$$(foreach index,$$(call lu-getvalues,INDEXES_$$$(type),$(1),$(2)), \
647          $$$(eval $$$(call lu-master-texflavor-index-vars,$(1),$(2),$$$(type),$$$(index),$(3))))
648  endif #####
649  define lu-master-texflavor-rules # MASTER FLAVOR ext(.dvi/.pdf)
650  $$$(call lu-show-rules,Defining flavor rules for $(1)/$(2))
651  $$$(call lu-getvalues,BBLFILES,$(1),$(2)): \
652  $$$(sort          $$$(call lu-getvalues,BIBFILES,$(1),$(2)) \
653  $$$(wildcard $$$(call lu-getvalues,BIBSTYLES,$(1),$(2))))
654  $(1)$(3): %$(3): \
655      $$$(call lu-getvalues,DEPENDS,$(1),$(2)) \
656      $$$(call lu-getvalues,REQUIRED,$(1),$(2)) \
657      $$$(if $$$(wildcard $(1)$(3)_FAILED),LU_FORCE,) \
658      $$$(if $$$(wildcard $(1)$(3)_NEED_REBUILD),LU_FORCE,) \
659      $$$(if $$$(wildcard $(1)$(3)_NEED_REBUILD_IN_PROGRESS),LU_FORCE,)
660  $$$(if $$$(filter-out $$$(LU_REC_LEVEL),$$$(call lu-getvalue,MAX_REC,$(1),$(2))),, \
661  $$$(warning *****) \
662  $$$(warning *****) \
663  $$$(warning *****) \
664  $$$(warning Stopping generation of $$@) \
665  $$$(warning I got max recursion level $$$(LU_$(1)_$(2)_MAX_REC)) \
666  $$$(warning Set LU_$(1)_$(2)_MAX_REC, LU_MAX_REC_$(1) or LU_MAX_REC if you need it) \
667  $$$(warning *****) \
668  $$$(warning *****) \
669  $$$(warning *****) \
670  $$$(error Aborting generation of $$@))
671  $$$(MAKE) LU_REC_MASTER="$(1)" LU_REC_FLAVOR="$(2)" LU_REC_TARGET="$$@"\
672  LU_WATCH_FILES_SAVE
673  $$$(COMMON_PREFIX)$$$(call _lu-do-latex\
674  ,$(1),$(2),$$$(call lu-getvalue-master,MAIN,$(1)),$(3))
675  $$$(MAKE) LU_REC_MASTER="$(1)" LU_REC_FLAVOR="$(2)" LU_REC_TARGET="$$@"\
676  LU_WATCH_FILES_RESTORE
677  $$$(MAKE) LU_REC_MASTER="$(1)" LU_REC_FLAVOR="$(2)" LU_REC_TARGET="$$@"\
678  $(1)$(3)_NEED_REBUILD
679  ifneq ($(LU_REC_TARGET),)
680  $(1)$(3)_NEED_REBUILD_IN_PROGRESS:
681  $$$(COMMON_HIDE)touch $(1)$(3)_NEED_REBUILD_IN_PROGRESS
682  $$$(addprefix LU_rebuild_,$$$(call lu-getvalues,REBUILD_RULES,$(1),$(2))): \
683  $(1)$(3)_NEED_REBUILD_IN_PROGRESS
684  .PHONY: $(1)$(3)_NEED_REBUILD

```

