

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

Vincent DANJEAN

Arnaud LEGRAND

2017/01/08

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	3
	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	6
	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	figdepth	27
5.3	gensubfig	28
5.4	svg2dev	29
5.5	latexfilter	30
5.6	svgdepth	31

*This file has version number v2.2.3, last revised 2017/01/08.

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 locally in the current directory. You can then commit these files into your control version system so all co-authors will be able to use \LaTeX -Make without installing it. However, note that:

- you won't benefit 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)

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.

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 L^AT_EX main source for a document

master_MAIN

Example: `figlatex_MAIN=figlatex.dtx`

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

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

Which flavors must be compiled

LU_FLAVORS

Example: `LU_FLAVORS=DVI DVIPDF`

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

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

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

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

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

Table 1: Predefined flavors

Which programs are called and with which options

prog/prog_OPTIONS

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

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

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

Per target programs and options

master_prog/master_prog_OPTIONS

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

Note that, if defined, *master_prog* will **replace** *prog* whereas *master_prog_OPTIONS* will **be added to** *prog_OPTIONS* (see section 3.2 for more details).

Global and per target dependencies

DEPENDS/master_DEPENDS

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

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

3 Reference manual

3.1 Flavors

3.1.1 What is a flavor ?

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

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

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

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

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

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

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

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

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

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

3.1.2 Defining a new flavor

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

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

For example, `LaTeX.mk` already defines:

DVI flavor

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

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

PDF flavor

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

LuaLaTeX flavor

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

PS flavor

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

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

DVIPDF flavor

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

3.2 Variables

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

3.2.1 Two kind of variables

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

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

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

1	<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 texdepends 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 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	<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 **LaTeX.mk** and the **texdepends** L^AT_EX package.

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

```

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

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

```

3.2.2 List of used variables

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

Name	Kind	Context of use	Description
MASTERS	ADD	Global	List of documents to compile. These values will be used as jobname. Default: basename of *.tex files containing the \documentclass pattern
FLAVORS	ADD	Document	List of flavors for each document. Default: PS PDF
MAIN	SET	Document	Master tex source file Default: master.tex
DEPENDS	ADD	Target	List of dependencies
<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 # list of messages categories to display
42 LU_SHOW ?= warning #info debug debug-vars
43
44 # Select GNU/BSD/MACOSX utils (cp, rm, mv, ...)
45 LU_UTILS ?= $(shell ( /bin/cp --help > /dev/null 2>&1 && echo GNU ) || echo BSD )
46 export LU_UTILS
47
48 #####[ End of configuration ]#####
49 # Modifying the remaining of this document may endanger you life!!! ;)
50
51 #-----
52 # Controlling verbosity
53 ifdef VERB
54 MAK_VERB := $(VERB)
```

```

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

```

```

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

```

```

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

```

```

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

```

```

280 $(eval $(call lu-define-addvar,-index,$$2$$$(call lu-getvalue-
      flavor,EXT,$$3)_$$4_$$5_,index $$2$$$(call lu-getvalue-flavor,EXT,$$3)/$$5[$$4],6,\
281   $$2$$$(call lu-getvalue-flavor,EXT,$$3)_ $$2_$$4_$$5_,\
282   $$$(eval $$$(call lu-def-addvar-inherited-ctx,$$1,$$2,$$3)) \
283   $$$(eval $$$(call lu-def-addvar-inherited-ctx-master-index,$$1,$$2,$$4,$$5)) \
284 ))
285
286
287
288
289
290
291 define lu-setvar-global # 1:name 2:value
292   _LU_$ (1) ?= $(2)
293   $$$(eval $$$(call lu-show-set-var,$(1),global,$(2)))
294 endef
295
296 define lu-setvar-flavor # 1:name 2:flavor 3:value
297   _LU_FLAVOR_$ (2)_$(1) ?= $(3)
298   $$$(eval $$$(call lu-show-set-var,$(1),flavor $(2),$(3)))
299 endef
300
301 define lu-setvar-master # 1:name 2:master 3:value
302   _LU_$ (2)_$(1) ?= $(3)
303   $$$(eval $$$(call lu-show-set-var,$(1),master $(2),$(3)))
304 endef
305
306 define lu-setvar # 1:name 2:master 3:flavor 4:value
307   _LU_$ (2)$$$(call lu-getvalue-flavor,EXT,$(3))_$(1)=$(4)
308   $$$(eval $$$(call lu-show-set-var,$(1),master/flavor $(2)/$(3),$(4)))
309 endef
310
311 define lu-getvalue # 1:name 2:master 3:flavor
312 $(call lu-show-readone-var,$(1),master/flavor $(2)/$(3),$(or \
313 $(LU_$ (2))$(call lu-getvalue-flavor,EXT,$(3))_$(1)), \
314 $(TD_$ (2))$(call lu-getvalue-flavor,EXT,$(3))_$(1)), \
315 $(LU_$ (2)_$(1)), \
316 $( $(2)_$(1)), \
317 $(LU_FLAVOR_$ (3)_$(1)), \
318 $(LU_$ (1)), \
319 $( $(1)), \
320 $( _LU_$ (2))$(call lu-getvalue-flavor,EXT,$(3))_$(1)), \
321 $( _LU_$ (2)_$(1)), \
322 $( _LU_FLAVOR_$ (3)_$(1)), \
323 $( _LU_$ (1))\
324 ))
325 endef
326
327 define lu-getvalue-flavor # 1:name 2:flavor
328 $(call lu-show-readone-var,$(1),flavor $(2),$(or \
329 $(LU_FLAVOR_$ (2)_$(1)), \
330 $(LU_$ (1)), \
331 $( $(1)), \
332 $( _LU_FLAVOR_$ (2)_$(1)), \
333 $( _LU_$ (1))\
334 ))
335 endef
336

```

