The LaTeX.mk Makefile and related script tools*

Vincent Danjean Arnaud Legrand

2012/03/17

Abstract

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

Contents

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

1 Introduction

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

These tools can be used to compile small LaTeX documents as well as big ones (such as, for example, a thesis with summary, tables of contents, list of figures, list of tabulars, multiple indexes and multiple bibliographies).

2 Quick start

2.1 First (and often last) step

When you want to use `latex-make`, most of the time you have to create a `Makefile` with the only line:

```
include LaTeX.mk
```

Then, the following targets are available: `dvi`, `ps`, `pdf`, *file*`.dvi`, *file*`.ps`, *file*`.pdf`, etc., `clean` and `distclean`.

All LaTeX documents of the current directory should be compilable with respect to their dependencies. If something fails, please, provide me the smallest example you can create to show me what is wrong.

Tip: If you change the dependencies inside your document (for example, if you change `\include{first}` into `\include{second}`), you may have to type `make distclean` before being able to recompile your document. Else, `make` can fail, trying to build or found the old `first.tex` file.

2.2 Customization

Of course, lots of things can be customized. Here are the most useful ones. Look at the section 3 for more detailed and complete possibilities.

Customization is done through variables in the `Makefile` set *before* including `LaTeX.mk`. Setting them after can sometimes work, but not always and it is not supported.

Which LaTeX documents to compile `LU_MASTERS`

Example: `LU_MASTERS=figlatex texdepends latex-make`

This variable contains the basename of the LaTeX documents to compile.

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

Which LaTeX main source for a document ***master*`_MAIN`**

Example: `figlatex_MAIN=figlatex.dtx`

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

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

[1] Dependencies are tracked with the help of the `texdepend.sty` package that is automatically loaded: no need to specify it with `\usepackage{}` in your documents.

Which flavors must be compiled `LU_FLAVORS`

Example: `LU_FLAVORS=DVI DVIPDF`

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

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

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

Flavor	dependency	program variable	Transformation
DVI		LATEX	`.tex` ⇒ `.dvi`
PS	DVI	DVIPS	`.dvi` ⇒ `.ps`
PDF		PDFLATEX	`.tex` ⇒ `.pdf`
DVIPDF	DVI	DVIPDFM	`.dvi` ⇒ `.pdf`

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

Table 1: Predefined flavors

Which programs are called and with which options ***prog*/*prog*_OPTIONS**

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

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

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

Per target programs and options ***master_prog*/*master_prog*_OPTIONS**

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

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

Global and per target dependencies DEPENDS/***master***_DEPENDS

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

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

3 Reference manual

3.1 Flavors

3.1.1 What is a flavor ?

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

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

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

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

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

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

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

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

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

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

3.1.2 Defining a new flavor

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

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

For example, `LaTeX.mk` already defines:

DVI flavor

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

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

PDF flavor

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

PS flavor

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

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

DVIPDF flavor

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

3.2 Variables

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

3.2.1 Two kind of variables

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

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

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

1	`LU_`*target_NAME*	per document and per flavor value;
2	`TD_`*target_NAME*	per document and per flavor value filled by the `texdepends` LaTeX package;
3	`LU_`*master_NAME*	per document value;
4	*master_NAME*	per document value;
5	`LU_FLAVOR_`*flavor_NAME*	per flavor value;
6	`LU_`*NAME*	global value;
7	*NAME*	global value;
8	`_LU_...`*NAME*	internal `LaTeX.mk` default values.

The first set variable will be used.

Tip: in case of flavor context or document context, only relevant variables will be checked. For example, the SET-variable `MAIN` that give the main source of the document will be evaluated in document context, so only 4, 5, 6, 7 and 8 will be used (and I cannot see any real interest in using 6 or 7 for this variable).

Tip2: in case of context of index (when building indexes or glossary), there exists several other variables per index to add to this list (mainly ending with *_kind_indexname_NAME* or *_kind_NAME*). Refer to the sources if you really need them.

ADD-variable. An ADD-variable is cumulative. The user can replace or add any values per document, per flavor, etc.