```

685 $(1)$(3)_NEED_REBUILD: \
686     $(1)$(3)_NEED_REBUILD_IN_PROGRESS \
687     $$ (addprefix LU_rebuild_, $(call lu-getvalues, REBUILD_RULES, $(1), $(2)))
688 $$ (COMMON_HIDE) $(RM) $(1)$(3)_NEED_REBUILD_IN_PROGRESS
689 $$ (COMMON_HIDE) if [ -f "$(1)$(3)_NEED_REBUILD" ]; then \
690 echo "*****" ; \
691 echo "***** New build needed *****" ; \
692 echo "*****" ; \
693 cat "$(1)$(3)_NEED_REBUILD" ; \
694 echo "*****" ; \
695 fi
696 $$ (MAKE) LU_REC_LEVEL = $(shell expr $(LU_REC_LEVEL) + 1) \
697 $$ (LU_REC_TARGET)
698 endif
699 clean-build-fig::
700 $$ (call lu-clean, $(foreach fig, \
701     $(basename $(wildcard $(filter %.fig, \
702     $(call lu-getvalues, FIGURES, $(1), $(2))))), \
703     $(addprefix $(fig), $(call lu-getvalues-flavor, CLEANFIGEXT, $(2)))) \
704     $(call lu-clean, $(foreach svg, \
705     $(basename $(wildcard $(filter %.svg, \
706     $(call lu-getvalues, FIGURES, $(1), $(2))))), \
707     $(addprefix $(svg), $(call lu-getvalues-flavor, CLEANSVGEXT, $(2)))) \
708 clean:: clean-build-fig
709 $$ (call lu-clean, $(call lu-getvalues, OUTPUTS, $(1), $(2)) \
710     $(call lu-getvalues, BBLFILES, $(1), $(2)) \
711     $(addsuffix .mk, $(call lu-getvalues, SUBFIGS, $(1), $(2)) \
712     $(patsubst %.bbl, %.blg, $(call lu-getvalues, BBLFILES, $(1), $(2)))) \
713     $(call lu-clean, $(wildcard $(1).log))
714 distclean::
715 $$ (call lu-clean, $(wildcard $(1)$(3) $(1)$(3)_FAILED \
716     $(1)$(3)_NEED_REBUILD $(1)$(3)_NEED_REBUILD_IN_PROGRESS))
717 $$ (foreach type, $(call lu-getvalues, INDEXES, $(1), $(2)), \
718     $(foreach index, $(call lu-getvalues, INDEXES, $(type), $(1), $(2)), \
719     $(eval $(call lu-master-texflavor-index-rules, $(1), $(2), $(type), $(index), $(3)))) \
720     endif #####
721 define lu-master-texflavor # MASTER FLAVOR ext(.dvi/.pdf)
722     $(eval $(call lu-master-texflavor-vars, $(1), $(2), $(3)))
723     $(eval $(call lu-master-texflavor-rules, $(1), $(2), $(3)))
724 endef
725 #####
726
727 #####
728 define lu-master-dviflavor-vars # MASTER FLAVOR ext(.ps)
729     $(call lu-show-rules, Setting flavor vars for \
730     $(1)/$(2)/$(call lu-getvalue-flavor, DEPFLAVOR, $(2)))
731     # $(eval $(call lu-addvar, VARPROG, $(1), $(2)))
732     # $(eval $(call lu-addvar, $(call lu-getvalue, VARPROG, $(1), $(2)), $(1), $(2)))
733     $(eval $(call lu-addto-var-flavor, TARGETS, $(2), $(1)$(3)))
734 endef #####
735 define lu-master-dviflavor-rules # MASTER FLAVOR ext(.ps)
736     $(call lu-show-rules, Defining flavor rules for \
737     $(1)/$(2)/$(call lu-getvalue-flavor, DEPFLAVOR, $(2)))
738     $(1)$(3): %$(3): %$(call lu-getvalue-flavor, EXT, $(call lu-getvalue-
739     flavor, DEPFLAVOR, $(2)))
739     $(call lu-call-prog-flavor, $(1), $(2)) -o $$@ $$<
740 distclean::
741 $$ (call lu-clean, $(wildcard $(1)$(3)))

```