```

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

```

```

395 #####
396 #####
397 #####
398 ##### Global variables #####
399 #####
400 #####
401 #####
402 #####
403 #####
404 #####
405
406 # Globals variables
407 $(eval $(call lu-setvar-global,LATEX,latex))
408 $(eval $(call lu-setvar-global,PDFLATEX,pdflatex))
409 $(eval $(call lu-setvar-global,LUALATEX,lualatex))
410 $(eval $(call lu-setvar-global,DVIPS,dvips))
411 $(eval $(call lu-setvar-global,DVIPDFM,dvipdfm))
412 $(eval $(call lu-setvar-global,BIBTEX,bibtex))
413 #$(eval $(call lu-setvar-global,MPOST,TEX="$(LATEX)" mpost))
414 $(eval $(call lu-setvar-global,FIG2DEV,fig2dev))
415 #$(eval $(call lu-setvar-global,SVG2DEV,svg2dev))
416 $(eval $(call lu-setvar-global,EPSTOPDF,epstopdf))
417 $(eval $(call lu-setvar-global,MAKEINDEX,makeindex))
418
419 # Look for local version, then texmfscript, then in PATH of our program
420 # At each location, we prefer with suffix than without
421 define _lu_which # VARNAME progname
422 ifeq ($(origin _LU_${1}_DEFAULT), undefined)
423 _LU_${1}_DEFAULT := $(firstword $(wildcard \
424     $(addprefix bin/,${2}) $(basename ${2})) \
425     $(addprefix ./,${2}) $(basename ${2})) \
426     $(shell kpsewhich -format texmfscripts ${2}) \
427     $(shell kpsewhich -format texmfscripts $(basename ${2})) \
428     $(foreach dir,$$(subst :, ,$(PATH)), \
429     $(dir)/${2} $(dir)/$(basename ${2})) \
430 ) $(2))
431 export _LU_${1}_DEFAULT
432 endif
433 $$$(eval $(call lu-setvar-global,$(1),$_LU_${1}_DEFAULT)))
434 endef
435
436 $(eval $(call _lu_which,GENSUBFIG,gensubfig.py))
437 $(eval $(call _lu_which,FIGDEPTH,figdepth.py))
438 $(eval $(call _lu_which,GENSUBSVG,gensubfig.py))
439 $(eval $(call _lu_which,SVGDEPTH,svgdepth.py))
440 $(eval $(call _lu_which,SVG2DEV,svg2dev.py))
441 $(eval $(call _lu_which,LATEXFILTER,latexfilter.py))
442
443 # Rules to use to check if the build document (dvi or pdf) is up-to-date
444 # This can be overruled per document manually and/or automatically
445 #REBUILD_RULES ?= latex texdepends bibtopic bibtopic_undefined_references
446 $(eval $(call lu-addtovar-global,REBUILD_RULES,latex texdepends))
447
448 # Default maximum recursion level
449 $(eval $(call lu-setvar-global,MAX_REC,6))
450
451 #####
452 #####

```

```

453 #####
454 #####
455 #####
456 #####          Flavors          #####
457 #####          #####
458 #####
459 #####
460 #####
461 #####
462 #####
463
464 define lu-create-texflavor # 1:name 2:tex_prog 3:file_ext
465   # 4:master_cible 5:fig_extention_to_clean
466   _LU_FLAVORS_DEFINED_TEX += $(1)
467   $(eval $(call lu-setvar-flavor,VARPROG,$(1),$(2)))
468   $(eval $(call lu-setvar-flavor,EXT,$(1),$(3)))
469   $(eval $(call lu-setvar-flavor,TARGETNAME,$(1),$(4)))
470   $(eval $(call lu-addtovar-flavor,CLEANFIGEXT,$(1),$(5)))
471   $(eval $(call lu-addtovar-flavor,CLEANSVGEXT,$(1),$(5)))
472 endef
473
474 define lu-create-dviflavor # 1:name 2:dvi_prog 3:file_ext
475   # 4:master_cible 5:tex_flavor_depend
476   $$$(eval $$$(call lu-define-flavor,$(5)))
477   _LU_FLAVORS_DEFINED_DVI += $(1)
478   $(eval $(call lu-setvar-flavor,VARPROG,$(1),$(2)))
479   $(eval $(call lu-setvar-flavor,EXT,$(1),$(3)))
480   $(eval $(call lu-setvar-flavor,TARGETNAME,$(1),$(4)))
481   $(eval $(call lu-setvar-flavor,DEPFLAVOR,$(1),$(5)))
482 endef
483
484 define lu-create-flavor # 1:name 2:type 3.:7:options
485   $$$(if $$$(filter $(1),$_LU_FLAVORS_DEFINED)), \
486   $$$(call lu-show-flavors,Flavor $(1) already defined), \
487   $$$(call lu-show-flavors,Creating flavor $(1) ($(2))) \
488   $$$(eval $$$(call lu-create-$(2)flavor,$(1),$(3),$(4),$(5),$(6),$(7))))
489 endef
490
491 define lu-define-flavor # 1:name
492   $$$(eval $$$(call lu-define-flavor-$(1)))
493 endef
494
495 define lu-flavor-rules # 1:name
496   $$$(call lu-show-flavors,Defining rules for flavor $(1))
497   $$$(if $$$(call lu-getvalue-flavor,TARGETNAME,$(1)), \
498   $$$(call lu-getvalue-flavor,TARGETNAME,$(1)): \
499   $$$(call lu-getvalues-flavor,TARGETS,$(1)))
500   $$$(if $$$(call lu-getvalue-flavor,TARGETNAME,$(1)), \
501   .PHONY: $$$(call lu-getvalue-flavor,TARGETNAME,$(1)))
502 endef
503
504 define lu-define-flavor-DVI #
505   $$$(eval $$$(call lu-create-flavor,DVI,tex,LATEX,.dvi,dvi,\
506   .pstex_t .pstex))
507 endef
508
509 define lu-define-flavor-PDF #
510   $$$(eval $$$(call lu-create-flavor,PDF,tex,PDFLATEX,.pdf,pdf,\

```

