Create Static build of Qt for gLAB GUI for Linux ARM64

This is a small guide for creating a statically linked build of Qt in Linux ARM64 (aarch64) cross-compiled in a Linux x64. This will allow creating a statically linked build of your Qt based application.

What is a cross-compilation and what is the reason to cross-compile?

Cross compiling is to compile a binary in a host that has a different architecture than the target computer. In this case, the compilation is done on an x64 host and the target is a Linux ARM 64 bit (aarch64), such as Raspberry Pi or Odroid. Although ARM devices can compile Qt themselves, it is much faster to do it in an x64 machine (even though the host is a virtual machine).

How does a cross-compilation work?

To do a cross-compilation, the following steps have to be done:

1. Download Qt sources
2. Download a cross-compiler (a compiler that runs on your host machine and produces code for the target machine).
3. Create a folder where a copy of the target operative system is installed along with all the necessary libraries to compile (this folder will contain all the folder hierarchy of a Linux OS)
4. Cross-compile using the cross-compiler and the libraries from the target machine.

What cross-compiler are you using?

The cross-compiler used is Linaro ToolChain “aarch64-linux-gnu” version 7.3.1 (https://releases.linaro.org/components/toolchain/binaries/latest-7/). It is based on GCC, an according to the release notes, only version 7 or later must be used, as GCC 5 and 6 has a bug that may produce incorrect code on ARM.

What is the most problematic part of the cross-compilation?

The most problematic part is that the cross-compiler has difficulties finding the target libraries. Therefore, it is necessary to manually provide the cross-compiler most of the folders where the libraries are saved (see step 21). If any path library wants to be added, both Qt and the application have to be rebuilt (repeat steps from step 21).

What is the difference between dynamic and static linking?

With dynamic linking, the necessary libraries to run our program are in external files, therefore you need to provide all the libraries along with your executable.

With static linking, the necessary libraries to run our program are all embedded inside the executable, therefore, our executable is a stand-alone program.

Does this guide also work for building other Qt programs (no gLAB)?

Yes, but you need to make sure that in step 10 all the necessary libraries are provided for your application. gLAB GUI uses only basic libraries of Qt.

What Qt version are you using for this guide?

In this guide, there are steps for compiling with Qt version 5.7.1 or the last Qt version available at git repository, although any Qt version may work.

Does this guide work with other Qt versions?

Yes, but you will need to use newer versions of Ubuntu (or other Linux) for higher Qt versions (5.6 or greater). Libraries needed in step 10 may change its name or additional libraries may be needed. Also, parameters for step 5 may change

What Linux version are you using for this guide?

There are two Linux versions in this guide. In this case, the host machine is an Ubuntu 14.04. The Linux version of the host machine is not important as long as it supports the Qt version you are using to compile.

The target host is an Ubuntu 14.04 for ARM64 (aarch64). The target Ubuntu version must be the minimum Linux version to be supported for your target device, as Linux is forward compatible but not backwards compatible (for instance, compiling in Ubuntu 14.04 will make your application on this version and newer ones, but not in older ones).

It is recommend following these guide using an Ubuntu virtual machine with 4 CPUS, 4GB of RAM and 30GB of hard drive.

Steps for a static build of Qt in Linux:

1) Open a terminal and become root user:

   sudo su -

2) Install programs in the host machine

   apt-get install build-essential perl python git sed xz-utils

3) Install tools to create a fake system filesystem

   apt-get install qemu-user-static debootstrap

4) The folder where the target filesystem will be mounted will be “/rootfsarm/root”, but it can be mounted anywhere else.

   ```
   export ROOTFS=/rootfsarm/root
   mkdir –p $ROOTFS #directory which will become /
   cd $ROOTFS
   ```

5) Create a new arm64 system of trusty (Ubuntu 14.04). This command will install the minimum number of required packages in order to boot the target system: Everything else (libraries needed for Qt) will be installed through apt command

   If user wants to use another Linux version, change the distribution name “trusty” to the desired one and the source URL (if it is not a Ubuntu distribution).

   qemu-debootstrap --arch=arm64 --variant=minbase trusty .
http://ports.ubuntu.com/ubuntu-ports

6) Add package sources to sources.list file of the arm64 distribution (if the user Linux distribution is different, change the distribution name and URL).

   echo "deb http://ports.ubuntu.com/ubuntu-ports/ trusty main restricted
deb-src http://ports.ubuntu.com/ubuntu-ports/ trusty main restricted

deb http://ports.ubuntu.com/ubuntu-ports/ trusty-updates main restricted
deb-src http://ports.ubuntu.com/ubuntu-ports/ trusty-updates main restricted

deb http://ports.ubuntu.com/ubuntu-ports/ trusty universe
deb-src http://ports.ubuntu.com/ubuntu-ports/ trusty universe
deb http://ports.ubuntu.com/ubuntu-ports/ trusty-updates universe
deb-src http://ports.ubuntu.com/ubuntu-ports/ trusty-updates universe

deb http://ports.ubuntu.com/ubuntu-ports/ trusty multiverse
deb-src http://ports.ubuntu.com/ubuntu-ports/ trusty multiverse
deb http://ports.ubuntu.com/ubuntu-ports/ trusty-updates multiverse
deb-src http://ports.ubuntu.com/ubuntu-ports/ trusty-updates multiverse