```

742 endif #####
743 define lu-master-dviflavor # MASTER FLAVOR ext(.ps)
744   $(eval $(call lu-master-dviflavor-vars,$(1),$(2),$(3)))
745   $(eval $(call lu-master-dviflavor-rules,$(1),$(2),$(3)))
746 endif
747 #####
748
749 #####
750 define lu-master-vars # MASTER
751   $(call lu-show-rules,Setting vars for $(1))
752   $(eval $(call lu-setvar-master,MAIN,$(1),$(1).tex))
753   $(eval $(call lu-addtovar-master,DEPENDS,$(1),\
754   $(call lu-getvalue-master,MAIN,$(1))))
755   _LU_$(1)_DVI_FLAVORS=$(filter $(call lu-getvalue-master,FLAVORS,$(1)),\
756   $(sort $(call lu-getvalues-master,FLAVORS,$(1))))
757   _LU_$(1)_TEX_FLAVORS=$(filter $(call lu-getvalue-master,FLAVORS,$(1)),\
758   $(sort $(call lu-getvalues-master,FLAVORS,$(1)) \
759   $(LU_REC_FLAVOR) \
760   $(foreach dvi,$(call lu-getvalues-master,FLAVORS,$(1)), \
761   $(call lu-getvalue-flavor,DEPFLAVOR,$(dvi)))))
762   $(foreach flav,$(_LU_$(1)_TEX_FLAVORS), $(eval $(call \
763   lu-master-texflavor-vars,$(1),$(flav),$(call lu-getvalue-flavor,EXT,$(flav))))
764   $(foreach flav,$(_LU_$(1)_DVI_FLAVORS), $(eval $(call \
765   lu-master-dviflavor-vars,$(1),$(flav),$(call lu-getvalue-flavor,EXT,$(flav))))
766   endif #####
767   define lu-master-rules # MASTER
768     $(call lu-show-rules,Defining rules for $(1))
769     $(foreach flav,$(_LU_$(1)_TEX_FLAVORS), $(eval $(call \
770     lu-master-texflavor-rules,$(1),$(flav),$(call lu-getvalue-flavor,EXT,$(flav))))
771     $(foreach flav,$(_LU_$(1)_DVI_FLAVORS), $(eval $(call \
772     lu-master-dviflavor-rules,$(1),$(flav),$(call lu-getvalue-flavor,EXT,$(flav))))
773   endif #####
774   define lu-master # MASTER
775     $(eval $(call lu-master-vars,$(1)))
776     $(eval $(call lu-master-rules,$(1)))
777   endif
778   #####
779
780   $(warning $(call LU_RULES,example))
781   $(eval $(call lu-addtovar-global,MASTERS,\
782   $(shell grep -l '\documentclass' *.tex 2>/dev/null | sed -e 's/\.tex$$$//'))
783   ifneq ($(LU_REC_TARGET),)
784     _LU_DEF_MASTERS = $(LU_REC_MASTER)
785     _LU_DEF_FLAVORS = $(LU_REC_FLAVOR) $(FLAV_DEPFLAVOR_$(LU_REC_FLAVOR))
786   else
787     _LU_DEF_MASTERS = $(call lu-getvalues-global,MASTERS)
788     _LU_DEF_FLAVORS = $(sort $(foreach master,$(_LU_DEF_MASTERS),\
789     $(call lu-getvalues-master,FLAVORS,$(master))))
790   endif
791
792   $(foreach flav, $(_LU_DEF_FLAVORS), $(eval $(call lu-define-flavor,$(flav))))
793   $(foreach master, $(_LU_DEF_MASTERS), $(eval $(call lu-master-vars,$(master))))
794   $(foreach flav, $(_LU_FLAVORS_DEFINED), $(eval $(call lu-flavor-rules,$(flav))))
795   $(foreach master, $(_LU_DEF_MASTERS), $(eval $(call lu-master-rules,$(master))))
796
797   #####
798   # Gestion des subfigs
799

```

