mauromol
Frequent Advisor

Posts: 34
Registered:
09-02-2012

Re: Building packages for the new firmware... someone tried it? [Edited]
07-26-2014 06:03 AM - edited 07-26-2014 12:06 PM

How to successfully build packages for WD My Cloud from source

The WD My Cloud comes with a Debian wheezy system on it. However WD customized that in a way that it is hardly possible (if not impossible at all) to install new packages using the standard "apt-get install <package name>" way, especially when you update the device firmware to version 4.x or later. The latest firmware, in fact, uses a modified Debian system with 64K sized memory pages: if you install a package using "apt-get install <package name>" from the standard repositories, almost certainly it won't run and produce just a laconic "Killed" output.
So, to install new packages on the My Cloud you need to build them from source on another system, copy the obtained deb packages on the My Cloud and install.
WD provides a GPL source package of the latest firmware on their website, which contains a build environment to perform this operation. This is totally at the end-user own risk, since no support at all is provided by WD and installing 3rd party software may void warrany.

An additional problem is that the build environment provided by WD has a "minor" problem that causes the building process of most packages to fail because of a GCC compiler segmentation fault.
This guide shows how to deal with this problem and, in general, how to create a build environment to easily do the task.
Some Linux knowledge is needed.

NOTE 1: as of now, this procedure is surely needed to build packages for the 4.x firmware; however, by experience, I find out that it's a useful procedure even if you are sticking with the 3.x firmware; so the guide explains how to build packages for both firmware versions
NOTE 2: in principle, it should be possible to build packages directly on the My Cloud, instead of using an external system; however I don't think it's a good idea to try this, because of many reasons, including: the My Cloud system is surely slower than a regular PC to compile packages; you would need to install all the development tools on the My Cloud itself (and this may require in turn to build them on another system first...)

Step 1: prepare the build system (required only once)

The build environment must be a Linux system, either Debian-based or Ubuntu-based. I personally suggest to create a virtual machine with such a system in it. This guide will take this route.
Download and install VirtualBox (**https://www.virtualbox.org/**) on your phisical system (either Linux, Windows or Mac). Start VirtualBox and create a new virtual machine for a Debian 64-bit. The default settings suggested by VirtualBox regarding memory, disk size and VM configuration should be ok to start.
After the VM is ready, download a Debian Wheezy 64-bit ISO image; I suggest the netinst image, which is the smaller one. Right now, Debian 7.0.6 is available and the direct links are:

- **http://cdimage.debian.org/debian-cd/7.6.0/amd64/iso-cd/debian-7.6.0-amd64-netinst.iso** (download via HTTP)
- **ttp://cdimage.debian.org/debian-cd/7.6.0/amd64/bt-cd/debian-7.6.0-amd64-netinst.iso.torrent** (download via BitTorrent)

Boot the VM with the downloaded ISO mounted in and install just the Debian base system (nothing else is required). Refer to VirtualBox and Debian websites for documentation on how to perform these operations.

Once the guest VM is installed, we need to make just a couple little tunings. First of all,

Debian Wheezy comes with qemu-user-static package version 1.1.2. Qemu is an environment needed to emulate an actual ARM system on another platform, like the AMD64 platform our build system consists of. It's a good idea to update Qemu to version 2.x from the wheezy-backports repository. To do this, start the build system and login. Then:

```
# sudo su
# echo "deb http://ftp.debian.org/debian wheezy-backports main
contrib non-free" >>/etc/apt/sources.list
# apt-get update
# apt-get -t wheezy-backports install qemu-user-static
```

This should also install binfmt-support, which is another packages needed by the build environment. If this is not the case, also type:

```
# apt-get install binfmt-support
```

Now, let's prepare the actual build environment. Let's create a folder in the /root directory of the build system and download the WD My Cloud 4.x firmware source package from WD website.

```
# cd /root
# mkdir wdmc-build
# cd wdmc-build
# wget http://download.wdc.com/gpl/gpl-source-sequoia-04.
00.00-607.zip
```