deb http://ports.ubuntu.com/ubuntu-ports/ trusty-backports main restricted universe multiverse
deb-src http://ports.ubuntu.com/ubuntu-ports/ trusty-backports main restricted universe multiverse

deb http://archive.canonical.com/ubuntu trusty partner
deb-src http://archive.canonical.com/ubuntu trusty partner

deb http://ports.ubuntu.com/ubuntu-ports/ trusty-security main restricted
deb-src http://ports.ubuntu.com/ubuntu-ports/ trusty-security main restricted
deb http://ports.ubuntu.com/ubuntu-ports/ trusty-security universe
deb-src http://ports.ubuntu.com/ubuntu-ports/ trusty-security universe
deb http://ports.ubuntu.com/ubuntu-ports/ trusty-security multiverse
deb-src http://ports.ubuntu.com/ubuntu-ports/ trusty-security multiverse" > ${ROOTFS}/etc/apt/sources.list

7) Change the root directory, which will make to effectively work in the target filesystem.

    chroot .

8) Check the architecture of your distribution is arm (the output of the command should be aarch64)

    uname -m

9) Update repositories in the target filesytem

    apt-get update
    apt-get upgrade

10) Install libraries on target machine

    apt-get install libstdc++-4.8-dev libc6-dev libssl-dev perl
    apt-get install build-essential linux-headers-$(uname -r) python
    apt-get install libgl1-mesa-dev libfontconfig1-dev libfreetype6-dev libx11-dev libxext-dev libxfixes-dev libxi-dev libxrender-dev libxcb1-dev libxcb1 libx11-xcb-dev libx11-xcb1 libxcb-glx0-dev libxcb-keysyms1-dev libxcb-image0-dev libxcb-shm0-dev libxcb-icccm4-dev libxcb-sync-dev libxcb-xfixes0-dev libxcb-shape0-dev libxcb-randr0-dev libxcb-render-util0-dev libjasper-dev libmng-dev libjpeg-dev libpng12-dev libtiff5-dev libgif-dev libxcb-xinerama0-dev libxcb-xinerama0-dev '^libxcb.*-dev' '^libxcb.*' libx11-xcb-dev libglu1-mesa-dev libxrender-dev libxi-dev libxrender-dev libxi-dev
    apt-get install libxcb1 libxcb-util0 libpam-dev libcairo-dev libxcb-xinerama0 libev-dev libx11-dev libx11-xcb-dev libxkbcommon0 libxkbcommon-x11-dev libxkbcommon-dev libxcb-dpms0-dev libxcb-xinerama0-dev libxkbfile-dev libxcb-util0-dev libxcb-image0-dev fontconfig libfreetype6 libfreetype6-dev

11) Exit chroot (return to host filesystem)

    exit

12) Exit root user (back to normal user in host machine)

```
exit
```

13) Create files directory (In this case “/files” will be used, which will be a folder accessible by any user)

```
sudo mkdir /files
sudo chown $USER:$USER /files
```

14) Change to “/files” directory

```
cd /files
```

15) Download cross compiler (Linaro version aarch64-linux-gnu 7.3.1) and uncompress it

```
mkdir linaro
cd linaro
wget https://releases.linaro.org/components/toolchain/binaries/latest-7/aarch64-linux-gnu/gcc-linaro-7.3.1-2018.05-x86_64_aarch64-linux-gnu.tar.xz
tar -xvJf gcc-linaro-7.3.1-2018.05-x86_64_aarch64-linux-gnu.tar.xz
```

16) Install Qt. There are two options:

A) Install last Qt version from git

a. Change to “/files” directory

```
cd /files
```

b. Clone Qt git repository and initialize it

```
git clone git://code.qt.io/qt/qt5.git
cd qt5
./init-repository #This will take some minutes
```

c. Install Qt creator

```
sudo apt-get install qtcreator
```

B) Download Qt installator and install Qt. In this case Qt 5.7.1 is used. Make sure that Qt is installed in “/files/qt5” folder and that during installation “Sources” and “QtCreator” options is selected.

a. Change to “/files” directory

```
cd /files
```

b. Download Qt5.71 installator

wget http://mirrors.ukfast.co.uk/sites/qt.io/archive/qt/5.7/5.7.1/qt-opensource-linux-x64-5.7.1.run

c. Add executable permissions to Qt installator

chmod +x qt-opensource-linux-x64-5.7.1.run

d. Install Qt

./qt-opensource-linux-x64-5.7.1.run

17) Become root again

sudo su -

18) Add environment variables to current session and to .bashrc file

```
export PATH=$PATH:/files/linaro/gcc-linaro-7.3.1-2018.05-x86_64_aarch64-linux-gnu/bin/
ROOTFS=/rootfsarm/root
TRIPLET=$(aarch64-linux-gnu-g++ -dumpmachine) #becomes aarch64-linux-gnu
export PKG_CONFIG_LIBDIR=${ROOTFS}/usr/lib/pkgconfig:${ROOTFS}/usr/share/pkgconfig:${ROOTFS}/usr/lib/aarch64-linux-gnu/pkgconfig:${ROOTFS}/usr/lib/aarch64-linux-gnu:${ROOTFS}/lib/aarch64-linux-gnu
echo “export PATH=$PATH:/files/linaro/gcc-linaro-7.3.1-2018.05-x86_64_aarch64-linux-gnu/bin/”>> /root/.bashrc
echo "ROOTFS=/rootfsarm/root" >> /root/.bashrc
echo “TRIPLET=$(aarch64-linux-gnu-g++ -dumpmachine)” >> /root/.bashrc
echo “export PKG_CONFIG_LIBDIR=${ROOTFS}/usr/lib/pkgconfig:${ROOTFS}/usr/share/pkgconfig:${ROOTFS}/usr/lib/aarch64-linux-gnu/pkgconfig:${ROOTFS}/usr/lib/aarch64-linux-gnu:${ROOTFS}/lib/aarch64-linux-gnu” >> /root/.bashrc
```

