How to make Debian Packages for Qt C++ based Applications

10 November 2014 By Bhavyanshu Parasher

```
Short Link: http://goo.gl/ejk24v
```


Overview

Recently I released my own Qt app and it was a bit of a struggle to package it properly so that end users can install the app easily without much of a problem. I am going to share everything I learned from this little experience of mine. Please remember that the deb package you make through this article may not make lintian happy the very first time you make it as it might generate some warnings by the end of it. Those warnings can be corrected by doing little bit of search on google. Also you may need to modify the project structure a bit and organize it properly so make sure you keep a backup of the original source code in case you mess things up. For reference look at my project.

Project Structure

You have to make your project structure look exactly like shown below,

```
/packagename-1.0.0
 |-->src/                //Main directory where your source code is
 |-->debian/     //Contains important packaging files
 |-->COPYING     // App license file
 |-->appname.desktop //Desktop Entry Point file
 |-->README      //Optional
 |-->packagename.pro //This is not the pro file from source code. We wi
ll create it
```

- **src** directory - The src directory contains your main application which you have developed in Qt Creator. In the src directory, rename your .pro file to src.pro. You will understand soon why I am focusing on project structure so much. Keep rest of the files as it is.
- **COPYING** file - This is the license that you would provide with your package. In my project, it is called COPYING. Name doesn't matter. For OSS we generally use COPYING or LICENSE.
- **appname.desktop** - This is the desktop entry file. The one shown below is an example. Remember few

points here. Visualize how you would want your application to install on users desktop. For example, the binary file must go in /usr/bin folder. The icons for the executable generally go in /usr/share/pixmaps. So here *Exec* and *Icon* are crucial fields. Rest of them are important too but these two deserve special mention. Rest of them are pretty understandable.

```
[Desktop Entry]
Version=1.0
Type=Application
Terminal=false
Name=Editor
Exec=/usr/bin/Editor %F
Comment=Text Editor
Icon=/usr/share/pixmaps/editor.ico
StartupNotify=false
Encoding=UTF-8
Categories=TextEditor;WordProcessor;Development;GTK;GNOME;
```

- **packagename.pro** - The .pro file which is crucial for packaging. Do not forget to add this file. The contents of the file are important too. The SUBDIRS tells where the source code .pro file is kept so that build can occur. CONFIG is similar to what you write in your source code .pro file. So simply just copy the below contents in packagename.pro file.

```
SUBDIRS += src
TEMPLATE = subdirs
CONFIG += ordered warn_on qt debug_and_release
```

---

src Directory

Now the **src** directory must be having your src.pro file which is the main .pro file for your Qt application. Now let us tweak it a little to help us package it in a proper manner.

```
QT += core gui
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = Editor
TEMPLATE = app
CONFIG += warn_on

DESTDIR = ../bin
MOC_DIR = ../build/moc
RCC_DIR = ../build/rcc
UI_DIR = ../build/ui
unix:OBJECTS_DIR = ../build/o/unix
win32:OBJECTS_DIR = ../build/o/win32
macx:OBJECTS_DIR = ../build/o/mac

SOURCES += main.cpp\
                .
                . and HEADERS, FORMS etc goes below...
```

So the most important part is everything between **DESTDIR** and **OBJECTS_DIR**. You can just copy this part as it is in your .pro file and leave it there. No need to modify anything else in there. Refer to my app .pro file to understand this better.

---

debian Directory

You will have to be very careful about what all things you put in this directory. I would recommend you to not read any other tutorial on debian packaging for Qt Apps because that will leave you in confusion. I have made this tutorial from my bitter experience and I don't want anyone else to waste so much time just to package a .deb file and still be able to keep lintian happy.

The **debian** directory structure should look like this. You must create all these files. I would not recommend you to run `dh_make` (explained later) command at this time even if some other tutorial tells you to do so. Just follow on and you can copy paste from examples below.

```
/debian
|-->source/format //format is a file name
|--> control
|--> rules
|--> compat
|--> changelog
|--> copyright
|--> dirs
|--> packagename.install
|--> menu
|--> watch
```

Actually if you run `dh_make` command, it will generate all these files with lot of template data and also it will put in lot of .ex files which are example files. We don't need any of it. That is the reason I am telling you why you should strictly follow this tutorial or don't follow it at all. Otherwise you might end up in lot of confusion. Now let us take one file at a time.