In case the link changes, refer to WD My Cloud support page to find the new one:
http://support.wdc.com/product/download.asp?groupid=904&lang=en

I then suggest to create different folders for different build scenarios. These are the possibilities:

- the target system may be firmware 3.x (4k) or firmware 4.x (64k)
- the source package base may be wheezy (Debian stable) or jessie (Debian testing); wheezy contains older packages, but that should run happily in My Cloud (which has a Wheezy in it!), while jessie contains newer packages that might also work and provide updated versions of many applications; I would personally recommend to build packages from wheezy, unless you absolutely need a newer version that is only in jessie

I don't recommend to mix things, so I would create different folders for any different combination. You're free to create just the one you are interested in, so among the following commands type fhe first ones, then only the block of commands of the combination(s) you're interested in, then the last command:

```
# cd /root/wdmc-build
# unzip gpl-source-sequoia-04.00.00-607.zip packages/build_tools
/debian/*

# mkdir 64k-wheezy
# cp -R packages/build_tools/debian/* ./64k-wheezy
# echo '#!/bin/bash' >>64k-wheezy/build.sh
# echo './build-armhf-package.sh --pagesize=64k $1 wheezy' >>64k-
wheezy/build.sh
# chmod a+x ./64k-wheezy/build.sh

# mkdir 64k-jessie
# cp -R packages/build_tools/debian/* ./64k-jessie
# echo '#!/bin/bash' >>64k-jessie/build.sh
# echo './build-armhf-package.sh --pagesize=64k $1 jessie' >>64k-
jessie/build.sh
# chmod a+x ./64k-jessie/build.sh


# mkdir 4k-wheezy
# cp -R packages/build_tools/debian/* ./4k-wheezy
# echo '#!/bin/bash' >>4k-wheezy/build.sh
# echo './build-armhf-package.sh --pagesize=4k $1 wheezy'
>>4k-wheezy/build.sh
# chmod a+x ./4k-wheezy/build.sh

# mkdir 4k-jessie
# cp -R packages/build_tools/debian/* ./4k-jessie
# echo '#!/bin/bash' >>4k-jessie/build.sh
```

```
# echo './build-armhf-package.sh --pagesize=4k $1 jessie'
>>4k-jessie/build.sh
# chmod a+x ./4k-jessie/build.sh

# rm -rf packages/
```

In this way, in every folder will be created a build.sh script that passes the right parameters to the WD provided script, requiring only the name of the package to build. This would work straight away if there weren't the problem with qemu I mentioned in the beginning, so another step is required to finish the prepare phase. Again, only type commands for the scenario(s) you're interested in:

64k-wheezy:

```
# cd /root/wdmc-build/64k-wheezy
# ./setup.sh bootstrap/wheezy-bootstrap_1.24.14_armhf.tar.gz build
# mv build/usr/bin/qemu-arm-static build/usr/bin/qemu-
arm-static_orig
# cp /usr/bin/qemu-arm-static build/usr/bin/qemu-arm-static
```

64k-jessie:

```
# cd /root/wdmc-build/64k-jessie
# ./setup.sh bootstrap/jessie-bootstrap_5.14.14_armhf.tar.gz build
# cp /usr/bin/qemu-arm-static build/usr/bin/qemu-arm-static
```

4k-wheezy:

```
# cd /root/wdmc-build/4k-wheezy
# ./setup.sh bootstrap/wheezy-bootstrap_1.24.14_armhf.tar.gz build
# mv build/usr/bin/qemu-arm-static build/usr/bin/qemu-
arm-static_orig
# cp /usr/bin/qemu-arm-static build/usr/bin/qemu-arm-static
```

4k-jessie:

```
# cd /root/wdmc-build/4k-jessie
# ./setup.sh bootstrap/jessie-bootstrap_5.14.14_armhf.tar.gz build
# cp /usr/bin/qemu-arm-static build/usr/bin/qemu-arm-static
```

The meaning of the above is the following: prepare an emulated ARM system and replace the qemu-arm-static binary provided by the bootstrap with the recent one we've installed in our actual build system.
Ignore any errors produced by `setup.sh`: that script is really buggy and many things it tries to do seem to be useless, unless we apply the mentioned qemu fix.

