Cross-Compiling Qt 5.15 for Raspberry Pi 4

This is a guide for Cross-compiling Qt 5.15 for a Raspberry Pi 4 running Ubuntu 22.04 Server from a host machine running Ubuntu 22.04.
Tested with below prerequisites.

Hardware:

Host: Intel® Core™ i7-10700 CPU @ 2.90GHz × 16
Target: Raspberry Pi 4 Model B

Software:

Host: Ubuntu 22.04 LTS (64 bits)
Target: OS Ubuntu 20.04 Server (64 bits)
Cross Compiler: gcc-9-aarch64-linux-gnu && g++-9-aarch64-linux-gnu

It is assumed that you have a SD card with ubuntu 22.04 Server installed in your Raspberry Pi, otherwise download it and follow the installation guide.

we have downloaded the image and prepared the SD card on Ubuntu machine.

The following list summarizes the main steps to cross-compile Qt 5.15 for Raspberry Pi, we will be describing each of them in this guide.

1 Flash the image onto on SD card

2 Update the system with the following commands-[RPI4]

3 Install development libraries-[RPI4]

4 Prepare target folder –[RPI4]

5 Prepare Ubuntu Host Machine –[Host]

6 Create working folder download QT Resources and set a toolchain–[Host]

7 Create and configure a sysroot [Host]

8 Configure Qt for cross compilation –[Host]

9 Setup Qt Creator for Raspberry Pi cross compilation

1 Flash the image onto on SD card

To flash ubuntu 22.04 server os on SD card we used Raspberry Pi imager. On Raspberry Pi imager select required options opearting system, storage device, settings to configure os.

https://www.raspberrypi.com/software

2 Update the system with the following commands- [RPI4]

```
$ sudo apt-get update && sudo apt-get upgrade
$ reboot
```

#configure wpa_supplicant:(optional if you configured on RPi imager)

Create or edit the /etc/wpa_supplicant/wpa_supplicant.conf file to configure your WiFi network. Replace your_network_name and your_network_password with your actual SSID (network name) and password.

```
$ sudo apt install wpasupplicant
$ sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
$ sudo nano/etc/wpa_supplicant/wpa_supplicant.conf
```

```
network={ ssid="your_network_
name"
psk="your_network_password"
}
```

```
$ sudo nano /etc/netplan/50-cloud-init.yaml
```

```
network:
  version: 2
  wifis:
    wlan0:
      access-points:
        "your_network_name":
          password: "your_network_password"
      dhcp4: true
      optional: true
```

```
$ sudo netplan apply
```

if no output then reboot the system

```
$ journalctl -xe | grep -iE "wpa_supplicant|network|wifi"
$ sudo systemctl start systemd-networkd
$ sudo systemctl status systemd-networkd
$ ping google.com
```

#Enable UART Communication

The UART should already be enabled by default on the Raspberry Pi 4.
To check the status of the UART hardware itself,you can use the
ls /dev command.
UART devices are typically named/dev/ttyS0(UART0)and/dev/ttyAMA0(UART1).

```
$ ls /dev/ttyS* /dev/ttyAMA*
```

If you see these devices listed , it means UART hardware is recognized
by the system.

