

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

Vincent DANJEAN

Arnaud LEGRAND

2016/02/09

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

*This file has version number v2.2.2, last revised 2016/02/09.

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

396 # Globals variables
397 $(eval $(call lu-setvar-global,LATEX,latex))
398 $(eval $(call lu-setvar-global,PDFLATEX,pdflatex))
399 $(eval $(call lu-setvar-global,DVIPS,dvips))
400 $(eval $(call lu-setvar-global,DVIPDFM,dvipdfm))
401 $(eval $(call lu-setvar-global,BIBTEX,bibtex))
402 #$(eval $(call lu-setvar-global,MPOST,TEX="$(LATEX)" mpost))
403 $(eval $(call lu-setvar-global,FIG2DEV,fig2dev))
404 #$(eval $(call lu-setvar-global,SVG2DEV,svg2dev))
405 $(eval $(call lu-setvar-global,EPSTOPDF,epstopdf))
406 $(eval $(call lu-setvar-global,MAKEINDEX,makeindex))
407
408 # Look for local version, then texmfscript, then in PATH of our program
409 # At each location, we prefer with suffix than without
410 define _lu_which # VARNAME progname
411 ifeq ($(origin _LU_$(1)_DEFAULT), undefined)
412 _LU_$(1)_DEFAULT := $(firstword $(wildcard \
413     $(addprefix bin/, $(2) $(basename $(2))) \
414     $(addprefix ./, $(2) $(basename $(2))) \
415     $(shell kpsewhich -format texmfscripts $(2)) \
416     $(shell kpsewhich -format texmfscripts $(basename $(2))) \
417     $(foreach dir, $(subst :, , $(PATH)), \
418     $(dir)/$(2) $(dir)/$(basename $(2))) \
419 ) $(2))
420 export _LU_$(1)_DEFAULT
421 endif
422 $$$(eval $(call lu-setvar-global,$(1),$_LU_$(1)_DEFAULT))
423 endef
424
425 $(eval $(call _lu_which,GENSUBFIG,gensubfig.py))
426 $(eval $(call _lu_which,FIGDEPTH,figdepth.py))
427 $(eval $(call _lu_which,GENSUBSVG,gensubfig.py))
428 $(eval $(call _lu_which,SVGDEPTH,svgdepth.py))
429 $(eval $(call _lu_which,SVG2DEV,svg2dev.py))
430 $(eval $(call _lu_which,LATEXFILTER,latexfilter.py))
431
432 # Rules to use to check if the build document (dvi or pdf) is up-to-date
433 # This can be overruled per document manually and/or automatically
434 #REBUILD_RULES ?= latex texdepends bibtopic bibtopic_undefined_references
435 $(eval $(call lu-addtovar-global,REBUILD_RULES,latex texdepends))
436
437 # Default maximum recursion level
438 $(eval $(call lu-setvar-global,MAX_REC,6))
439
440 #####
441 #####
442 #####
443 #####
444 #####
445 ##### Flavors #####
446 #####
447 #####
448 #####
449 #####
450 #####
451 #####
452
453 define lu-create-texflavor # 1:name 2:tex_prog 3:file_ext

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

This part is not needed: already checked with the lu_rebuild_latex rule
903  $\backslash$ *notused)
904 $(COMMON_HIDE)if grep -sq 'Rerun to get indentation of bibitems right'\
905 "$$(LU_REC_LOGFILE)" ; then \
906 $(call lu-rebuild-needed,"$@: new run needed") \
907 fi
908 $(COMMON_HIDE)if grep -sq 'Rerun to get cross-references right'\
909 "$$(LU_REC_LOGFILE)" ; then \
910 $(call lu-rebuild-needed,"$@: new run needed") \
911 fi
912  $\backslash$ /notused)
913  $\backslash$ *makefile)

```

