

Documentation de CDBS

Copyright © 2004-2009 DuckCorp

Permission is granted to copy, distribute and/or modify this document under the terms of the [GNU General Public License](#), Version 3 or any later version published by the Free Software Foundation.

INDEXATION DU DOCUMENT

	<i>TITRE :</i> Documentation de CDBS		
<i>ACTION</i>	<i>NOM</i>	<i>DATE</i>	<i>SIGNATURE</i>
RÉDIGÉ PAR	Marc (Duck) Dequènes et Arnaud (Rtp) Patard	29/03/2005	

SUIVI DU DOCUMENT

INDICE	DATE	MODIFICATIONS	NOM
0.1.0	03/04/2005	Première version publique (pour CDBS V0.4.27-3)	
0.1.1	07/06/2005	Mise à jour pour CDBS V0.4.30 (gestion des dépendances de construction dans la classe Perl, script cdbs-edit-patch)	
0.1.2	05/07/2005	Ajout d'avertissement d'usage pour <code>DEB_CONFIGURE_SCRIPT_ENV</code> , correction d'un typo.	
0.1.3	16/09/2005	Ajout d'information à propos des besoins de l'extension dpatch (inclusions supplémentaires + ordre d'inclusion). Ajout d'un avertissement et d'une méthode de contournement pour les problèmes liés à <code>DEB_AUTO_UPDATE_DEBIAN_CONTROL</code> (voir #311724). Correction d'une typo.	
0.1.4	02/10/2005	Added info about quilt extension for patching sources.	
0.2.0	05/01/2006	Added info about Ruby classes (Team & setup.rb). Reordered makefile and autotools class, and explained relationship. Document extraordinary use of <code>DEB_MAKE_ENVVARS</code> in autotools class.	

SUIVI DU DOCUMENT

INDICE	DATE	MODIFICATIONS	NOM
0.3.0	23/04/2006	Fixed typo (s/foo-date/foo-data/ reported by tioui). Warned of possible breakage if spaces in CURDIR (see #306941). Removed hacks in examples because of #300149, #284231 and #239128 / #341275. Updated for CDBS 0.4.37 (document DEB_MAKE_MAKEFILE, and special case when DEB_MAKE_CLEAN_TARGET can be empty + dh_installmime and dh_instal catalogs now called in debhelper class + s/DEB_ANT_TEST_TARGET/DEB_ANT_CHECK_TARGET/ which was a mistake in code, see #307813 + document DEB_CLEAN_EXCLUDE + default compat mode changed to 5 and DEB_DH_STRIP_ARGS usage too). Warn rules MUST come after CDBS includes (see #273835). Improved documentation of common build options.	
0.4.0	24/04/2006	Updated for CDBS 0.4.39 (ability to use uuencoded patches + dh_installudev now called in debhelper class + KDE class improvements + config.* left over autotools files not removed anymore + new DEB_DH_COMPAT_DISABLE variable + improved scrollkeeper and Python cleanup + updated common variables available in debian/rules). Updated some examples accordingly. Improved part on compat. Improved fixes related to DEB_AUTO_UPDATE_DEBIAN_CONTROL problems.	
0.5.0	2009-04-18	Updated for CDBS 0.4.56 (Python class update + updated DEB_CONFIGURE_SCRIPT_ENV usage + compat level 7 + documented new cmake and qmake classes). Made clear cdbs-edit-patch is for the simple-patchsys patch system. Added dh_icons to list of debhelper scripts handled by the GNOME class. Improved documentation of DEB_AUTO_UPDATE_* variables. Improved docbook tags a lot. Fixed punctuation a bit.	

Table des matières

1	Introduction	1
1.1	Un peu d'histoire	1
1.2	Pourquoi CDBS ?	1
2	Premiers pas	2
2.1	Convertir un paquet à CDBS	2
2.2	Réglages de base et variables disponibles	2
2.3	Règles de construction simples et sur-mesures	3
2.4	Options de compilation usuelles	4
2.5	Les astuces de Debhelper	4
2.5.1	Ne pas s'occuper de Debhelper	4
2.5.2	Paramètres de Debhelper	5
2.5.3	Règles personnalisées de construction pour debhelper	6
3	Tâches courantes	7
3.1	Appliquer une rustine (en utilisant simple-patchsys)	7
3.2	Appliquer des rustines (en utilisant dpatch)	7
3.3	Appliquer des rustines (en utilisant quilt)	8
3.4	Gestion automatique des tarball	8
4	Personnalisation avancée	10
4.1	debian/control management	10
4.2	Utilisation de la classe Makefile	11
4.3	Utilisation de la classe Autotools	11
4.4	Utilisation de la classe Perl	13
4.5	Utilisation de la classe Python	13
4.6	Utilisation de la classe Ruby setup.rb	15
4.7	Utilisation de la classe de l'équipe Debian Ruby Extras	15
4.8	Utilisation de la classe GNOME	16
4.9	Utilisation de la classe de l'équipe Debian GNOME	16
4.10	Utilisation de la classe KDE	17

4.11	Utilisation de la classe Ant	17
4.12	Utilisation de la classe HBuild	18
4.13	Using the CMake class	18
4.14	Using the qmake class	18
5	Panthéon des exemples	19
5.1	Exemple de GNOME + autotools + patchsys simple	19
5.2	Exemple Python	21
5.3	Exemple Makefile + Dpatch	22
5.4	Exemple Perl	23
6	Outils utiles	25
6.1	cdbs-edit-patch (provided with CDBS)	25
7	Conclusion	26

Liste des tableaux

2.1	Common variables available in <code>debian/rules</code>	3
2.2	Scripts Debhelper communément gérés	5
4.1	Scripts Debhelper gérés par la classe GNOME	16

Préface

Cette documentation décrit ce que nous avons réussi à apprendre de l'utilisation de CDBS, avec le plus de détails possible. Néanmoins, nous n'utilisons pas le jeu complet de fonctionnalités disponibles, et certaines parties de cette documentation ont été écrites par pur soucis pratique et d'exhaustivité.

Please note some examples in this documentation contain folded content which is necessary to keep the pages at a reasonable width; take care to unfold them when necessary before using them (eg: `debian/control` content must not be folded or build will fail or result be incorrect).

If you find mistakes or missing information, feel free to contact Marc Dequènes (Duck) duck@duckcorp.org.

Chapitre 1

Introduction

1.1 Un peu d'histoire

CDBS a été écrit par Jeff Bailey et Colin Waters en mars 2003, plus tard rejoins pas 4 autre développeurs.

Des informations simples peuvent être trouvées sur leur [page de projet](#). Dans le paquet est fourni [un petit jeu d'exemples](#) (aussi disponible dans le paquet ici : `/usr/share/doc/cdbsexamples/`).

Depuis que nous expérimentions CDBS, il était devenu évident que le manque de documentation nous empêchait de l'utiliser plus largement dans nos paquets. Alors nous avons commencé à écrire quelques notes sur l'utilisation de CDBS, qui ont rapidement grossi jusqu'à devenir plusieurs pages. Cette documentation est une version révisée de [la page wiki originale de la DuckCorp](#).

1.2 Pourquoi CDBS ?

CDBS is designed to simplify the maintainer's work so that they only need to think about packaging and not maintaining a `debian/rules` file that keeps growing bigger and more complicated. So CDBS can handle for you most of common rules and detect some parts of your configuration.

CDBS n'utilise que des règles simples de makefile et est aisément extensible en utilisant des classes. Des classes pour gérer les autotools, l'application de rustines aux sources, les programmes GNOME, l'installation de Python, et bien d'autres sont disponibles.