19) Soft-links in the target filesystem “/usr/lib” (in the host directory $ROOTFS/usr/lib) must have relative soft links, otherwise the soft link will point to the host rather than the target filesystem (these commands have to be executed again if user installs more libraries in the target filesystem after applying these commands).

```
cd /files
git clone https://github.com/rm5248/cross-compile-tools.git
./cross-compile-tools/fixQualifiedLibraryPaths ${ROOTFS} /files/linaro/gcc-linaro-7.3.1-2018.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu-g++
cd $ROOTFS/usr/lib/aarch64-linux-gnu
ls -l |grep  ^l|awk '$NF~/^\// {print "rm "$(NF-2)" && ln -s ../../.."$NF,$(NF-2) }'|bash
```

20) This step is only for Ubuntu 14.04: In Ubuntu 14.04, the library "$ROOTFS/usr/include/aarch64-linux-gnu/bits/mathcalls.h" has a bug which makes the compilation fail. The steps to fix the library (add 4 lines of codes) is provided in https://patches.linaro.org/patch/59550/, but also the user can just substitute the library by the file embedded in this pdf

mathcalls.zip

21) Create Qt mkspec for ARM. It is critical in this phase to add the paths of the libraries to be used by the linker, as if they are not manually provided, probably the linker will not find them. These paths are provided using the Qt variables "QMAKE_LFLAGS" and "LIBS". For Qt builds that use more Qt libraries than gLAB, maybe more paths must be added.

```
cd /files/qt5
cp -r qtbase/mkspecs/linux-arm-gnueabi-g++ qtbase/mkspecs/aarch64-linux-gnu-g++
sed -i 's/arm-linux-gnueabi/aarch64-linux-gnu/g' qtbase/mkspecs/aarch64-linux-gnu-g++/qmake.conf
echo "QMAKE_LFLAGS        += -Wl,-rpath-link,${ROOTFS}/lib/aarch64-linux-gnu
QMAKE_LFLAGS        += -Wl,-rpath-link,${ROOTFS}/usr/lib/aarch64-linux-gnu" >> qtbase/mkspecs/aarch64-linux-gnu-g++/qmake.conf
echo "LIBS += -L/rootfsarm/root/usr/lib/aarch64-linux-gnu/qt5/plugins/platforms
LIBS += -L${ROOTFS}/usr/lib/aarch64-linux-gnu/qt5/plugins/imageformats
LIBS += -L${ROOTFS}/usr/lib/aarch64-linux-gnu/qt5/plugins/bearer
LIBS += -L${ROOTFS}/usr/lib/aarch64-linux-gnu" >> qtbase/mkspecs/aarch64-linux-gnu-g++/qmake.conf
```

22) Create the Qt Makefile using the "configure" command. It is recommended to disable all Qt libraries not used in your application, as Qt often links to all the system libraries required by the Qt modules for all Qt modules available even though the application does not use them (for instance, "libudev" or "libts"). The system library usually cannot be static linked, therefore the more system libraries linked in your application, the more libraries the target system has to be installed, whilst the objective is to have the minimum libraries to be installed in the target system.
The target Qt5 installation directory is "/usr/share/qt5" (defined in parameter "-hostprefix").

1. If Qt from git is installed

```
./configure -static -opensource -release -confirm-license -no-compile-examples -nomake tests -qt-zlib -qt-libpng -qt-libjpeg -qt-freetype -qt-pcre -qt-harfbuzz -prefix /usr -bindir /usr/lib/$TRIPLET/qt5/bin -libdir /usr/lib/$TRIPLET -headerdir /usr/include/$TRIPLET/qt5 -datadir /usr/share/qt5 -archdatadir /usr/lib/$TRIPLET/qt5 -plugindir /usr/lib/$TRIPLET/qt5/plugins -importdir /usr/lib/$TRIPLET/qt5/imports -translationdir /usr/share/qt5/translations -hostdatadir /usr/share/qt5 -xplatform aarch64-linux-gnu-g++ -platform linux-g++ -sysroot $ROOTFS -hostprefix /usr/share/qt5 -no-opengl -no-assimp -skip qtdeclarative -skip qtlocation -skip qtmultimedia -skip qtquickcontrols -skip qtsensors  -skip qtwebchannel -skip qtwebengine -no-qt3d-simd -no-sql-sqlite -skip wayland -skip xmlpatterns
```

