RaspberryPi2EGLFS (original article for 5.6 https://wiki.qt.io/RaspberryPi2EGLFS)

A modern guide for cross-compiling Qt 5.7 for HW accelerated OpenGL with eglfs on Raspbian Jessie and setting up Qt Creator

Note that this is not intended for running desktop-style, windowed Qt apps under X11, but rather for the real embedded/device creation use case where the Qt app runs fullscreen on top of dispmanx/EGL using the Broadcom drivers.

For detailed, generic information about eglfs and Qt on Embedded Linux check the Qt documentation.

Step by step

1. Get the latest image from https://downloads.raspberrypi.org/raspbian_latest

2. Unzip and write it to a memory card. Replace ... with the SD card device (check with lsblk or dmesg eg. mmcblk0).

```
sudo dd if=2015-09-24-raspbian-jessie.img of=... bs=4M
```

3. Boot it up, run raspi-config, change it to boot to the console instead of X, change the GPU memory to 256 MB, install a bunch of development files (for simplicity we use build-dep, not everything is really needed, but it is easier this way), and prepare our target directory:

```
sudo raspi-config
sudo nano /etc/apt/sources.list
```
and uncomment the deb-src line.
```
sudo apt-get update
sudo apt-get build-dep qt4-x11
sudo apt-get build-dep libqt5gui5
sudo apt-get install libudev-dev libinput-dev libts-dev libxcb-xinerama0-dev

sudo mkdir /usr/local/qt5pi
sudo chown pi:pi /usr/local/qt5pi
```

4. Back on the host PC, create our working directory and get a toolchain:

```
mkdir ~/raspi
cd ~/raspi
git clone https://github.com/raspberrypi/tools
```

5. Create a sysroot. Using rsync we can properly keep things synchronized in the future as well. Replace IP with the address of the Pi.

```
mkdir sysroot sysroot/usr sysroot/opt
rsync -avz pi@IP:/lib sysroot
rsync -avz pi@IP:/usr/include sysroot/usr
rsync -avz pi@IP:/usr/lib sysroot/usr
rsync -avz pi@IP:/opt/vc sysroot/opt
```

6. Adjust symlinks to be relative. Instead of the old fixQualifiedLibraryPaths get a script that works:

```
wget https://raw.githubusercontent.com/riscv/riscv-
poky/master/scripts/sysroot-relativelinks.py
chmod +x sysroot-relativelinks.py
./sysroot-relativelinks.py sysroot
```

7. Get qtbase 5.6 and configure Qt. The target directory is /usr/local/qt5pi on the Pi, the host tools like qmake will go to ~/raspi/qt5, while make install will target ~/raspi/qt5pi (this is what we will sync to the device). If using the old Pi instead of the Pi 2, change -device to linux-rasp-pi-g++. Don't forget to adjust the path to your ~/raspi directory if you changed that. For some reason the ~/ in the paths may not work, if this the case just use full paths.

```
git clone git://code.qt.io/qt/qtbase.git -b 5.6
```

if this doesn't work :

download the source manually and exctract into ~/raspi/qt5

http://download.qt.io/archive/qt/5.7/5.7.0/single/qt-everywhere-opensource-src-5.7.0.tar.xz

Before you start compiling you need to edit the file :
qtbase/mkspecs/devices/linux-rasp-pi**2**-g++/qmake.conf

Add the following lines to the file : (source https://forum.qt.io/topic/62264/failed-to-cross-compile-qt-5-6-on-rpi-2-fatal-error-bcm_host-h-no-such-file-or-directory/7)

```
INCLUDEPATH += $[QT_SYSROOT]/opt/vc/include
INCLUDEPATH += $[QT_SYSROOT]/opt/vc/include/interface/vcos
INCLUDEPATH += $[QT_SYSROOT]/opt/vc/include/interface/vcos/pthreads
INCLUDEPATH += $[QT_SYSROOT]/opt/vc/include/interface/vmcs_host/linux
```

After we save the file we can now start to make QT5.7

```
cd qtbase
./configure -release -opengl es2 -device linux-rasp-pi2-g++ \
-device-option CROSS_COMPILE=~/raspi/tools/arm-bcm2708/gcc-linaro-arm-linux-
gnueabihf-raspbian/bin/arm-linux-gnueabihf- \
-sysroot ~/raspi/sysroot -opensource -confirm-license -make libs \
-prefix /usr/local/qt5pi -extprefix ~/raspi/qt5pi -hostprefix ~/raspi/qt5 -v

make
make install
```