1	`LU_`*target_NAME*	replacing per document and per flavor values;
2	*target_NAME*	cumulative per document and per flavor values;
3	`LU_`*master_NAME*	replacing per document values;
4	*master_NAME*	cumulative per document values;
5	`LU_FLAVOR_`*flavor_NAME*	replacing per flavor values;
6	`FLAVOR_`*flavor_NAME*	cumulative per flavor values;
7	`LU_`*NAME*	replacing global values;
8	*NAME*	cumulative global values;

Tip: if not defined, `LU_`*variable* defaults to "`$(`*variable*`) $(_LU_`*variable*`)`" and `_LU_`*variable* contains default values managed by `LaTeX.mk` and the `texdepends` LaTeX package.

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

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

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

3.2.2 List of used variables

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

Name	Kind	Context of use	Description
`MASTERS`	ADD	Global	List of documents to compile. These values will be used as jobname. **Default:** basename of `*.tex` files containing the `\documentclass` pattern
`FLAVORS`	ADD	Document	List of flavors for each document. **Default:** `PS PDF`
`MAIN`	SET	Document	Master `tex` source file **Default:** *master*`.tex`
`DEPENDS`	ADD	Target	List of dependencies
progvarname	SET	Target	Program to launch for the corresponding flavor
progvarname`_OPTIONS`	ADD	Target	Options to use when building the target
`STYLE`	SET	Index	Name of the index/glossary style file to use (`.ist`, etc.)
`TARGET`	SET	Index	Name of the index/glossary file to produce (`.ind`, `.gls`, etc.)
`SRC`	SET	Index	Name of the index/glossary file source (`.idx`, `.glo`, etc.)
`FIGURES`	ADD	Target	Lists of figures included
`BIBFILES`	ADD	Target	Lists of bibliography files used (`.bib`)
`BIBSTYLES`	ADD	Target	Lists of bibliography style files used (`.bst`)
`BBLFILES`	ADD	Target	Lists of built bibliography files (`.bbl`)
`INPUT`	ADD	Target	Lists of input files (`.cls`, `.sty`, `.tex`, etc.)
`OUTPUTS`	ADD	Target	Lists of output files (`.aux`, etc.)
`GRAPHICSPATH`	ADD	Target	`\graphicspath{}` arguments
`GPATH`	ADD	Target	List of directories from `GRAPHICSPATH` without `{` and `}`, separated by spaces
`INDEXES`	ADD	Target	Kinds of index (`INDEX`, `GLOSS`, etc.)
`INDEXES_`*kind*	ADD	Target	List of indexes or glossaries
`WATCHFILES`	ADD	Target	List of files that trigger a rebuild if modified (`.aux`, etc.)
`REQUIRED`	ADD	Target	List of new dependencies found by the `texdepends` LaTeX package
`MAX_REC`	SET	Target	Maximum level of recursion authorized
`REBUILD_RULES`	ADD	Target	List of rebuild rules to use (can be modified by the `texdepends` LaTeX package
`EXT`	SET	Flavor	Target file extension of the flavor
`DEPFLAVOR`	SET	Flavor	TEX-flavor a DVI-flavor depend upon
`CLEANFIGEXT`	ADD	Flavor	Extensions of figure files to remove on clean

4 FAQ

4.1 No rule to make target 'LU_WATCH_FILES_SAVE'

⇒ *When using `LaTeX.mk`, I got the error:*
`make[1]: *** No rule to make target 'LU_WATCH_FILES_SAVE'. Stop.`

`make` is called in such a way that does not allow correct recursive calls. As one can not know by advance how many times LaTeX, bibTeX, etc. will need to be run, `latex-make` use recursive invocations of `make`. This means that in the `LaTeX.mk` makefile, there exist rules such as:

```
$(MAKE) INTERNAL_VARIABLE=value internal_target
```

In order `latex-make` to work, this invocation of `make` must read the same rules and variable definitions as the main one. This means that calling "`make -f LaTeX.mk foo.pdf`" in a directory with only `foo.tex` will not work. Recursive invocations of make will not load `LaTeX.mk`, will search for a `Makefile` in the current directory and will complain about being unable to build the `LU_WATCH_FILES_SAVE` internal target.

