

'Embedded Debian'

Neil Williams Fosdem 2008. Copyright 2008 Neil Williams

GNU GENERAL PUBLIC LICENSE Version 3, 29 June 2007

This document is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Table of Contents

Acknowledgements

1. Debian Update

- Introduction

- “Disc” space is not cheap

- Conventions used within Emdebian

- Common problems for cross builds

- Using dpkg-architecture to detect a cross build

- Changes to locate the cross compiler

- Removing documentation

- Preventing build checks

- Omitting udeb packages

- Complications involving maintainer scripts

2. Remaining workarounds

- Building internal libraries

- Working with dpkg in a cross build

3. Tdeb translation packages

- tdebs in Debian and Emdebian

- Current Debian practice for translations

- Repackaging translations into individual locales per package.

- Handling a ten fold increase in binary package numbers

- Translator upload support (outline only)

List of Examples

- 1.1. Cross building when using ./configure
- 1.2. Cross building using \$(MAKE)
- 2.1. Specifying relinking path for internal objects in CDBS
- 2.2. Specifying relinking path for internal objects via \$(MAKE)
- 2.3. Using debian/xcontrol with xcontrol

Acknowledgements

<http://www.toby-churchill.com/>

Many individuals :

- Wookey,
- Phillipe de Swert,
- Hector Oron,
- Neil Williams,
- Peter Naulls,
- Jon Masters,
- Justin Cormack,
- Nikita Y.Youshchenko,
- Raphael Bossek,
- Allen Curtis,
- Benjamin Henrion

and anyone I might have forgotten.

Chapter 1. Debian Update

Cross building Debian

Introduction

This paper describes recent developments with the Emdebian project to create a set of Debian packages that can be installed on embedded devices. Emdebian now has a basic root filesystem and a set of scripts to customise the rootfs further for other requirements, a set of toolchains for amd64, i386 and powerpc and a set of GUI packages based on the G Palmtop Environment (GPE) and Gtk+2.

Details of how these GUI packages are being prepared and configured and how Emdebian utilises “tdeb” packages is the subject of a separate talk. This paper describes common reasons to cross build Debian, problems when cross building and the current implementation of solutions devised within Emdebian.

“Disc” space is not cheap

The typical Debian model for packaging is that all working modes of a package should be enabled and all forms of documentation and support packaged for all users.

Embedded devices need a very different approach. The model involves identifying only those modes of each package that are essential to the purpose of the device itself, removing dependencies, removing content that cannot be utilised on an embedded device (manpages, info, README etc.) and overall increasing the granularity of the packages. Emdebian seeks to make a lot more, smaller, binaries out of the current set of source packages to support more choice about just which functions and components are installed on a specific device, initially within the core Debian packages provided by the typical debootstrap set but also within GUI package sets like Gtk+2.

Storage space on an embedded device is very expensive, unnecessary writes to storage should be avoided and removing a feature of one package that is unused on a particular device could allow the same device to support a new, useful, feature of another package. Emdebian seeks to reduce the default size of a Debian installation by between 70 and 90%.

Some components of a package can be removed simply by repackaging the .deb file - stripping out manpages etc. - but this is rarely sufficient to allow Debian packages to fit onto an embedded device. Cross building allows Emdebian to modify the dependencies of the package, modify the feature set (converting --with to --without in calls to ./configure), implement new package splits and remove certain components of a package completely.

Emdebian cross builds seek to:

- Replace 'Essential: yes' with a device-specific list of essential packages that is much smaller than the Debian set. Perl is not included and modifications to the Debian busybox configuration allow it to replace the much larger Debian equivalents.
- Package each component and subcomponent separately.
- Drop or disable any component or feature that cannot be packaged separately or which drags in a long list of dependencies.
- Remove all documentation from the main package binaries. (If documentation is needed on specific devices, it can be added later in a shortened, bespoke, format.)
- Repackage all gettext-compliant translation data as individual tdeb packages, one per locale.
- Create new package splits where necessary to redistribute package binaries in ways that allow individual devices to omit binaries that would otherwise be unused.

A package split does not have to be about benefits to the archive, splitting Architecture: any from Architecture: all or simply providing a -dev, -dbg and -doc version of a package. Package splits in embedded usage need to be about splitting feature sets, making it possible to install specific features of a package without having to install the dependencies of unused features.

Current state.

Emdebian has so far cross built just under 200 Debian source packages, producing just under 600 binary packages and over 1,200 tdeb packages. Working installations of Emdebian chroots show a 70% reduction in download and installed package sizes. Emdebian has a root filesystem built from Debian packages that is a 15Mb download and a 24Mb installation. Work continues to reduce this further to below 10Mb installed. (A typical debbootstrap would require about 180Mb installed). These further reductions are likely to require multiple package splits in source packages like glibc and tzdata.