3 Install development libraries-[RPI4]

We need to install some development libraries, so the first thing to do is to allow the system to install source packages

```
$ sudo apt-get install -y qtbase5-dev qt5-qmake
$ sudo apt-get install -y libqt5webengine-data
```

```
$ sudo apt-get install libboost-all-dev libudev-dev libinput-dev libinput-dev

libts- dev libmtdev-dev libjpeg-dev libfontconfig1-dev

$ sudo apt-get install libssl-dev libdbus-1-dev libglib2.0-dev libxkbcommon-dev
libegl1-mesa-dev libgbm-dev libgles2-mesa-dev mesa-common-dev


$ sudo apt-get install libasound2-dev libpulse-dev gstreamer1.0-omx
libgstreamer1.0-dev libgstreamer-plugins-base1.0-dev gstreamer1.0-alsa
```

```
$ sudo apt-get install libasound2-dev libpulse-dev gstreamer1.0-omx libgstreamer1.0-dev libgstreamer-plugins-base1.0-dev gstreamer1.0-alsa

$ sudo apt-get install libasound2-dev libpulse-dev gstreamer1.0-omx libgstreamer1.0-dev libgstreamer-plugins-base1.0-dev gstreamer1.0-alsa

$ sudo apt-get install libsrtp2-1 && sudo apt-get install "^libxcb.*"

$ sudo apt-get install flex bison libxslt-dev ruby gperf libbz2-dev libcups2-dev libatkmm-1.6-dev libxi6 libxcomposite1

$ sudo apt-get install libfreetype6-dev libicu-dev libsqlite3-dev libxslt1-dev libavcodec-dev libavformat-dev libswscale-dev
$ sudo apt-get install libx11-xcb-dev libxcb-keysyms1 libxcb-keysyms1-dev libxcb-image0 libxcb-image0-dev libxcb-shm0 libxcb-shm0-dev libxcb-icccm4 libxcb-icccm4-dev
$ sudo apt-get install libxcb-glx0-dev libxi-dev libdrm-dev libssl-dev libxcb-xinerama0 libxcb-xinerama0-dev
$ sudo apt-get install libatspi2.0-0
$ sudo apt-get install libxcb-sync1 libxcb-sync-dev libxcb-render-util0 libxcb-render-util0-dev libxcb-xfixes0-dev libxrender-dev libxcb-shape0-dev libxcb-randr0-dev


# Optional package for multimedia
$ sudo apt-get install libgstreamer1.0-dev libgstreamer-plugins-base1.0-dev libgstreamer-plugins-bad1.0-dev gstreamer1.0-plugins-base gstreamer1.0-plugins-good gstreamer1.0-plugins-bad gstreamer1.0-plugins-ugly gstreamer1.0-libav gstreamer1.0-tools gstreamer1.0-x gstreamer1.0-alsa gstreamer1.0-gl gstreamer1.0-gtk3 gstreamer1.0-qt5 gstreamer1.0-pulseaudio
```

4 Prepare target folder –[RPI4]

This step just involves creating a folder in our Raspberry Pi for the pi user. This folder (/usr/local/qt5pi) will be used to deploy Qt from our computer to the Raspberry Pi.

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

5 Prepare Ubuntu Host Machine –[Host]

First of all, bring Ubuntu up to date and install some required libraries:
ping your target device

```
$ ping 192.168.16.25 (target board ip)
$ sudo apt-get update && sudo apt-get upgrade
$ sudo bash
$ apt-get install gcc git bison python3 gperf pkg-config
$ apt install make
$ apt install libclang-dev
$ apt install build-essential
```

6 Create working folder download QT Resources and set a toolchain–[Host]

```
$ mkdir /opt/qt5pi
$ chown username:username/opt/qt5pi
$ cd /opt/qt5pi/
$ wget https://download.qt.io/official_releases/qt/5.15/5.15.2/single/qt-everywhere-src-5.15.2.tar.xz
$ tar xf qt-everywhere-src-5.15.2.tar.xz


$ sudo apt install gcc-9-aarch64-linux-gnu
$ sudo apt install g++-9-aarch64-linux-gnu
```

#Add Device config for linux-rasp-pi4b-v3d-g++

Add below content in $ nano qmake.conf

qmake configuration for the Raspberry Pi 4 (64-

bit) include(../common/linux_device_pre.conf)

QT_QPA_DEFAULT_PLATFORM = xcb

QMAKE_LIBS_EGL   += -lEGL

QMAKE_LIBS_OPENGL_ES2 += -lGLESv2 -lEGL

DISTRO_OPTS   += aarch64

QMAKE_CFLAGS   = -march=armv8-a -mtune=cortex-

a72 QMAKE_CXXFLAGS   = $$QMAKE_CFLAGS

EGLFS_DEVICE_INTEGRATION = eglfs_kms

include(../common/linux_arm_device_post.conf)

load(qt_config)

Add below content in $nano qplatformdefs.h

#include "../../linux-g++/qplatformdefs.h"