As a final step, I would recommend to edit the sources file list within the armhf build subsystem in order to be able to build packages that are in any of the distribution repositories. To do this, type the following:

```
# cd /root/wdmc-build
# nano <scenario>/build/etc/apt/sources.list
```

by replacing `<scenario>` with the desired one (64k-wheezy, 64k-jessie, 4k-wheezy, 4k-jessie); the nano editor will open, then replace the contents of the existing file with the following:

For 64k-wheezy and 4k-wheezy:

```
deb http://security.debian.org/ wheezy/updates main contrib non-free
deb-src http://security.debian.org/ wheezy/updates main contrib
non-free
deb http://ftp.debian.org/debian wheezy-updates main contrib
non-free
deb-src http://ftp.debian.org/debian wheezy-updates main contrib
non-free
```

```
deb http://ftp.debian.org/debian wheezy main contrib non-free
deb-src http://ftp.debian.org/debian wheezy main contrib non-free

#deb http://ftp.debian.org/debian wheezy-backports main contrib
non-free
#deb http://ftp.debian.org/debian wheezy-backports main contrib
non-free
```

For 64k-jessie and 4k-jessie:

```
deb http://security.debian.org/ jessie/updates main contrib non-free
deb-src http://security.debian.org/ jessie/updates main contrib
non-free
deb http://ftp.debian.org/debian jessie-updates main contrib
non-free
deb-src http://ftp.debian.org/debian jessie-updates main contrib
non-free
deb http://ftp.debian.org/debian jessie main contrib non-free
deb-src http://ftp.debian.org/debian jessie main contrib non-free
```

In case of wheezy, uncomment the last two lines if you want to build package versions from wheezy-backports. I don't know however if additional changes to the build.sh script (or better to the WD provided one) are needed to instruct apt to download and build the source from the backports repository. I don't have tried it yet. Anyway, I would recommend to leave those two lines commented unless you actually need something from the backports repository.

Save the file by hitting Ctrl+X, Y, Enter.

Now you're ready to build your first package!

Optional additional step (not strictly required) for wheezy scenarios
You may also want to use an updated C++ compiler to build packages in the wheezy scenarios. Debian Wheezy provides g++ package 4.6, but 4.7 is also available. With the following commands you can install the new version and then switch from one to the other using update-alternatives:

```
# cd /root/wdmc-build/<scenario>
# chroot build
# apt-get update
# apt-get install g++ g++-4.7
# update-alternatives --install /usr/bin/g++ g++ /usr/bin/g++.4.6 10
# update-alternatives --install /usr/bin/g++ g++ /usr/bin/g++-4.7 20
# update-alternatives --install /usr/bin/gcc g++ /usr/bin/gcc-4.6 10
# update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-4.7 20
# rm /usr/bin/cpp
# update-alternatives --install /usr/bin/cpp cpp /usr/bin/cpp-4.6 10
# update-alternatives --install /usr/bin/cpp cpp /usr/bin/cpp-4.7 20
# exit
```

After these commands, the default C++ compiler will be version 4.7. You can then switch to the old version by typing:

```
# cd /root/wdmc-build/<scenario>
# chroot build
# update-alternatives --set cpp /usr/bin/cpp-4.6
# update-alternatives --set gcc /usr/bin/gcc-4.6
# update-alternatives --set g++ /usr/bin/g++-4.6
# exit
```

Or use `update-alternatives --config <command>` to get an interactive prompt.

Continues on the next message

Content
(1,517 Views)