```

511 .pdfTeX_t .$$(_LU_PDFTEX_EXT)))
512 endif
513
514 define lu-define-flavor-LUALATEX #
515   $$ (eval $$ (call lu-create-flavor,LUALATEX,tex,LUALATEX,.pdf,pdf,\
516 .pdfTeX_t .$$(_LU_PDFTEX_EXT)))
517 endif
518
519 define lu-define-flavor-PS #
520   $$ (eval $$ (call lu-create-flavor,PS,dvi,DVIPS,.ps,ps,DVI))
521 endif
522
523 define lu-define-flavor-DVIPDF #
524   $$ (eval $$ (call lu-create-flavor,DVIPDF,dvi,DVIPDFM,.pdf,pdf,DVI))
525 endif
526
527 $(eval $(call lu-addtovar-global,FLAVORS,PDF PS))
528
529 #####
530 #####
531 #####
532 #####
533 #####
534 ##### Masters #####
535 #####
536 #####
537 #####
538 #####
539 #####
540 #####
541
542 define _lu-do-latex # 1:master 2:flavor 3:source.tex 4:ext(.dvi/.pdf)
543   exec 3>&1; \
544   run() { \
545   printf "Running:" 1>&3 ; \
546   for arg; do \
547   printf "%s" " '$$arg' " 1>&3 ; \
548   done ; echo 1>&3 ; \
549   "$$@" ; \
550   }; \
551   doit() { \
552   $(RM) -v "$$(1)$$(4)_FAILED" \
553   "$$(1)$$(4)_NEED_REBUILD" \
554   "$$(1)$$(4).mk" ;\
555   ( echo X | \
556   run $(call lu-call-prog-flavor,$(1),$(2)) \
557   --interaction errorstopmode \
558   --jobname "$(1)" \
559   '\RequirePackage[extension='"$$(4)"']{texdepends}\input'{"$(3)}" || \
560   touch "$$(1)$$(4)_FAILED" ; \
561   if grep -sq '^! LaTeX Error:' "$(1).log" ; then \
562   touch "$$(1)$$(4)_FAILED" ; \
563   fi \
564   ) | $(call lu-call-prog,LATEXFILTER,$(1),$(2)) ; \
565   NO_TEXDEPENDS_FILE=0 ;\
566   if [ ! -f "$$(1)$$(4).mk" ]; then \
567   NO_TEXDEPENDS_FILE=1 ;\
568   fi ;\

```

```

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

```

```

625 $$ (addsuffix .ilg,$$(basename \
626 $(call lu-getvalue-index,SRC,$(1),$(2),$(3),$(4))))
627 endif #####
628 define lu-master-texflavor-index # MASTER FLAVOR INDEX ext(.dvi/.pdf)
629 $$ (eval $(call lu-master-texflavor-index-vars,$(1),$(2),$(3),$(4)))
630 $$ (eval $(call lu-master-texflavor-index-rules,$(1),$(2),$(3),$(4)))
631 endif
632 #####
633
634 #####
635 define lu-master-texflavor-vars # MASTER FLAVOR ext(.dvi/.pdf)
636 $$ (call lu-show-rules,Setting flavor vars for $(1)/$(2))
637 -include $(1)$(3).mk
638 $$ (eval $(call lu-addtovar,DEPENDS,$(1),$(2), \
639         $(call lu-getvalues,FIGURES,$(1),$(2)) \
640         $(call lu-getvalues,BIBFILES,$(1),$(2)) \
641         $(wildcard $(call lu-getvalues,INPUTS,$(1),$(2))) \
642         $(wildcard $(call lu-getvalues,BIBSTYLES,$(1),$(2))) \
643         $(call lu-getvalues,BBLFILES,$(1),$(2))\
644 ))
645
646 $$ (eval $(call lu-addtovar-flavor,TARGETS,$(2),$(1)$(3)))
647
648 $$ (eval $(call lu-addtovar,GPATH,$(1),$(2), \
649         $(subst },,,$$(subst {,,,$$(subst }{, ,\
650 $(call lu-getvalue,GRAPHICSPATH,$(1),$(2))))))
651
652 $$ (if $(sort $(call lu-getvalues,SUBFIGS,$(1),$(2))), \
653 $(eval include $(addsuffix .mk,$$(sort \
654 $(call lu-getvalues,SUBFIGS,$(1),$(2))))))
655
656 $$ (eval $(call lu-addtovar,WATCHFILES,$(1),$(2), \
657 $(filter %.aux, $(call lu-getvalues,OUTPUTS,$(1),$(2))))
658
659 $$ (foreach type,$$(call lu-getvalues,INDEXES,$(1),$(2)), \
660     $(foreach index,$$(call lu-getvalues,INDEXES,$(type),$(1),$(2)), \
661         $(eval $(call lu-master-texflavor-index-vars,$(1),$(2),$(type),$(index),$(3))))
662 endif #####
663 define lu-master-texflavor-rules # MASTER FLAVOR ext(.dvi/.pdf)
664 $$ (call lu-show-rules,Defining flavor rules for $(1)/$(2))
665 $$ (call lu-getvalues,BBLFILES,$(1),$(2)): \
666 $(sort $(call lu-getvalues,BIBFILES,$(1),$(2)) \
667 $(wildcard $(call lu-getvalues,BIBSTYLES,$(1),$(2))))
668 $(1)$(3): %$(3): \
669     $(call lu-getvalues,DEPENDS,$(1),$(2)) \
670     $(call lu-getvalues,REQUIRED,$(1),$(2)) \
671     $(if $(wildcard $(1)$(3)_FAILED),LU_FORCE,) \
672     $(if $(wildcard $(1)$(3)_NEED_REBUILD),LU_FORCE,) \
673     $(if $(wildcard $(1)$(3)_NEED_REBUILD_IN_PROGRESS),LU_FORCE,)
674 $$ (if $(filter-out $(LU_REC_LEVEL),$(call lu-getvalue,MAX_REC,$(1),$(2))),, \
675 $(warning *****) \
676 $(warning *****) \
677 $(warning *****) \
678 $(warning Stopping generation of $$@) \
679 $$ (warning I got max recursion level $(call lu-getvalue,MAX_REC,$(1),$(2))) \
680 $(warning Set LU_$(1)_$(2)_MAX_REC, LU_MAX_REC=$(1) or LU_MAX_REC if you need it) \
681 $(warning *****) \
682 $(warning *****) \

```