Symlinks for compiler

$ ln -s /usr/bin/aarch64-linux-gnu-g++-9 /usr/bin/aarch64-linux-gnu-g++

$ ln -s /usr/bin/aarch64-linux-gnu-gcc-9 /usr/bin/aarch64-linux-gnu-gcc

7 Create and configure a sysroot [Host]

```
$ mkdir sysroot sysroot/usr sysroot/opt
$ rsync -avz pi@192.168.16.25:/lib sysroot
$ rsync -avz pi@192.168.16.25:/usr/include sysroot/usr
```

```
$ rsync -avz pi@192.168.16.25:/usr/lib sysroot/usr
$ rsync -avz pi@192.168.16.25:/usr/lib/ sysroot/opt

$ ls sysroot/usr/lib/aarch64-linux-gnu/libEGL.so.1.1.0

$ mv sysroot/usr/lib/aarch64-linux-gnu/libEGL.so.1.1.0
sysroot/usr/lib/aarch64- linux-gnu/libEGL.so.1.1.0_backup

$ ln -s sysroot/usr/lib/aarch64-linux-gnu/libEGL.so sysroot/usr/lib/aarch64-linux-
gnu/libEGL.so.1.1.0

 $ mv sysroot/usr/lib/aarch64-linux-gnu/libGLESv2.so.2.1.0
sysroot/usr/lib/aarch64-linux-gnu/libGLESv2.so.2.1.0_backup
  $ ln -s sysroot/usr/lib/aarch64-linux-gnu/libGLESv2.so
sysroot/usr/lib/aarch64- linux-gnu/libGLESv2.so.2.1.0

$ ln -s sysroot/usr/lib/aarch64-linux-gnu/libEGL.so sysroot/usr/lib/aarch64-linux-
gnu/libEGL.so.1
$ ln -s sysroot/usr/lib/aarch64-linux-gnu/libEGLSv2.so
sysroot/usr/lib/aarch64- linux-gnu/libEGLSv2.so.2
```

Next, we need to adjust our symbolic links in sysroot to be relative since this folder structure is in both our computer and Raspberry Pi.

```
 $ wget https://raw.githubusercontent.com/riscv/riscv-poky/master/scripts/
sysroot-relativelinks.py
 $ chmod +x sysroot-relativelinks.py
 $ ./sysroot-relativelinks.py sysroot
$ rsync -avz pi@192.168.16.25:/lib sysroot
$ rsync -avz pi@192.168.16.25:/usr/include sysroot/usr
$ rsync -avz pi@192.168.16.25:/usr/lib sysroot/usr
$ rsync -avz pi@192.168.16.25:/usr/lib sysroot/opt
$ ./sysroot-relativelinks.py sysroot
```

8 Configure Qt for cross compilation –[Host]

```
$ mkdir qt5build
$ cd qt5build/

 $../qt-everywhere-src-5.15.2/configure -opengl es2 -eglfs -device linux-rasp-pi4b-v3d-g++ -device-option CROSS_COMPILE=/usr/bin/aarch64-linux-gnu- -sysroot /opt/qt5pi/sysroot -prefix /usr/local/qt5pi -opensource -confirm-license -skip qtscript -skip qtwayland -skip qtdatavis3d -nomake examples -make libs -pkg-config -no-use-gold-linker -v

After a few minutes, the script should be finished and the following conditions should be set or not set.

QPA backends:
```

```
DirectFBno

EGLFS ..... yes  [

EGLFS details:

EGLFS OpenWFDno

EGLFS i.Mx6no

EGLFS i.Mx6 Waylandno

EGLFS RCARno

EGLFS EGLDeviceyes

EGLFS GBMyes

EGLFS VSP2no

EGLFS Malino

EGLFS Raspberry Pino

EGLFS X11yes

LinuxFByes

VNCyes
```

The exact compilation time depends on your computer performance. This process can take up to 2 or 3 hours in nowadays common computers. Compiler processes can be executed in parallel by means of the j flag. For four make processes, execute:make -j14 depends on your machine config.

```
$ make -j8
$ make install
```