8. Deploy to the device. We simply sync everything from ~/raspi/qt5pi to the prefix we configured above.

```
rsync -avz qt5pi pi@IP:/usr/local
```

9. Build an example:

```
cd qtbase/examples/opengl/qopenglwidget
~/raspi/qt5/bin/qmake
make
scp qopenglwidget pi@IP:/home/pi
```

10. On the device, let the linker find the Qt libs:

```
echo /usr/local/qt5pi/lib | sudo tee /etc/ld.so.conf.d/qt5pi.conf
sudo ldconfig
```

11. Still on the device, fix the EGL/GLES library nonsense:

The device may have the Mesa version of libEGL and libGLESv2 in /usr/lib/arm-linux-gnueabihf, resulting Qt apps picking these instead of the real thing from /opt/vc/lib. This may be fine for X11 desktop apps not caring about OpenGL performance but is totally useless for windowing system-less, fullscreen embedded apps.

```
sudo rm /usr/lib/arm-linux-gnueabihf/libEGL.so.1.0.0 /usr/lib/arm-linux-gnueabihf/libGLESv2.so.2.0.0
sudo ln -s /opt/vc/lib/libEGL.so /usr/lib/arm-linux-gnueabihf/libEGL.so.1.0.0
sudo ln -s /opt/vc/lib/libGLESv2.so /usr/lib/arm-linux-gnueabihf/libGLESv2.so.2.0.0
```

In case you are using qtwebengine please make sure to add missing symbolic links.

```
sudo ln -s /opt/vc/lib/libEGL.so /opt/vc/lib/libEGL.so.1
sudo ln -s /opt/vc/lib/libGLESv2.so /opt/vc/lib/libGLESv2.so.2
```

You may want to save the originals somewhere, just in case.

12. Run qopenglwidget we deployed to /home/pi. At this point it should just work at fullscreen with 60 FPS and mouse, keyboard, and possibly touch support.

13. Build other Qt modules as desired, the steps are the same always:

Open the folder of the desired module , eg `~/raspi/qt5/qtserialport`

Then compile and install the module and deploy to pi

```
~/raspi/qt5/bin/qmake -r
make
make install
```

then deploy the new files by running
`rsync -avz qt5pi pi@IP:/usr/local`
in ~/raspi

Additional notes

Frequently asked question: I only get a low resolution like 640x480 or even 576x416 with black boxes. How to fix this?

As a quick fix, try adding disable_overscan=1 to /boot/config.txt and after a reboot check with /opt/vc/bin/tvservice what modes are available. For example, to switch to 1024x768:

```
$ /opt/vc/bin/tvservice -m DMT
Group DMT has 4 modes:
           mode 4: 640x480 @ 60Hz 4:3, clock:25MHz progressive
           mode 9: 800x600 @ 60Hz 4:3, clock:40MHz progressive
           mode 16: 1024x768 @ 60Hz 4:3, clock:65MHz progressive
           mode 35: 1280x1024 @ 60Hz 5:4, clock:108MHz progressive
$ /opt/vc/bin/tvservice -e "DMT 16 HDMI"
$ fbset -xres 1024 -yres 768
```

Troubleshooting

Enabling the logging categories under qt.qpa is a good idea in general. This will show some debug prints both from eglfs and the input handlers.

```
export QT_LOGGING_RULES=qt.qpa.*=true
./qopenglwidget
```

A typical output would like like this:

```
qt.qpa.egldeviceintegration: EGL device integration plugin keys:
("eglfs_brcm", "eglfs_kms")
qt.qpa.egldeviceintegration: EGL device integration plugin keys (sorted):
("eglfs_brcm", "eglfs_kms")
qt.qpa.egldeviceintegration: Trying to load device EGL integration
"eglfs_brcm"
qt.qpa.egldeviceintegration: Using EGL device integration "eglfs_brcm"
Unable to query physical screen size, defaulting to 100 dpi.
To override, set QT_QPA_EGLFS_PHYSICAL_WIDTH and QT_QPA_EGLFS_PHYSICAL_HEIGHT
(in millimeters).
qt.qpa.input: libinput: input device 'Logitech Optical USB Mouse',
/dev/input/event0 is a pointer caps = relative-motion button
qt.qpa.input: libinput: input device 'Apple Inc. Apple Keyboard',
/dev/input/event1 is a keyboard
qt.qpa.input: libinput: input device 'Apple Inc. Apple Keyboard',
/dev/input/event2 is a keyboard
qt.qpa.input: libinput: input device 'Raspberry Pi Sense HAT Joystick',
/dev/input/event3 is a keyboard
qt.qpa.input: Using xkbcommon for key mapping
```

Verify that eglfs_brcm is in use and that input devices are correctly found.