```

683 $$ (warning *****) \
684 $$ (error Aborting generation of $$@)
685 $$ (MAKE) LU_REC_MASTER="$ (1)" LU_REC_FLAVOR="$ (2)" LU_REC_TARGET="$ $@" \
686 LU_WATCH_FILES_SAVE
687 $$ (COMMON_PREFIX) $$ (call _lu-do-latex \
688 , $ (1), $ (2), $$ (call lu-getvalue-master, MAIN, $ (1)), $ (3))
689 $$ (MAKE) LU_REC_MASTER="$ (1)" LU_REC_FLAVOR="$ (2)" LU_REC_TARGET="$ $@" \
690 LU_WATCH_FILES_RESTORE
691 $$ (MAKE) LU_REC_MASTER="$ (1)" LU_REC_FLAVOR="$ (2)" LU_REC_TARGET="$ $@" \
692 $ (1) $ (3)_NEED_REBUILD
693 ifneq ($ (LU_REC_TARGET),)
694 $ (1) $ (3)_NEED_REBUILD_IN_PROGRESS:
695 $$ (COMMON_HIDE) touch $ (1) $ (3)_NEED_REBUILD_IN_PROGRESS
696 $$ (addprefix LU_rebuild_, $$ (call lu-getvalues, REBUILD_RULES, $ (1), $ (2))) : \
697 $ (1) $ (3)_NEED_REBUILD_IN_PROGRESS
698 .PHONY: $ (1) $ (3)_NEED_REBUILD
699 $ (1) $ (3)_NEED_REBUILD: \
700 $ (1) $ (3)_NEED_REBUILD_IN_PROGRESS \
701 $$ (addprefix LU_rebuild_, $$ (call lu-getvalues, REBUILD_RULES, $ (1), $ (2)))
702 $$ (COMMON_HIDE) $ (RM) $ (1) $ (3)_NEED_REBUILD_IN_PROGRESS
703 $$ (COMMON_HIDE) if [ -f "$ (1) $ (3)_NEED_REBUILD" ]; then \
704 echo "*****" ; \
705 echo "***** New build needed *****" ; \
706 echo "*****" ; \
707 cat "$ (1) $ (3)_NEED_REBUILD" ; \
708 echo "*****" ; \
709 fi
710 $$ (MAKE) LU_REC_LEVEL=$$ (shell expr $$ (LU_REC_LEVEL) + 1) \
711 $$ (LU_REC_TARGET)
712 endif
713 clean-build-fig::
714 $$ (call lu-clean, $$ (foreach fig, \
715 $$ (basename $$ (wildcard $$ (filter %.fig, \
716 $$ (call lu-getvalues, FIGURES, $ (1), $ (2))))), \
717 $$ (addprefix $$ (fig), $$ (call lu-getvalues-flavor, CLEANFIGEXT, $ (2)))) \
718 $$ (call lu-clean, $$ (foreach svg, \
719 $$ (basename $$ (wildcard $$ (filter %.svg, \
720 $$ (call lu-getvalues, FIGURES, $ (1), $ (2))))), \
721 $$ (addprefix $$ (svg), $$ (call lu-getvalues-flavor, CLEANSVGEXT, $ (2)))) \
722 clean:: clean-build-fig
723 $$ (call lu-clean, $$ (call lu-getvalues, OUTPUTS, $ (1), $ (2)) \
724 $$ (call lu-getvalues, BBLFILES, $ (1), $ (2)) \
725 $$ (addsuffix .mk, $$ (call lu-getvalues, SUBFIGS, $ (1), $ (2)) \
726 $$ (patsubst %.bbl, %.blg, $$ (call lu-getvalues, BBLFILES, $ (1), $ (2)))) \
727 $$ (call lu-clean, $$ (wildcard $ (1).log))
728 distclean::
729 $$ (call lu-clean, $$ (wildcard $ (1) $ (3) $ (1) $ (3)_FAILED \
730 $ (1) $ (3)_NEED_REBUILD $ (1) $ (3)_NEED_REBUILD_IN_PROGRESS))
731 $$ (foreach type, $$ (call lu-getvalues, INDEXES, $ (1), $ (2)), \
732 $$ (foreach index, $$ (call lu-getvalues, INDEXES_$$ (type), $ (1), $ (2)), \
733 $$ (eval $$ (call lu-master-texflavor-index-rules, $ (1), $ (2), $$ (type), $$ (index), $ (3)))))
734 endif #####
735 define lu-master-texflavor # MASTER FLAVOR ext(.dvi/.pdf)
736 $$ (eval $$ (call lu-master-texflavor-vars, $ (1), $ (2), $ (3)))
737 $$ (eval $$ (call lu-master-texflavor-rules, $ (1), $ (2), $ (3)))
738 endif
739 #####
740

```