```

800 %<<MAKEFILE
801 %.subfig.mk: %.subfig %.fig
802 $(COMMON_PREFIX)$(call lu-call-prog,GENSUBFIG) \
803 -p '$$(COMMON_PREFIX)$(call lu-call-prog,FIGDEPTH) < $$< > $$$@' \
804 -s $*.subfig $*.fig < $^ > $@
805 %MAKEFILE
806
807 %<<MAKEFILE
808 %.subfig.mk: %.subfig %.svg
809 $(COMMON_PREFIX)$(call lu-call-prog,GENSUBSVG) \
810 -p '$$(COMMON_PREFIX)$(call lu-call-prog,SVGDEPTH) < $$< > $$$@' \
811 -s $*.subfig $*.svg < $^ > $@
812 %MAKEFILE
813
814 clean::
815 $(call lu-clean,$(FIGS2CREATE_LIST))
816 $(call lu-clean,$(FIGS2CREATE_LIST:%.fig=%.pstex))
817 $(call lu-clean,$(FIGS2CREATE_LIST:%.fig=%.pstex_t))
818 $(call lu-clean,$(FIGS2CREATE_LIST:%.fig=%.$(_LU_PDFTEX_EXT)))
819 $(call lu-clean,$(FIGS2CREATE_LIST:%.fig=%.pdftex_t))
820 $(call lu-clean,$(FIGS2CREATE_LIST:%.svg=%.pstex))
821 $(call lu-clean,$(FIGS2CREATE_LIST:%.svg=%.pstex_t))
822 $(call lu-clean,$(FIGS2CREATE_LIST:%.svg=%.$(_LU_PDFTEX_EXT)))
823 $(call lu-clean,$(FIGS2CREATE_LIST:%.svg=%.pdftex_t))
824
825 .PHONY: LU_FORCE clean distclean
826 LU_FORCE:
827 @echo "Previous compilation failed. Rerun needed"
828
829 #$(warning $(MAKEFILE))
830
831 distclean:: clean
832
833 %<<MAKEFILE
834 %.eps: %.fig
835 $(COMMON_PREFIX)$(call lu-call-prog,FIG2DEV) -L eps $< $@
836
837 %.pdf: %.fig
838 $(COMMON_PREFIX)$(call lu-call-prog,FIG2DEV) -L pdf $< $@
839
840 %.pstex: %.fig
841 $(COMMON_PREFIX)$(call lu-call-prog,FIG2DEV) -L pstex $< $@
842
843 %.pstex: %.svg
844 $(COMMON_PREFIX)$(call lu-call-prog,SVG2DEV) -L pstex $< $@
845
846
847 .PRECIOUS: %.pstex
848 %.pstex_t: %.fig %.pstex
849 $(COMMON_PREFIX)$(call lu-call-prog,FIG2DEV) -L pstex_t -p $*.pstex $< $@
850
851 %.pstex_t: %.svg %.pstex
852 $(COMMON_PREFIX)$(call lu-call-prog,SVG2DEV) -L pstex_t -p $*.pstex $< $@
853
854
855 %.$(_LU_PDFTEX_EXT): %.fig
856 $(COMMON_PREFIX)$(call lu-call-prog,FIG2DEV) -L pdftex $< $@
857

```