Les avantages de CDBS :

- short, readable and efficient `debian/rules`
 - automatise l'usage de debhelper et des autotools pour que vous n'ayez pas à vous préoccuper de cette déplaisante et répétitive tâche.
 - le mainteneur peut se concentrer sur les vrais problèmes de paquets car CDBS vous aide sans vous limiter
 - les classes utilisées dans CDBS ont été bien testées donc vous faites usage de règles sans erreurs et évitez de faire de sales correctifs pour résoudre des problèmes banals
 - migrer vers CDBS est facile
 - can be used to generate Debian files (like `debian/control` for GNOME Team Uploaders inclusion)
 - CDBS est facilement extensible
 - Il l'0>< !!!
-

Chapitre 2

Premiers pas

2.1 Convertir un paquet à CDBS

Converting to CDBS is easy; A simple `debian/rules` for a C/C++ software with no extra rules would be written as this :

```
#!/usr/bin/make -f

include /usr/share/cdb/1/rules/debhelper.mk
include /usr/share/cdb/1/class/autotools.mk
```

No, i'm not joking, this is sufficient to handle autotools management, like updating `config.{guess|sub}`, cleanup temp files after build and launch all common debhelper stuff.

Just use compat level 7 (or lower if needed, but not lower than 4 or it may not work), create your `pkg.install`, `pkg.info`, etc as you usually do with `dh_*` scripts, and CDBS would call them if necessary, autodetecting a lot of things. In case of a missing compat information, CDBS would create a `debian/compat` file with compatibility level 7. If you are using an obsolete `DH_COMPAT` variable in your `debian/rules`, you should get rid of it. In this case, or in case you would like CDBS not to create a `debian/compat` file, you can disable this feature by setting `DEB_DH_COMPAT_DISABLE` to a non-void value.



Important

If `debian/control` management is activated (see below), build dependency on **cdbs** is automatically added, if not, you will have to do it yourself.



AVERTISSEMENT

Beware your working directory *MUST NOT* have spaces or CDBS would probably fail; see [#306941](#)

2.2 Réglages de base et variables disponibles

Remember you can get the `pkg` directory using the `CURDIR` variable.

You can change common build parameters this way :

```
# where sources are
DEB_SRCDIR = $(CURDIR)/src
# in which directory to build
DEB_BUILDDIR = $(DEB_SRCDIR)/build
```

```
# in which directory to install the software
DEB_DESTDIR = $(CURDIR)/plop/
```

Some various variables you can use in `debian/rules` :

DEB_SOURCE_PACKAGE	nom du paquet source
DEB_VERSION	version Debian complète
DEB_NOEPOCH_VERSION	version Debian sans epoch
DEB_UPSTREAM_VERSION	version amont
DEB_ISNATIVE	non-vide si le paquet est natif
DEB_ALL_PACKAGES	liste de tous les paquets binaires
DEB_INDEP_PACKAGES	liste des paquets binaires indépendants de l'architecture
DEB_ARCH_PACKAGES	liste des paquets binaires dépendants de l'architecture
DEB_PACKAGES	liste des paquets binaires normaux (non-udeb)
DEB_UDEB_PACKAGES	liste des paquets binaires udeb, s'ils existent
DEB_HOST_GNU_TYPE	type GNU de la machine hôte
DEB_HOST_GNU_SYSTEM	partie système de type GNU de la machine hôte
DEB_HOST_GNU_CPU	partie CPU du type GNU de la machine hôte
DEB_HOST_ARCH	nom de l'architecture Debian sur la machine hôte
DEB_HOST_ARCH_CPU	partie CPU du nom de l'architecture Debian sur la machine hôte
DEB_HOST_ARCH_OS	partie OS du nom de l'architecture Debian sur la machine hôte
DEB_BUILD_GNU_TYPE	type GNU pour cette construction
DEB_BUILD_GNU_SYSTEM	partie système du type GNU pour cette construction
DEB_BUILD_GNU_CPU	partie CPU du type GNU pour cette construction
DEB_BUILD_ARCH	nom de l'architecture Debian pour cette construction
DEB_BUILD_ARCH_CPU	partie CPU du nom de l'architecture Debian pour cette construction
DEB_BUILD_ARCH_OS	partie OS du nom de l'architecture Debian pour cette construction
DEB_ARCH	vieux nom de l'architecture Debian
	<i>! deprecated, only use to provide backward compatibility</i>
	<i>! deprecated, only use to provide backward compatibility</i>
	(voir la page de manuel de <code>dpkg-architecture</code> pour plus d'information)

TABLE 2.1 – Common variables available in `debian/rules`

2.3 Règles de construction simples et sur-mesures



AVERTISSEMENT

Prenez soin d'ajouter les règles *après* les inclusions CDBS nécessaires.

Suppose you want custom rules for the source package `foo`, creating `foo (arch-dep)` and `foo-data (arch-indep)`, you simply need to add some lines to `debian/rules`.

To add pre-configure actions :

```
makebuilddir/foo::
    ln -s plop plop2
```

To add post-configure actions :

```
configure/foo::
    sed -ri 's/PLOP/PLIP/' Makefile

configure/foo-data::
    touch src/z.xml
```

/!\ in this case we are talking about package configuration and NOT about a configure script made with autotools.

To add post-build actions :

```
build/foo::
    /bin/bash debian/scripts/toto.sh

build/foo-data::
    $(MAKE) helpfiles
```

To add post-install actions :

```
install/foo::
    cp debian/tmp/myfoocmd debian/foo/foocmd
    find debian/foo/ -name "CVS" -depth -exec rm -rf {} \;

install/foo-data::
    cp data/*.png debian/foo-data/usr/share/foo-data/images/
    dh_stuff -m ipot -f plop.bz3 debian/foo-data/libexec/
```

To add post deb preparation actions :

```
binary/foo::
    strip --remove-section=.comment --remove-section=.note --strip-unnneeded \
        debian/foo/usr/lib/foo/totoz.so
```

To add pre-clean actions :

```
cleanbuilddir/foo::
    rm -f debian/fooman.1
```

2.4 Options de compilation usuelles

Default optimization is set using `DEB_OPT_FLAG` which default to `"-O2"`; you can override it. `CFLAGS` and `CXXFLAGS` are set to `"-g -Wall $(DEB_OPT_FLAG)"`, `CPPFLAGS` is untouched from environment, but you can override these settings on a per-package basis using `CFLAGS_package`, `CXXFLAGS_package`, and `CPPFLAGS_package` variables.

`DEB_BUILD_OPTIONS` is a well known Debian environment variable, not a CDBS one, containing special build options (a comma-separated list of keywords). CDBS does check `DEB_BUILD_OPTIONS` to take these options into account; see details in each class.

2.5 Les astuces de Debhelper

2.5.1 Ne pas s'occuper de Debhelper

Oui, CDBS fait presque tout pour vous :-).

Just add this line to the beginning of your `debian/rules` file :

```
include /usr/share/cdb/1/rules/debhelper.mk
```

CDBS debhelper rules handle the following Debhelper scripts for each binary package automatically : Other Debhelper scripts

dh_builddeb	dh_installcatalogs	dh_installdocs	dh_installogrotate	dh_link
dh_clean	dh_installchangelogs	dh_installemacsen	dh_installman	dh_makeshlibs
dh_compress	dh_installdoc	dh_installexamples	dh_instalmenu	dh_md5sums
dh_fixperms	dh_installdeb	dh_installinfo	dh_instalmime	dh_perl
dh_prep	dh_gencontrol	dh_installdebconf	dh_instalinit	dh_installpam
dh_shlibdeps	dh_install	dh_installdirs	dh_installogcheck	dh_instaludev
dh_strip				

TABLE 2.2 – Scripts Debhelper communément gérés

can be handled in specific classes or may be called in custom rules.