Note that download sizes are just as important as installed sizes - a device that only has 32Mb of total storage space must not be expected to download 30Mb of packages and then strip out 16Mb during installation - even unpacking one package at a time would result in a failed install or upgrade. Instead, the packages in the archive must be rebuilt to be as small as possible before download and this includes user upgrades after installation as well as the more specialised installation procedure itself.

Conventions used within Emdebian

build

Big or Desktop - the machine running the cross compiler, generally amd64, i386 or powerpc.

Host

Handheld - the device which will run the built package, e.g. ARM

Common problems for cross builds

Gradually, these changes are being implemented within the Debian packages themselves by providing patches to the long term mass bug filing for cross building support and the guideline described here is based on the existing recommendations within `/usr/share/doc/autotools-dev/README.Debian.gz`, part of the `autotools-dev` package. This includes the use of `--host` only when cross compiling, not as a default part of all builds on the basis of just making the patch smaller. `--build` should always be used. The mass bug filing also implements similar support for packages that do not use `autotools` but simply implement a `Makefile`.

Using `dpkg-architecture` to detect a cross build

```
DEB_HOST_GNU_TYPE=$(shell dpkg-architecture -qDEB_HOST_GNU_TYPE)
DEB_BUILD_GNU_TYPE=$(shell dpkg-architecture -qDEB_BUILD_GNU_TYPE)
```

Changes to locate the cross compiler

(in `debian/rules`)

Example 1.1. Cross building when using `./configure`

```
ifneq ($(DEB_HOST_GNU_TYPE),$(DEB_BUILD_GNU_TYPE))
CROSS= --build $(DEB_BUILD_GNU_TYPE) --host $(DEB_HOST_GNU_TYPE)
else
CROSS= --build $(DEB_BUILD_GNU_TYPE)
endif
```

and add the variable to `./configure`:

```
./configure $(CROSS) ...
```

Example 1.2. Cross building using `$(MAKE)`

```
ifneq ($(DEB_HOST_GNU_TYPE),$(DEB_BUILD_GNU_TYPE))
CROSS=CC=$(DEB_HOST_GNU_TYPE) -gcc
else
CROSS=
endif
```

and add the variable to all invocations of `$(MAKE)` (except `'make *clean'`).

```
$(MAKE) $(CROSS) ...
```

Removing documentation

Embedded devices such as routers and handhelds will not be able to utilise manpages or info documents in a usable manner. Other documentation like `README`, changelogs, examples and API documentation is also unnecessary. Whilst a small amount of help content may be necessary in some environments (tooltips etc.), this is generally embedded within the program binaries and is

easily localised using gettext and tdeb packages.

It is better, wherever possible, to prevent the generation of such documentation during the cross build itself because this reduces the complexity of the build, reduces the build dependencies and improves the speed of the cross build itself. To this end, the emdebian-tools build wrapper emdebuild, uses the “nodoc” DEB_BUILD_OPTION and patches to prevent the generation or remove the pre-generated documentation in the Debian package.

A Debian package supporting “nodoc” should, from an embedded perspective, not generate or package any documentation of any kind. Bug #448615 requests support from debhelper to adjust the way that copyright files are handled so that only a compressed copyright file is retained in the package.

Support within debian-xcontrol (described later) will allow cross builds to ignore build dependencies merely required to generate documentation that is already disabled.

Preventing build checks

During a cross build, it is impossible to execute any binary that has been compiled for the host architecture, therefore all attempts to run any build checks (e.g. ‘make check’) must be disabled. Emdebuild uses the “nocheck” DEB_BUILD_OPTION which packages can support to avoid running build checks in such environments. Packages that do not already support “nocheck” are patched by Emdebian to remove the ‘make check’ command. The existing long term mass bug filing for cross build support will include bugs for such packages, with patches to include support for “nocheck”.

Omitting udeb packages

udeb packages are already very specialized and are not needed by Emdebian. The different dependencies of udeb packages and the specialized requirements of the D-I team mean that these packages need to be omitted from Emdebian builds.

Emdebuild does use a “noudeb” DEB_BUILD_OPTION but this is not commonly supported in Debian packages yet. Once other issues have been resolved, the long term mass bug filing will embrace “noudeb” as well. As with documentation, there will be a role here for debian-xcontrol to prune the build dependencies of packages related to udeb builds.

Complications involving maintainer scripts

Many maintainer scripts will need patches for Emdebian, reasons include (but are not limited to):

perl

Maintainer scripts are often written in shell but may use perl or any interpreter if the dependencies are handled appropriately. Scripts that are written in perl will need to be reimplemented in shell or simply removed with Emdebian patches. So far, this has not been a particular problem - the

kind of packages that have complex perl maintainer scripts are not typically needed on an embedded device.

update-alternatives

Maintainer scripts may call update-alternatives with details of commands, man pages and info pages - these calls need to be removed until a non-perl form of update-alternatives is available. By ignoring "Essential", this becomes less of a problem because the embedded device does not have to install more than one version of the same functionality and dependencies can be adjusted to force replacement rather than alternative. Eventually, some form of non-perl support for 'alternatives' may well become necessary.

install-info

Maintainer scripts may call install-info to put info pages into usable locations. Without info pages in the package, install-info will fail so these calls must be removed.