```

914 $(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) *\[^\]*\)$$/\1/p};d' \
915 "$(LU_REC_LOGFILE)" | while read file ; do \
916 touch $$file.aux ; \
917 $(call lu-rebuild-needed,"bibtopic: $$file.bbl outdated") \
918 done
919
920 LU_rebuild_bibttopic_undefined_references:
921 $(call lu-rebuild-head)
922 $(COMMON_HIDE)if grep -sq 'There were undefined references'\
923 "$(MASTER_$(LU_REC_MASTER)).log" ; then \
924 $(call lu-rebuild-needed,"@: new run needed") \
925 fi
926
927 .PHONY: LU_WATCH_FILES_SAVE LU_WATCH_FILES_RESTORE
928 LU_WATCH_FILES_SAVE:
929 $(COMMON_HIDE)$(foreach file, $(sort \
930 $(call lu-getvalues,WATCHFILES,$(LU_REC_MASTER),$(LU_REC_FLAVOR))), \
931     $(call lu-save-file,$(file),$(file).orig);)
932
933 LU_WATCH_FILES_RESTORE:
934 $(COMMON_HIDE)$(foreach file, $(sort \
935 $(call lu-getvalues,WATCHFILES,$(LU_REC_MASTER),$(LU_REC_FLAVOR))), \
936     $(call lu-cmprestaure-file,"$(file)","$(file).orig",\
937 echo "New $(file) file" >> $(LU_REC_GENFILE)_NEED_REBUILD;\
938 );)
939
940 endif
941
942 %<<MAKEFILE
943 %.bbl: %.aux
944 $(COMMON_PREFIX)$(call lu-call-prog,BIBTEX) $*
945 %MAKEFILE
946
947 _LaTeX_Make_GROUPS=BIN TEX
948 _LaTeX_Make_BIN = figdepth.py gensubfig.py svg2dev.py svgdepth.py latexfilter.py
949 _LaTeX_Make_BINDIR=bin
950 _LaTeX_Make_BINORIGDIR= /FIXME_TDS_ROOT/scripts/latex-make
951 _LaTeX_Make_TEX = figlatex.sty pdfswitch.sty texdepends.sty texgraphicx.sty
952 _LaTeX_Make_TEXDIR=.
953 _LaTeX_Make_TEXORIGDIR= /FIXME_TDS_ROOT/tex/latex/latex-make
954
955 .PHONY: LaTeX-Make-local-install LaTeX-Make-local-uninstall
956 .PHONY: _LaTeX-Make-local-install-done
957 _LaTeX-Make-local-install-done:
958
959 LaTeX-Make-local-uninstall::
960 $(foreach g,$(_LaTeX_Make_GROUPS),\
961 $(foreach f,$(_LaTeX_Make_$(g)), \
962 $(LU_RM) $(_LaTeX_Make_$(g)DIR)/$f && \
963 ) (rmdir $(_LaTeX_Make_$(g)DIR) || true) && \
964 ) $(LU_RM) LaTeX.mk
965
966 LaTeX-Make-local-install:: _LaTeX-Make-local-install-done
967 $(foreach g,$(_LaTeX_Make_GROUPS),\
968 mkdir -p $(_LaTeX_Make_$(g)DIR) && \
969 $(foreach f,$(_LaTeX_Make_$(g)), \

```

```

970 $(LU_CP) $( _LaTeX_Make_$(g)ORIGDIR)/$f $( _LaTeX_Make_$(g)DIR) && \
971 )) $(LU_CP) $( _LaTeX_Make_BINORIGDIR)/LaTeX.mk .
972 @echo >> LaTeX.mk
973 @echo "_LaTeX-Make-local-install-done:" >> LaTeX.mk
974 @echo " @echo " >> LaTeX.mk
975 @echo " @echo 'You must remove (at least) the locally installed La-
    TeX.mk file if you wish to'" >> LaTeX.mk
976 @echo " @echo 'restart the installation.'" >> LaTeX.mk
977 @echo " @echo 'You can try \"make LaTeX-Make-local-uninstall\"'" >> LaTeX.mk
978 @echo " @echo " >> LaTeX.mk
979 @echo " @exit 1" >> LaTeX.mk
980 @echo
981 @echo "=> All LaTeX-Make files are locally copied"
982 @echo
983
984 </makefile>

```

5.2 figdepth

```

985 (*figdepth)
986 #!/usr/bin/env python
987 #coding=utf8
988
989 """
990
991 stdin : the original xfig file
992 stdout : the output xfig file
993 args : all depths we want to keep
994
995 """
996
997 import optparse
998 import os.path
999 import sys
1000
1001 def main():
1002     parser = optparse.OptionParser()
1003     (options, args) = parser.parse_args()
1004
1005     depths_to_keep = set()
1006     for arg in args:
1007         depths_to_keep.add(arg)
1008
1009     comment = ''
1010     display = True
1011     def show(depth, line):
1012         if depth in depths_to_keep:
1013             print comment+line,
1014             return True
1015         else:
1016             return False
1017     for line in sys.stdin:
1018         if line[0] == '#':
1019             comment += line
1020             continue
1021         if line[0] in "\t ":
1022             if display:
1023                 print line
1024         else:

```

```

1025         Fld = line.split(' ', 9999)
1026         if not Fld[0] or Fld[0] not in ('1', '2', '3', '4', '5'):
1027             print comment+line
1028             display = True
1029         elif Fld[0] == '4':
1030             display = show(Fld[3], line)
1031         else:
1032             display = show(Fld[6], line)
1033         comment = ''
1034
1035 if __name__ == "__main__":
1036     main()
1037 </figdepth>

```

5.3 gensubfig