Important

If `debian/control` management is activated (see below), build dependency on **debhelper** is automatically added, if not, you will have to do it yourself.

Having a versioned dependency on **debhelper** is recommended as it will ensure people will use the version providing the necessary features (CDBS `debian/control` management will do it).

2.5.2 Paramètres de Debhelper

The following parameters allow debhelper calls customization while most common calls are still handled without writing any rule. Some of them apply on all binary packaged, like `DEB_INSTALL_DOCS_ALL`, and some apply only to a specific package, like `DEB_SHLIBDEPS_LIBRARY_pkg` (where *pkg* is the name of the binary package). Read the comments in `/usr/share/cdb/1/rules/debhelper.mk` for a complete listing. Some non-exhaustive examples follow.

To specify a tight dependency on a package containing shared libraries:

```
DEB_DH_MAKESHLIBS_ARGS_libfoo := -V"libfoo (>= 0.1.2-3)"
DEB_SHLIBDEPS_LIBRARY_arkrpg := libfoo
DEB_SHLIBDEPS_INCLUDE_arkrpg := debian/libfoo/usr/lib/
```

To install a changelog file with an uncommon name as `ProjectChanges.txt.gz`:

```
DEB_INSTALL_CHANGELOGS_ALL := ProjectChanges.txt
```

To avoid compressing files with `.py` extension :

```
DEB_COMPRESS_EXCLUDE := .py
```

To register a debug library package `libfoo-dbg` for `libfoo` (which needs unstripped `.so`) in compat mode 4:

```
DEB_DH_STRIP_ARGS := --dbg-package=libfoo
```

Starting from compat mode 5, CDBS automatically detect `-dbg` packages and pass the needed arguments to `dh_strip`; `DEB_DH_STRIP_ARGS` can still be useful to pass additional parameters like excluded items (`--exclude=item`).

Perl-specific debhelper options (`dh_perl` call is always performed):

```
# Add a space-separated list of paths to search for perl modules
DEB_PERL_INCLUDE := /usr/lib/perl-z
# Like the above, but for the 'libperl-stuff' package
DEB_PERL_INCLUDE_libperl-stuff := /usr/lib/perl-plop

# Overrides options passed to dh_perl
DEB_DH_PERL_ARGS := -d
```

To avoid loosing temporary generated files in `dh_clean` processing (rarely useful):

```
# files containing these pattern would not be deleted
# (beware CDBS~changelog has a typo while highlighting new DEB_CLEAN_EXCLUDE*S* feature)
DEB_CLEAN_EXCLUDE := precious keep
```

2.5.3 Règles personnalisées de construction pour debhelper

Les règles de CDBS pour debhelper ajoutent aussi des règles plus judicieuses pour la construction du paquet.

To add post deb preparation (including debhelper stuff) actions :

```
binary-install/foo::
    chmod a+x debian/foo/usr/bin/pouet
```

To add post clean actions :

```
clean::
    rm -rf plop.tmp
```

D'autres règles existent, mais nous ne les avons pas testé :

- `binary-strip/foo` (appelé après la suppression des symboles de débogage)
- `binary-fixup/foo` (appelé après la compression et la correction des permissions)
- `binary-predeb` (appelé juste avant la construction du `.deb`)

Chapitre 3

Tâches courantes

3.1 Appliquer une rustine (en utilisant simple-patchsys)

Tout d'abord, l'application de rustines directement dans les sources est MAL™, donc vous avez besoin d'une façon d'appliquer les rustines sans toucher aux fichiers. Ces règles, inspirées pas le système Dpatch, sont très similaires et efficaces. Tout ce que vous avez à connaître est l'usage de diff et patch, CDBS s'occupe du reste.

C'est très dur, donc écoutez attentivement et préparez vous pour l'examen.

First, add this line to your `debian/rules` :

```
include /usr/share/cdb/1/rules/simple-patchsys.mk
```

And then use it !

Create the directory `debian/patches` and put your patches in it. Files should be named so as to reflect in which order they have to be applied, and must finish with the `.patch` or `.diff` suffix. The class would take care of patching before configure and unpatch after clean. It is possible to use patch level 0 to 3, and CDBS would try them and use the right level automatically. The system can handle compressed patch with additional `.gz` or `.bz2` suffix and uu-encoded patches with additional `.uu` suffix.

You can customize the directories where patches are searched, and the suffix like this : (defaults are: `.diff .diff.gz .diff.bz2 .diff.uu .patch .patch.gz .patch.bz2 .patch.uu`)

```
DEB_PATCHDIRS := debian/mypatches
DEB_PATCH_SUFFIX := .plop
```

In case of errors when applying, for example `debian/patches/01_hurd_ftbfs_pathmax.patch`, you can read the log for this patch in `debian/patches/01_hurd_ftbfs_pathmax.patch.level-0.log` ('0' because a level 0 patch).



Important

If `debian/control` management is activated (see below), build dependency on **patchutils** is automatically added, if not, you will have to do it yourself.

3.2 Appliquer des rustines (en utilisant dpatch)

To use Dpatch as an alternative to the CDBS included patch system, just add his line to your `debian/rules` :

```
include /usr/share/cdb/1/rules/dpatch.mk
# needed to use the dpatch tools (like dpatch-edit-patch)
include /usr/share/dpatch/dpatch.make
```

Now you can use Dpatch as usual and CDBS would call it automatically.

**AVERTISSEMENT**

Vous devriez inclure `dpatch.mk` *APRÈS* `autotools.mk` ou `gnome.mk` pour que l'extension `dpatch` fonctionne correctement.

**Important**

If `debian/control` management is activated (see below), build dependency on **dpatch** and **patchutils** is automatically added, if not, you will have to do it yourself.

3.3 Appliquer des rustines (en utilisant quilt)

To use **Quilt** as an alternative to the CDBS included patch system, just add his line to your `debian/rules` :

```
include /usr/share/cdb/1/rules/patchsys-quilt.mk
```

Now you can use Quilt as usual and CDBS would call it automatically.

**Important**

If `debian/control` management is activated (see below), build dependency on **quilt** and **patchutils** is automatically added, if not, you will have to do it yourself.

3.4 Gestion automatique des tarball

To use the CDBS tarball system, just add his line to your `debian/rules`, and specify the name of the top directory of the extracted tarball :

```
include /usr/share/cdb/1/rules/tarball.mk
```

```
DEB_TAR_SRCDIR := foosoft
```

CDBS will recognize tarballs with the following extensions: `.tar` `.tgz` `.tar.gz` `.tar.bz` `.tar.bz2` `.zip`

The tarball location is autodetected if in the top source directory, or can be specified :

```
DEB_TARBALL := $(CURDIR)/tarballdir/mygnustuff_beta-1.2.3.tar.gz
```

CDBS will handle automatic uncompression and cleanups, automatically set `DEB_SRCDIR` and `DEB_BUILDDIR` for you, and integrate well with other CDBS parts (like `autotools` class).

Moreover, if you want sources to be cleaned up from dirty SCM-specific dirs and file, just add this at the top of your `debian/rules`, before any `include` :

```
DEB_AUTO_CLEANUP_RCS := yes
```

**AVERTISSEMENT**

The `DEB_AUTO_CLEANUP_RCS` feature has been removed for no good reason since version 0.4.39. Feel free to bugreport if you want it resurrected.

