Documentação do CDBS

Marc (Duck) Dequènes and Arnaud (Rtp) Patard

July 5, 2005

Revision History

Revision 0.1.0	2005-04-03	
Primeira release pública (para CDBS V0.4.27-3)		

Prefácio

Esta documentação descreve nossa experiência ao aprender sobre a utilização do CDBS, com tantos detalhes quanto forem possíveis. FIXMENevertheless, we are not using the whole set of available features ourselves, and some parts of this documentation were written for mere convinience and completeness.

Observe que alguns exemplos nesta documentação contêm o conteúdo quebrado, que é necessário para deixar as páginas com uma largura razoável. Tenha o cuidado de juntá-los quando necessário antes de utilizá-los. (ex: o conteúdo de 'debian/control' não deve ser quebrado senão sua construção irá falhar ou resultar em algo incorreto).

Caso você encontre erros ou falta de informação, sinta-se livre para contactar Marc Dequènes (Duck) duck@duckcorp.org <mailto:duck@duckcorp.org>.

Contents

List of Figures

List of Tables

Chapter 1

Introdução

1.1 Um pouco da história

O CDBS foi escrito por Jeff Bailey e Colin Walters em março de 2003, posteriormente com a participação de outros 4 desenvolvedores.

Informção básica pode ser encontrada nas suas páginas do projeto <http://alioth.debian.org/projects/build-common/>. No pacote estão disponíveis uma série de pequenos exemplos <http://cvs.alioth.debian.org/cgi-bin/cvsweb.cgi/cdbs/examples/?cvsroot=build-common> (também disponíveis no pacote aqui: /usr/share/doc/cdbs/examples/).

Desde que nós estivemos experimentando o CDBSS, foi seguramente a falta de documentação que nos restringiu utilizá-los nos nossos pacotes. Assim nós começamos a escrever algumas anotações sobre o uso do CDBS, e rapidamente passou a ser utilizado em diversos pacotes. Esta documentação é uma versão revisada da original escrita na wiki da DuckCorp <https://wiki.duckcorp.org/DebianPackagingTutorial_2fCDBS>.

1.2 Por que CDBS?

CDBS é desenhado para simplificar o trabalho dos mantenedores, pelo fato deles precisarem pensar somente sobre o empacotamento e não sobre a manutenção de um arquivo 'debian/rules' que tende a ficar cada vez maior e mais complicado. Assim o CDBS pode manipular para você a maior parte das regras comuns e detectar algumas partes da sua configuração.

CDBS utiliza apenas simples regras do makefile e é facilmente extensível para utilização de classes. Classes para manipular autotools buildsys, aplicar patches, softwares para gnome, instalar programas python e muitas outras disponíveis.

Vantagens do CDBS :

- Um 'debian/rules' pequeno, legível e eficiente
- Automatiza o debhelper e autotools para você, assim você não precisa se incomodar com essas tarefas repetitivas e desagradáveis.
- O mantenedor pode focar nos problemas reais do empacotamento porque o CDBS te ajuda mas não restringe personalização.
- As classes utilizadas no CDBS têm sido bem testadas então você está usando regras "à prova de falhas" e evitando "hacks" sujos para resolver problemas comuns.
- Migrar para CDBS é fácil
- Pode ser usado para gerar arquivos debian (como um 'debian/control' para inclusão no time de uploaders do GNOME)

- CDBS é facilmente extensível
- Ele $|70><$!!!

Chapter 2

Primeiros passos

2.1 Converter um pacote para CDBS

Converter para CDBS é fácil; Um simples 'debian/rules' para um programa em C/C++ sem regras extras pode ser escrito dessa forma:

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

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

Não, eu não estou brincando, isto é suficiente para manipular o gerenciamento do autotools, como atualizar o config.{guess|sub}, limpar arquivos temporários depois da construção e chamar todos os recursos comuns do debhelper.

Apenas use a compatibilidade nível 4 (eu não sei se a 3 funciona), crie seu <pkg>.install, <pkg>.info, etc como você geralmente faz com os scripts dh_*, e o CDBS poderá chamá-los quando necessário, detectando automaticamente uma série de coisas (no caso de faltar informações sobre incompatibilidade, o CDBS irá criar o arquivo 'debian/compat'com compatibilidade nível 4).

IMPORTANT

Se o gerenciamento do 'debian/control' está ativado (veja abaixo), a dependência de construção (build dependency) no 'cdbs' é automaticamente adicionada, se não, você mesmo terá que fazer isso.

2.2 Configurações básicas e variáveis disponíveis

Lembre que você pode pegar o diretório do pacote usando a variável $(CURDIR).