2. If Qt 5.7.1 is installed
   ./configure -static -opensource -release -confirm-license -no-compile-examples -nomake tests -qt-zlib -qt-libpng -qt-libjpeg -qt-freetype -qt-pcre -qt-harfbuzz -largefile -qt-xcb -qt-xkbcommon -prefix /usr -bindir /usr/lib/$TRIPLET/qt5/bin -libdir /usr/lib/$TRIPLET -headerdir /usr/include/$TRIPLET/qt5 -datadir /usr/share/qt5 -archdatadir /usr/lib/$TRIPLET/qt5 -plugindir /usr/lib/$TRIPLET/qt5/plugins -importdir /usr/lib/$TRIPLET/qt5/imports -translationdir /usr/share/qt5/translations -hostdatadir /usr/share/qt5 -xplatform aarch64-linux-gnu-g++ -platform linux-g++ -sysroot $ROOTFS -hostprefix /usr/share/qt5 -no-opengl -no-qml-debug -no-sql-sqlite -skip qtdeclarative -skip qtlocation -skip qtmultimedia -skip qtquickcontrols -skip qtsensors -skip qtwebchannel -skip qtwebengine -skip wayland -skip xmlpatterns

NOTES:

- If we want to be able to have a debug version of the static build, we need to add the parameter "-debug".

23) Compile Qt. With parameter "-j N" you can set the number of cores to used (N is the number of cores). Depending on the processing power and the number of cores, this process may take several hours to complete.

    1. If Qt from git is installed

       make -j 4

    2. If Qt 5.7.1 is installed. With Qt 5.7.1, there are some errors building QML libraries, which gLAB does not need, therefore, these errors can be skipped.

       make -j 4 --ignore-errors

24) Install the new version of Qt in the folder specified in the "-prefix" parameter in step 5.

    1. If Qt from git is installed

       make install -j 4

    2. If Qt 5.7.1 is installed. With Qt 5.7.1, there are some errors building QML libraries, which gLAB does not need, therefore, these errors can be skipped.

       make install -j 4 --ignore-errors

NOTE: If user wants to rerun the "configure" command and build again, it is mandatory to clean the data folder from previous builds. This can be done with the following commands:

a) If Qt from git is installed

    1. git submodule foreach --recursive "git clean -dfx" && git clean –dfx

    2. Redo step 21

b) If Qt 5.7.1 is installed. With Qt 5.7.1, there are some errors building QML libraries, which gLAB does not need, therefore, these errors can be skipped.

```
make clean -j 4
make distclean -j 4
```

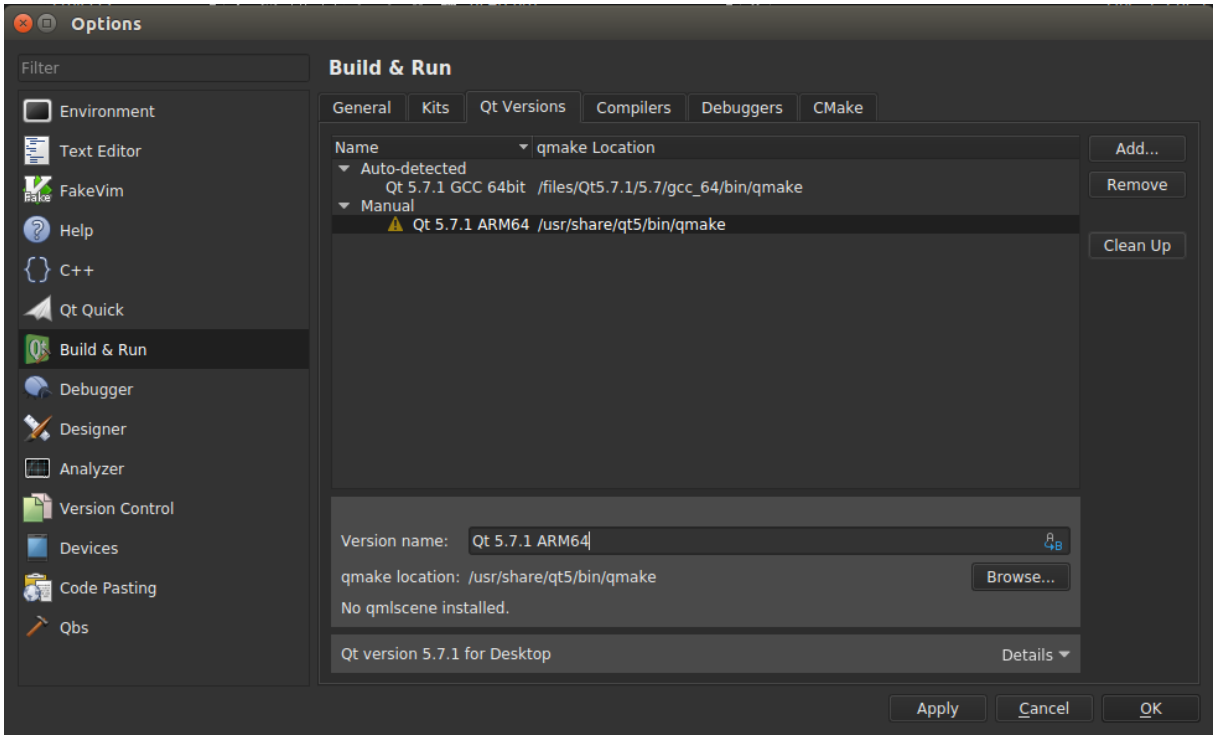
25) Exit root user (back to normal user in host machine)

```
exit
```

26) Open Qt creator (with normal user)