**Important**

If needed, and if `debian/control` management is activated (see below), build dependency on **bzip2** or **unzip** is automatically added, if not, you will have to do it yourself.

Chapitre 4

Personnalisation avancée

4.1 debian/control management

Attention

Automatic `debian/control` generation using any tool is permitted into Debian as long as it is triggered manually by the developer and the latter checks the result carefully.



Autogenerating `debian/control` without any human intervention could be harmful in some ways detailed in [#311724](#). This is not allowed in Debian.

We then urge you to avoid using `DEB_AUTO_UPDATE_DEBIAN_CONTROL` directly and instead invoke the autogeneration rules manually after you modified `debian/control.in` (this way users or builddds wouldn't have different Build-Depends when building, avoiding many problems). Do not forget to proofread the result before any upload.

Manual `debian/control` regeneration:

```
DEB_AUTO_UPDATE_DEBIAN_CONTROL=yes fakeroot debian/rules clean
```

Cette fonction permet :

- à CDBS de gérer automatiquement certains « build-related Build-Depends »
 - l'utilisation de commandes shell intégrées.
 - use of CPU and System criterias to specify architecture (*EXPERIMENTAL*)
- « Build-related Build-Depends » sont des dépendances introduites avec l'utilisation de certaines fonctions CDBS, ou de besoins détectés automatiquement.

Embedded shell commands allows including hacks like :

```
Build-Depends: libgpm-dev ['type-handling any linux-gnu']
```

CPU and System criterias implements support for Cpu/System fields, as a replacement for the Architecture field (which is to be implemented in `dpkg` in the long term, but still *EXPERIMENTAL*). Here is an exemple, before :

```
Architecture: all
```

and after :

```
Cpu: all
System: all
```

If these fields are used, it is also possible to include special tags to easily take advantage of the **type-handling** tool, like in this example :

```
Build-Depends: @cdbse, procps [system: linux], plop [cpu: s390]
```

(look at the **type-handling** package documentation, for more information)

Voici la recette :

1. Rename `debian/control` into `debian/control.in`.
2. Replace `cdbs / debhelper / ... Build-Depends` with `@cdbs@` in your `debian/control.in` like this :

```
Build-Depends-Indep: @cdbs@, python-dev (>= 2.3), python-soya (>= 0.9), \
    python-soya (<< 0.10), python-openal(>= 0.1.4-4), gettext
```

3. Then manually (re)generate `debian/control` as explained above (see the caution part).

4.2 Utilisation de la classe Makefile

Cette classe est destinée à ceux qui ont uniquement un fichier Makefile (aucun autotools disponible) pour construire le programme. Vous avez uniquement besoin de quatre règles dans le Makefile :

- une pour nettoyer le répertoire de construction (c'est-à-dire « mrproper »)
- une pour construire votre logiciel (c'est-à-dire « myprog »)
- une pour vérifier que votre logiciel fonctionne correctement (c'est-à-dire « check »)
- une pour installer votre logiciel (c'est-à-dire « install »)

Pour être honnête, la présence des règles d'installation n'est pas essentielle, mais cela aide toujours beaucoup lorsque vous les avez.

The first operation, is to write the `debian/rules`. First, we add the include lines:

```
include /usr/share/cdbs/1/class/makefile.mk
```

Now, it remains to tell `cdbs` the name of our four Makefile rules. For the previous examples it gives:

```
DEB_MAKE_CLEAN_TARGET      := mrproper
# if you detect authors's loss of sanity, tell CDBS not to try running the nonexistent ↔
# clean rule, and do the job yourself in debian/rules
DEB_MAKE_CLEAN_TARGET      :=
DEB_MAKE_BUILD_TARGET       := myprog
DEB_MAKE_INSTALL_TARGET     := install DESTDIR=$(CURDIR)/debian/tmp/
# no check for this software
DEB_MAKE_CHECK_TARGET      :=

# allow changing the makefile filename in case of emergency exotic practices
DEB_MAKE_MAKEFILE          := MaKeFiLe
# example when changing environnement variables is necessary :
DEB_MAKE_ENVVARS           := CFLAGS="-fomit-frame-pointer"
```

`DEB_BUILD_OPTIONS` is checked for the following options :

- `noopt`: utilise `-O0` au lieu de `-O2`
- `nocheck`: skip the check rule

If your Makefile doesn't support the `DESTDIR` variable, take a look in it and find the variable responsible for setting installation directory. If you don't find some variable to do this, you'll have to patch the file...

C'est tout :)

4.3 Utilisation de la classe Autotools

This class is able to use configure scripts and makefiles generated with autotools (and possibly libtool). All rules are called automatically and clean rules to remove generated files during build are also added. This class in fact improves the makefile class to support autotools features and provide good defaults.

To use it, just add this line to your `debian/rules`

```
include /usr/share/cdb/1/class/autotools.mk
```

CDBS automatically handles common flags to pass to the configure script, but it is possible to give some extra parameters :

```
DEB_CONFIGURE_EXTRA_FLAGS := --with-ipv6 --with-foo
```

If the build system uses non-standard configure options you can override CDBS default behavior :

```
COMMON_CONFIGURE_FLAGS := --program-dir=/usr
```

(notice that `DEB_CONFIGURE_EXTRA_FLAGS` would still be appended)

If some specific environment variables need to be setup, use :

```
DEB_CONFIGURE_SCRIPT_ENV += BUILDOPT="someopt"
```



AVERTISSEMENT

Prefer use of `+=` instead of `:=` not to override other environment variables (`CC` / `CXX` / `CFLAGS` / `CXXFLAGS` / `CPPFLAGS` / `LDFLAGS`) propagated in the CDBS default.

CDBS will automatically update `config.sub`, `config.guess`, and `config.rpath` before build and restore the old ones at clean stage (even if using the tarball system). If needed, and if `debian/control` management is activated, **autotools-dev** and/or **gnulib** will then be automatically added to the build dependencies (needed to find updated versions of the files).

If the program does not use the top source directory to store autoconf files, you can teach CDBS where it is to be found :

```
DEB_AC_AUX_DIR = $(DEB_SRCDIR)/autoconf
```

CDBS can be asked to update `libtool`, `autoconf`, and `automake` files, but this behavior is likely to break the build system and is **STRONGLY** discouraged. Nevertheless, if you still want this feature, set the following variables :

- `DEB_AUTO_UPDATE_LIBTOOL`: `pre` to call `libtoolize`, or `post` to copy system-wide **libtool** after configure is done
- `DEB_AUTO_UPDATE_ACLOCAL`: **aclocal** version to use
- `DEB_AUTO_UPDATE_AUTOCONF`: **autoconf** version to use
- `DEB_AUTO_UPDATE_AUTOHEADER`: **autoheader** version to use
- `DEB_AUTO_UPDATE_AUTOMAKE`: **automake** version to use
- `DEB_AUTOMAKE_ARGS`: extra arguments to **automake** call

(les dépendances de construction correspondantes seront ajoutées automatiquement)

The following make parameters can be overridden :

```
# these are the defaults CDBS provides
DEB_MAKE_INSTALL_TARGET := install DESTDIR=$(DEB_DESTDIR)
DEB_MAKE_CLEAN_TARGET := distclean
DEB_MAKE_CHECK_TARGET :=

# example to work around dirty makefile
DEB_MAKE_INSTALL_TARGET := install prefix=$(CURDIR)/debian/tmp/usr

# example with unexistent install rule for make
DEB_MAKE_INSTALL_TARGET :=

# example to activate check rule
DEB_MAKE_CHECK_TARGET := check

# overriding make-only environment variables :
# (should never be necessary in a clean build system)
# (example borrowed from the bioapi package)
DEB_MAKE_ENVVARS := "SKIPCONFIG=true"
```

DEB_BUILD_OPTIONS is checked for the following options :

- noopt: utilise -O0 au lieu de -O2
- nocheck: skip the check rule

If you are using CDBS version < 0.4.39, it automatically cleans autotools files generated during build (config.cache, config.log, and config.status). Since version 0.4.39, CDBS leave them all considering it is not his job to correct an upstream buildsys misbehavior (but you may remove them in the clean rule if necessary before you get the issue solved by authors).