```

1038 (*gensubfig)
1039 #!/usr/bin/env python
1040 #coding=utf8
1041
1042 """
1043
1044 Arguments passes :
1045     - fichier image (image.fig ou image.svg)
1046     - -s fichier subfig (image.subfig)
1047     - -p chemin du script pour generer les sous-images (svgdepth.py ou figdepth.py)
1048
1049 Sortie standard :
1050     - makefile pour creer les sous-images (au format .fig ou .svg), et pour les sup-
      primer
1051
1052 """
1053
1054 from optparse import OptionParser
1055 import os.path
1056
1057 def main():
1058     parser = OptionParser(usage='usage: %prog [options] svg file', descrip-
      tion='Creates a\
1059 Makefile generating subfigures using figdepth.py or svgdepth.py')
1060     parser.add_option("-s", "--subfig", dest="subfig", help="subfig file")
1061     parser.add_option("-p", "--depth", dest="depth", help="full path of depth script")
1062     (options, args) = parser.parse_args()
1063     if len(args) < 1:
1064         parser.error("incorrect number of arguments")
1065     if not options.subfig:
1066         parser.error("no subfig file specified")
1067     if not options.depth:
1068         parser.error("no depth script specified")
1069
1070     (root, ext) = os.path.splitext(args[0])
1071     sf_name = options.subfig
1072     ds_name = options.depth
1073     varname = '%s_FIGS' % root.upper()
1074
1075     subfigs = []
1076     for line in open(options.subfig, 'r'):
1077         t = line.find('#') # looking for comments
1078         if t > -1: line = line[0:t] # remove comments...

```

```

1079         line = line.strip() #remove blank chars
1080         if line == '': continue
1081         subfigs.append(line)
1082
1083         count = 1
1084         for subfig in subfigs:
1085             print "%s_%d%s: %s%s %s" % (root, count, ext, root, ext, sf_name)
1086             print "\t%s %s" % (ds_name, subfig)
1087             print ""
1088             count += 1
1089         print "%s := $(foreach n, " % varname,
1090               count = 1
1091         for subfig in subfigs:
1092             print '%d ' % count,
1093             count += 1
1094         print ", %s_$(n)%s)" % (root, ext)
1095         print "FILES_TO_DISTCLEAN += $(%s)" % varname
1096         print "FIGS2CREATE_LIST += $(%s)" % varname
1097         print "$ (TEMPORAIRE): $(%s)" % varname
1098         print "$ (TEMPORAIRE): $(%s)" % varname
1099
1100 if __name__ == "__main__":
1101     main()
1102 </gensubfig>

```

5.4 svg2dev

```

1103 <svg2dev>
1104 #!/usr/bin/env python
1105 #coding=utf8
1106
1107 from optparse import OptionParser
1108 import shutil
1109 import subprocess
1110
1111
1112 svg2eps = 'inkscape %s -z -C --export-eps=%s --export-latex'
1113 svg2pdf = 'inkscape %s -z -C --export-pdf=%s --export-latex'
1114
1115
1116 def create_image(input_filename, output_filename, mode):
1117     subprocess.Popen(mode % (input_filename, output_filename),
1118                     stdout=subprocess.PIPE, shell=True).communicate()[0]
1119     n1 = output_filename + '_tex'
1120     n2 = output_filename + '_t'
1121     shutil.move(n1, n2)
1122
1123
1124 def main():
1125     parser = OptionParser()
1126     parser.add_option("-L", "--format", dest="outputFormat",
1127                     metavar="FORMAT", help="output format", default="spstex")
1128     parser.add_option("-p", "--portrait", dest="portrait", help="dummy arg")
1129     (options, args) = parser.parse_args()
1130     if len(args) != 2: return
1131     (input_filename, output_filename) = args
1132     fmt = options.outputFormat
1133     portrait = options.portrait
1134

```

```

1135     if fmt == 'eps':
1136         create_image(input_filename, output_filename, svg2eps)
1137     elif fmt == 'spstex' or fmt == 'pstex':
1138         create_image(input_filename, output_filename, svg2eps)
1139     elif fmt == 'spstex_t' or fmt == 'pstex_t':
1140         pass
1141     elif fmt == 'spdfTEX' or fmt == 'pdfTEX':
1142         create_image(input_filename, output_filename, svg2pdf)
1143     elif fmt == 'spdfTEX_t' or fmt == 'pdfTEX_t':
1144         pass
1145
1146
1147 if __name__ == "__main__":
1148     main()
1149
1150 </svg2dev>

```

5.5 latexfilter

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