The solution is to call `make` so that recursive invocations will read the same variables and rules. For example:
`make -f LaTeX.mk MAKE="make -f LaTeX.mk" foo.pdf`
or (if there is no `Makefile` in the directory):
`env MAKEFILES=LaTeX.mk make foo.pdf`

5 Implementation

5.1 LaTeX.mk

```
⟨*makefile⟩

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

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

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

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

# For global site options
-include LaTeX.mk.conf

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

# Select GNU/BSD/MACOSX utils (cp, rm, mv, ...)
LU_UTILS ?= GNU

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

#-----------------------------------------------------------------------
# Controling verbosity
```

```
ifdef VERB
MAK_VERB :=  $(VERB)
else
#MAK_VERB :=  verbose
#MAK_VERB :=  normal
MAK_VERB :=  quiet
#MAK_VERB :=  silent
endif

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

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

###############################################################################
# Utilities
LU_CP=$(LU_CP_$(LU_UTILS))
LU_MV=$(LU_MV_$(LU_UTILS))
LU_RM=$(LU_RM_$(LU_UTILS))
LU_CP_GNU ?= cp -a --
LU_MV_GNU ?= mv --
LU_RM_GNU ?= rm -f --
LU_CP_BSD ?= cp -p
LU_MV_BSD ?= mv
LU_RM_BSD ?= rm -f
LU_CP_MACOSX ?= /bin/cp -p
LU_MV_MACOSX ?= /bin/mv
LU_RM_MACOSX ?= /bin/rm -f
```

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

```
# _LU_FLAVORS_DEFINED : list of available flavors
# _LU_FLAV_*_'flavname' : per flavor variables
#   where * can be :
#   PROGNAME : variable name for programme (and .._OPTIONS for options)
#   EXT : extension of created file
#   TARGETNAME : global target
#   DEPFLAVOR : flavor to depend upon
#   CLEANFIGEXT : extensions to clean for fig figures
_LU_FLAVORS_DEFINED = $(_LU_FLAVORS_DEFINED_TEX) $(_LU_FLAVORS_DEFINED_DVI)

# INDEXES_TYPES = GLOSS INDEX
# INDEXES_INDEX = name1 ...
# INDEXES_GLOSS = name2 ...
# INDEX_name1_SRC
# GLOSS_name2_SRC

define _lu-getvalues# 1:VAR 2:CTX (no inheritage)
$(if $(filter-out undefined,$(origin LU_$2$1)),$(LU_$2$1),$($2$1) $(_LU_$2$1_MK) $(TD_$2$1))
endef
define lu-define-addvar # 1:suffix_fnname 2:CTX 3:disp-debug 4:nb_args 5:inher-
  ited_ctx 6:ctx-build-depend
  define lu-addtovar$1 # 1:VAR 2:... $4: value
    _LU_$2$$1_MK+=$$($4)
    $$(call lu-show-add-var,$$1,$3,$$(value $4))
  endef
  define lu-def-addvar-inherited-ctx$1 # 1:VAR 2:...
    $6
    _LU_$2$$1_INHERITED_CTX=$$(sort \
      $$(foreach ctx,$5,$$(ctx) $$(if $$(filter-out undefined,$$(origin \
          LU_$$(ctx)$$1)),,\
        $$(_LU_$$(ctx)$$1_INHERITED_CTX))))
    $$$$(call lu-show-var,_LU_$2$$1_INHERITED_CTX)
  endef
  define lu-getvalues$1# 1:VAR 2:...
$$(if $$(filter-out undefined,$$(origin _LU_$2$$1_INHERITED_CTX)),,$$(eval \
  $$(call lu-def-addvar-inherited-ctx$1,$$1,$$2,$$3,$$4,$$5,$$6)\
))$$(call lu-show-read-var,$$1,$3,$$(foreach ctx,\
    $(if $2,$2,GLOBAL) $$(if $$(filter-out undefined,$$(origin LU_$2$$1)),,\
            $$(_LU_$2$$1_INHERITED_CTX))\
    ,$$(call _lu-getvalues,$$1,$$(filter-out GLOBAL,$$(ctx)))))
  endef
endef

# Global variable
# VAR (DEPENDS)
$(eval $(call lu-define-addvar,-global,,global,2))

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

# Per master variable
# $2_VAR (source_DEPENDS)
```