Backwards compatibility

Maintainer scripts are often written to handle issues that only arise when upgrading from an old version to a fixed version. These sections can be removed from the Emdebian scripts, especially if any of the above problems exist in these sections or if the compatibility code makes the script particularly long.

Chapter 2. Remaining workarounds

Building internal libraries

Gradually, changes are being made to the Debian package building tools to make cross building easier in Debian. Some problems remain and emdebuid uses three main workarounds:

gccross : from dpkg-cross. This script sits between \$(MAKE) and the cross compiler. When the cross compiler is called, all -I and -L paths must be set for the cross packages installed by dpkg-cross so that the compiler can find the right headers and the right shared objects. In most cases, a combination of pkg-config and libtool ensure that these paths are available. Packages that build internal libraries and then link other libraries or executables against those object files will experience problems when installing the object file during the Debian package building. During the final install, object files are relinked for the package directories - except when cross building, the relinking gets confused and tries to relink against libraries in /usr/lib/ instead of /usr/\$arch-triplet/lib/ and the build fails.

For some years, dpkg-cross has contained a utility called gccross that rewrites the paths passed to the cross compiler. It does this by pretending to be the cross compiler and then rewriting the @ARGV before passing control to the real compiler.

Since emdebian-tools (>=0.7.3), gccross is optional - packages need to opt-in to gccross support using a flag in debian/xcontrol : **X-Build-Cross-Libtool: yes** (or Build-Cross-Libtool: yes but hopefully this will be fixed before this field needs to become an official dpkg control field).

Packages can implement their own support for cross building their internal objects by specifying -l flags to libtool:

Example 2.1. Specifying relinking path for internal objects in CDBS

This example is from libqof1.

```
ifneq ($(DEB_BUILD_GNU_TYPE),$(DEB_HOST_GNU_TYPE))
# internal libraries need to be redirected to cross version
DEB_MAKE_INVOKE+="QOF_LIBS=-L/usr/$(DEB_HOST_GNU_TYPE)/lib -lqof"
endif
```

Example 2.2. Specifying relinking path for internal objects via \$(MAKE)

For an example package, FOO using -lfoo during native relinking and passing \${FOO_LIBS} for libfoo_la_LIBADD in the upstream Makefile.am.

```
Ifneq ($(DEB_BUILD_GNU_TYPE),$(DEB_HOST_GNU_TYPE))
RELINK="FOO_LIBS=-L/usr/$(DEB_HOST_GNU_TYPE)/lib -lfoo"
endif
```

Upstream may need to be patched. Note that upstream needs to support a

variable already – if the upstream Makefile.am hardcodes -lfoo then a patch will be needed to replace this value with a suitable variable.

These problems will also be reported under the long term mass bug filing for cross building support agreed on debian-devel in November 2007.

emdebian-tools (specifically emdebuild) already has outline support for debian/xcontrol and this will be enhanced with full support for the debian-xcontrol package in future releases.

Example 2.3. Using debian/xcontrol with xcontrol

emdebuild initially improvised a Build-Cross-Depends: field in debian/xcontrol and this will continue to be supported until emdebian-tools is fully integrated with xcontrol from the debian-xcontrol package. For compatibility with xcontrol, use:

```
Build-Depends: libsqlite0-dev, libglib2.0-dev, libgpewidget-dev
Build-Depends-Tools: cdb, debhelper (>= 5), autotools-dev,
automake1.9, dpkg-dev (>= 1.13.19)
```

instead of:

```
Build-Depends: cdb, debhelper (>= 5), autotools-dev, libsqlite0-
dev, libglib2.0-dev, dpkg-dev (>= 1.13.19), libgpewidget-dev,
automake1.9
```

Only packages that need to be installed with apt-cross are now listed in Build-Depends and packages that only need to be installed on the build machine (including all packages that are Architecture: all) go into Build-Depends-Tools. Xcontrol build can then be used to integrate the two sections for normal Debian builds.

Working with dpkg in a cross build

Outstanding dpkg bugs and the Emdebian workarounds

- LD_LIBRARY_PATH : this is a workaround for dpkg-shlibdeps and when bug # 453267 is fixed, this workaround will disappear from emdebuild. The basis of this workaround is to make /usr/\${crossprefix}/lib/ available to dpkg-shlibdeps where \${crossprefix} is the architecture triplet; for ARM the triplet is arm-linux-gnu.
- PKG_CONFIG_LIBDIR : this is a workaround for dpkg-buildpackage -a and when bug #439979 is fixed, this workaround will disappear from emdebuild. The basis of this workaround is to make /usr/\${crossprefix}/include/ and /usr/\${crossprefix}/lib available to gcc where \${crossprefix} is the architecture triplet; for ARM the triplet is arm-linux-gnu.