```

741 #####
742 define lu-master-dviflavor-vars # MASTER FLAVOR ext(.ps)
743   $(call lu-show-rules,Setting flavor vars for \
744 $(1)/$(2)/$(call lu-getvalue-flavor,DEPFLAVOR,$(2)))
745 #   $(eval $(call lu-addvar,VARPROG,$(1),$(2)))
746 #   $(eval $(call lu-addvar,$$(call lu-getvalue,VARPROG,$(1),$(2)),$(1),$(2)))
747   $(eval $(call lu-addtovar-flavor,TARGETS,$(2),$(1)$(3)))
748 endef #####
749 define lu-master-dviflavor-rules # MASTER FLAVOR ext(.ps)
750   $(call lu-show-rules,Defining flavor rules for \
751 $(1)/$(2)/$(call lu-getvalue-flavor,DEPFLAVOR,$(2)))
752   $(1)$(3): %$(3): %$(call lu-getvalue-flavor,EXT,$$(call lu-getvalue-
       flavor,DEPFLAVOR,$(2)))
753   $(call lu-call-prog-flavor,$(1),$(2)) -o $$@ $$<
754   distclean::
755   $(call lu-clean,$$(wildcard $(1)$(3)))
756 endef #####
757 define lu-master-dviflavor # MASTER FLAVOR ext(.ps)
758   $(eval $(call lu-master-dviflavor-vars,$(1),$(2),$(3)))
759   $(eval $(call lu-master-dviflavor-rules,$(1),$(2),$(3)))
760 endef
761 #####
762
763 #####
764 define lu-master-vars # MASTER
765   $(call lu-show-rules,Setting vars for $(1))
766   $(eval $(call lu-setvar-master,MAIN,$(1),$(1).tex))
767   $(eval $(call lu-addtovar-master,DEPENDS,$(1),\
768   $(call lu-getvalue-master,MAIN,$(1))))
769   _LU_$(1)_DVI_FLAVORS=$(filter $(call lu-getvalue-master,MAIN,$(1)),\
770   $(call lu-getvalues-master,FLAVORS,$(1)))
771   _LU_$(1)_TEX_FLAVORS=$(filter $(call lu-getvalue-master,MAIN,$(1)),\
772   $(call lu-getvalues-master,FLAVORS,$(1)) \
773   $(LU_REC_FLAVOR) \
774   $(foreach dvi,$$(call lu-getvalues-master,FLAVORS,$(1)), \
775   $(call lu-getvalue-flavor,DEPFLAVOR,$(dvi))))
776   $(foreach flav,$$(call lu-getvalues-master,FLAVORS,$(1)), \
777   $(eval $(call lu-master-texflavor-vars,$(1),$(flav),$(call lu-getvalue-flavor,EXT,$(flav))))
778   $(foreach flav,$$(call lu-getvalues-master,FLAVORS,$(1)), \
779   $(eval $(call lu-master-dviflavor-vars,$(1),$(flav),$(call lu-getvalue-flavor,EXT,$(flav))))
780 endef #####
781 define lu-master-rules # MASTER
782   $(call lu-show-rules,Defining rules for $(1))
783   $(foreach flav,$$(call lu-getvalues-master,FLAVORS,$(1)), \
784   $(eval $(call lu-master-texflavor-rules,$(1),$(flav),$(call lu-getvalue-flavor,EXT,$(flav))))
785   $(foreach flav,$$(call lu-getvalues-master,FLAVORS,$(1)), \
786   $(eval $(call lu-master-dviflavor-rules,$(1),$(flav),$(call lu-getvalue-flavor,EXT,$(flav))))
787 endef #####
788 define lu-master # MASTER
789   $(eval $(call lu-master-vars,$(1)))
790   $(eval $(call lu-master-rules,$(1)))
791 endef
792 #####
793
794 #$(warning $(call LU_RULES,example))
795 $(eval $(call lu-addtovar-global,MASTERS,\
796 $(shell grep -l '\documentclass' *.tex 2>/dev/null | sed -e 's/\.\tex$$$$//'))
797 ifneq ($(LU_REC_TARGET),)

```

```

798 _LU_DEF_MASTERS = $(LU_REC_MASTER)
799 _LU_DEF_FLAVORS = $(LU_REC_FLAVOR) $(FLAV_DEPFLAVOR_$(LU_REC_FLAVOR))
800 else
801 _LU_DEF_MASTERS = $(call lu-getvalues-global,MASTERS)
802 _LU_DEF_FLAVORS = $(sort $(foreach master,$(_LU_DEF_MASTERS),\
803 $(call lu-getvalues-master,FLAVORS,$(master))))
804 endif
805
806 $(foreach flav, $_LU_DEF_FLAVORS, $(eval $(call lu-define-flavor,$(flav))))
807 $(foreach master, $_LU_DEF_MASTERS, $(eval $(call lu-master-vars,$(master))))
808 $(foreach flav, $_LU_FLAVORS_DEFINED, $(eval $(call lu-flavor-rules,$(flav))))
809 $(foreach master, $_LU_DEF_MASTERS, $(eval $(call lu-master-rules,$(master))))
810
811 #####"
812 # Gestion des subfigs
813
814 %<<MAKEFILE
815 %.subfig.mk: %.subfig %.fig
816 $(COMMON_PREFIX)$(call lu-call-prog,GENSUBFIG) \
817 -p '$$(COMMON_PREFIX)$(call lu-call-prog,FIGDEPTH) < $$< > $$@' \
818 -s $*.subfig $*.fig < $^ > $@
819 %MAKEFILE
820
821 %<<MAKEFILE
822 %.subfig.mk: %.subfig %.svg
823 $(COMMON_PREFIX)$(call lu-call-prog,GENSUBSVG) \
824 -p '$$(COMMON_PREFIX)$(call lu-call-prog,SVGDEPTH) < $$< > $$@' \
825 -s $*.subfig $*.svg < $^ > $@
826 %MAKEFILE
827
828 clean::
829 $(call lu-clean,$(FIGS2CREATE_LIST))
830 $(call lu-clean,$(FIGS2CREATE_LIST:%.fig=%.pstex))
831 $(call lu-clean,$(FIGS2CREATE_LIST:%.fig=%.pstex_t))
832 $(call lu-clean,$(FIGS2CREATE_LIST:%.fig=%.(_LU_PDFTEX_EXT)))
833 $(call lu-clean,$(FIGS2CREATE_LIST:%.fig=%.pdftex_t))
834 $(call lu-clean,$(FIGS2CREATE_LIST:%.svg=%.pstex))
835 $(call lu-clean,$(FIGS2CREATE_LIST:%.svg=%.pstex_t))
836 $(call lu-clean,$(FIGS2CREATE_LIST:%.svg=%.(_LU_PDFTEX_EXT)))
837 $(call lu-clean,$(FIGS2CREATE_LIST:%.svg=%.pdftex_t))
838
839 .PHONY: LU_FORCE clean distclean
840 LU_FORCE:
841 @echo "Previous compilation failed. Rerun needed"
842
843 #$(warning $(MAKEFILE))
844
845 distclean:: clean
846
847 %<<MAKEFILE
848 %.eps: %.fig
849 $(COMMON_PREFIX)$(call lu-call-prog,FIG2DEV) -L eps $< $@
850
851 %.pdf: %.fig
852 $(COMMON_PREFIX)$(call lu-call-prog,FIG2DEV) -L pdf $< $@
853
854 %.pstex: %.fig
855 $(COMMON_PREFIX)$(call lu-call-prog,FIG2DEV) -L pstex $< $@

```