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

# Per target variable
# $2$(EXT of $3)_VAR (source.dvi_DEPENDS)
# 2: master name
# 3: flavor name
# Inherit from $2_VAR FLAVOR_$3_VAR (source_DEPENDS FLAVOR_DVI_DEPENDS)
$(eval $(call lu-define-addvar,,$$2$$(call lu-getvalue-flavor,EXT,$$3)_,target $$2$$(call lu-getvalue-flavor,EXT,$$3),4,\
  $$2_ FLAVOR_$$3_,\
  $$(eval $$(call lu-def-addvar-inherited-ctx-master,$$1,$$2)) \
  $$(eval $$(call lu-def-addvar-inherited-ctx-flavor,$$1,$$3)) \
))

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

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

# Per target and per index/glossary variable
# $(2)$(EXT of $3)_$(4)_$(5)_VAR (source.dvi_INDEX_source_DEPENDS)
# 2: master name
# 3: flavor name
# 4: type (INDEX, GLOSS, ...)
# 5: index name
# Inherit from $2$(EXT of $3)_VAR $(2)_$(3)_$(4)_VAR
# (source.dvi_DEPENDS source_INDEX_source_DEPENDS)
$(eval $(call lu-define-addvar,-index,$$2$$(call lu-getvalue-flavor,EXT,$$3)_$$4_$$5_,index $$2$$(call lu-getvalue-flavor,EXT,$$3)/$$5[$$4],6,\
  $$2$$(call lu-getvalue-flavor,EXT,$$3)_ $$2_$$4_$$5_,\
  $$(eval $$(call lu-def-addvar-inherited-ctx,$$1,$$2,$$3)) \
  $$(eval $$(call lu-def-addvar-inherited-ctx-master-index,$$1,$$2,$$4,$$5)) \
))
```

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