4.4 Utilisation de la classe Perl

This class can manage standard perl build and install with MakeMaker method.

To use this class, add this line to your debian/rules :

```
include /usr/share/cdb/1/class/perlmodule.mk
```

Optionally, it can take care of using dh_perl, depending the debhelper class is declared before the perl class or not.

Install path defaults to 'first_pkg/usr' where first_pkg is the first package in debian/control.

You can customize build options like this :

```
# change MakeMaker defaults (should never be usefull)
DEB_MAKE_BUILD_TARGET := build-all
DEB_MAKE_CLEAN_TARGET := realclean
DEB_MAKE_CHECK_TARGET :=
DEB_MAKE_INSTALL_TARGET := install PREFIX=debian/stuff

# add custom MakeMaker options
DEB_MAKEMAKER_USER_FLAGS := --with-ipv6
```

Common makefile or general options can still be overridden: DEB_MAKE_ENVVARS, DEB_BUILDDIR (must match DEB_SRCDIR for Perl)

Référez-vous aux options de debhelper spécifiques à Perl décrites ci-dessus.



Important

If debian/control management is activated (see below), build dependency on **perl** is automatically added, if not, you will have to do it yourself.

4.5 Utilisation de la classe Python

This class can manage common Python builds using **distutils** automatically.



AVERTISSEMENT

Since 0.4.53, this class does not handle old-policy packages (pre-Etch) anymore.

With the new policy all versionned packages (python_{ver-app}) are collapsed into a single package (python-_{app}). The class is able to move python scripts and .so files in the new locations automatically. You can use the auto control file generation feature to ensure your Build-Depends are set correctly for the new needed tools.

To use this class, add these lines to your debian/rules :

```
# select the python system you want to use : pysupport or pycentral
# (this MUST be done before including the class)
DEB_PYTHON_SYSTEM = pysupport
include /usr/share/cdb/1/class/python-distutils.mk
```

The class also takes care of calling `dh_pysupport` or `dh_pycentral`.

You can customize build options like this :

```
##### These variables are additions for the new policy

# packages holding the collapsed content of all supported versions
# CDBS defaults to the first non -doc/-dev/-common package listed in "debian/control"
# this variable allows to override automatic detection
DEB_PYTHON_MODULE_PACKAGES = mypyapp

# list of private modules private directoris (needed to automatically handle ↔
# bytecompilation)
DEB_PYTHON_PRIVATE_MODULES_DIRS = /usr/share/mypyapp/my-pv-module

# overrides the default Python installation root directory
DEB_PYTHON_DESTDIR = $(CURDIR)/debian/python-stuff

##### These variables have not changed (same in the old and new policy)

# change the Python build script name (default is 'setup.py')
DEB_PYTHON_SETUP_CMD := install.py

# clean options for the Python build script
DEB_PYTHON_CLEAN_ARGS = -all

# build options for the Python build script
DEB_PYTHON_BUILD_ARGS = --build-base="$(DEB_BUILDDIR)/specific-build-dir"

# common additional install options for all binary packages
# ('--root' option is always set)
DEB_PYTHON_INSTALL_ARGS_ALL = --no-compile --optimize --force
```

You may use some read-only (meaning you **MUST NOT** alter them) variables in your `debian/rules` :

- `cdbs_python_current_version`: current python version number (defined only if selected package is a module)
- `cdbs_python_build_versions`: list of space separated version numbers for which the selected module/extension is gonna be built
- `cdbs_python_destdir`: Python installation root directory (works even when you didn't used `DEB_PYTHON_DESTDIR`)
- `cdbs_python_packages`: list of all Python packages
- `cdbs_python_arch_packages`: list of all Python architecture dependent packages
- `cdbs_python_indep_packages`: list of all Python architecture independent packages

Complete `debian/rules` example using `python-support` for a module (editobj):

```
#!/usr/bin/make -f
# -*- mode: makefile; coding: utf-8 -*-

include /usr/share/cdb/1/rules/debhelper.mk
DEB_PYTHON_SYSTEM = pysupport
include /usr/share/cdb/1/class/python-distutils.mk
include /usr/share/cdb/1/rules/patchsys-quilt.mk

DEB_COMPRESS_EXCLUDE := .py

$(patsubst %,install/%,$(cdbs_python_packages))
```

```

mv debian/$(cdbs_curpkg)/usr/lib/python*/site-packages/editobj/icons \
    debian/$(cdbs_curpkg)/usr/share/$(cdbs_curpkg)

$(patsubst %,binary-install/%,$(cdbs_python_packages))
find debian/$(cdbs_curpkg)/usr/share/ -type f -exec chmod -R a-x {} \;

```

AVERTISSEMENT



Do not use the `DEB_PYTHON_MODULE_PACKAGE` variable anymore, it has been obsoleted. Use the `DEB_PYTHON_MODULE_PACKAGES` to force the list of module packages, or `cdbs_python_packages` variable in rules generation.

Do not use the `cdbs_python_support_path` and `cdbs_python_module_arch` variables anymore, they have been obsoleted. `cdbs_python_support_path` is not useful anymore, just install files in the standard way and `python-support` would do the rest (messing into `python-support` internals was only a workaround when it was incomplete). `cdbs_python_module_arch` can easily be replaced by a rule rewrite or a membership test using `cdbs_python_arch_packages` or `cdbs_python_indep_packages`.

4.6 Utilisation de la classe Ruby setup.rb

Cette classe peut gérer automatiquement les programmes d'installation `setup.rb` usuels.

To use this class, add this line to your `debian/rules` :

```
include /usr/share/ruby-pkg-tools/1/class/ruby-setup-rb.mk
```

Optionally, it can take care of using `dh_rdoc`, to generate and install Rdoc documentation, depending the debhelper class is declared before the `ruby setup.rb` class or not.

Most ruby packages are architecture all, and then don't need being build for multiple ruby versions; your package should then be called `'libfoo-ruby'` or `'foo'` and CDBS would automatically use the current Debian ruby version to build it. If your package contains a compiled part or a binding to an external lib, then you will have packages named `'libfoo-ruby1.6'`, `'libfoo-ruby1.8'`, and so on, then CDBS would automatically build each package with the corresponding ruby version. In this case, don't forget to add a `'libfoo-ruby'` convenience dummy package depending on the current Debian ruby version. If you have documentation you want split into a separate package, then call it `'libfoo-ruby-doc'`. If this is Rdoc documentation, you may want to include the debhelper class, as explained before, to have it generated and installed automatically.

You can customize build options like this :

```

# force using a specific ruby version for build
# (should not be necessary)
DEB_RUBY_VERSIONS := 1.9

# use ancestor
DEB_RUBY_SETUP_CMD := install.rb

# config options for the ruby build script
# (older setup.rb used --site-ruby instead of --siteruby)
DEB_RUBY_CONFIG_ARGS = --site-ruby=/usr/lib/ruby/1.9-beta

```

4.7 Utilisation de la classe de l'équipe Debian Ruby Extras

Si vous appartenez à l'équipe Ruby Extras, ou que vous avez une équipe comme « uploaders », et que maintenir la liste des développeurs vous ennuie, cette classe est faite pour vous.