```

1151 (*latexfilter)
1152 #!/usr/bin/env python
1153 #coding=utf8
1154
1155 """
1156
1157 stdin : the original xfig file
1158 stdout : the output xfig file
1159 args : all depths we want to keep
1160
1161 """
1162
1163 from __future__ import print_function
1164 import optparse
1165 import os.path
1166 import re
1167 import sys
1168
1169 def main():
1170     parser = optparse.OptionParser()
1171     (options, args) = parser.parse_args()
1172
1173     display = 0
1174     in_display = 0
1175     start_line = ''
1176     warnerror_re = re.compile(r"^(LaTeX|Package|Class)(.*)? (Warning:|Error:)")
1177     fullbox_re = re.compile(r"^(Underfull|Overfull) \\[hv\]box")
1178     accu = ''
1179     for line in sys.stdin:
1180         if display > 0:
1181             display -= 1
1182         if line[0:4].lower() in ('info', 'warn') or line[0:5].lower() == 'error':
1183             display = 0
1184         line_groups = warnerror_re.match(line)
1185         if line_groups:
1186             start_line = line_groups.group(3)
1187         if not start_line:
1188             start_line = ''

```

```

1189         if line_groups.group(2):
1190             start_line = "(" + start_line + ")"
1191             display = 1
1192             in_display = 1
1193         elif (start_line != '') and (line[0:len(start_line)] == start_line):
1194             display = 1
1195         elif line == "\n":
1196             in_display = 0
1197         elif line[0:4] == 'Chap':
1198             display = 1
1199         elif fullbox_re.match(line):
1200             display = 2
1201         if display:
1202             print(accum, end="")
1203             accum = line
1204         elif in_display:
1205             print(accum[0:-1], end="")
1206             accum = line
1207
1208 if __name__ == "__main__":
1209     main()
1210
1211 </latexfilter>

```

5.6 svgdepth

```

1212 (*svgdepth)
1213 #!/usr/bin/env python
1214 #coding=utf8
1215
1216 import sys
1217 import xml.parsers.expat
1218
1219
1220 layers = []
1221 for arg in sys.argv:
1222     layers.append(arg)
1223
1224 parser = xml.parsers.expat.ParserCreate()
1225 class XmlParser(object):
1226     def __init__(self, layers):
1227         self.state_stack = [True]
1228         self.last_state = True
1229         self.layers = layers
1230     def XmlDeclHandler(self, version, encoding, standalone):
1231         sys.stdout.write("<?xml version='%s' encoding='%s'>\n" % (version, encoding))
1232     def StartDoctypeDeclHandler(self, doctypeName, systemId, publi-
1233 cId, has_internal_subset):
1234         if publicId != None: sys.stdout.write("<!DOCTYPE %s PUBLIC \"%s\" \"%s\">\n" % \
1235 (doctypeName, publicId, systemId))
1236         else: sys.stdout.write("<!DOCTYPE %s \"%s\">\n" % (doctypeName, systemId))
1237     def StartElementHandler(self, name, attributes):
1238         if name.lower() == 'g':
1239             r = self.last_state and ('id' not in attributes or \
1240 attributes['id'] in self.layers)
1241             self.last_state = r
1242             self.state_stack.append(r)
1243         if not self.last_state: return

```

```

1243         s = ""
1244         for k, v in attributes.items(): s += ' %s="%s"' % (k, v)
1245         sys.stdout.write("<%s%s>" % (name, s))
1246     def EndElementHandler(self, name):
1247         r = self.last_state
1248         if name.lower() == 'g':
1249             self.state_stack = self.state_stack[0:-1]
1250             self.last_state = self.state_stack[-1]
1251         if not r: return
1252         sys.stdout.write("</%s>" % (name))
1253     def CharacterDataHandler(self, data):
1254         if not self.last_state: return
1255         sys.stdout.write(data)
1256
1257 my_parser = XmlParser(layers)
1258
1259 parser.XmlDeclHandler = my_parser.XmlDeclHandler
1260 parser.StartDoctypeDeclHandler = my_parser.StartDoctypeDeclHandler
1261 parser.StartElementHandler = my_parser.StartElementHandler
1262 parser.EndElementHandler = my_parser.EndElementHandler
1263 parser.CharacterDataHandler = my_parser.CharacterDataHandler
1264
1265 for line in sys.stdin:
1266     parser.Parse(line, False)
1267 parser.Parse('', True)
1268
1269
1270 </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

\ " 977, 1233, 1235

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