When using a touchscreen, setting the correct physical screen size may be essential to get properly scaled, finger friendly user interface elements with Qt Quick Controls. When using ordinary monitors via HDMI, the default 100 dpi may be acceptable.

It is also wise to carefully check the output of configure (saved as config.summary) before proceeding to build qtbase. Check that the following are marked as 'yes':

```
Support enabled for:
  ...
  Evdev .................. yes
  FontConfig ............. yes
  FreeType ............... yes (system library)
  ...
  libinput................ yes
  ...
  OpenGL / OpenVG:
    EGL .................. yes
    OpenGL ............... yes (OpenGL ES 2.0+)
  QPA backends:
    EGLFS ................ yes
      ...
      EGLFS Raspberry Pi . yes
    ...
    LinuxFB .............. yes
    ...
  udev ................... yes
  xkbcommon-evdev......... yes
```

Regarding keyboard input

By default Qt attempts to disable the keyboard and hide the cursor on application startup. This is very handy since this way keystrokes will not go to the console (which is enabled by default in a normal Raspbian image) "underneath". However, at the moment this will all silently fail when starting an application remotely via ssh. This is the explanation for the (harmless) 9;15] and similar prints and the keyboard input unexpectedly going to the console as well. As a workaround, start apps directly on the console. Alternatively, you could experiment with getting rid of the first TTY by doing sudo systemctl disable getty@tty1.service and reboot. (tty2 and others remain usable)

Qt Creator

Once Qt is on the device, Qt Creator can be set up to build, deploy, run and debug Qt apps directly on the device with one click.

```
Go to Options -> Devices
  Add
    Generic Linux Device
    Enter IP address, user & password
    Finish
```

```
Go to Options -> Compilers
  Add
    GCC
    Compiler path: ~/raspi/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-
raspbian/bin/arm-linux-gnueabihf-g++
Go to Options -> Debuggers
  Add
    ~/raspi/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian-
x64/bin/arm-linux-gnueabihf-gdb
Go to Options -> Qt Versions
  Check if an entry with ~/raspi/qt5/bin/qmake shows up. If not, add it.

Go to Options -> Build & Run
  Kits
    Add
      Generic Linux Device
      Device: the one we just created
      Sysroot: ~/raspi/sysroot
      Compiler: the one we just created
      Debugger: the one we just created
      Qt version: the one we saw under Qt Versions
      Qt mkspec: leave empty
```

Done. At this point you should be able to start a new project with the new kit, build and deploy it, etc.

Note: While things will usually just work for applications, developing libraries and in particular, Qt modules may be problematic when it comes to deployment. The per-project Run tab under Projects -> Build & Run is your friend. In some cases the target deployment paths will just be wrong. Life is too short for worrying about all the intricate details of Creator and the build system. Therefore, if all else fails, use Add Deploy Step ("Make" and "Custom Process Step" are extremely handy, anything can be made working with a combination of make install, rsync and scp) and change Run configuration to Custom Executable.

Sense HAT

To access the sensors and leds on the Sense HAT, you can use the unofficial Qt Sense HAT module on qt-labs. Check it out via git and build it like any other Qt module.

The joystick shows up as an ordinary evdev device providing key events so it will just work. Regarding the leds and sensors, see the README.

Qt Multimedia

Unfortunately the GStreamer-based multimedia stuff is not quite usable at the time of writing - accelerated video works only sometimes (and with glitches), while the camera is just broken.

As an alternative to Qt Multimedia, try using OpenMAX directly. For an open implementation refer to this project. For the Raspberry Pi Camera Module take a look at the raspistill application sources for an example on how to get the camera preview image into an OpenGL texture.

If you wish to experiment with Qt Multimedia, try the following:

Before building Qt Multimedia 5.6, make sure the following are installed:

```
sudo apt-get install gstreamer1.0-omx libgstreamer1.0-dev libgstreamer-plugins-base1.0-dev
```

Do not forget to sync the headers and libs back to the sysroot on the host PC and re-run the sysroot-relativelinks.py script.

Once the GStreamer 1.0 dependencies are in place, make sure Qt Multimedia is built with 1.0 support:

```
~/raspbi/qt5/bin/qmake -r GST_VERSION=1.0
```

To verify that the accelerated OpenMAX path is used for H.264 videos, do *export GST_DEBUG=omx:4* before running a video playback app like the qmlvideofx example.

Red highlighted text are additions to the original article

Yellow highlighted text are command inputs