Debian devkit

Valtteri Rahkonen

valtteri.rahkonen@movial.fi

Debian devkit

by Valtteri Rahkonen

Revision history

Version:	Author:	Description:
2004-04-29	Rahkonen	Added references to doctools and rootstrap documents
2004-04-22	Rahkonen	Initial version

Table of Contents

Chapter 1. Introduction

Development kits provide Linux distribution specific envinment and set of tools that are installed to ease building of the distribution or invidual packages. More general tools (such as Autotools) are already present in Scratchbox core.

The Debian Devkit consists of a bunch of additional tools for Scratchbox mainly Debian packaging tool dpkg related things. Thus Debian devkit provides a set of tools to build Debian packages. Build tools are provided as a host (x86) executables to speed up building process.

Debian devkit is divided into three parts:

- Debian devkit package contains Debian specific build and package handling tools such as dpkg. Debian devkit provides all necessary tools required to build a Debian package. However this package does not provide documentation generation tools or libraries that are needed to build packages because they are provided by doctools and rootstrap packages.
- doctools provide a common document geration tools (for example groff). This package is build expecially Debian in mind thus it provides tools that are used when generating Debian packages documentation. However many other distributions and packages use same tools not just Debian. For this reason document genaration tools are divided to their own package and can be used without a Debian devkit.
- rootstrap provides a set of libraries and tools that are compiled for specific platform. Rootstrap is provided to developers as so they do not need to compile dependency libraries themselves and thus rootstrap speeds up development process even further. Rootstrap is currently provided for ARM target platform but i386 and PowerPC support is under way. If libraries for other target is needed, libraries must be installed to Scratchbox by other means.

This document describes what tools Debian devkit provides and how it can be updated. Doctools are documented on *Scratchbox's Documentation Tools* document [1]. For using this document it is assumed that you have a working Scratchbox installation. For installing and configuring Scratchbox see *Installing Scratchbox* document [2].

Chapter 2. Debian devkit tools

In addition to general build tools that are provided by Scratchbox and toolchains Debian devkit provides Debian specific tools. Debian specific tools provide most common executables and scripts that are used to build Debian packages.

Debian devkits focus is to provide host executables for speeding up build process (not all things needs to be emulated or executed on target platform). Currently following packages are provided by Debian devkit:

- alien-tools provides tools to convert other Linux distributions packages into Debian packages, which can be installed with dpkg. It can also generate packages of any of the other formats. Alien is suitable only for binary packages.
- apt is Debian's front-end for the dpkg package manager. It provides the apt-get utility to install and upgrade packages.
- console-common provides keymap handling utilities.
- dbs provides an alternative approach for source packages which want to provide original source's and patches against them separately. Dbs allows the distribution of multiple patches inside one package that can be applied during the building.
- debhelper provides a collection of programs that can be used in a debian/rules file to automate common tasks related to building debian packages. There are programs to install various files into your package, compress files, fix file permissions, integrate your package with the debian menu system, debconf, doc-base, etc. Most debian packages use debhelper as part of their build process.
- debianutils provides a number of small utilities which are used primarily by the installation scripts of Debian packages.
- devscripts provides helper scripts to update, build and test Debian packages.
- dh-make provides tools to generates a Debian style source package from regular source code archive. Customizes control files that are used to build a Debian package. Also provides example setup for debhelper utilities that is usually usable with minimal editing.
- dpatch is an easy to use patch system for Debian packages. Dpatch is similar to the dbs package, but much simpler to use. Also provides support for applying patches only on specific architectures if needed.
- dpkg contains the programs which handle the installation and removal of packages on the system. Also provides tools required to unpack, build and upload Debian source packages.
- mawk is an interpreter for the AWK programming language. The AWK language is useful for manipulation of data files, text retrieval and processing, and for prototyping and experimenting with algorithms. Many Debian packages depends on mawk and thus this package is needed on Debian devkit even though Scratchbox already provides more general gawk interpreter.