Você pode mudar parâmetros comuns de construção desta forma:

```
# onde estão os fontes
DEB_SRCDIR = $(CURDIR)/src
# em que diretório construir
DEB_BUILDDIR = $(DEB_SRCDIR)/build
# em que diretório instalar o software
```

```
DEB_DESTDIR = $(CURDIR)/plop/
```

Você pode usar diversas variáveis no 'debian/rules' :

Table 2.1: Variáveis comuns disponíveis no 'debian/rules'

DEB_SOURCE_PACKAGE	name of the source package
DEB_VERSION	full Debian version
DEB_NOEPOCH_VERSION	Debian version without epoch
DEB_ISNATIVE	non-empty if package is native
DEB_ALL_PACKAGES	list of all binary packages
DEB_INDEP_PACKAGES	list of architecture independant binary packages
DEB_ARCH_PACKAGES	list of architecture dependant binary packages
DEB_PACKAGES	list of normal (non-udeb) binary packages
DEB_UDEB_PACKAGES	list of udeb binary packages, if any
DEB_ARCH	the old Debian architecture name
	/!\ deprecated, only use to provide backward compatibility /!
	(see man dpkg-architecture for more information)
DEB_HOST_GNU_TYPE	the GNU system type of the host machine
DEB_HOST_GNU_SYSTEM	the CPU part of DEB_HOST_GNU_TYPE

2.3 Basic custom build rules

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-date::
   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-unneeded \
      debian/foo/usr/lib/foo/totoz.so
```

To add pre-clean actions :

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

2.4 Common Build Options

CFLAGS and CXXFLAGS are set to ”-g -Wall -O2” by default.

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 Debhelper stuff

2.5.1 Not managing Debhelper

Yes, CDBS is doing almost everything for you :) .

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

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

CDBS debhelper rules handle the following dh_* scripts for each binary package automatically :

Table 2.2: Debhelper scripts commonly managed

dh_builddeb	dh_installchangelogs	dh_installemacsen	dh_installman	dh_perl
dh_clean	dh_installcron	dh_installexamples	dh_installmenu	dh_shlibdeps
dh_compress	dh_installdeb	dh_installinfo	dh_installpam	dh_strip
dh_fixperms	dh_installdebconf	dh_installinit	dh_link	
dh_gencontrol	dh_installdirs	dh_installlogcheck	dh_makeshlibs	
dh_install	dh_installdocs	dh_installlogrotate	dh_md5sums	

Other dh_* scripts 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 recommanded as it will ensure people will use the version providing the necessary features (CDBS 'debian/control' management will do it).

2.5.2 Debhelper parameters

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/cdbs/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') :

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

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
```

2.5.3 Debhelper custom build rules

CDBS debhelper rules also add more adequate build rules.

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
```

Several other rules exists, but we have not tested them :

- binary-strip/foo (called after stripping)
- binary-fixup/foo (called after gzipping and fixing permissions)
- binary-predeb (called just before creating .deb)

Chapter 3

Common tasks

3.1 Patching sources (using simple-patchsys)

First, patching sources directly is really BAD(tm), so you need a way to apply patches without touching any files. These rules, inpired by the Dpatch system, are quite similar and powerful. All you need is diff/patch knowledge, CDBS is doing the rest.

That's quite hard, so please listen carefully and prepare for examination.

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

```
include /usr/share/cdbs/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 additionnal '.gz' or '.bz2' suffix.

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

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

In case of errors when applying, for example 'debian/pacthes/01_hurd_ftbfs_pathmax.patch', you can read the log for this patch in 'debian/pacthes/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 Patching sources (using dpatch)

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

```
include /usr/share/cdbs/1/rules/dpatch.mk
```

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

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 Automatic tarball management

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/cdbs/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, automagically 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
```

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.

Chapter 4

Advanced customisation

4.1 'debian/control' management

This feature allow :

- CDBS to automatically manage some build-related Build-Depends automatically
- use of embedded shell commands
- use of CPU and System criterias to specify architecture (*EXPERIMENTAL*)

Build-related Build-Depends are dependencies introduced by the use of certain CDBS features, or autodetected needs.

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: @cdbs@, procps [system: linux], plop [cpu: s390]
```

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

Here is the recipe :.

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. Add the following line to 'debian/rules', before *any* include :

```
DEB_AUTO_UPDATE_DEBIAN_CONTROL := yes
```

4. Then do a "**debian/rules clean**" run to (re)generate 'debian/control'.

4.2 Using the Autotools class

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.