Once Qt is compiled, it can be deployed to your Raspberry Pi using the rsync command.

```
$ cd /opt/qt5pi/
$ rsync -avz --rsync-path=" rsync" sysroot/usr/local/qt5pi
pi@192.168.16.25:/usr/local
```

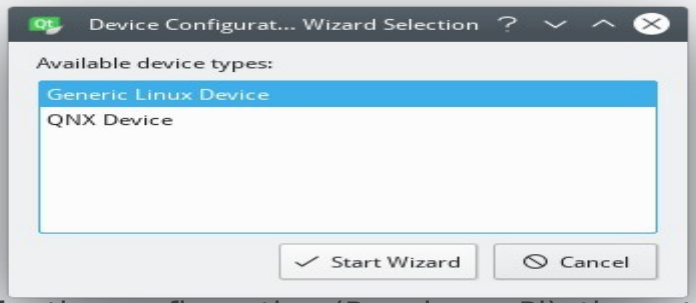
9 Setup Qt Creator for Raspberry Pi cross compilation

If you do not have installed Qt Creator, install it with cmd

```
$ sudo apt install qtcreator
```

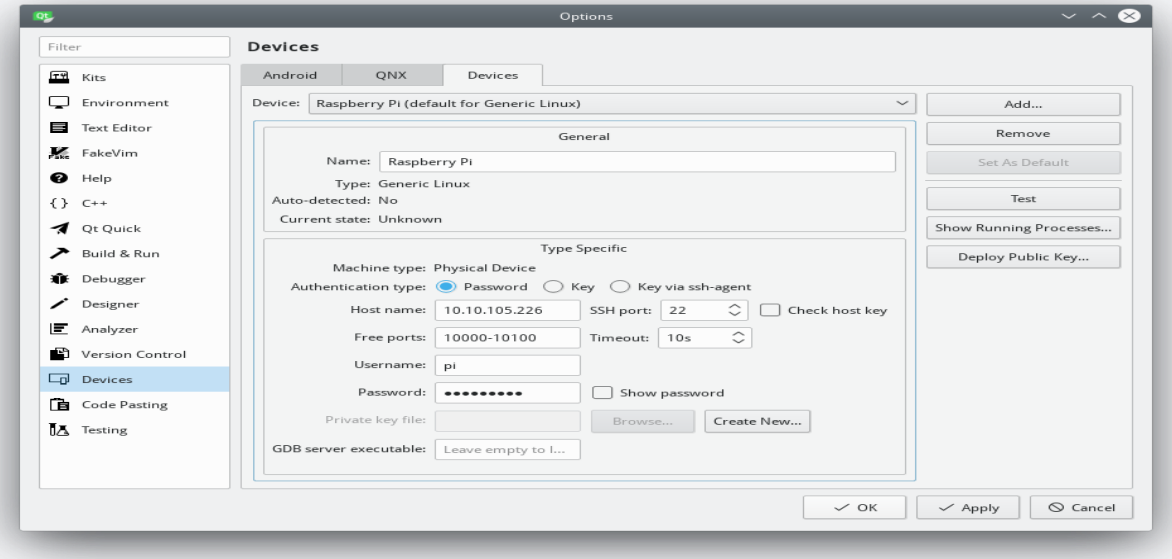
then, follow these instructions to configure cross-compilation support.

First, open Qt Creator, go to the Tools -> Options menu, select the Devices section and devices tab.Add a new Generic Linux Device.