In addition Debian devkit contains environment package that makes changes to Scratchbox environment when Debian devkit is selected. It is environment packages responsibility to take care of setting up apt (sources.list needs to be created) and dpkg (cache files for packages), update user account information to system files and set up Debian package building for correct target platform.

Chapter 3. Adding packages to Debian devkit

3.1. Debian devkits repository

Debian devkit is located in Scratchbox CVS repository and it can be through CVS Viewer located at Scratchbox website [4]. This is the only public access to Scratchbox repository that is currently offered.

3.2. Debian devkits GAR build system

Scratchbox is built using GAR build system [6]. It is a mechanism for automating the compilation and installation of third-party source code. It appears in the form of a tree of directories containing Makefiles and other ancillary bookkeeping files (such as installation manifests and checksum lists). Scratchbox GAR build system is available trough CVS located on Scratchbox site [4].

Adding software to GAR build system is fairly straightforward. Adding new tools to Debian devkit can be done with following steps:

1. Obtain sources for package. Sources should be fetched with **apt-get source** command because Debian can have Debian specific patches against original sources. Otherwise there might be need to apply patches manually.
2. Create new directory for package to Debian devkits GAR directory 'scratchbox/debian_tools/$PACKAGENAME'.
3. Copy Debian patched source tarball to $PACKAGENAME/files directory.

   Note: Source tarball should be at Scratchbox site at 'download/files/sbox-files/' directory for enabling GAR to download package automatically. For testing purposes a local copy is sufficient.

4. Create makefile under $PACKAGENAME diretory. In GAR build systems package needs to have some specific variables for enabling package to build correctly. At least following list should be included, for more information about GAR build system see [6].
   - GARNAME defines package's name.
   - GARVERSION defines package's version.
   - CATEGORIES defines package's category. In Debian devkits case this must be 'debian_tools'.
   - DISTFILES defines source (tar) package that is used to build this package. This is usually combination of GARNAME, GARVERSION and source packages extension (usually tar.gz).
   - LIBDEPS tells which libraries this package depends. These packages are compiled before compiling this package.

- DEPENDS defines packages that must be compiled before this package can be compiled properly. This might contain tools that are required during building process of this package.
- DESCRIPTION contains description of this package.
- CONFIGURE_ENV defines some Scratchbox specific variables. For example Scratchboxs perl and python should be defined here if package uses them.
- CONFIGURE_ARGS defines options that are passed to packages configure script. This should contain at least prefix to install this package properly and Debian specific options (they should be in 'debian/rules' file inside packages source tarball).
- BUILD_ARGS defines arguments that are used to build this package. This should contain at least Scratchbox specific LDFLAGS, CC and PERL or PYTHON paths if they are used.
- INSTALL_ARGS contains arguments that are used when package is installed. Usually they are the same as BUILD_ARGS.
- CONFIGURE_SCRIPTS defines script that used to configure building process. Typically this is packages configure script.
- BUILD_SCRIPT is used to define packages makefile.
- INSTALL_SCRIPT defines packages install script. Usually this is packages makefile.
- In addition to package specific options packages GAR makefile should include correct GAR category file. In Scratchbox this is done by including '../category.mk' file after variable definitions. Also there must be 'pre-patch' and 'pre-configure' directives in Scratchbox GAR makefiles. These are used to add Scratchbox specific patches if necessary.