To use this class, add this line to your `debian/rules` :

```
include /usr/share/ruby-pkg-tools/1/rules/uploaders.mk
```

Rename your `debian/control` file to `debian/control.in` and run the clean rule (`./debian/rules clean`) to regenerate the `debian/control` file, replacing the `@RUBY_TEAM@` tag with the list of developers automatically.

4.8 Utilisation de la classe GNOME

This class adds a make environment variable : `GCONF_DISABLE_MAKEFILE_SCHEMA_INSTALL = 1` ("This is necessary because the Gconf schemas have to be registered at install time. In the case of packaging, this registration cannot be done when building the package, so this variable disable schema registration in **make install**. This procedure is deferred until `gconftool-2` is called in `debian/postinst` to register them, and in `debian/prerm` to unregister them. The `dh_gconf` script is able to add the right rules automatically for you.")

It can handle the following Debhelper scripts automatically :

<code>dh_desktop</code>	<code>dh_gconf</code>	<code>dh_icons</code>
-------------------------	-----------------------	-----------------------

TABLE 4.1 – Scripts Debhelper gérés par la classe GNOME

Moreover it adds some more clean rules to remove:

- fichiers générés par intltool
- scrollkeeper generated files (left over `.omf.out` files in `doc` and `help` directories)

To use it, just add this line to your `debian/rules`, after the debhelper class include :

```
include /usr/share/cdb/1/class/gnome.mk
```

Pour de plus amples informations sur les règles d’empaquetage spécifiques à GNOME, référez-vous à la [Politique d’empaquetage Debian GNOME](#) (en anglais).

4.9 Utilisation de la classe de l’équipe Debian GNOME

Si vous appartenez à l’équipe GNOME, ou si vous avez l’équipe comme « uploaders », et que maintenir la liste des développeurs vous ennuie, cette classe est faite pour vous.

To use this class, add this line to your `debian/rules` :

```
include /usr/share/gnome-pkg-tools/1/rules/uploaders.mk
```

Rename your `debian/control` file to `debian/control.in` and run the clean rule (`./debian/rules clean`) to regenerate the `debian/control` file, replacing the `'@GNOME_TEAM@'` tag with the list of developers automatically.

AVERTISSEMENT

If you are using the `debian/control` file management described below, please note this class will override this feature. To cope with this problem, allowing at least Build-Depends handling, use the following work-around (until it is solved in a proper way) :



```
# deactivate debian/control file management
#DEB_AUTO_UPDATE_DEBIAN_CONTROL := yes

# ...
# includes and other stuff
# ...

clean::
    sed -i "s/@cdbs@/$(CDBS_BUILD_DEPENDS)/g" debian/control
    # other clean stuff
```

4.10 Utilisation de la classe KDE

To use this class, add this line to your `debian/rules` file :

```
include /usr/share/cdb/1/class/kde.mk
```

CDBS exporte automatiquement les variables suivantes avec la valeur correcte :

- `kde_cgidir` (`/usr/lib/cgi-bin`)
- `kde_confdir` (`/etc/kde3`)
- `kde_html_dir` (`/usr/share/doc/kde/HTML`)

`DEB_BUILDDIR`, `DEB_AC_AUX_DIR` and `DEB_CONFIGURE_INCLUDEDIR` are set to KDE defaults.

Les fichiers suivants sont exclus de la compression :

- `.dcl`
- `.docbook`
- `-license`
- `.tag`
- `.sty`
- `.el`

(take care of them if you override the `DEB_COMPRESS_EXCLUDE` variable)

Il peut gérer des options de configuration spécifiques à KDE (sans oublier de désactiver `rpath` et activer `xinerama`), définir correctement le répertoire `autotools`, et lancer les règles `make` de manière adéquate.

You can enable APIDOX build by setting the `DEB_KDE_APIDOX` variable to a non-void value.

you can enable the final mode build by setting `DEB_KDE_ENABLE_FINAL` variable to a non-void value.

`DEB_BUILD_OPTIONS` is checked for the following options :

- `noopt`: disable optimisations (and KDE final mode, overriding `DEB_KDE_ENABLE_FINAL`)
- `nostrip`: enable KDE debug (and disable KDE final mode, overriding `DEB_KDE_ENABLE_FINAL`)

You can prepare the build using the 'buildprep' convenience target: **fakeroot debian/rules buildprep** (which is in fact calling the dist target of `admin/Makefile.common`).

4.11 Utilisation de la classe Ant

(Ant est un outil de construction basé sur java)

To use this class, add this include to your `debian/rules` and set the following variables :

```
include /usr/share/cdb/1/class/ant.mk

# Définissez soit un unique (JAVA_HOME) ou de multiples (JAVA_HOME_DIRS) emplacements java
JAVA_HOME~:= /usr/lib/kaffe
# ou définissez JAVACMD si vous n'utilisez pas le chemin «~<JAVA_HOME>/bin/java~» par ↔
défaut
#JAVACMD~:= /usr/bin/java

# Définissez un emplacement Ant
ANT_HOME~:= /usr/share/ant-cvs
```

You may add additional JARs like in the following example :

```
# list of additional JAR files ('.jar' extension may be omitted)
# (path must be absolute or relative to '/usr/share/java')
DEB_JARS := /usr/lib/java-bonus/ldap-connector adml-adapter.jar
```

The property file defaults to `debian/ant.properties`.

You can provide additional JVM arguments using `ANT_OPTS`. You can provide as well additional Ant command line arguments using `ANT_ARGS` (global) and/or `ANT_ARGS_pkg` (for package `pkg`), thus overriding the settings in `build.xml` and the property file.

CDBS will build and clean using defaults target from `build.xml`. To override these rules, or run the install / check rules, set the following variables to your needs :

```
# override build and clean target
DEB_ANT_BUILD_TARGET = makeitrule
DEB_ANT_CLEAN_TARGET = super-clean
# i want install and test rules to be run
DEB_ANT_INSTALL_TARGET = install-all
DEB_ANT_CHECK_TARGET = check
```

`DEB_BUILD_OPTIONS` is checked for the following options :

– `noopt`: définir l’option « `compile.optimize` » Ant à « `false` » (faux)

Vous devriez pouvoir obtenir plus d’informations sur ces outils de construction basées sur java sur le [site web Ant Apache](#) . (en anglais).

4.12 Utilisation de la classe HBuild

(HBuild est le mini-distutils Haskell)

CDBS can take care of `-hugs` and `-ghc` packages: invoke `Setup.lhs` properly for clean and install part.

To use this class, add this line to your `debian/rules` :

```
include /usr/share/cdb/1/class/hbuild.mk
```

Vous devriez pouvoir obtenir plus d’informations sur les distutils Haskell dans [ce fil de discussion](#) (en anglais).

4.13 Using the CMake class

This class is intended to handle build systems using [CMake](#).

To use this class, add this line to your `debian/rules` :

```
include /usr/share/cdb/1/class/cmake.mk
```

This class is an extension of the Makefile class, calling **cmake** in `DEB_SRCDIR` with classic parameters and then calling **make**. In the call to **cmake**, the install prefix and path to `CC/CXX` as well as `CFLAGS/CXXFLAGS` are given automatically. You can pass extra arguments using `DEB_CMAKE_EXTRA_FLAGS`, and in strange cases where you need a special configuration you can override the default arguments using `DEB_CMAKE_NORMAL_ARGS`.