```

858 %.$(_LU_PDFTEX_EXT): %.svg
859 $(COMMON_PREFIX)$(call lu-call-prog,SVG2DEV) -L pdftex $< $@
860
861 .PRECIOUS: %.$(_LU_PDFTEX_EXT)
862 %.pdftex_t: %.fig %.$(_LU_PDFTEX_EXT)
863 $(COMMON_PREFIX)$(call lu-call-prog,FIG2DEV) -L pdftex_t -p $*.$(_LU_PDFTEX_EXT) $< $@
864
865 %.pdftex_t: %.svg %.$(_LU_PDFTEX_EXT)
866 $(COMMON_PREFIX)$(call lu-call-prog,SVG2DEV) -L pdftex_t -p $*.$(_LU_PDFTEX_EXT) $< $@
867
868 %.pdf: %.eps
869 $(COMMON_PREFIX)$(call lu-call-prog,EPSTOPDF) --filter < $< > $@
870 %MAKEFILE
871
872 #####
873 # Les flavors
874 LU_REC_LEVEL ?= 1
875 ifneq ($(LU_REC_TARGET),)
876 export LU_REC_FLAVOR
877 export LU_REC_MASTER
878 export LU_REC_TARGET
879 export LU_REC_LEVEL
880 LU_REC_LOGFILE=$(LU_REC_MASTER).log
881 LU_REC_GENFILE=$(LU_REC_MASTER)$(call lu-getvalue-flavor,EXT,$(LU_REC_FLAVOR))
882
883 lu-rebuild-head=$(info *** Checking rebuild with rule '$(subst LU_rebuild_,,$@)')
884 lu-rebuild-needed=echo $(1) >> "$(LU_REC_GENFILE)_NEED_REBUILD" ;
885
886 .PHONY: $(addprefix LU_rebuild_,latex texdepends bibtex)
887 LU_rebuild_latex:
888 $(call lu-rebuild-head)
889 $(COMMON_HIDE)if grep -sq 'Rerun to get'\
890 "$(LU_REC_LOGFILE)" ; then \
891 $(call lu-rebuild-needed\
892 "$@: new run needed (LaTeX message 'Rerun to get...')") \
893 fi
894
895 LU_rebuild_texdepends:
896 $(call lu-rebuild-head)
897 $(COMMON_HIDE)if grep -sq '^Package texdepends Warning: .* Check dependen-
cies again.$$\
898 "$(LU_REC_LOGFILE)" ; then \
899 $(call lu-rebuild-needed,"$@: new depends required") \
900 fi
901
902 LU_rebuild_bibtopic:
903 $(call lu-rebuild-head)
904  $\backslash$ /makefile

This part is not needed: already checked with the lu_rebuild_latex rule
905  $\backslash$ *notused)
906 $(COMMON_HIDE)if grep -sq 'Rerun to get indentation of bibitems right'\
907 "$(LU_REC_LOGFILE)" ; then \
908 $(call lu-rebuild-needed,"$@: new run needed") \
909 fi
910 $(COMMON_HIDE)if grep -sq 'Rerun to get cross-references right'\
911 "$(LU_REC_LOGFILE)" ; then \
912 $(call lu-rebuild-needed,"$@: new run needed") \
913 fi

```

```

914  $\langle$ /notused $\rangle$ 
915  $\langle$ *makefile $\rangle$ 
916 $(COMMON_HIDE)sed -e '/^Package bibtopic Warning: Please (re)run Bib-
    TeX on the file(s):$$/,/^ (bibtopic) *and after that rerun La-
    TeX./{s/^ (bibtopic) *\[^\ ]*\)$$/\1/p};d' \
917 "$(LU_REC_LOGFILE)" | while read file ; do \
918 touch $$file.aux ; \
919 $(call lu-rebuild-needed,"bibtopic: $$file.bbl outdated") \
920 done
921
922 LU_rebuild_bibtopic_undefined_references:
923 $(call lu-rebuild-head)
924 $(COMMON_HIDE)if grep -sq 'There were undefined references'\
925 "$(MASTER_$(LU_REC_MASTER)).log" ; then \
926 $(call lu-rebuild-needed,"$@: new run needed") \
927 fi
928
929 .PHONY: LU_WATCH_FILES_SAVE LU_WATCH_FILES_RESTORE
930 LU_WATCH_FILES_SAVE:
931 $(COMMON_HIDE)$(foreach file, $(sort \
932 $(call lu-getvalues,WATCHFILES,$(LU_REC_MASTER),$(LU_REC_FLAVOR))), \
933     $(call lu-save-file,$(file),$(file).orig);)
934
935 LU_WATCH_FILES_RESTORE:
936 $(COMMON_HIDE)$(foreach file, $(sort \
937 $(call lu-getvalues,WATCHFILES,$(LU_REC_MASTER),$(LU_REC_FLAVOR))), \
938     $(call lu-cmprestaure-file,"$(file)","$(file).orig",\
939 echo "New $(file) file" >> $(LU_REC_GENFILE)_NEED_REBUILD;\
940 );)
941
942 endif
943
944  $\%<<$ MAKEFILE
945  $\%.$ bbl:  $\%.$ aux
946 $(COMMON_PREFIX)$(call lu-call-prog,BIBTEX)  $\%*$ 
947  $\%$ MAKEFILE
948  $\langle$ /makefile $\rangle$ 

```

5.2 LaTeX.mk.conf