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

```
include /usr/share/cdbs/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 environnement variables need to be setup, use :

```
DEB_CONFIGURE_SCRIPT_ENV += LDFLAGS=" -Wl,-z,defs -Wl,-O1"
```

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
- DEB_AUTO_UPDATE_AUTOCONF
- DEB_AUTO_UPDATE_AUTOMAKE

(corresponding build dependencies will automatically be added)

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 unexistant install rule for make
DEB_MAKE_INSTALL_TARGET :=

# example to activate check rule
DEB_MAKE_CHECK_TARGET := check
```

DEB_BUILD_OPTIONS is checked for the following options :

- noopt : use -O0 instead of -O2
- nocheck : skip the check rule

CDBS automagically cleans autotools files generated during build ('config.cache', 'config.log', and 'config.status').

4.3 Using the Makefile class

This class is for the guys who only have a Makefile to build the program. You only need to have four rules in the Makefile :

- one for cleaning the build directory (i.e. mrproper)
- one for building your software (i.e. myprog)
- one for checking if the software is working properly (i.e. check)
- one for installing your software (i.e. install)

To be honest, the install rules is not a must-have, but it always helps a lot when you've got it.

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
DEB_MAKE_BUILD_TARGET    := myprog
DEB_MAKE_INSTALL_TARGET  := install DESTDIR=$(CURDIR)/debian/tmp/
# no check for this software
DEB_MAKE_CHECK_TARGET :=

# example when changing environnement variables is necessary :
DEB_MAKE_ENVVARS    := CFLAGS="-pwet"
```

DEB_BUILD_OPTIONS is checked for the following options :

- noopt : use -O0 instead of -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...

That's all :)

4.4 Using the Perl class

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

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

```
include /usr/share/cdbs/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 overrided : DEB_MAKE_ENVVARS, DEB_BUILDDIR (must match DEB_SRCDIR for Perl)

Have a look at Perl-specific debhelper options described above.

4.5 Using the Python class

This class can manage common python builds using 'distutils' automatically.

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

```
include /usr/share/cdbs/1/class/python-distutils.mk
```

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

Most python packages are architecture all, and then don't need being build for multiple python versions ; your package should then be called 'python-<foo>' and CDBS would automatically use the current Debian python version to build it. If your package contains a compiled part or a binding to an external lib, then you will have packages named 'python2.3-<foo>', 'python2.4-<foo>', and so on, depending on ${python:Depends} (and perhaps other packages), then CDBS would automatically build each package with the corresponding python version. In this case, don't forget to add a 'python-<foo>' convenience dummy package depending on the curent Debian python version.

You can customize build options like this :

```
# force using a specific python version for build
# (should not be necessary)
DEB_PYTHON_COMPILE_VERSION := 2.3

# 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

# specific additional install options for binary package 'foo'
#   ('--root' option is always set)
DEB_PYTHON_INSTALL_ARGS_foo := --root=debian/foo-install-dir/
```

4.6 Using the GNOME class

This class adds a make environnement 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 if defered 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 dh_* scripts automagically :

Table 4.1: Debhelper scripts managed by the GNOME class

dh_desktop	dh_gconf	dh_scrollkeeper

Moreover it adds some more clean rules :

- to remove intltool generated files
- to remove scrollkeeper generated files

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

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

For more information on GNOME specific packaging rules, look at the Debian GNOME packaging policy <http://alioth.debian.org/docman/view.php/30194/18/gnome-policy-20030502-1.html>.

4.7 Using the Debian GNOME Team class

If you are part of the GNOME Team, or having the Team as Uploaders, and you feel bored maintaining the list of developpers, this class is made for you.

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 developpers automatically.

WARNING

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-arround (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
```

4.8 Using the KDE class

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

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

CDBS automatically exports the following variables with the right value :

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

DEB_BUILDDIR, DEB_AC_AUX_DIR and DEB_CONFIGURE_INCLUDEDIR are set to KDE defaults.

The following files are excluded from compression :

- .dcl
- .docbook
- -license

It can handle configure options specific to KDE (not forgeting disabling rpath and activating xinerama), set the correct autotools directory, and launch make rules adequately.

DEB_BUILD_OPTIONS is checked for the following options :

- nofinal : disable KDE final mode
- nostrip : enable KDE debug and disable KDE final mode

4.9 Using the Ant class

(Ant is a java-based build tool)

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

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

# Set either a single (JAVA_HOME) or multiple (JAVA_HOME_DIRS) java locations
JAVA_HOME := /usr/lib/kaffe
# or set JAVACMD if you don't use default '<JAVA_HOME>/bin/java' path
#JAVACMD := /usr/bin/java

# Set Ant location
ANT_HOME := /usr/share/ant-cvs
```

You may add additionnal JARs like in the following example :

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

WARNING

Due to a CDBS bug, you must always add "$(ANT_HOME)/lib/ant-launcher.jar" to DEB_JARS or Ant will fail.