```

856
857 %.pstex: %.svg
858 $(COMMON_PREFIX)$(call lu-call-prog,SVG2DEV) -L pstex $< $@
859
860
861 .PRECIOUS: %.pstex
862 %.pstex_t: %.fig %.pstex
863 $(COMMON_PREFIX)$(call lu-call-prog,FIG2DEV) -L pstex_t -p $*.pstex $< $@
864
865 %.pstex_t: %.svg %.pstex
866 $(COMMON_PREFIX)$(call lu-call-prog,SVG2DEV) -L pstex_t -p $*.pstex $< $@
867
868
869 %.$(_LU_PDFTEX_EXT): %.fig
870 $(COMMON_PREFIX)$(call lu-call-prog,FIG2DEV) -L pdftex $< $@
871
872 %.$(_LU_PDFTEX_EXT): %.svg
873 $(COMMON_PREFIX)$(call lu-call-prog,SVG2DEV) -L pdftex $< $@
874
875 .PRECIOUS: %.$(_LU_PDFTEX_EXT)
876 %.pdftex_t: %.fig %.$(_LU_PDFTEX_EXT)
877 $(COMMON_PREFIX)$(call lu-call-prog,FIG2DEV) -L pdftex_t -p $*.$(_LU_PDFTEX_EXT) $< $@
878
879 %.pdftex_t: %.svg %.$(_LU_PDFTEX_EXT)
880 $(COMMON_PREFIX)$(call lu-call-prog,SVG2DEV) -L pdftex_t -p $*.$(_LU_PDFTEX_EXT) $< $@
881
882 %.pdf: %.eps
883 $(COMMON_PREFIX)$(call lu-call-prog,EPSTOPDF) --filter < $< > $@
884 %MAKEFILE
885
886 #####
887 # Les flavors
888 LU_REC_LEVEL ?= 1
889 ifneq ($(LU_REC_TARGET),)
890 export LU_REC_FLAVOR
891 export LU_REC_MASTER
892 export LU_REC_TARGET
893 export LU_REC_LEVEL
894 LU_REC_LOGFILE=$(LU_REC_MASTER).log
895 LU_REC_GENFILE=$(LU_REC_MASTER)$(call lu-getvalue-flavor,EXT,$(LU_REC_FLAVOR))
896
897 lu-rebuild-head=$(info *** Checking rebuild with rule '$(subst LU_rebuild_,,$@)')
898 lu-rebuild-needed=echo $(1) >> "$(LU_REC_GENFILE)_NEED_REBUILD" ;
899
900 .PHONY: $(addprefix LU_rebuild_,latex texdepends bibtex)
901 LU_rebuild_latex:
902 $(call lu-rebuild-head)
903 $(COMMON_HIDE)if grep -sq 'Rerun to get'\
904 "$$(LU_REC_LOGFILE)" ; then \
905 $(call lu-rebuild-needed\
906 ,"$@: new run needed (LaTeX message 'Rerun to get...')") \
907 fi
908
909 LU_rebuild_texdepends:
910 $(call lu-rebuild-head)
911 $(COMMON_HIDE)if grep -sq '^Package texdepends Warning: .* Check dependen-
cies again.$$\`
912 "$$(LU_REC_LOGFILE)" ; then \

```

```

913 $(call lu-rebuild-needed,"$@: new depends required") \
914 fi
915
916 LU_rebuild_bibtopic:
917 $(call lu-rebuild-head)
918 /makefile

This part is not needed: already checked with the lu_rebuild_latex rule
919 (*notused)
920 $(COMMON_HIDE)if grep -sq 'Rerun to get indentation of bibitems right'\
921 "$(LU_REC_LOGFILE)" ; then \
922 $(call lu-rebuild-needed,"$@: new run needed") \
923 fi
924 $(COMMON_HIDE)if grep -sq 'Rerun to get cross-references right'\
925 "$(LU_REC_LOGFILE)" ; then \
926 $(call lu-rebuild-needed,"$@: new run needed") \
927 fi
928 (/notused)
929 (*makefile)
930 $(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) *\[^\]*\)$$/\1/p};d' \
931 "$(LU_REC_LOGFILE)" | while read file ; do \
932 touch $$file.aux ; \
933 $(call lu-rebuild-needed,"bibtopic: $$file.bbl outdated") \
934 done
935
936 LU_rebuild_bibtopic_undefined_references:
937 $(call lu-rebuild-head)
938 $(COMMON_HIDE)if grep -sq 'There were undefined references'\
939 "$(MASTER_$(LU_REC_MASTER)).log" ; then \
940 $(call lu-rebuild-needed,"$@: new run needed") \
941 fi
942
943 .PHONY: LU_WATCH_FILES_SAVE LU_WATCH_FILES_RESTORE
944 LU_WATCH_FILES_SAVE:
945 $(COMMON_HIDE)$(foreach file, $(sort \
946 $(call lu-getvalues,WATCHFILES,$(LU_REC_MASTER),$(LU_REC_FLAVOR))), \
947 $(call lu-save-file,$(file),$(file).orig);)
948
949 LU_WATCH_FILES_RESTORE:
950 $(COMMON_HIDE)$(foreach file, $(sort \
951 $(call lu-getvalues,WATCHFILES,$(LU_REC_MASTER),$(LU_REC_FLAVOR))), \
952 $(call lu-cmprestaure-file,"$(file)","$(file).orig",\
953 echo "New $(file) file" >> $(LU_REC_GENFILE)_NEED_REBUILD;\
954 );)
955
956 endif
957
958 %<<MAKEFILE
959 %.bbl: %.aux
960 $(COMMON_PREFIX)$(call lu-call-prog,BIBTEX) $*
961 %MAKEFILE
962
963 _LaTeX_Make_GROUPS=BIN TEX
964 _LaTeX_Make_BIN = figdepth.py gensubfig.py svg2dev.py svgdepth.py latexfilter.py
965 _LaTeX_Make_BINDIR=bin
966 _LaTeX_Make_BINORIGDIR= /FIXME_TDS_ROOT/scripts/latex-make
967 _LaTeX_Make_TEX = figlatex.sty pdfswitch.sty texdepends.sty texgraphicx.sty

```