Chapter 3. Tdeb translation packages

tdebs in Debian and Emdebian

Emdebian borrows a technique from OpenEmbedded to handle translations which is already outlined within Debian as “Tdeb” - translation debian packages.

Emdebian tdebs are not identical to the Debian tdeb plans

The Emdebian implementation of tdebs differs from the proposed tdebs for Debian because Emdebian does not care about manpages in general, let alone translated manpages. Once the ‘nodoc’ DEB_BUILD_OPTION is supported in debhelper, this will not be an issue as the tdebs can be built for Emdebian without any manpages. Other translated documentation would be omitted under ‘nodoc’ too.

Images containing translated text are relatively few. Therefore, the majority of the Emdebian tdeb implementation can be applicable to Debian.

Current Debian practice for translations

Debian packages collate all available translations into a single package which can easily lead to over 250Mb of unused translation files on a typical installation. I.e. The translations alone could take up four times the amount of space intended for the entire embedded operating system.

Repackaging translations into individual locales per package.

One locale, one source package, one tdeb

em_installtdeb is a new script in emdebian-tools based on debhelper, to create translation packages (tdebs). em_installtdeb is intended to separate out the individual translation files from the current Debian packages into packages without any translation files and a series of tdeb locale packages, one per translation. Generated packages use the syntax:

```
$package-locale-$languagecode_$version_all.deb.
```

Once a package uses em_installtdeb, translation files should be removed from all packages in the normal build. em_installtdeb runs as a second build (very small, very fast) that simply converts and packages the *.po files into tdebs.

A tdeb source package is created (.dsc, tdeb.tar.gz and .changes) alongside the existing build data. The source package can be used by translators to build updated or new tdeb packages. Tdeb .changes files need to be uploaded only to secondary locale repositories instead of the main Debian mirrors and these repositories can have much more relaxed upload policies. Tdeb packages have no dependencies and no packages may depend upon them.

Each package now built for Emdebian includes tdeb support so that if the package contains gettext-compliant translations, tdeb packages are created.

Emdebian also supports a locale repository with search support. Currently, certain compromises are made to allow tdebs to work without changes to packages like dpkg or reprepro - including the need to make two uploads, one for the main package (without translations) and one for the generated tdeb .changes file including generated tdeb source and all of the tdeb packages. For more information on em_installtdeb, see the em_installtdeb manpage.

Handling a ten fold increase in binary package numbers

Emdebian generates a single package for every translation of each Emdebian package, leading to a 70% reduction in installation size but a tenfold increase in the number of binary packages built from each source package. To solve this scalability problem, Emdebian provides a new C/C++ application (langupdate) handling the installation and update of tdeb packages according to the list of supported locales and the list of installed packages. Data cached by langupdate is temporary and is not intended to be stored between runs of langupdate (i.e. It can be stored completely in RAM).

The secondary locale repository organises the tdeb packages by locale root, e.g. 'en_GB' is beneath 'en' and 'sr@Latn' beneath 'sr'. Each locale root becomes a single apt source in the locale repository to support fallback from a specific locale to a the more general locale root. In this manner, each device only has to cope with cache data for a fraction of the total number of tdeb packages (roughly 1 part in 30). The tdeb cache itself is temporary so the only effect on the device package data is the small number of locale packages that are actually installed. Tdebs have no long description and no dependencies so the effect on the dpkg lists should be minimal.

Langupdate queries the secondary apt cache to get a list of all translation packages (tdebs) that match the list of locales supported on the device. This list is then compared with the list of installed binaries. The translation packages that match the installed package are then installed by passing options to apt-get. Langupdate uses a temporary file for the sources.list and updates it from the list of supported locales each time langupdate is run, I.e. It combines the effects of 'apt-get update' with 'apt-get upgrade'. Tdeb packages have no dependencies so 'dist-upgrade' is not required.

Langupdate is a native Emdebian package - it may appear in Debian at a later date when Debian supports tdebs.

Translator upload support (outline only)

em_installtdeb includes outline support for translator uploads. This is intended to allow translators to add the locale deb-src source to their apt sources lists and use apt-get source \$package-locale-\$lang to download the tdeb source: the POT file, existing PO files and enough metadata to be able to build a new or updated tdeb package for a specific language using em_installtdeb \$lang. The build generates a new (binary-only) .changes file suitable for upload direct to the locale repository. There are issues to be resolved within this support before it can be made widely available, relating to merging the new translations back into the Debian package, but everything related to tdebs in Emdebian should be considered "experimental".