4.14 Using the qmake class

This class is intended to handle build systems using [qmake](#), mostly used to build Qt applications.

To use this class, add this line to your `debian/rules` :

```
include /usr/share/cdb/1/class/qmake.mk
```

This class is an extension of the Makefile class, calling **qmake** in `DEB_BUILDDIR` with classic parameters and then calling **make**. In the call to **qmake**, the path to `CC/CXX` as well as `CPPFLAGS/CFLAGS/PPFLAGS/CXXFLAGS` are given automatically. Configuration options can be given using `DEB_QMAKE_CONFIG_VAL` (resulting in adding content to `CONFIG`), the later defaulting to `nostrip` if this option is set in `DEB_BUILD_OPTIONS`.

Chapitre 5

Panthéon des exemples

5.1 Exemple de GNOME + autotools + patchsys simple

(exemple du paquet « gnome-panel »)

debian/control.in:

```
Source: gnome-panel
Section: gnome
Priority: optional
Maintainer: Marc Dequènes (Duck) <Duck@DuckCorp.org>
Uploaders: Sebastien Bacher <seb128@debian.org>, Arnaud Patard \
  <arnaud.patard@rtp-net.org>, @GNOME_TEAM@
Standards-Version: 3.6.1.1
Build-Depends: @cdbs@, liborbit2-dev (>= 2.10.2-1.1), intltool, gnome-pkg-tools, \
  libglade2-dev (>= 1:2.4.0), libwnck-dev (>= 2.8.1-3), scrollkeeper \
  (>= 0.3.14-9.1), libgnome-desktop-dev (>= 2.8.3-2), libpng3-dev, sharutils, \
  libbonobo2-dev (>= 2.8.0-3), libxmu-dev, autotools-dev, libedata-cal-dev \
  (>= 1.0.2-3)

Package: gnome-panel
Architecture: any
Depends: ${shlibs:Depends}, ${misc:Depends}, gnome-panel-data \
  (= ${Source-Version}), gnome-desktop-data (>= 2.8.1-2), gnome-session \
  (>= 2.8.1-4), gnome-control-center (>= 1:2.8.1-3)
Conflicts: gnome-panel2, quick-lounge-applet (<= 0.98-1), system-tray-applet, \
  metacity (<= 2.6.0), menu (<< 2.1.9-1)
Recommends: gnome-applets (>= 2.8.2-1)
Suggests: menu (>= 2.1.9-1), yelp, gnome2-user-guide, gnome-terminal | \
  x-terminal-emulator, gnome-system-tools
Description: launcher and docking facility for GNOME 2
 This package contains toolbar-like "panels" which can be attached to
 the sides of your X desktop, or left "floating". It is designed to be
 used in conjunction with the Gnome Desktop Environment. Many features
 are provided for use with the panels - including an application menu,
 clock, mail checker, network monitor, quick launch icons and the like.

Package: libpanel-applet2-0
Section: libs
Architecture: any
Depends: ${shlibs:Depends}
Replaces: gnome-panel (<< 2.6.0-2)
Description: library for GNOME 2 panel applets
 This library is used by GNOME 2 panel applets.
```

```

Package: libpanel-applet2-dbg
Section: libdevel
Architecture: any
Depends: libpanel-applet2-0 (= ${Source-Version})
Description: library for GNOME 2 panel applets - library with debugging symbols
  This library is used by GNOME 2 panel applets.
  .
  This package contains unstripped shared libraries. It is provided primarily
  to provide a backtrace with names in a debugger, this makes it somewhat
  easier to interpret core dumps. The libraries are installed in
  /usr/lib/debug and can be used by placing that directory in
  LD_LIBRARY_PATH.
  Most people will not need this package.

```

```

Package: libpanel-applet2-dev
Section: libdevel
Architecture: any
Depends: libpanel-applet2-0 (= ${Source-Version}), libgnomeui-dev (>= 2.7.1-1)
Replaces: gnome-panel (<< 2.6.0-2), gnome-panel-data (<< 2.6.0)
Description: library for GNOME 2 panel applets - development files
  This packages provides the include files and static library for the GNOME 2
  panel applet library functions.

```

```

Package: libpanel-applet2-doc
Section: doc
Architecture: all
Suggests: doc-base
Replaces: libpanel-applet2-dev (<= 2.0.11-1)
Description: library for GNOME 2 panel applets - documentation files
  This packages provides the documentation files for the GNOME 2 panel applet
  library functions.

```

```

Package: gnome-panel-data
Section: gnome
Architecture: all
Depends: gnome-panel (= ${Source-Version}), scrollkeeper (>= 0.3.14-9.1), \
  ${misc:Depends}
Conflicts: gnome-panel-data2, gnome-core (<< 1.5)
Replaces: gnome-desktop-data (<= 2.2.2-1), gnome-panel (<< 2.6.0-2)
Description: common files for GNOME 2 panel
  This package includes some files that are needed by the GNOME 2 panel
  (Pixmaps, .desktop files and internationalization files).

```

debian/rules:

```

#!/usr/bin/make -f

# Gnome Team
include /usr/share/gnome-pkg-tools/1/rules/uploaders.mk

include /usr/share/cdb/1/rules/debhelper.mk
# L'inclusion ce fichier nous donne un système de correctifs simple. Vous pouvez ←
  simplement
# déposer les correctifs dans «~debian/patches~», et ils seront automatiquement
# appliqués et désappliqués.
include /usr/share/cdb/1/rules/simple-patchsys.mk
# Inclure ceci nous donne un nombre de règles typique pour un programme
# GNOME, incluant la définition de GCONF_DISABLE_MAKEFILE_SCHEMA_INSTALL=1.
# Notez que cette classe hérite d'autotools.mk et de docbookxml.mk,
# donc vous n'avez pas besoin de les inclure.
include /usr/share/cdb/1/class/gnome.mk

```

```

DEB_CONFIGURE_SCRIPT_ENV~:= LDFLAGS="-Wl,-z,defs -Wl,-O1"
DEB_CONFIGURE_EXTRA_FLAGS~:= --enable-eds

# debug lib
DEB_DH_STRIP_ARGS~:= --dbg-package=libpanel-applet-2

# tight versioning
DEB_NOREVISION_VERSION~:= $(shell dpkg-parsechangelog | egrep '^Version:' | \
    cut -f 2 -d ' ' | cut -f 1 -d '-')
DEB_DH_MAKESHLIBS_ARGS_libpanel-applet2-0~:= -V"libpanel-applet2-0 \
    (>= $(DEB_NOREVISION_VERSION))"
DEB_SHLIBDEPS_LIBRARY_gnome-panel:= libpanel-applet2-0
DEB_SHLIBDEPS_INCLUDE_gnome-panel~:= debian/libpanel-applet2-0/usr/lib/

binary-install/gnome-panel::
    chmod a+x debian/gnome-panel/usr/lib/gnome-panel/*

binary-install/gnome-panel-data::
    chmod a+x debian/gnome-panel-data/etc/menu-methods/gnome-panel-data
    find debian/gnome-panel-data/usr/share -type f -exec chmod -R a-x {} \;

binary-install/libpanel-applet2-doc::
    find debian/libpanel-applet2-doc/usr/share/doc/libpanel-applet2-doc/ \
    -name ".arch-ids" -depth -exec rm -rf {} \;

clean::
    # L'équipe GNOME «~uploaders.mk~» ne devrait pas outrepasser ce comportement
    # voici un contournement~:
    sed -i "s/@cdbs@/$(CDBS_BUILD_DEPENDS)/g" debian/control
    # cleanup not done by buildsys
    -find . -name "Makefile" -exec rm -f {} \;

```

5.2 Exemple Python

(exemple de « python-dice », un paquet non-officiel de DC)

debian/control.in:

```

Source: python-dice
Section: python
Priority: optional
Maintainer: Marc Dequènes (Duck) <Duck@DuckCorp.org>
Standards-Version: 3.6.1.1
Build-Depends: @cdbs@, python2.3-dev, python2.4-dev, swig, libdice2-dev \
    (>= 0.6.2.fixed.1)

Package: python-dice
Architecture: all
Depends: python2.3-dice
Description: python bindings for dice rolling and simulation library
PyDice is a python module for dice rolling and simulation (using fuzzy
logic).
.
It provides a Python API to the libdice2 library.
.
This is a dummy package automatically selecting the current Debian
python version.

