**gsview / Qt: A Guide to the Perplexed**

Fred Ross-Perry

January, 2015

This document contains instructions for building the QT-based version of gsview for Mac OS X and Linux.

**Obtaining source and building**

1. Download and install the latest Qt. As of this writing, that’s version 5.4:

[http://download.qt-project.org/official\_releases/online\_installers](http://download.qt-project.org/official_releases/online_installers/qt-opensource-mac-x64-online.dmg)

1. For OS X, Install and configure the latest XCode and command line tools.

For Linux, the standard GCC tools.

1. Use git to clone the source.

git clone ghostscript.com:/home/fred/repos/qt-gsview.git

1. mupdf and ghostpdl are included as submodules. Update them:

cd qt-gsview

git submodule update --init

1. Now update mupdf’s submodules and build it, but without X11:

cd qt-gsview/mupdf

git submodule update –init

make HAVE\_X11=no XCFLAGS="-DSUPPORT\_GPROOF"

This results in static libraries that will be linked in when you build gsview.

1. Build ghostpdl, xps and gs. This will produce gs and gxps executables that are included in the build.

cd qt-gsview/ghostpdl

make clean

./autogen.sh --with-libiconv=no --disable-cups \

--without-libpaper --disable-fontconfig

make

1. Open gsview.pro using Qt Creator (just installed in step 2). The first time, you’ll be asked to configure the project. For OS X, select the kit called “Desktop Qt 5.4.0 clang 64bit” and click “Configure Project”. For Linux, select “Desktop Qt 5.3 GCC 64bit”.
2. One more step needs to be done, one time. Switch to Projects Mode by clicking the “Projects” button on the left. Now change the shadow build path to “build/debug” for the debug configuration, and “build/release” for the release configuration.
3. Select a configuration and “Run qmake” (found under the Build menu). This creates the shadow build directory and creates the makefile that will be used.
4. Now build by running “Build All” (found under the Build menu).

**Project File (.pro) Highlights**

For Linux release, we’re linking using –rpath so the Qt shared libraries can be placed into a “libs” subfolder when installed.

# look for shared libs in ./libs

# unix only, release only

CONFIG(release,debug|release) {

unix:!macx {

QMAKE\_LFLAGS += "-Wl,-rpath,\'\$$ORIGIN/libs\'"

}

}

defining this will cause the code to use CUPS for printing when it can. However, with the addition of a new print dialog, this is not working well, and so is currently disabled.

unix: DEFINES += USE\_CUPS

Control whether we use native file dialogs. Debugging can be easier if we don’t. But it should be set to true for deployment.

DEFINES += USE\_NATIVE\_FILE\_DIALOGS=true

Libraries to link with. Order is important.

LIBS += -L$$PWD/mupdf/build/debug/

unix: LIBS += -lmupdf -lfreetype -ljbig2dec -ljpeg

-lopenjpeg -lz -lmujs -lcups

macx: LIBS += -lssl -lcrypto

gs and gxps executables are copied into an adjacent apps folder:

unix {

QMAKE\_POST\_LINK += $$quote(mkdir -p ./apps $$escape\_expand(\n\t))

QMAKE\_POST\_LINK += $$quote(cp $$PWD/ghostpdl/gs/bin/gs ./apps/gs $$escape\_expand(\n\t))

QMAKE\_POST\_LINK += $$quote(cp $$PWD/ghostpdl/xps/obj/gxps ./apps/gxps $$escape\_expand(\n\t))

}

OSX icon and Info.plist file:

macx {

ICON = resources/gsview\_app.icns

QMAKE\_POST\_LINK += $$quote(cp $$PWD/resources/gsview\_mac.plist $$OUT\_PWD/gsview.app/Contents/Info.plist $$escape\_expand(\n\t))

}

**Deploying – OS X**

There are several scripts in the **mac\_deploy** folder:

**1-build-release.command**

**2-code-sign.command**

**3-verify-code-sign.command**

**4-build-dmg.command**

Most of the scripts define **QTBIN** or **QTDIR**. You’ll want to edit these to reflect the location of Qt on your system.

Open a terminal window and run, in this order,

**1-build-release.command**. This will produce a release build in the **build/release** folder, and puts a copy in the **mac\_deploy** folder. As part of this step, Qt frameworks and plugins are added to the .app package.

**2-code-sign.command.** This signs the app using an identity that should be installed on your system. This step runs **rebundle.py**, a Python program that reorganizes the app package contents if necessary to comply with signing requirements.

**3-verify-code-sign.command.** Run this to verify that the signing is valid.

**4-build-dmg.command.** This prepares a .DMG file containing the app and a license agreement.

**Deploying - Linux**

There are two flavors to build for Linux, 64-bit and 32-bit. Each has a folder, **linux-deploy** and **linux-deploy-32**. In each folder are two scripts, **build-release.sh** and **build-installer.sh**.

Edit **build-release.sh** and change the value of **QTBIN** so it reflects the place where the gcc folder is. Mine are

64-bit: **$HOME/Qt5.4.1/5.4/gcc\_64/bin**

32-bit: **$HOME/Qt5.4.1/5.4/gcc/bin**

In a Terminal window, execute **build-release.sh**. This will produce a release build in the **build/release** folder.

Then, execute **build-installer.sh**. This will create an installer program that the end user will run on their system.

**Notes**

If you’re using Qt 5.3, you’ll need to also install the OS X 10.8 SDK, which is no longer included in the latest XCode releases. It should be installed inside the XCode app package, here:

/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX10.8.sdk

If you install XCode updates in the future, you’ll lose the 10.8 SDK. So save it somewhere so you can restore it.

It looks like this is no longer necessary with 5.4.

When using 5.3, the debugger may not be set up properly. To fix this, clone the kit called “Desktop Qt 5.3 clang 64bit” and add the debugger.