```
endef

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

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

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

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

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

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

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

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

```
fi; \
if [ "$$NO_TEXDEPENDS_FILE" = 1 ]; then \
echo "************************************" ;\
echo "texdepends does not seems be loaded" ;\
echo "Either your (La)TeX installation is wrong or you found a bug." ;\
echo "If so, please, report it (with the result of shell command 'kpsepath tex')";\
echo "Aborting compilation" ;\
echo "************************************" ;\
touch "$(1)$(4)_FAILED" ; \
return 1 ;\
fi ;\
    }; doit
endef

.PHONY: clean-build-fig

############################################################
define lu-master-texflavor-index-vars # MASTER FLAVOR TYPE INDEX ext(.dvi/.pdf)
 $$(call lu-show-rules,Setting flavor index vars for $(1)/$(2)/[$(3)]$(4))
 $$(eval $$(call lu-addtovar,DEPENDS,$(1),$(2), \
    $$(call lu-getvalue-index,TARGET,$(1),$(2),$(3),$(4))))
 $$(eval $$(call lu-addtovar,WATCHFILES,$(1),$(2), \
    $$(call lu-getvalue-index,SRC,$(1),$(2),$(3),$(4))))
endef ####################################################
define lu-master-texflavor-index-rules # MASTER FLAVOR TYPE INDEX ext(.dvi/.pdf)
 $$(call lu-show-rules,Setting flavor index rules for $(1)/$(2)/[$(3)]$(4))
 $$(if $$(_LU_DEF_IND_$$(call lu-getvalue-index,TARGET,$(1),$(2),$(3),$(4))), \
   $$(call lu-show-rules,=> Skipping: already defined in fla-
vor $$(_LU_DEF_IND_$$(call lu-getvalue-index,TARGET,$(1),$(2),$(3),$(4)))), \
   $$(eval $$(call _lu-master-texflavor-index-rules\
,$(1),$(2),$(3),$(4),$(5),$$(call lu-getvalue-index,TARGET,$(1),$(2),$(3),$(4)))))
endef
define _lu-master-texflavor-index-rules # MASTER FLAVOR TYPE INDEX ext TARGET
 $(6): \
    $$(call lu-getvalue-index,SRC,$(1),$(2),$(3),$(4)) \
    $$(wildcard $$(call lu-getvalue-index,STYLE,$(1),$(2),$(3),$(4)))
$$(COMMON_PREFIX)$$(call lu-call-prog-index,MAKEINDEX,$(1),$(2),$(3),$(4)) \
  $$(addprefix -s ,$$(call lu-getvalue-index,STYLE,$(1),$(2),$(3),$(4))) \
  -o $$@ $$<
 _LU_DEF_IND_$(6)=$(2)
 clean::
$$(call lu-clean,$$(call lu-getvalue-index,TARGET,$(1),$(2),$(3),$(4)) \
$$(addsuffix .ilg,$$(basename \
$$(call lu-getvalue-index,SRC,$(1),$(2),$(3),$(4)))))
endef ####################################################
define lu-master-texflavor-index # MASTER FLAVOR INDEX ext(.dvi/.pdf)
 $$(eval $$(call lu-master-texflavor-index-vars,$(1),$(2),$(3),$(4)))
 $$(eval $$(call lu-master-texflavor-index-rules,$(1),$(2),$(3),$(4)))
endef
############################################################

############################################################
define lu-master-texflavor-vars # MASTER FLAVOR ext(.dvi/.pdf)
 $$(call lu-show-rules,Setting flavor vars for $(1)/$(2))
 -include $(1)$(3).mk
 $$(eval $$(call lu-addtovar,DEPENDS,$(1),$(2), \
               $$(call lu-getvalues,FIGURES,$(1),$(2)) \
               $$(call lu-getvalues,BIBFILES,$(1),$(2)) \
```

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

```
 $(1)$(3)_NEED_REBUILD: \
    $(1)$(3)_NEED_REBUILD_IN_PROGRESS \
    $$(addprefix LU_rebuild_,$$(call lu-getvalues,REBUILD_RULES,$(1),$(2)))
$$(COMMON_HIDE)$(RM) $(1)$(3)_NEED_REBUILD_IN_PROGRESS
$$(COMMON_HIDE)if [ -f "$(1)$(3)_NEED_REBUILD" ];then\
echo "*****************************************" ;\
echo "*********** New build needed ***************" ;\
echo "*****************************************" ;\
cat "$(1)$(3)_NEED_REBUILD" ; \
echo "*****************************************" ;\
fi
$$(MAKE) LU_REC_LEVEL=$$(shell expr $$(LU_REC_LEVEL) + 1) \
$$(LU_REC_TARGET)
endif
 clean-build-fig::
$$(call lu-clean,$$(foreach fig, \
   $$(basename $$(wildcard $$(filter %.fig, \
$$(call lu-getvalues,FIGURES,$(1),$(2))))), \
   $$(addprefix $$(fig),$$(call lu-getvalues-flavor,CLEANFIGEXT,$(2)))))
$$(call lu-clean,$$(foreach svg, \
   $$(basename $$(wildcard $$(filter %.svg, \
$$(call lu-getvalues,FIGURES,$(1),$(2))))), \
   $$(addprefix $$(svg),$$(call lu-getvalues-flavor,CLEANSVGEXT,$(2)))))
 clean:: clean-build-fig
$$(call lu-clean,$$(call lu-getvalues,OUTPUTS,$(1),$(2)) \
$$(call lu-getvalues,BBLFILES,$(1),$(2)) \
$$(addsuffix .mk,$$(call lu-getvalues,SUBFIGS,$(1),$(2))) \
    $$(patsubst %.bbl,%.blg,$$(call lu-getvalues,BBLFILES,$(1),$(2))))
$$(call lu-clean,$$(wildcard $(1).log))
 distclean::
$$(call lu-clean,$$(wildcard $(1)$(3) $(1)$(3)_FAILED \
$(1)$(3)_NEED_REBUILD $(1)$(3)_NEED_REBUILD_IN_PROGRESS))
 $$(foreach type,$$(call lu-getvalues,INDEXES,$(1),$(2)), \
   $$(foreach index,$$(call lu-getvalues,INDEXES_$$(type),$(1),$(2)), \
    $$(eval $$(call lu-master-texflavor-index-rules,$(1),$(2),$$(type),$$(index),$(3)))))
endef ######################################################
define lu-master-texflavor # MASTER FLAVOR ext(.dvi/.pdf)
 $$(eval $$(call lu-master-texflavor-vars,$(1),$(2),$(3)))
 $$(eval $$(call lu-master-texflavor-rules,$(1),$(2),$(3)))
endef
############################################################

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