```

949  $\langle$ *makefile-config $\rangle$ 
950 # Choose between GNU/BSD utilities (cp, rm, ...)
951 # LU_UTILS = GNU
952 # LU_UTILS = BSD
953  $\langle$ /makefile-config $\rangle$ 

```

5.3 figdepth

```

954  $\langle$ *figdepth $\rangle$ 
955 #!/usr/bin/env python
956 #coding=utf8
957
958 """
959
960 stdin : the original xfig file
961 stdout : the output xfig file
962 args : all depths we want to keep
963
964 """
965

```

```

966 import optparse
967 import os.path
968 import sys
969
970 def main():
971     parser = optparse.OptionParser()
972     (options, args) = parser.parse_args()
973
974     depths_to_keep = set()
975     for arg in args:
976         depths_to_keep.add(arg)
977
978     comment = ''
979     display = True
980     def show(depth, line):
981         if depth in depths_to_keep:
982             print comment+line,
983             return True
984         else:
985             return False
986     for line in sys.stdin:
987         if line[0] == '#':
988             comment += line
989             continue
990         if line[0] in "\t ":
991             if display:
992                 print line
993         else:
994             Fld = line.split(' ', 9999)
995             if not Fld[0] or Fld[0] not in ('1', '2', '3', '4', '5'):
996                 print comment+line
997                 display = True
998             elif Fld[0] == '4':
999                 display = show(Fld[3], line)
1000             else:
1001                 display = show(Fld[6], line)
1002             comment = ''
1003
1004 if __name__ == "__main__":
1005     main()
1006 <\/figdepth>

```

5.4 gensubfig

```

1007 (*gensubfig)
1008 #!/usr/bin/env python
1009 #coding=utf8
1010
1011 """
1012
1013 Arguments passes :
1014 - fichier image (image.fig ou image.svg)
1015 - -s fichier subfig (image.subfig)
1016 - -p chemin du script pour generer les sous-images (svgdepth.py ou figdepth.py)
1017
1018 Sortie standard :
1019 - makefile pour creer les sous-images (au format .fig ou .svg), et pour les sup-
  primer
1020

```

