

Cross-Compiling Qt 6.8 for Raspberry Pi 64-bit (Bookworm)

A step-by-step guide: build the ARM64 GCC toolchain, fix QtWebEngine, and compile every Qt module for host (x86_64) and target (arm64).

Qt 6.8.3 | GCC 12.4.0 / glibc 2.36 | Raspberry Pi OS bookworm 64-bit

Tested on Raspberry Pi OS bookworm (64-bit) · by Ed Chapman

Note: This guide is for **Linux** users. If you have a question, and it is about how to do things on a Windows machine, I will not answer you — not because I am rude, but because I don't know a thing about Windows. I am old, and have not the number of brain cells left to move out of the Unix/Linux world.

Contact: You can contact me — if you are a Linux user — on my webpage www.vremsoft.com.

1. Overview

This guide walks through building a complete Qt 6.8 (including QtWebEngine) cross-compilation environment for a Raspberry Pi 4 or 5 running 64-bit Raspberry Pi OS **bookworm** (Debian 12). All compilation happens on an x86_64 Linux host; only the finished binaries are copied to the Pi.

At the end you will have:

- `/opt/cross-pi-gcc` — a self-built aarch64 cross-compiler (GCC 12.4.0, glibc 2.36).
- A **sysroot** copied from the Pi (glibc 2.36) used as the target file system.
- `/opt/Trolltech/Qt6/qt-host` — Qt 6.8.3 built for x86_64 (host tools + libs).
- `/opt/Trolltech/Qt6/qt-raspi` — Qt 6.8.3 built for arm64 (deployed to the Pi).
- QtWebEngine built and working for both host and target.

The golden rule

The cross-compiler's glibc **must be ≤ the Pi's glibc**. Bookworm ships glibc 2.36, so the toolchain links against glibc 2.36. If the toolchain were built against a newer glibc (for example the `gcc-aarch64-linux-gnu` package on Ubuntu 24.04, which bundles glibc 2.39), the resulting binaries would require `GLIBC_2.37`-and-later symbols and **fail to run** on the Pi.

Versions used

Component	Version
binutils	2.40
gcc	12.4.0

glibc	2.36
Qt	6.8.3 (Qt 6.8 LTS)
Pi OS	bookworm (Debian 12), 64-bit / aarch64
Host	x86_64 Linux (Ubuntu 24.04 / Pop!_OS)

2. Prerequisites

On the host

```
sudo apt update && sudo apt install -y \
    build-essential cmake ninja-build git rsync python3 python3-pip \
    bison flex gperf ruby perl python3-html5lib \
    libssl-dev libncurses-dev texinfo wget pkg-config
```

You also need SSH access to the Pi with passwordless **sudo** (so **rsync** can run as root on the remote side).

On the Raspberry Pi

Install a current 64-bit Raspberry Pi OS **bookworm** image, then update it:

```
sudo apt update && sudo apt upgrade && sudo reboot
```

The build dependencies for the Pi are installed in **Part 2** (they populate the sysroot with the headers and libraries the cross-build needs).

3. Part 1 — Build the GCC cross-compiler

This builds **binutils**, **gcc**, and **glibc** for target **aarch64-linux-gnu**, installed under **/opt/cross-pi-gcc**. The build is done in stages because the compiler and the C library each need the other:

1. Linux kernel headers
2. binutils
3. gcc (bootstrap — compiler only, no libc)
4. glibc (bootstrap headers + startup objects)
5. finish **libgcc**
6. finish **glibc**
7. finish **gcc** (full C/C++ + libstdc++)

3.1 Download and unpack the sources

```
mkdir -p ~/gcc_all && cd ~/gcc_all
wget https://ftpmirror.gnu.org/binutils/binutils-2.40.tar.gz
wget https://ftpmirror.gnu.org/glibc/glibc-2.36.tar.gz
wget https://ftpmirror.gnu.org/gcc/gcc-12.4.0/gcc-12.4.0.tar.gz
```

```
git clone --depth=1 https://github.com/raspberrypi/linux

tar xf binutils-2.40.tar.gz
tar xf glibc-2.36.tar.gz
tar xf gcc-12.4.0.tar.gz

cd gcc-12.4.0
./contrib/download_prerequisites # pulls gmp, mpfr, mpc, isl
```

3.2 Create the install prefix

```
sudo mkdir -p /opt/cross-pi-gcc
sudo chown $USER /opt/cross-pi-gcc
export PATH=/opt/cross-pi-gcc/bin:$PATH
```

3.3 Install the Linux kernel headers

```
cd ~/gcc_all/linux
make ARCH=arm64 INSTALL_HDR_PATH=/opt/cross-pi-gcc/aarch64-linux-gnu
headers_install
```

3.4 Build binutils

```
cd ~/gcc_all && mkdir build-binutils && cd build-binutils
../binutils-2.40/configure --prefix=/opt/cross-pi-gcc \
    --target=aarch64-linux-gnu --with-arch=armv8 --disable-multilib
make -j$(nproc)
make install
```

3.5 Build gcc (bootstrap — compiler only)

Optional (only if you later hit a `PATH\_MAX` error in libsanitizer): edit

`gcc-12.4.0/libsanitizer/asan/asan_linux.cpp` and add at the top:

```
#ifndef PATH_MAX
#define PATH_MAX 4096
#endif

cd ~/gcc_all && mkdir build-gcc && cd build-gcc
../gcc-12.4.0/configure --prefix=/opt/cross-pi-gcc \
    --target=aarch64-linux-gnu --enable-languages=c,c++ --disable-multilib
make -j$(nproc) all-gcc
make install-gcc
```