```

968 _LaTeX_Make_TEXDIR=.
969 _LaTeX_Make_TEXORIGDIR= /FIXME_TDS_ROOT/tex/latex/latex-make
970
971 .PHONY: LaTeX-Make-local-install LaTeX-Make-local-uninstall
972 .PHONY: _LaTeX-Make-local-install-done
973 _LaTeX-Make-local-install-done:
974
975 LaTeX-Make-local-uninstall::
976 $(foreach g,$(_LaTeX_Make_GROUPS),\
977 $(foreach f,$(_LaTeX_Make_$(g)), \
978 $(LU_RM) $_LaTeX_Make_$(g)DIR)/$f && \
979 ) (rmdir $_LaTeX_Make_$(g)DIR || true) && \
980 ) $(LU_RM) LaTeX.mk
981
982 LaTeX-Make-local-install:: _LaTeX-Make-local-install-done
983 $(foreach g,$(_LaTeX_Make_GROUPS),\
984 mkdir -p $_LaTeX_Make_$(g)DIR && \
985 $(foreach f,$(_LaTeX_Make_$(g)), \
986 $(LU_CP) $_LaTeX_Make_$(g)ORIGDIR)/$f $_LaTeX_Make_$(g)DIR && \
987 )) $(LU_CP) $_LaTeX_Make_BINORIGDIR)/LaTeX.mk .
988 @echo >> LaTeX.mk
989 @echo "_LaTeX-Make-local-install-done:" >> LaTeX.mk
990 @echo " @echo " >> LaTeX.mk
991 @echo " @echo 'You must remove (at least) the locally installed La-
    TeX.mk file if you wish to'" >> LaTeX.mk
992 @echo " @echo 'restart the installation.'" >> LaTeX.mk
993 @echo " @echo 'You can try \"make LaTeX-Make-local-uninstall\"'" >> LaTeX.mk
994 @echo " @echo " >> LaTeX.mk
995 @echo " @exit 1" >> LaTeX.mk
996 @echo
997 @echo "=> All LaTeX-Make files are locally copied"
998 @echo
999
1000 </makefile>

```

5.2 figdepth

```

1001 (*figdepth)
1002 #!/usr/bin/env python
1003 #coding=utf8
1004
1005 """
1006
1007 stdin : the original xfig file
1008 stdout : the output xfig file
1009 args : all depths we want to keep
1010
1011 """
1012
1013 import optparse
1014 import os.path
1015 import sys
1016
1017 def main():
1018     parser = optparse.OptionParser()
1019     (options, args) = parser.parse_args()
1020
1021     depths_to_keep = set()
1022     for arg in args:

```

```

1023         depths_to_keep.add(arg)
1024
1025     comment = ''
1026     display = True
1027     def show(depth, line):
1028         if depth in depths_to_keep:
1029             print comment+line,
1030             return True
1031         else:
1032             return False
1033     for line in sys.stdin:
1034         if line[0] == '#':
1035             comment += line
1036             continue
1037         if line[0] in "\t ":
1038             if display:
1039                 print line
1040         else:
1041             Fld = line.split(' ', 9999)
1042             if not Fld[0] or Fld[0] not in ('1', '2', '3', '4', '5'):
1043                 print comment+line
1044                 display = True
1045             elif Fld[0] == '4':
1046                 display = show(Fld[3], line)
1047             else:
1048                 display = show(Fld[6], line)
1049             comment = ''
1050
1051 if __name__ == "__main__":
1052     main()
1053 </figdepth>

```

5.3 gensubfig

```

1054 <*gensubfig>
1055 #!/usr/bin/env python
1056 #coding=utf8
1057
1058 """
1059
1060 Arguments passes :
1061     - fichier image (image.fig ou image.svg)
1062     - -s fichier subfig (image.subfig)
1063     - -p chemin du script pour generer les sous-images (svgdepth.py ou figdepth.py)
1064
1065 Sortie standard :
1066     - makefile pour creer les sous-images (au format .fig ou .svg), et pour les sup-
      primer
1067
1068 """
1069
1070 from optparse import OptionParser
1071 import os.path
1072
1073 def main():
1074     parser = OptionParser(usage='usage: %prog [options] svg file', descrip-
      tion='Creates a\
1075 Makefile generating subfigures using figdepth.py or svgdepth.py')
1076     parser.add_option("-s", "--subfig", dest="subfig", help="subfig file")

```

```

1077     parser.add_option("-p", "--depth", dest="depth", help="full path of depth script")
1078     (options, args) = parser.parse_args()
1079     if len(args) < 1:
1080         parser.error("incorrect number of arguments")
1081     if not options.subfig:
1082         parser.error("no subfig file specified")
1083     if not options.depth:
1084         parser.error("no depth script specified")
1085
1086     (root, ext) = os.path.splitext(args[0])
1087     sf_name = options.subfig
1088     ds_name = options.depth
1089     varname = '%s_FIGS' % root.upper()
1090
1091     subfigs = []
1092     for line in open(options.subfig, 'r'):
1093         t = line.find('#') # looking for comments
1094         if t > -1: line = line[0:t] # remove comments...
1095         line = line.strip() #remove blank chars
1096         if line == '': continue
1097         subfigs.append(line)
1098
1099     count = 1
1100     for subfig in subfigs:
1101         print "%s_%d%s: %s%s %s" % (root, count, ext, root, ext, sf_name)
1102         print "\t%s %s" % (ds_name, subfig)
1103         print ""
1104         count += 1
1105     print "%s := $(foreach n, " % varname,
1106     count = 1
1107     for subfig in subfigs:
1108         print '%d ' % count,
1109         count += 1
1110     print ", %s$(n)%s)" % (root, ext)
1111     print "FILES_TO_DISTCLEAN += $(%s)" % varname
1112     print "FIGS2CREATE_LIST += $(%s)" % varname
1113     print "$ (TEMPORAIRE): $(%s)" % varname
1114     print "$ (TEMPORAIRE): $(%s)" % varname
1115
1116 if __name__ == "__main__":
1117     main()
1118 </gensubfig>

```

5.4 svg2dev

```

1119 <svg2dev>
1120 #!/usr/bin/env python
1121 #coding=utf8
1122
1123 from optparse import OptionParser
1124 import shutil
1125 import subprocess
1126
1127
1128 svg2eps = 'inkscape %s -z -C --export-eps=%s --export-latex'
1129 svg2pdf = 'inkscape %s -z -C --export-pdf=%s --export-latex'
1130
1131
1132 def create_image(input_filename, output_filename, mode):

```