```

```
Package: python2.3-dice
Architecture: any
Depends: ${python:Depends}
Description: python bindings for dice rolling and simulation library
  PyDice is a python module for dice rolling and simulation (using fuzzy
  logic).
  .
  It provides a Python API to the libdice2 library.
```

```
Package: python2.4-dice
Architecture: any
Depends: ${python:Depends}
Description: python 2.4 bindings for dice rolling and simulation library
  PyDice is a python module for dice rolling and simulation (using fuzzy
  logic).
  .
  It provides a Python 2.4 API to the libdice2 library.
```

```
debian/rules:
```

```
#!/usr/bin/make -f
```

```
include /usr/share/cdb/1/rules/debhelper.mk
include /usr/share/cdb/1/class/python-distutils.mk
```

5.3 Exemple Makefile + Dpatch

(exemple du paquet « apg »)

```
debian/control.in:
```

```
Source: apg
Section: admin
Priority: optional
Maintainer: Marc Haber <mh+debian-packages@zugsclus.de>
Build-Depends: @cdb/@
Standards-Version: 3.6.1

Package: apg
Architecture: any
Depends: ${shlibs:Depends}
Description: Automated Password Generator - Standalone version
  APG (Automated Password Generator) is the tool set for random
  password generation. It generates some random words of required type
  and prints them to standard output. This binary package contains only
  the standalone version of apg.
Advantages:
  * Built-in ANSI X9.17 RNG (Random Number Generator) (CAST/SHA1)
  * Built-in password quality checking system (now it has support for Bloom
    filter for faster access)
  * Two Password Generation Algorithms:
    1. Pronounceable Password Generation Algorithm (according to NIST
      FIPS 181)
    2. Random Character Password Generation Algorithm with 35
      configurable modes of operation
  * Configurable password length parameters
  * Configurable amount of generated passwords
  * Ability to initialize RNG with user string
```

```

* Support for /dev/random
* Ability to crypt() generated passwords and print them as additional output.
* Special parameters to use APG in script
* Ability to log password generation requests for network version
* Ability to control APG service access using tcpd
* Ability to use password generation service from any type of box (Mac,
  WinXX, etc.) that connected to network
* Ability to enforce remote users to use only allowed type of password
  generation
The client/server version of apg has been deliberately omitted.
.
Upstream URL: http://www.adel.nursat.kz/apg/download.shtml

```

debian/rules:

```

#!/usr/bin/make -f

DEB_MAKE_CLEAN_TARGET    ~= clean
DEB_MAKE_BUILD_TARGET    ~= standalone
DEB_MAKE_INSTALL_TARGET  ~= install INSTALL_PREFIX=$(CURDIR)/debian/apg/usr

include /usr/share/cdb/1/rules/debhelper.mk
include /usr/share/cdb/1/rules/dpatch.mk
include /usr/share/cdb/1/class/makefile.mk

cleanbuilddir/apg::
    rm -f build-stamp configure-stamp php.tar.gz

install/apg::
    mv $(CURDIR)/debian/apg/usr/bin/apg \
    $(CURDIR)/debian/apg/usr/lib/apg/apg
    tar --create --gzip --file php.tar.gz --directory \
    $(CURDIR)/php/apgonline/ .
    install -D --mode=0644 php.tar.gz \
    $(CURDIR)/debian/apg/usr/share/doc/apg/php.tar.gz
    rm php.tar.gz
    install -D --mode=0755 $(CURDIR)/debian/apg.wrapper \
    $(CURDIR)/debian/apg/usr/bin/apg
    install -D --mode=0644 $(CURDIR)/debian/apg.conf \
    $(CURDIR)/debian/apg/etc/apg.conf

```

5.4 Exemple Perl

(exemple du paquet « libmidi-perl »)

debian/control:

```

Source: libmidi-perl
Section: interpreters
Priority: optional
Build-Depends: cdb (>= 0.4.4), debhelper (>= 4.1.0), perl (>= 5.8.0-7)
Maintainer: Mario Lang <mlang@debian.org>
Standards-Version: 3.5.10

Package: libmidi-perl
Architecture: all
Depends: ${perl:Depends}
Description: read, compose, modify, and write MIDI files in Perl
  This suite of Perl modules provides routines for reading, composing,

```

```
modifying, and writing MIDI files.
```

```
debian/rules:
```

```
#!/usr/bin/make -f
```

```
include /usr/share/cdb/1/rules/debhelper.mk  
include /usr/share/cdb/1/class/perlmodule.mk
```

Chapitre 6

Outils utiles

6.1 `cdbs-edit-patch` (provided with CDBS)

This script is intended to help lazy people edit or easily create patches for the simple-patchsys patch system.

Invoke this script with the name of the patch as argument, and you will enter a copy of your work directory in a subshell where you can edit sources. When your work is done and you are satisfied with your changes, just exit the subshell and you will get back to normal world with `debian/patches/patch_name.patch` created or modified accordingly. The script takes care to apply previous patches (ordered patches needed!), current patch if already existing (in case you want to update it), then generate an incremental diff to only get desired modifications. If you want to cancel the patch creation / modification, you only need to exit the subshell with a non-zero value and the diff will not be generated (only cleanups will be done).

Chapitre 7

Conclusion

CDBS résoud la plupart des problème et est très plaisant à utiliser. De plus en plus de DD l'utilisent, non pas parce qu'ils y sont obligé, mais parce qu'ils l'ont goûté et ont trouvé qu'ils pouvaient améliorer leurs paquets et éviter de perdre du temps à concevoir des règles idiotes et complexes.

CDBS is not perfect, the BTS entry is not clear, but fixing a single bug most of the time fix a problem for plenty of other packages. CDBS is not yet capable of handling very complicated situations (like packages where multiple C/C++ builds with different options and/or patches are required), but this only affects a very small number of packages. These limitations would be solved in CDBS2, which is work in progress (please contact Jeff Bailey jbailey@raspberryinger.com if you want to help).

Utiliser CDBS plus largement améliorerait la qualité globale de Debian. N'hésitez pas à l'essayer, à en parler à vos amis, et à contribuer.

Bon amusement avec CDBS !!! :-)

Merci

Merci à Jeff pour sa patience et avoir répondu à tant de questions.

Un merci spécial à GuiHome pour son aide pour relire cette documentation.

This document is a **DocBook** application, checked using `xmllint` (from `libxml2`), produced using `xsltproc` (from `libxslt`), using the **N. Walsh** and `dblatex` XLSL stylesheets, and converted with **LaTeX** tools (`latex`, `mkindex`, `pdflatex` & `dvips`) / `pstotext` (with `GS`).