- **source** directory - Create a *source* directory inside debian. In the *source* directory, add a file called format, with just one line in it `3.0 (quilt)`. Example

- **control** file - The content of the file are

```
  Source: sourcepackagename
  Section: editors
  Priority: optional
  Maintainer: Full Name <your@email.id>
  Build-Depends: debhelper (>= 9), qtbase5-dev, qtbase5-dev-tools, q
t5-qmake
  Standards-Version: 3.9.6
  Homepage: https://source-code-homepage.com

  Package: packagename
  Architecture: any
```

```
  Depends: libqt5gui5, libqt5widgets5, libqt5core5a, ${misc:Depend
s}, ${shlibs:Depends}
  Description: AppnameEditor is an editor program.
   Editor is an editor app based on Qt and C++.
   It is ....... It has ...... It supports ..........
   It is licensed under licensename. See so and so file for more inf
ormation.
```

This is a very general template for the control file. The *Source* field is "The name of the source package that this binary package came from, if different than the name of the package itself." The *Depends* field is what packages/libraries your package depends on for execution. This is still a very vague statement but it is easier to understand like that. You can read the documentation from Debian Maintainer Guide. **Notice** how I have left single space right before I began writing long description. First you mention one line abstract for your app. Then from next line, leaving a space in the beginning, you start writing description for your app. For syntax and other key explanation refer to the control manual.

- **rules** file - This is again a very crucial file. Luckily, debhelper understants Qt projects very well. So we do not actually need to tweak this. Just copy contents shown below in your rules file

```
  #!/usr/bin/make -f
  export QT_SELECT := qt5

  %:
      dh $@
```

NOTE : If you have any shared object library dependencies in your project then you can add `override_dh_shlibdeps:dh_shlibdeps -l$(shell pwd)/usr/lib` to help debhelper find the required library in a particular directory. Before dpkg-shlibdeps is run, LD_LIBRARY_PATH will have added to it just like you have to do while running app outside the Qt Creator directly through terminal. This is only in the case of shared object (.so) files. If you don't have it, you won't need this.

- **compat** file - The compat file defines the debhelper compatibility level. Just run this command `echo 9 > compat`. It will write 9 to the file. That is all what is required in this.

- **changelog** file - Copy paste below contents in it and edit the text according to your need. Make sure you don't modify the syntax or alignment of it. Also make sure to bump the version. Just add a debian revision (like below I have added -1 to the version).

```
  packagename (1.0.0-1) UNRELEASED; urgency=low

    * Initial Release

   -- Full Name <your@email.id>  Fri, 07 Nov 2014 10:15:05 +0530
```

That's all. Save this file and move on to next file.

- **copyright** file - You can refer to the templates in your own system (Go to usr/share/doc/dh_make/) or this copyright file. My project uses GPLv3. You can replace that text with your license terms. Keep the syntax and the alignment correct.

- **dirs** file - This file specifies any directories which we need but which are not created by the normal installation procedure. Refer to docs. The contents of file are :

```
  usr/bin
  usr/share/applications
```

  These two will always be present in all linux systems but just making sure because our binary file goes in usr/bin and our .desktop file goes in usr/share/applications as mentioned before.

- **packagename.install** file - This .install file has one line per file installed, with the name of the file (relative to the top build directory) then a space then the installation directory. So when we edited our .pro file, you saw we had set few variables in it like DESTDIR etc., so this is the reason why we had defined specific paths in our .pro file. Now we can easily write .install file without worrying about paths. Just go on and copy paste the below contents in it.

```
  bin/GeneratedBinary usr/bin
  appname.desktop usr/share/applications
  src/appicon.ico usr/share/pixmaps
```

  My icon file was in *src* directory only. So you too can keep it there. GeneratedBinary is the name of the generated binary file which is defined in src.pro file by the TARGET attribute. Use it exactly as it is (Don't ignore capital letters). appname.desktop is the one that is in top directory. The one we created above. Simple, right?

- **menu** file - This is actually an optional file. X Window System users usually have a window manager with a menu that can be customized to launch programs. If they have installed the Debian menu package, a set of menus for every program on the system will be created for them. Taken from docs. The contents can be :

```
  ?package(packagename):needs="x11" \
      section="Applications/Editors" \
      title="AppName" \
      command="/usr/bin/GeneratedBinary"
```

  Again, you just need to modify the text of it. Keep the syntax and alignment same. Replace packagename with name of your package, AppName with name of your application and GeneratedBinary by the name of whatever executable binary you have to execute your app.