```
endef ##################################################
define lu-master-dviflavor # MASTER FLAVOR ext(.ps)
 $$(eval $$(call lu-master-dviflavor-vars,$(1),$(2),$(3)))
 $$(eval $$(call lu-master-dviflavor-rules,$(1),$(2),$(3)))
endef
##########################################################

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

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

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

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

```

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

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

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

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

#$(warning $(MAKEFILE))

distclean:: clean

%<<MAKEFILE
%.eps: %.fig
$(COMMON_PREFIX)$(call lu-call-prog,FIG2DEV) -L eps $< $@

%.pdf: %.fig
$(COMMON_PREFIX)$(call lu-call-prog,FIG2DEV) -L pdf $< $@

%.pstex: %.fig
$(COMMON_PREFIX)$(call lu-call-prog,FIG2DEV) -L pstex $< $@

%.pstex: %.svg
$(COMMON_PREFIX)$(call lu-call-prog,SVG2DEV) -L pstex $< $@


.PRECIOUS: %.pstex
%.pstex_t: %.fig %.pstex
$(COMMON_PREFIX)$(call lu-call-prog,FIG2DEV) -L pstex_t -p $*.pstex $< $@

%.pstex_t: %.svg %.pstex
$(COMMON_PREFIX)$(call lu-call-prog,SVG2DEV) -L pstex_t -p $*.pstex $< $@


%.$(_LU_PDFTEX_EXT): %.fig
$(COMMON_PREFIX)$(call lu-call-prog,FIG2DEV) -L pdftex $< $@

```

```
%.$(_LU_PDFTEX_EXT): %.svg
$(COMMON_PREFIX)$(call lu-call-prog,SVG2DEV) -L pdftex $< $@

.PRECIOUS: %.$(_LU_PDFTEX_EXT)
%.pdftex_t: %.fig %.$(_LU_PDFTEX_EXT)
$(COMMON_PREFIX)$(call lu-call-prog,FIG2DEV) -L pdftex_t -p $*.$(_LU_PDFTEX_EXT) $< $@

%.pdftex_t: %.svg %.$(_LU_PDFTEX_EXT)
$(COMMON_PREFIX)$(call lu-call-prog,SVG2DEV) -L pdftex_t -p $*.$(_LU_PDFTEX_EXT) $< $@

%.pdf: %.eps
$(COMMON_PREFIX)$(call lu-call-prog,EPSTOPDF) --filter < $< > $@
%MAKEFILE

########################################################################
# Les flavors
LU_REC_LEVEL ?= 1
ifneq ($(LU_REC_TARGET),)
export LU_REC_FLAVOR
export LU_REC_MASTER
export LU_REC_TARGET
export LU_REC_LEVEL
LU_REC_LOGFILE=$(LU_REC_MASTER).log
LU_REC_GENFILE=$(LU_REC_MASTER)$(call lu-getvalue-flavor,EXT,$(LU_REC_FLAVOR))

lu-rebuild-head=$(info *** Checking rebuild with rule '$(subst LU_rebuild_,,$@)')
lu-rebuild-needed=echo $(1) >> "$(LU_REC_GENFILE)_NEED_REBUILD" ;

.PHONY: $(addprefix LU_rebuild_,latex texdepends bibtex)
LU_rebuild_latex:
$(call lu-rebuild-head)
$(COMMON_HIDE)if grep -sq 'Rerun to get'\
"$(LU_REC_LOGFILE)" ; then \
$(call lu-rebuild-needed\
,"$@: new run needed (LaTeX message 'Rerun to get...')") \
fi

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

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

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