```

1021 """
1022
1023 from optparse import OptionParser
1024 import os.path
1025
1026 def main():
1027     parser = OptionParser(usage='usage: %prog [options] svg file', descrip-
1028         tion='Creates a\
1029 Makefile generating subfigures using figdepth.py or svgdepth.py')
1029     parser.add_option("-s", "--subfig", dest="subfig", help="subfig file")
1030     parser.add_option("-p", "--depth", dest="depth", help="full path of depth script")
1031     (options, args) = parser.parse_args()
1032     if len(args) < 1:
1033         parser.error("incorrect number of arguments")
1034     if not options.subfig:
1035         parser.error("no subfig file specified")
1036     if not options.depth:
1037         parser.error("no depth script specified")
1038
1039     (root, ext) = os.path.splitext(args[0])
1040     sf_name = options.subfig
1041     ds_name = options.depth
1042     varname = '%s_FIGS' % root.upper()
1043
1044     subfigs = []
1045     for line in open(options.subfig, 'r'):
1046         t = line.find('#') # looking for comments
1047         if t > -1: line = line[0:t] # remove comments...
1048         line = line.strip() #remove blank chars
1049         if line == '': continue
1050         subfigs.append(line)
1051
1052     count = 1
1053     for subfig in subfigs:
1054         print "%s_%d%s: %s%s %s" % (root, count, ext, root, ext, sf_name)
1055         print "\t%s %s" % (ds_name, subfig)
1056         print ""
1057         count += 1
1058     print "%s := $(foreach n, " % varname,
1059     count = 1
1060     for subfig in subfigs:
1061         print '%d ' % count,
1062         count += 1
1063     print ", %s_$(n)%s)" % (root, ext)
1064     print "FILES_TO_DISTCLEAN += $(%s)" % varname
1065     print "FIGS2CREATE_LIST += $(%s)" % varname
1066     print "$(TEMPORAIRE): $(%s)" % varname
1067     print "$(TEMPORAIRE): $(%s)" % varname
1068
1069 if __name__ == "__main__":
1070     main()
1071 /gensubfig

```

5.5 svg2dev

```

1072 (*svg2dev)
1073 #!/usr/bin/env python
1074 #coding=utf8
1075

```

```

1076 from optparse import OptionParser
1077 import shutil
1078 import subprocess
1079
1080
1081 svg2eps = 'inkscape %s -z -C --export-eps=%s --export-latex'
1082 svg2pdf = 'inkscape %s -z -C --export-pdf=%s --export-latex'
1083
1084
1085 def create_image(input_filename, output_filename, mode):
1086     subprocess.Popen(mode % (input_filename, output_filename),
1087         stdout=subprocess.PIPE, shell=True).communicate()[0]
1088     n1 = output_filename + '_tex'
1089     n2 = output_filename + '_t'
1090     shutil.move(n1, n2)
1091
1092
1093 def main():
1094     parser = OptionParser()
1095     parser.add_option("-L", "--format", dest="outputFormat",
1096         metavar="FORMAT", help="output format", default="spstex")
1097     parser.add_option("-p", "--portrait", dest="portrait", help="dummy arg")
1098     (options, args) = parser.parse_args()
1099     if len(args) != 2: return
1100     (input_filename, output_filename) = args
1101     fmt = options.outputFormat
1102     portrait = options.portrait
1103
1104     if fmt == 'eps':
1105         create_image(input_filename, output_filename, svg2eps)
1106     elif fmt == 'spstex' or fmt == 'pstex':
1107         create_image(input_filename, output_filename, svg2eps)
1108     elif fmt == 'spstex_t' or fmt == 'pstex_t':
1109         pass
1110     elif fmt == 'spdfTeX' or fmt == 'pdfTeX':
1111         create_image(input_filename, output_filename, svg2pdf)
1112     elif fmt == 'spdfTeX_t' or fmt == 'pdfTeX_t':
1113         pass
1114
1115
1116 if __name__ == "__main__":
1117     main()
1118
1119 </svg2dev>

```

5.6 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.

```

1120 (*latexfilter)
1121 #!/usr/bin/env python
1122 #coding=utf8
1123
1124 """
1125
1126 stdin : the original xfig file
1127 stdout : the output xfig file
1128 args : all depths we want to keep
1129

```