- **watch** file (Optional for the beginning but important if you want to submit it to official debian) - It generates some lintian warning if watch file is not present. Hence, it is important if you want to submit your package to official debian. This is highly dependent on where you host your original source code and how you have done the versioning. If you understand regex, you can easily modify it. Like i use debian compatible versioning and my source is hosted on github. I use tags to identify new releases. The major release is always v1.0 and the debian revision is v1.0.1, v1.0.2 like that. So my watch file will be,

```
  version=3
  https://github.com/bhavyanshu/LightMd_Editor/tags .*/archive/[a-z]
(\d\S*)\.tar\.gz
```

So it will automatically detect if the current build is from latest source or not.

> Please note that there are other files too. Like manpages, docs etc. sp please refer to this document for details on how to write these. You must try to satisfy debian policy. Also, it is recommended to generate a key pair to later on sign .dsc and .changes files.

Creating a clean chroot environment

For this I have used pdebuild.

```
sudo apt-get install pdebuild
```

pdebuild is a wrapper for pbuilder which lets you easily create a clean chroot environment. Basically we want to do this step so that we can ensure that build dependencies which you have mentioned in control file are satisfied and our package must not need anything else to build apart from what you have mentioned in control file.

Now first create a *.pbuilderrc* file. In that add the following

```
BASETGZ=$HOME/pbuilder/base.tgz
BUILDPLACE=$HOME/pbuilder/build/
BUILDRESULT=$HOME/pbuilder/result/
DEBEMAIL='Your Name <your@emaild.id>'
BUILDUSERNAME=username
AUTO_DEBSIGN=yes
APTCACHEHARDLINK=no
DEBBUILDOPTS="-sa"
```

Here username is the username that you use to log into your linux desktop. For example, for me it `BUILDUSERNAME=bhavyanshu`. In DEBBUILDOPTS, **-sa** : means that source always includes orig & **-sd** : source is diff and .dsc only. Next create a directory by name *pbuilder* in your *home*.

Now use the following command

```
sudo pbuilder --create --mirror "https://archive.ubuntu.com/ubuntu"
```

Since I am on linux mint, i had to specify mirror url. Ubuntu users don't need to specify mirror argument as far as i know. It is a long process. It will take time so meanwhile please refer to how to make a gpg key pair and create it. You will be needing it soon. Once the above operation is completed, you will see *build/* and *result/* in *home/pbuilder* directory along with *base.tgz* file. Do not remove this file or else you will need to execute above command again and it's going to take hell lot of time again. It is pointless to do it over and over again.

Time to pack things up

Note that upstream should be having all the files that we just now created. Make sure of that. If you don't know what is upstream, then let me quickly explain you. Upstream is basically what your master branch on github looks like. It should always be build ready and should represent stable. Now the time has finally come to package it and start distributing. If you followed above structure strictly, you should get a .deb file for your app ready in no time.

1. Go to the parent folder of *packagename-1.0.0*. Create **packagename_1.0.0.orig.tar.gz** by compressing the *packagename-1.0.0* folder. Ofcourse you can create this with dh_make but I prefer to do it manually. Now your parent directory should have two things. One is the package_1.0.0.orig.tar.gz and the other is the your package-1.0.0 folder.
2. Open terminal, `cd ./packagename-1.0.0/` hit enter to get back into your package folder. Now execute the following command.

```
dh_make -e your@email.id -f ../packagename_1.0.0.orig.tar.gz -s
```

3. Option *-s* in above command is for single binary. Do not use *-s* if you have multiple binary. Refer to dh_make manual and then finally

```
pdebuild --debbuildopts -sa
```

The pdebuild command will automatically try to sign .dsc and .changes file with the provided key id (If it can find any). The -sa is for including source with binary. It is requires since you will be packaging for the first time. Run lintian on the generated file and look for errors/warnings. You have to fix those before you can move ahead with filling an ITP bug. You must also read the debian policy as mentioned above. If the build result is a satisfactory one, sign the .dsc and .changes files with your private GPG key using the debsign command and your .deb file will be ready for distribution. If there is a signing issue, please read the document on signing the sources.

I have spent a lot of time documenting this in a hope that it will help someone. I too looked online for some Qt specific packaging tutorials but the ones I found were too outdated. Leave a comment below if you are stuck somewhere. I will help you out with it.