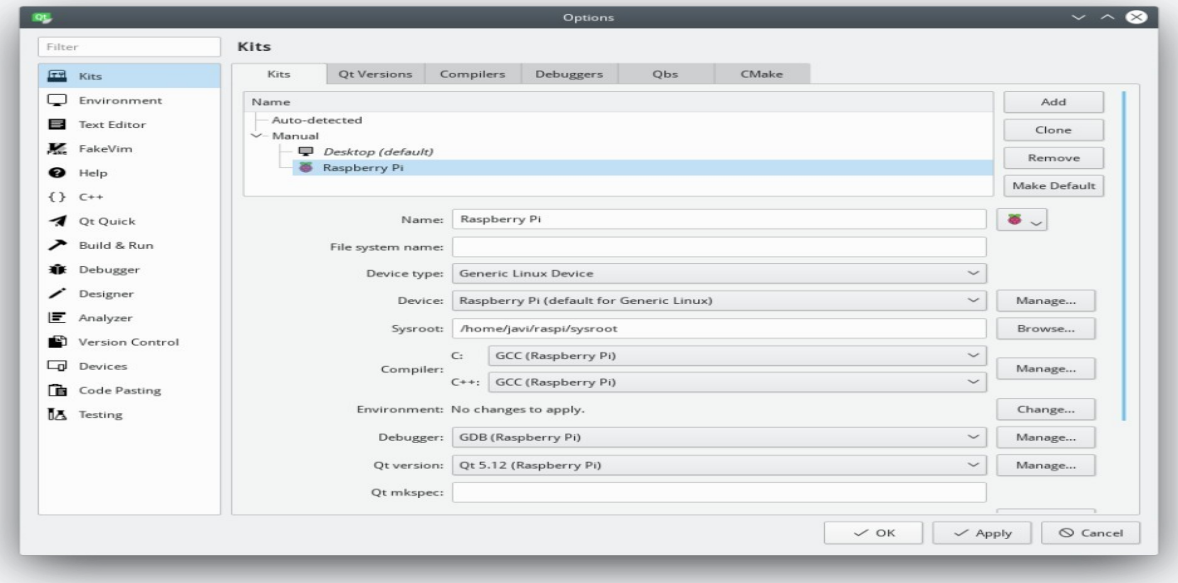
Set a name for the configuration (Raspberry Pi), the network name or IP, the username

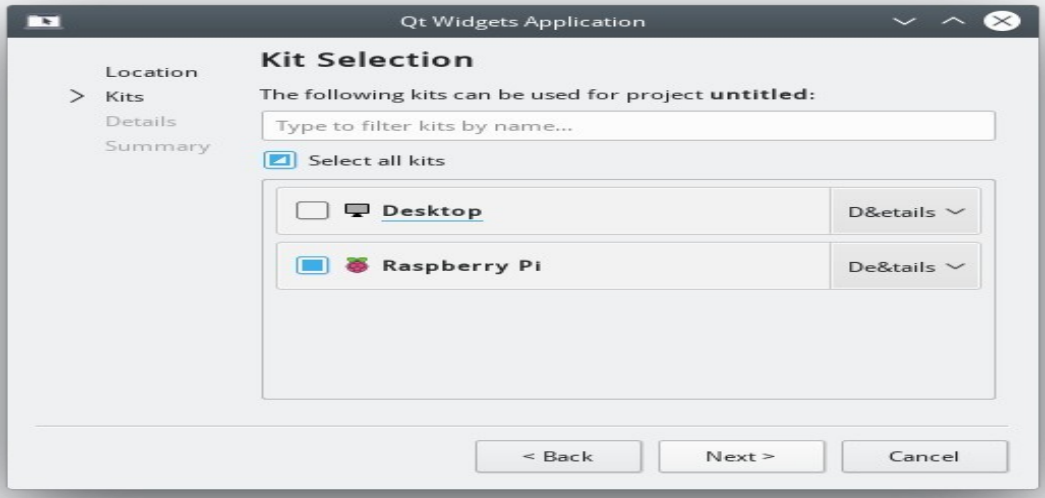
Before you finish the wizard, the connection to your Raspberry Pi will be verified and you will have then a new device configured in Qt Creator.



Next, go to the Kits section and Compilers tab, Add GCC C and C++ cross compilers for Raspberry Pi.
The path for the GCC C Raspberry Pi compiler is /usr/bin/aarch64-linux-gnu

Finally, go to the Kits tab, Add a new kit, set the name and icon you prefer, and pay attention to the following configuration





Congratulations! You can now design, build and deploy your Qt Raspberry Pi apps in you computer and, execute and debug them directly in your Raspberry Pi.