```

1130 """
1131
1132 from __future__ import print_function
1133 import optparse
1134 import os.path
1135 import re
1136 import sys
1137
1138 def main():
1139     parser = optparse.OptionParser()
1140     (options, args) = parser.parse_args()
1141
1142     display = 0
1143     in_display = 0
1144     start_line = ''
1145     warnerror_re = re.compile(r"^(LaTeX|Package|Class)(.*)? (Warning:|Error:)")
1146     fullbox_re = re.compile(r"^(Underfull|Overfull) \\[hv]box")
1147     accu = ''
1148     for line in sys.stdin:
1149         if display > 0:
1150             display -= 1
1151             if line[0:4].lower() in ('info', 'warn') or line[0:5].lower() == 'error':
1152                 display = 0
1153             line_groups = warnerror_re.match(line)
1154             if line_groups:
1155                 start_line = line_groups.group(3)
1156                 if not start_line:
1157                     start_line = ''
1158                 if line_groups.group(2):
1159                     start_line = "(" + start_line + ")"
1160             display = 1
1161             in_display = 1
1162         elif (start_line != '') and (line[0:len(start_line)] == start_line):
1163             display = 1
1164         elif line == "\n":
1165             in_display = 0
1166         elif line[0:4] == 'Chap':
1167             display = 1
1168         elif fullbox_re.match(line):
1169             display = 2
1170         if display:
1171             print(accu, end="")
1172             accu = line
1173         elif in_display:
1174             print(accu[0:-1], end="")
1175             accu = line
1176
1177 if __name__ == "__main__":
1178     main()
1179
1180 </latexfilter>

```

5.7 svgdepth

```

1181 <svgdepth>
1182 #!/usr/bin/env python
1183 #coding=utf8
1184

```

```

1185 import sys
1186 import xml.parsers.expat
1187
1188
1189 layers = []
1190 for arg in sys.argv:
1191     layers.append(arg)
1192
1193 parser = xml.parsers.expat.ParserCreate()
1194 class XmlParser(object):
1195     def __init__(self, layers):
1196         self.state_stack = [True]
1197         self.last_state = True
1198         self.layers = layers
1199     def XmlDeclHandler(self, version, encoding, standalone):
1200         sys.stdout.write("<?xml version='%s' encoding='%s'?>\n" % (version, encoding))
1201     def StartDoctypeDeclHandler(self, doctypeName, systemId, publi-
        cId, has_internal_subset):
1202         if publicId != None: sys.stdout.write("<!DOCTYPE %s PUBLIC \"%s\" \"%s\">\n" %\
1203             (doctypeName, publicId, systemId))
1204         else: sys.stdout.write("<!DOCTYPE %s \"%s\">\n" % (doctypeName, systemId))
1205     def StartElementHandler(self, name, attributes):
1206         if name.lower() == 'g':
1207             r = self.last_state and ('id' not in attributes or \
1208                 attributes['id'] in self.layers)
1209             self.last_state = r
1210             self.state_stack.append(r)
1211             if not self.last_state: return
1212             s = ""
1213             for k, v in attributes.items(): s += ' %s="%s"' % (k, v)
1214             sys.stdout.write("<%s%s>" % (name, s))
1215     def EndElementHandler(self, name):
1216         r = self.last_state
1217         if name.lower() == 'g':
1218             self.state_stack = self.state_stack[0:-1]
1219             self.last_state = self.state_stack[-1]
1220             if not r: return
1221             sys.stdout.write("</%s>" % (name))
1222     def CharacterDataHandler(self, data):
1223         if not self.last_state: return
1224         sys.stdout.write(data)
1225
1226 my_parser = XmlParser(layers)
1227
1228 parser.XmlDeclHandler = my_parser.XmlDeclHandler
1229 parser.StartDoctypeDeclHandler = my_parser.StartDoctypeDeclHandler
1230 parser.StartElementHandler = my_parser.StartElementHandler
1231 parser.EndElementHandler = my_parser.EndElementHandler
1232 parser.CharacterDataHandler = my_parser.CharacterDataHandler
1233
1234 for line in sys.stdin:
1235     parser.Parse(line, False)
1236 parser.Parse('', True)
1237
1238
1239 </svgdepth>

```

Index

Numbers written in *italic* refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in *roman* refer to the code lines where the entry is used.

Symbols

\ " 1202, 1204

Change History

v2.0.0	v2.1.1
General: First autocommented version ... 1	General: Improve error message 1
v2.1.0	v2.1.2
General: That's the question 1	General: Switch from perl to python 1