References

- Structure of a debian package
- Debian New Maintainers' Guide
- Other files under debian directory
- Debian Policy Manual
- Lintian

← Previous | Archive | Next →

Join the discussion…

David Schmider • 4 months ago

Hi, Well I have to say this is a great tutorial and exactly the thing I have been searching for! However I am having a problem on the last stage of pdebuild.
Right at the end it fails:
dpkg-genchanges: error: cannot read files list file: No such file or directory
dpkg-buildpackage: error: dpkg-genchanges gave error exit status 2
E: Failed autobuilding of package

prior to that though there is a mistake in the email address it uses: (i have put xxx to change whats not relevant)
dh binary
dpkg-genchanges -sa -eDavid Schmider <dvid.schmider@xx.me>
>../package_4.0.3.0_amd64.changes

notice the missing 'a' in david on the email address. It happens further up as well. However, nowhere in my source or my ket is there a file with this.
Totally got me stumped!
Any ideas?
Cheers
David

^ v • Reply • Share ›

Bhavyanshu Parasher Mod ➚ David Schmider • 4 months ago

Hi David,

For issue 1) This is most likely caused because of wrongly written debian/rules file. Can you post your rules file below so I can take a look if in case you have modified it and are not using the same version as mine?

For issue 2) In your .bashrc file (Located in Home folder), simply add the following lines for proper email and name

DEBEMAIL="Email@address.com"
DEBFULLNAME="Your full name here"
export DEBEMAIL DEBFULLNAME

Because by default the pre-build commands take default sys name. To override this, use this .bashrc method and try again. Make sure you run this

. ~/.bashrc
once you are done editing the bashrc.

^ v • Reply • Share ›

David Schmider ➚ Bhavyanshu Parasher • 4 months ago

Thanks for replying so quickly!

I looked in the rules file and the % was not indented, so I recopied yours again:

#!/usr/bin/make -f

export QT_SELECT := qt5

%:
dh $@

This now has changed the error to: (and Ive contained more of the build output):

dpkg-source: info: extracting ercompanion in ercompanion-4.0.3.0
dpkg-source: info: unpacking ercompanion_4.0.3.0.orig.tar.gz
dpkg-source: info: applying ercompanion_4.0.3.0.diff.gz
I: Building the package
I: Running cd tmp/buildd/*/ && env PATH="/usr/sbin:/usr/bin:/sbin:/bin" dpkg-buildpackage -us -uc "-eDavid Schmider <dvid.schmider@lprs.co.uk>" -sa -rfakeroot

---

see more

• Reply • Share ›

Bhavyanshu Parasher Mod ➚ David Schmider • 4 months ago

Okay, so here are few tips first of all.
The original tutorial needs to be updated too. I will do it today itself. Thanks for reminding me.

1) Solve "dpkg-source: warning: no source format specified in debian/source/format" by creating a debian/source/format file and adding the following line

3.0 (quilt)

Refer to this example file https://github.com/bhavyanshu/...

2) The rules file - Look at this raw file from my project. I still feel there is some issue between copy paste hence I suggest using the raw file. But delete the "override_dh_shlibdeps" from it because I am sure you don't need it.

https://raw.githubusercontent....

If in case you still get same error again, I might need to look into your project structure myself. If it is on github (or any other) then let me know. I will definitely look into it.

• Reply • Share ›

David Schmider ➚ Bhavyanshu Parasher • 3 months ago

Hi again,
Sorry its been a few days, as I've been forced to be on other things.
I made the changes, added the format file, and re pasted from your rules file, which did shift the formatting slightly as mine were indented by a space.
Anyway still the same issue occurs:
dpkg-source -b ercompanion-4.0.3.0
dpkg-source: warning: version does not contain a revision

dpkg-source: info: using source format `3.0 (quilt)'
dpkg-source: info: building ercompanion using existing ./ercompanion_4.0.3.0.orig.tar.gz
dpkg-source: info: building ercompanion in ercompanion_4.0.3.0.debian.tar.gz
dpkg-source: info: building ercompanion in ercompanion_4.0.3.0.dsc
debian/rules build
dh build
fakeroot debian/rules binary
dh binary

see more

• Reply • Share ›

Bhavyanshu Parasher Mod ➚ David Schmider • 3 months ago

Google drive or dropbox? I just need to take a look at project structure and make tweaks there itself. I might not be able to compile it cause I still run i386 machine. For GD & Dropbox, share folder by sending invite to bhavyanshu.spl@gmail.com
I will look into it.

• Reply • Share ›

Published
10 November 2014 By Bhavyanshu Parasher - ask@bhavyanshu.me

Tags

debian [1]

.

Glide To Top