Following example makefile is from Debian devkits dpkg package:

```
GARNAME = dpkg
GARVERSION = 1.10.20
CATEGORIES = debian_tools
DISTFILES = $(GARNAME)_$(GARVERSION).tar.gz

LIBDEPS =
DEPENDS = tools/perl

DESCRIPTION = dpkg

CONFIGURE_ENV = PERL=/scratchbox/tools/bin/perl
CONFIGURE_ARGS = --prefix=$(prefix) --exec-prefix=$(prefix) \
         --sbindir=$(prefix)/bin --with-admindir=/var/lib/dpkg

BUILD_ARGS = -j1 LDFLAGS="$(TOOLS_LDFLAGS) -L../optlib -L../lib" \
         CC="$(TOOLS_CC)" PERL=/scratchbox/tools/bin/perl

INSTALL_ARGS = $(BUILD_ARGS)

CONFIGURE_SCRIPTS = $(WORKSRC)/configure
BUILD_SCRIPTS = $(WORKSRC)/Makefile
INSTALL_SCRIPTS = $(WORKSRC)/Makefile

include ../category.mk
```

```
pre-patch:
        $(MAKECOOKIE)

pre-configure:
        patch -d $(WORKSRC) -p1 < files/dpkg-sbox.patch
        $(MAKECOOKIE)
```

Above example should a give good picture about necessary elements in dpkg packages GAR makefile.

5. Type **make makesums** in packages directory to generate checksum information for package.
6. Package is now ready to be compiled. Compiling and installing package can be done with **make install** command.

Note: If package does not compile properly see packages Debian rules file for Debian specific configuring options and check that you have necessary library dependencies in order. In addition to Scratchbox specific configuring options also Debian might have some required options.

3.3. Testing a new tool

First thing is to check that new tool has been linked properly. This can be done inside Scratchbox with command **ldd**. Actual command is something like this:

```
[sbox-MYTARGET: ~] > ldd /scratchbox/devkits/debian/bin/$PROGRAMNAME
```

New tool should have been linked with libraries under '/scratchbox/host_shared'. If tool has links to other libraries outside '/scratchbox/host_shared' it is propably broken (usually inside Scratchbox libraries under '/lib' and '/usr/lib' directories are compiled for target platform and will not work with tools compiled for i386) and needs to be build again with correct build options. If tool was linked properly it can be tested that it works correctly.

Best way to test a new tool is to build a package that depends on that specific tool. If build process completes successfully and package can be installed and removed correctly test is considered passed and the new tool can be committed to devkits repository. Debian package building is described more closely on *Debian Development with Scratchbox* [3].

3.4. Inserting package to repository

After package is compiled, installed and tested properly it is ready to be put into Debian devkits repository. Adding new packages to Debian devkits repository should be done with following steps:

- If you are not a member of Scratchbox development team you do not have rights to insert packages to repository. If this is the case contact Scratchbox development team (contact information can be found at Scratchbox websites contact page [5]).
- Upload packages source tarball into 'download/files/sbox-files/' directory in Scratchbox website.
- Cleanup packages directory with 'make clean' command inside packages GAR directory.
- Use 'cvs add $PACKAGEDIRECTORY' command for packages GAR directory. This will insert packages directory to CVS repository.
- Insert packages makefile and checksums file to repository with 'cvs add Makefile checksums' command.
- If package contains patches files directory and the patch in files directory they should be also added to repository with 'cvs add' command.
- Add an entry to ChangeLog file that describes what has been added. This way modifications can be tracked later on.
- After necessary files are added modifications to repository should be committed. This can be done with 'cvs commit' command.
- New package should be also added to 'meta/debian_tools/' directory's makefiles dependency list so that package would be compiled automatically. Modifications to this file should also be committed with 'cvs commit' command.
- After package has been added to repository it is recommended that building and installing packages are verified. This can be done by checking out clean repository and testing packages with same tests than in Section 3.3.

References

[1] *Scratchbox's Documentation Tools (http://www.scratchbox.org/documentation/docbook/doctools.html)* , Janne Langi.

[2] *Installing Scratchbox (http://www.scratchbox.org/documentation/docbook/installdoc.html)* , Valtteri Rahkonen.

[3] *Debian Development with Scratchbox (http://www.scratchbox.org/documentation/docbook/debiandevelopment.html)* , Lauri Arimo.

[4] *Scratchbox website (http://www.scratchbox.org/)* .

[5] *Scratchbox devolopment team contacts (http://www.scratchbox.org/contact/)* .

[6] *GAR Architecture (http://www.lnx-bbc.org/garchitecture.html)* .