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

You can provide additionnal JVM arguments using ANT_OPTS. You can provide as well additionnal 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_TEST_TARGET = check
```

DEB_BUILD_OPTIONS is checked for the following options :

- noopt : set 'compile.optimize' Ant option to false

You should be able to fetch some more information on this java-based build tool in the Ant Apache web site <http://ant.apache.org/>.

4.10 Using the HBuild class

(HBuild is the Haskell mini-distutils)

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/cdbs/1/class/hbuild.mk
```

You should be able to fetch some more information on Haskell distutils in this thread <http://www.haskell.org/pipermail/libraries/2003-July/001239.html>.

Chapter 5

Hall of examples

5.1 GNOME + autotools + simple patchsys example

(example from the 'gnome-panel' package)

'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 floatingâ. It is designed to be
 used in conjunction with the Gnome Desktop Environment. Many features
 are provided for use with the panels â 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/cdbs/1/rules/debhelper.mk
# Including this file gets us a simple patch system.  You can just
# drop patches in debian/patches, and they will be automatically
# applied and unapplied.
include /usr/share/cdbs/1/rules/simple-patchsys.mk
```

```
# Including this gives us a number of rules typical to a GNOME
# program, including setting GCONF_DISABLE_MAKEFILE_SCHEMA_INSTALL=1.
# Note that this class inherits from autotools.mk and docbookxml.mk,
# so you don't need to include those too.
include /usr/share/cdbs/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::
        # GNOME Team 'uploaders.mk' should not override this behavior
        #   here is a workarround :
        sed -i "s/@cdbs@/$(CDBS_BUILD_DEPENDS)/g" debian/control
        # cleanup not done by buildsys
        -find help -name '*omf.out' -exec rm -f {} \;
        -find . -name "Makefile" -exec rm -f {} \;
        # binary unpatch
        uudecode -o po/fr.gmo debian/maintfiles/fr.gmo.uu
        uudecode -o po/or.gmo debian/maintfiles/or.gmo.uu
        uudecode -o po/uk.gmo debian/maintfiles/uk.gmo.uu
```

5.2 Python example

(example from 'python-dice', an unofficial DC package)

'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

DEB_AUTO_UPDATE_DEBIAN_CONTROL := yes

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

clean::
        # hack (CDBS bug -- see #300149)
        -rm -rf build
```

5.3 Makefile + Dpatch example

(example from the 'apg' package)

'debian/control.in':

```
Source: apg
Section: admin
```

```
Priority: optional
Maintainer: Marc Haber <mh+debian-packages@zugschlus.de>
Build-Depends: @cdbs@
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_AUTO_UPDATE_DEBIAN_CONTROL := yes

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

include /usr/share/cdbs/1/rules/debhelper.mk
include /usr/share/cdbs/1/rules/dpatch.mk
include /usr/share/cdbs/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

# bug #284231
unpatch: deapply-dpatches
```

5.4 Perl example

(example from the 'libmidi-perl' package)

'debian/control':

```
Source: libmidi-perl
Section: interpreters
Priority: optional
Build-Depends: cdbs (>= 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/cdbs/1/rules/debhelper.mk
include /usr/share/cdbs/1/class/perlmodule.mk
```

Chapter 6

Conclusion

CDBS solves most common problems and is very pleasant to use. More and more DD are using it, not because they are obliged to, but because they tasted and found it could improve their packages and avoid loosing time on designing silly and complicated rules.

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@raspberryginger.com <mailto:jbailey@raspberryginger.com> if you want to help).

Using CDBS more widely would improve Debian's overall quality. Don't hesitate trying it, talking to your friends about it, and contributing.

Have a Lot of FUN with CDBS !!! :-)

Thanks

Thanks to Jeff for his patience and for replying my so many questions.

Special thanks to GuiHome for his help to review this documentation.

This document is a DocBook <http://docbook.org/> application, checked using xmllint (from libxml2 <http://www.xmlsoft.org/>), produced using xsltproc (from libxslt <http://xmlsoft.org/XSLT/>), using the N. Walsh <http://nwalsh.com/> and DB2LaTeX <http://db2latex.sourceforge.net/> XLST stylesheets, and converted with LaTeX <http://www.latex-project.org/> tools (latex, mkindex, pdflatex & dvips) / pstotext <http://research.compaq.com/SRC/virtualpaper/pstotext.html> (with GS <http://www.cs.wisc.edu/~ghost/>).