```
⟨*notused⟩
$(COMMON_HIDE)if grep -sq 'Rerun to get indentation of bibitems right'\
"$(LU_REC_LOGFILE)" ; then \
$(call lu-rebuild-needed,"$@: new run needed") \
fi
$(COMMON_HIDE)if grep -sq 'Rerun to get cross-references right'\
"$(LU_REC_LOGFILE)" ; then \
$(call lu-rebuild-needed,"$@: new run needed") \
fi
```

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

5.2 LaTeX.mk.conf

```
949 ⟨*makefile-config⟩
950 # Choose between GNU/BSD utilities (cp, rm, ...)
951 # LU_UTILS = GNU
952 # LU_UTILS = BSD
953 ⟨/makefile-config⟩
```

5.3 figdepth

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

```
import optparse
import os.path
import sys

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

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

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

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

5.4 gensubfig

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

"""

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

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

```

```
"""

from optparse import OptionParser
import os.path

def main():
    parser = OptionParser(usage='usage: %prog [options] svg file', description='Creates a\
Makefile generating subfigures using figdepth.py or svgdepth.py')
    parser.add_option("-s", "--subfig", dest="subfig", help="subfig file")
    parser.add_option("-p", "--depth", dest="depth", help="full path of depth script")
    (options, args) = parser.parse_args()
    if len(args) < 1:
        parser.error("incorrect number of arguments")
    if not options.subfig:
        parser.error("no subfig file specified")
    if not options.depth:
        parser.error("no depth script specified")

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

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

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

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

5.5 svg2dev

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

```

```
from optparse import OptionParser
import shutil
import subprocess


svg2eps = 'inkscape %s -z -C --export-eps=%s --export-latex'
svg2pdf = 'inkscape %s -z -C --export-pdf=%s --export-latex'


def create_image(input_filename, output_filename, mode):
    subprocess.Popen(mode % (input_filename, output_filename),
        stdout=subprocess.PIPE, shell=True).communicate()[0]
    n1 = output_filename + '_tex'
    n2 = output_filename + '_t'
    shutil.move(n1, n2)


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

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


if __name__ == "__main__":
    main()

```
⟨/svg2dev⟩

5.6 latexfilter

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

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

"""

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

```

```
"""

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

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

    display = 0
    in_display = 0
    start_line = ''
    warnerror_re = re.compile(r"^(LaTeX|Package|Class)( (.*))? (Warning:|Error:)")
    fullbox_re = re.compile(r"^(Underfull|Overfull) \\[hv]box")
    accu = ''
    for line in sys.stdin:
        if display > 0:
            display -= 1
        if line[0:4].lower() in ('info', 'warn') or line[0:5].lower() == 'error':
            display = 0
        line_groups = warnerror_re.match(line)
        if line_groups:
            start_line = line_groups.group(3)
            if not start_line:
                start_line = ''
            if line_groups.group(2):
                start_line = "(" + start_line + ")"
            display = 1
            in_display = 1
        elif (start_line != '') and (line[0:len(start_line)] == start_line):
            display = 1
        elif line == "\n":
            in_display = 0
        elif line[0:4] == 'Chap':
            display = 1
        elif fullbox_re.match(line):
            display = 2
        if display:
            print(accu, end="")
            accu = line
        elif in_display:
            print(accu[0:-1], end="")
            accu = line

if __name__ == "__main__":
    main()
```

⟨/latexfilter⟩

5.7 svgdepth

⟨∗svgdepth⟩

```
#!/usr/bin/env python
#coding=utf8

```

```
import sys
import xml.parsers.expat


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

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

my_parser = XmlParser(layers)

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

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


⟨/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

Change History