```

1133     subprocess.Popen(mode % (input_filename, output_filename),
1134         stdout=subprocess.PIPE, shell=True).communicate()[0]
1135     n1 = output_filename + '_tex'
1136     n2 = output_filename + '_t'
1137     shutil.move(n1, n2)
1138
1139
1140 def main():
1141     parser = OptionParser()
1142     parser.add_option("-L", "--format", dest="outputFormat",
1143         metavar="FORMAT", help="output format", default="spstex")
1144     parser.add_option("-p", "--portrait", dest="portrait", help="dummy arg")
1145     (options, args) = parser.parse_args()
1146     if len(args) != 2: return
1147     (input_filename, output_filename) = args
1148     fmt = options.outputFormat
1149     portrait = options.portrait
1150
1151     if fmt == 'eps':
1152         create_image(input_filename, output_filename, svg2eps)
1153     elif fmt == 'spstex' or fmt == 'pstex':
1154         create_image(input_filename, output_filename, svg2eps)
1155     elif fmt == 'spstex_t' or fmt == 'pstex_t':
1156         pass
1157     elif fmt == 'spdfTEX' or fmt == 'pdfTEX':
1158         create_image(input_filename, output_filename, svg2pdf)
1159     elif fmt == 'spdfTEX_t' or fmt == 'pdfTEX_t':
1160         pass
1161
1162
1163 if __name__ == "__main__":
1164     main()
1165
1166 </svg2dev>

```

5.5 latexfilter

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

```

1167 (*latexfilter)
1168 #!/usr/bin/env python
1169 #coding=utf8
1170
1171 """
1172
1173 stdin : the original xfig file
1174 stdout : the output xfig file
1175 args : all depths we want to keep
1176
1177 """
1178
1179 from __future__ import print_function
1180 import optparse
1181 import os.path
1182 import re
1183 import sys
1184
1185 def main():
1186     parser = optparse.OptionParser()

```

```

1187     (options, args) = parser.parse_args()
1188
1189     display = 0
1190     in_display = 0
1191     start_line = ''
1192     warnerror_re = re.compile(r"^(LaTeX|Package|Class)(.*)? (Warning:|Error:)")
1193     fullbox_re = re.compile(r"^(Underfull|Overfull) \\[hv]box")
1194     accu = ''
1195     for line in sys.stdin:
1196         if display > 0:
1197             display -= 1
1198             if line[0:4].lower() in ('info', 'warn') or line[0:5].lower() == 'error':
1199                 display = 0
1200             line_groups = warnerror_re.match(line)
1201             if line_groups:
1202                 start_line = line_groups.group(3)
1203                 if not start_line:
1204                     start_line = ''
1205                 if line_groups.group(2):
1206                     start_line = "(" + start_line + ")"
1207             display = 1
1208             in_display = 1
1209             elif (start_line != '') and (line[0:len(start_line)] == start_line):
1210                 display = 1
1211             elif line == "\n":
1212                 in_display = 0
1213             elif line[0:4] == 'Chap':
1214                 display = 1
1215             elif fullbox_re.match(line):
1216                 display = 2
1217             if display:
1218                 print(accu, end="")
1219                 accu = line
1220             elif in_display:
1221                 print(accu[0:-1], end="")
1222                 accu = line
1223
1224 if __name__ == "__main__":
1225     main()
1226
1227 </latexfilter>

```

5.6 svgdepth

```

1228 <svgdepth>
1229 #!/usr/bin/env python
1230 #coding=utf8
1231
1232 import sys
1233 import xml.parsers.expat
1234
1235
1236 layers = []
1237 for arg in sys.argv:
1238     layers.append(arg)
1239
1240 parser = xml.parsers.expat.ParserCreate()
1241 class XmlParser(object):

```

```

1242     def __init__(self, layers):
1243         self.state_stack = [True]
1244         self.last_state = True
1245         self.layers = layers
1246     def XmlDeclHandler(self, version, encoding, standalone):
1247         sys.stdout.write("<?xml version='%s' encoding='%s'?>\n" % (version, encoding))
1248     def StartDoctypeDeclHandler(self, doctypeName, systemId, publi-
        cId, has_internal_subset):
1249         if publicId != None: sys.stdout.write("<!DOCTYPE %s PUBLIC \"%s\" \"%s\">\n" %\
1250             (doctypeName, publicId, systemId))
1251         else: sys.stdout.write("<!DOCTYPE %s \"%s\">\n" % (doctypeName, systemId))
1252     def StartElementHandler(self, name, attributes):
1253         if name.lower() == 'g':
1254             r = self.last_state and ('id' not in attributes or \
1255                 attributes['id'] in self.layers)
1256             self.last_state = r
1257             self.state_stack.append(r)
1258             if not self.last_state: return
1259             s = ""
1260             for k, v in attributes.items(): s += ' %s="%s"' % (k, v)
1261             sys.stdout.write("<%s%s>" % (name, s))
1262     def EndElementHandler(self, name):
1263         r = self.last_state
1264         if name.lower() == 'g':
1265             self.state_stack = self.state_stack[0:-1]
1266             self.last_state = self.state_stack[-1]
1267             if not r: return
1268             sys.stdout.write("</%s>" % (name))
1269     def CharacterDataHandler(self, data):
1270         if not self.last_state: return
1271         sys.stdout.write(data)
1272
1273 my_parser = XmlParser(layers)
1274
1275 parser.XmlDeclHandler = my_parser.XmlDeclHandler
1276 parser.StartDoctypeDeclHandler = my_parser.StartDoctypeDeclHandler
1277 parser.StartElementHandler = my_parser.StartElementHandler
1278 parser.EndElementHandler = my_parser.EndElementHandler
1279 parser.CharacterDataHandler = my_parser.CharacterDataHandler
1280
1281 for line in sys.stdin:
1282     parser.Parse(line, False)
1283 parser.Parse('', True)
1284
1285
1286 </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

\ " 993, 1249, 1251

Change History

v2.0.0		v2.2.0	
General: First autocommented version . . .	1	General: Support to install LaTeX-Make	
v2.1.0		locally	1
General: That's the question	1	v2.2.1	
v2.1.1		General: Improve configure	1
General: Improve error message	1	v2.2.2	
v2.1.2		General: Fix bugs	1
General: Switch from perl to python	1	v2.2.3	
		General: Add LuaLaTeX support	1