27) In Qt creator, the new Qt Static version has to be added:

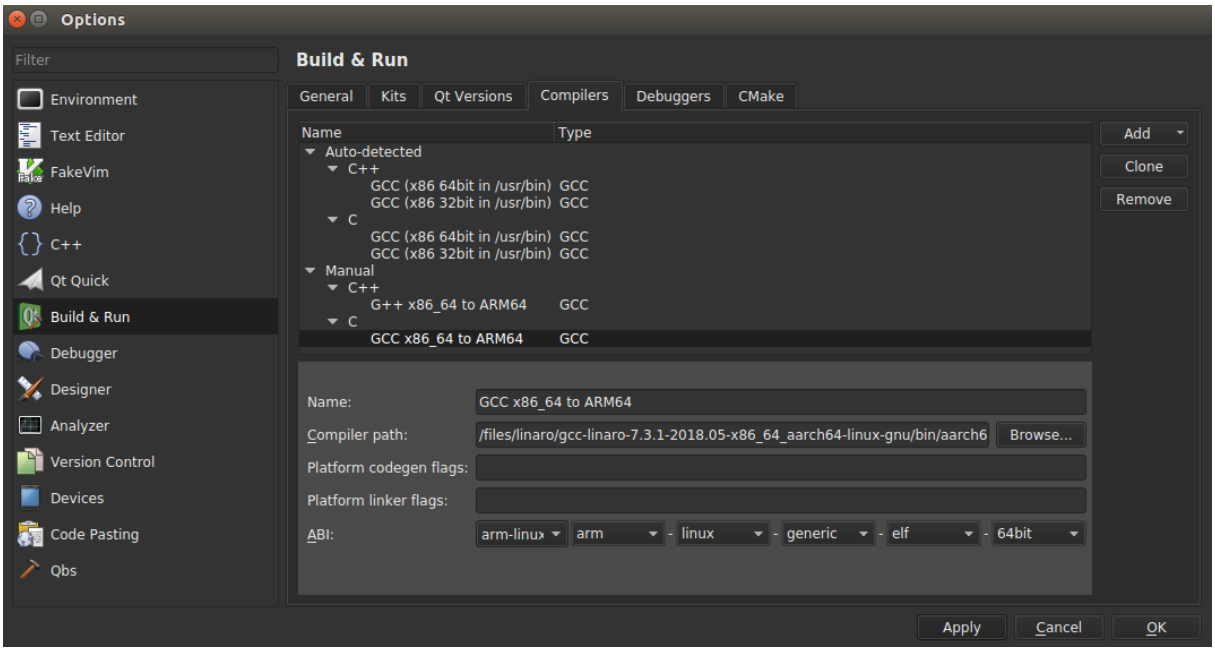
- Go to Tools→Options.
- In the left pane, select “Build & Run”.
- Select the “Qt Versions” tab.
- Click in the “Add” button, and search for the “qmake” file, which will be in the “bin” folder where the static Qt was installed. In this case, it should be in ““/usr/share/qt5/bin" folder.
- Set a name for this version (any name is valid). For instance, “Qt 5.7.1 ARM64”
- Click in “Apply” in the bottom of the window.
- The result should look like this:



Ignore the warnings.

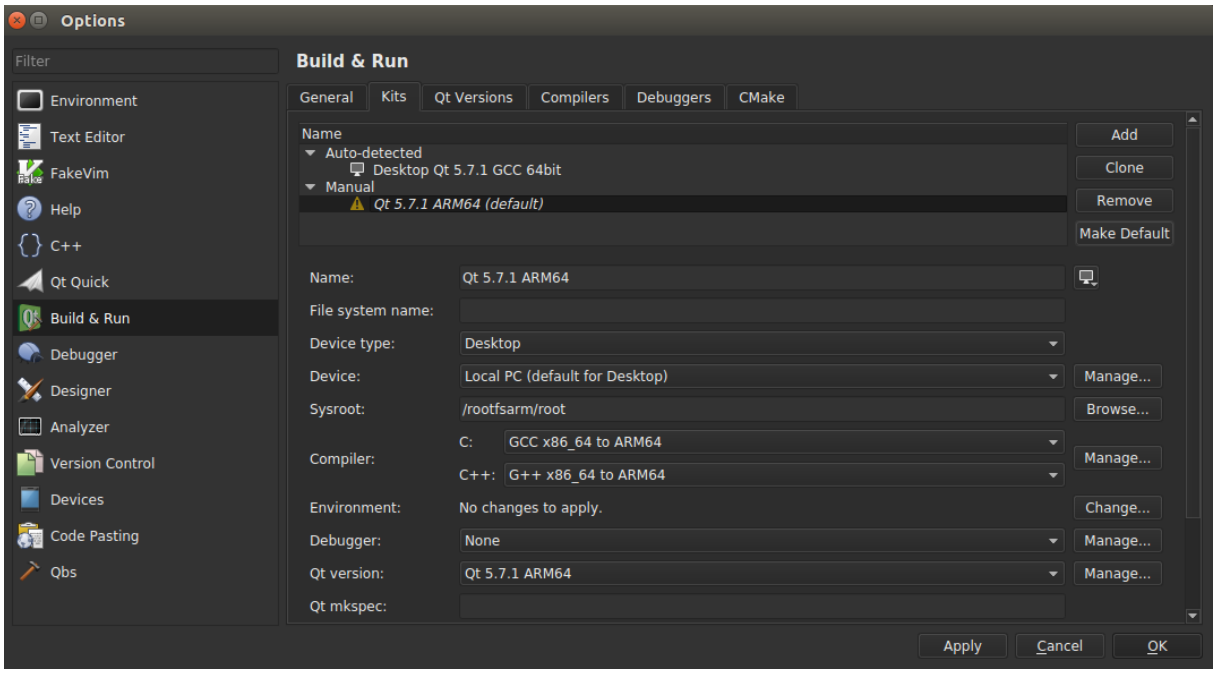
28) Add the cross-compilers:

- Click in the “Compilers” tab.
- Click in the “Add->GCC->C” buttons
- In the compiler path below, click in “browse” and select the file “/files/linaro/gcc-linaro-7.3.1-2018.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu-gcc”
- Set the name for the compiler as “GCC x86_64 to ARM64”
- Click in the “Add->GCC->C++” buttons
- In the compiler path below, click in “browse” and select the file “/files/linaro/gcc-linaro-7.3.1-2018.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu-g++”
- Set the name for the compiler as “G++ x86_64 to ARM64”
- Click in “Apply” in the bottom of the window.
- The result should look like this:



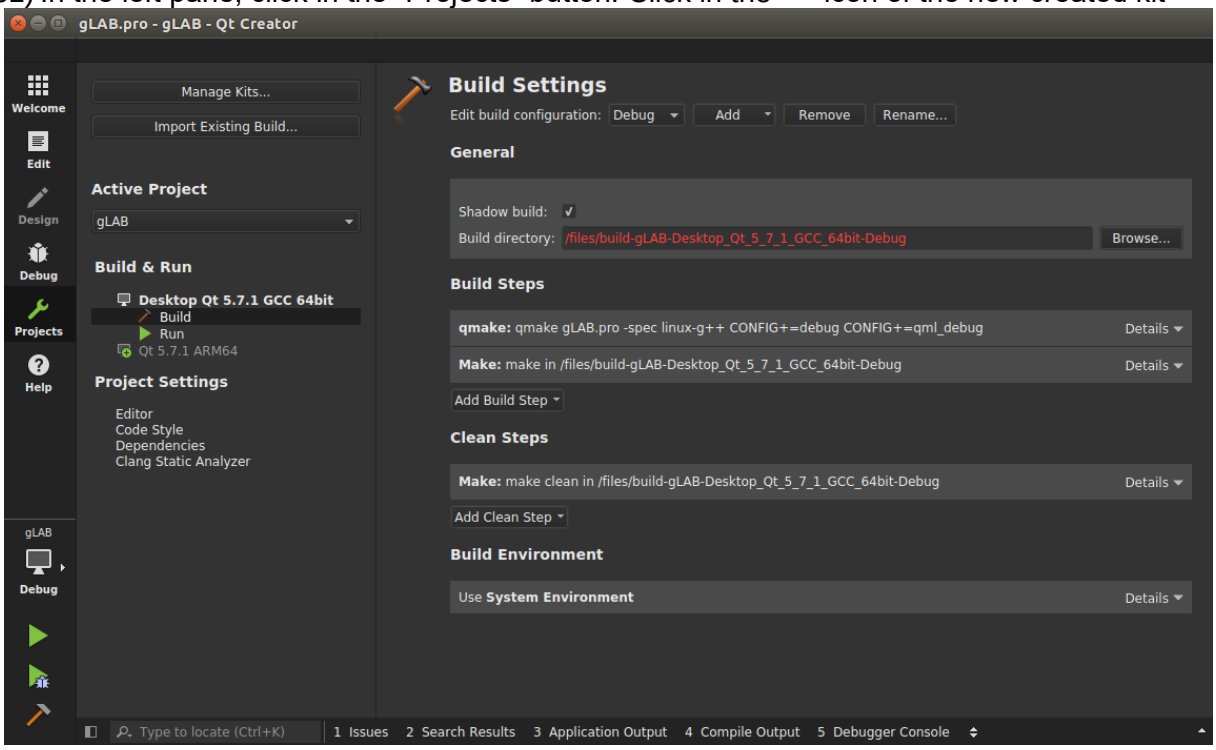
29) Add a new build kit with the static Qt ARM version and the cross compiler:

- Click in the “Kits” tab.
- Click in the “Add” button
- Set a name for the kit (any name). For instance “Qt 5.7.1 Static ARM64”
- In the compiler section, set “GCC x86_64 to ARM64” for C and “G++ x86_64 to ARM64” for C++
- In the Qt version, select the one created in step 27. For instance, “Qt 5.7.1 ARM64”
- In the “Sysroot”, and the path “/rootfsarm/root” (the directory were the target filesystem was installed).
- The result should look like this:

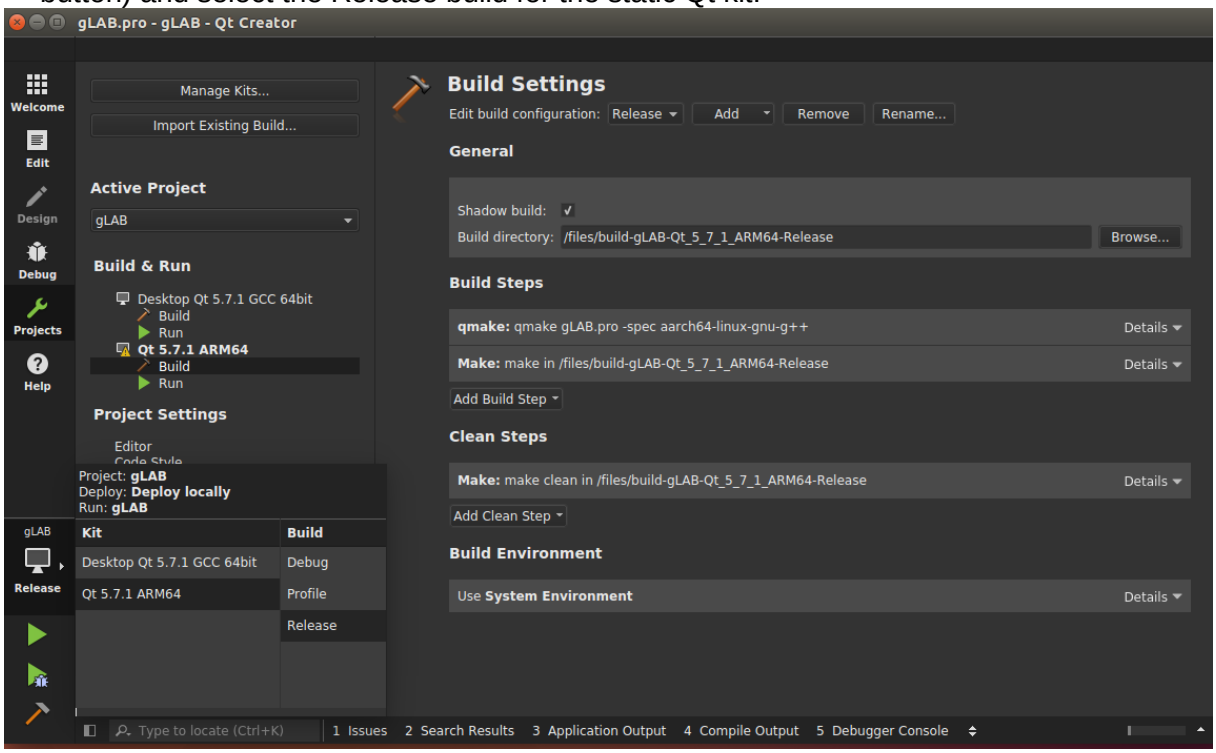


30) Click in “OK” in the bottom part of the window.

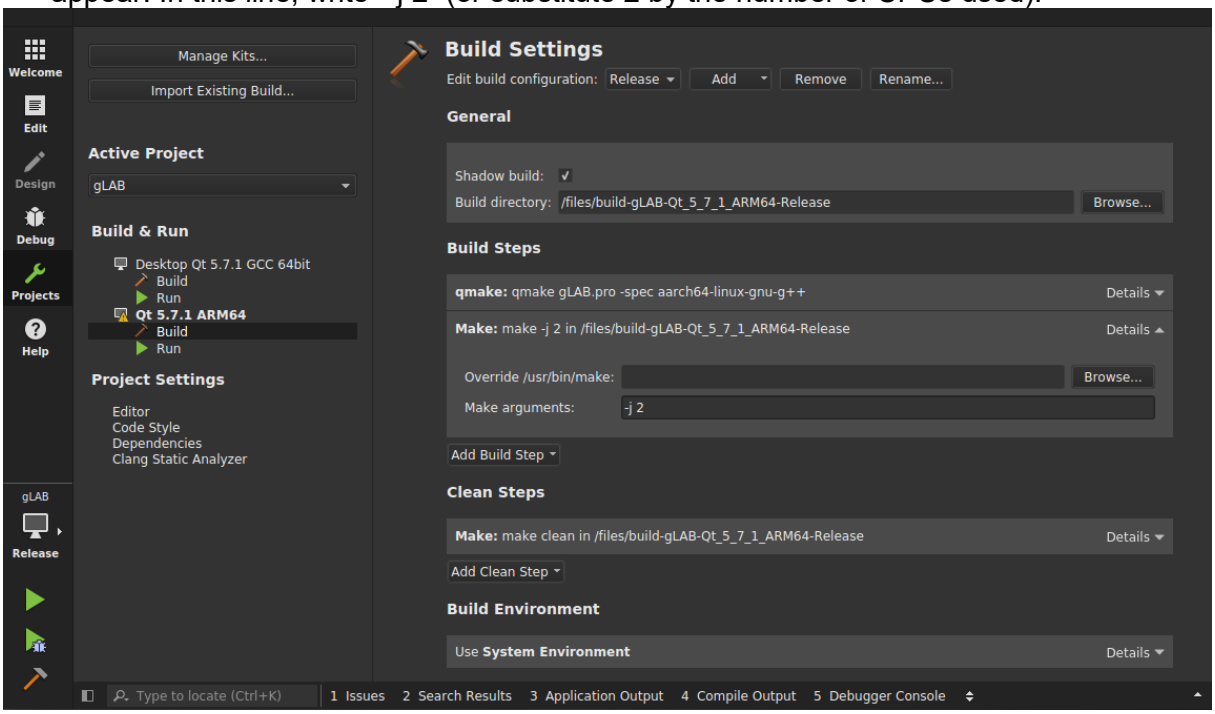
31) In the left pane, click in the “Projects” button. Click in the “+” icon of the new created kit



32) The new kit is ready to build, but we need to make sure we select the Release build, as the Debug build will not be available (unless the “-debug” parameter was provided in step 22. To select the Release build, click in the button with the monitor icon (over the play button) and select the Release build for the static Qt kit:



33) Optional: To compile with more than one CPU, click in the “Projects” button, select the “Build” line of Static ARM Qt Kit, then in the right side, below the “Build Steps” section, click in the “Details” button of the “Make” line. The line with “Make arguments” will appear. In this line, write “-j 2” (or substitute 2 by the number of CPUs used).



34) Optional: When compiling with a target machine of Ubuntu 14 to 16, the system library “libjasper” is linked, but it is not available any more in Ubuntu 18. To force this library to be statically linked, the Makefile created by Qt Creator has to be modified. A new Makefile will be created to avoid Qt Creator overwriting the modified file.

1. Open a terminal (with the default user)

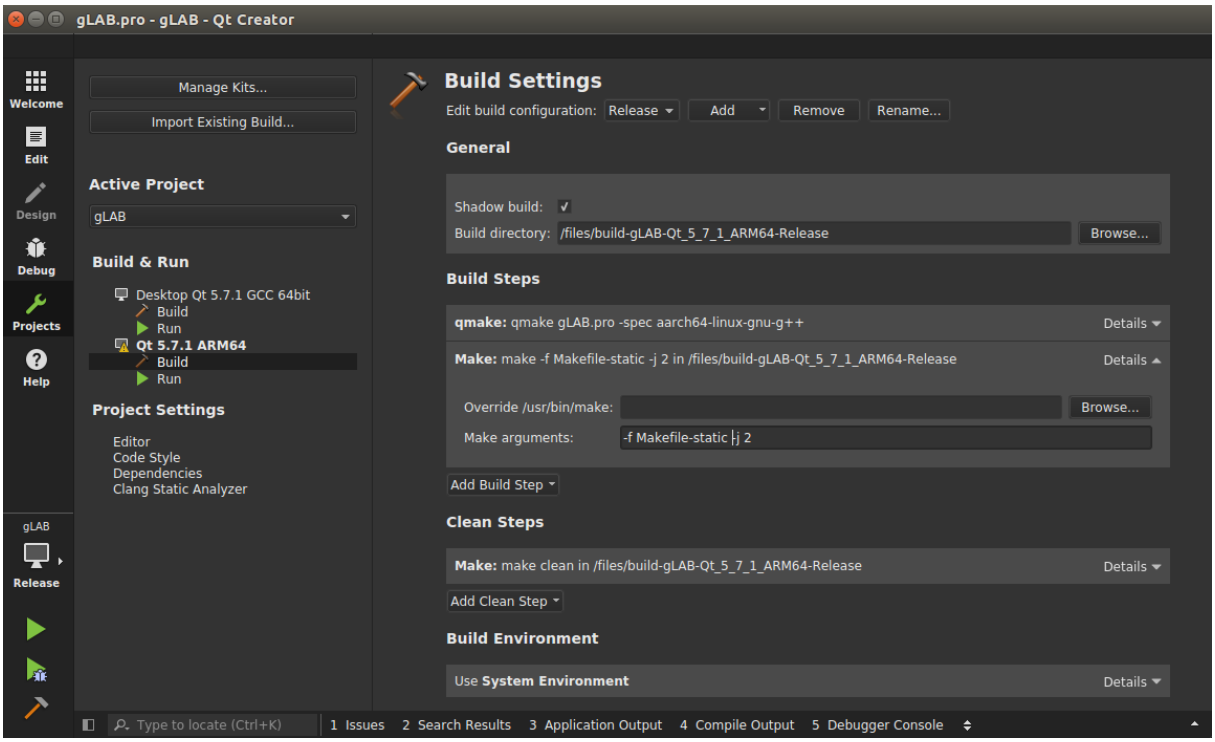
2. Change to the build directory created by Qt Creator (it is shown in the “Build directory” in the “Build Settings”, as in images on steps 31,32 and 33). In this example it is “/files/build-gLAB-Qt_5_7_1_Static_ARM64-Release”

   cd /files/build-gLAB-Qt_5_7_1_Static_ARM64-Release

3. Add the flag for libjasper for forcing to be statically built. To do this, the flag “-ljasper” has to be replaced for “-Wl,-Bstatic -ljasper -Wl,-Bdynamic” with the following command:
   sed -e 's/-ljasper/-Wl,-Bstatic -ljasper -Wl,-Bdynamic/' Makefile > Makefile-static

4. As in step 33, in the “Make arguments”, add at the beginning (prior to the “-j” parameter, if provided) the text “-f Makefile-static”

5. The result should look like this:



35) Compile the application with the “Play” button.

36) To test the that the executable does not have any Qt dependency, go to the folder where the binary is created (in our case, as we can see in the screenshot above, the binary will be in the path “/files/build-gLAB-Qt_5_7_1_Static_ARM64-Release”), execute the following command:

```
cd /files/build-gLAB-Qt_5_7_1_Static_ARM64-Release
ldd gLAB_GUI | grep –i qt
```

No output should appear (that is, no Qt library dependencies)

End of Document