3.6 Build glibc (bootstrap headers + startup objects)

```
cd ~/gcc_all && mkdir build-glibc && cd build-glibc
../glibc-2.36/configure --prefix=/opt/cross-pi-gcc/aarch64-linux-gnu \
    --build=$MACHTYPE --host=aarch64-linux-gnu --target=aarch64-linux-gnu \
    --with-headers=/opt/cross-pi-gcc/aarch64-linux-gnu/include \
    --disable-multilib libc_cv_forced_unwind=yes
```

```
make install-bootstrap-headers=yes install-headers
make -j$(nproc) csu/subdir_lib
install csu/crt1.o csu/crti.o csu/crtn.o /opt/cross-pi-gcc/aarch64-linux-gnu/lib

aarch64-linux-gnu-gcc -nostdlib -nostartfiles -shared -x c /dev/null \
    -o /opt/cross-pi-gcc/aarch64-linux-gnu/lib/libc.so
touch /opt/cross-pi-gcc/aarch64-linux-gnu/include/gnu/stubs.h
```

3.7 Finish libgcc

```
cd ~/gcc_all/build-gcc
make -j$(nproc) all-target-libgcc
make install-target-libgcc
```

3.8 Finish glibc

```
cd ~/gcc_all/build-glibc
make -j$(nproc)
make install
```

3.9 Finish gcc (full C/C++ + libstdc++)

```
cd ~/gcc_all/build-gcc
make -j$(nproc)
make install
```

You can now delete `~/gcc_all` if you want the disk space back.

3.10 Verify the toolchain

```
/opt/cross-pi-gcc/bin/aarch64-linux-gnu-gcc --version
# expect: aarch64-linux-gnu-gcc (GCC) 12.4.0

/opt/cross-pi-gcc/bin/aarch64-linux-gnu-g++ --version
```

4. Part 2 — Prepare the sysroot

The sysroot is a copy of the Pi's `/usr/include`, `/usr/lib`, `/lib`, `/etc`, and `/usr/share` so the cross-compiler can link against the same libraries the Pi has.

4.1 Install build dependencies on the Pi

Run the following **on the Pi** (this populates the `-dev` headers/libs that will be copied back into the sysroot). It is idempotent — safe to re-run.

```
#!/usr/bin/env bash
set -euo pipefail
```

```

ARCH=$(dpkg --print-architecture)
[ "$ARCH" = arm64 ] || { echo "need arm64"; exit 1; }
[ "$(id -u)" = 0 ] || { echo "run as root"; exit 1; }

apt-get update

# (1) build toolchain
apt-get install -y \
    build-essential cmake ninja-build pkg-config \
    perl python3 python3-html5lib bison flex gperf ruby nodejs git curl wget

# (2) Qt 6 base (X11 / EGL / GL / core libs)
apt-get install -y \
    libfontconfig1-dev libfreetype6-dev \
    libx11-dev libx11-xcb-dev libxext-dev libxfixes-dev libxi-dev libxrender-
dev \
    libxcb1-dev 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 \
    libxcb-xinerama0-dev libxcb-xkb-dev \
    libxkbcommon-dev libxkbcommon-x11-dev \
    libgl1-mesa-dev libgles2-mesa-dev libegl1-mesa-dev \
    libdrm-dev libgbm-dev libwayland-dev wayland-protocols \
    libssl-dev zlib1g-dev libsqlite3-dev libdbus-1-dev libglib2.0-dev \
    libpcre2-dev libicu-dev libsystemd-dev libudev-dev \
    libinput-dev libmtdev-dev libasound2-dev libpulse-dev \
    libjpeg-dev libpng-dev libwebp-dev

# (3) QtWebEngine / Chromium build dependencies
apt-get install -y \
    libnss3-dev libnspr4-dev libevent-dev \
    libatk1.0-dev libgtk-3-dev \
    libcups2-dev libkrb5-dev libharfbuzz-dev \
    libopus-dev libvpx-dev libsnappy-dev libre2-dev libminizip-dev \
    liblcms2-dev libopenjp2-7-dev \
    libpci-dev libcap-dev \
    libxcomposite-dev libxcursor-dev libxdamage-dev libxrandr-dev \
    libxss-dev libxtst-dev libxkbfile-dev libxshmfence-dev \
    libxml2-dev libxslt1-dev

```

4.2 Pull the sysroot with rsync

Run on the **host**. The `--exclude` flags keep the sysroot lean — in particular `*.a` is excluded because Debian's static libraries are non-PIC and break Qt shared-object linking (a classic source of `relocation R_AARCH64_ADR_PREL_PG_HI21 ... recompile with -fPIC` errors).

```

mkdir -p sysroot

RSYNC='rsync -az --info=progress2 --rsync-path="sudo rsync" \
    --exclude=*.a --exclude=doc/ --exclude=man/ --exclude=info/ \
    --exclude=locale/ --exclude=modules/'

```

```
eval $RSYNC pi@<pi-ip>:/usr/share      sysroot/usr
eval $RSYNC pi@<pi-ip>:/lib           sysroot
eval $RSYNC pi@<pi-ip>:/usr/include   sysroot/usr
eval $RSYNC pi@<pi-ip>:/usr/lib       sysroot/usr
eval $RSYNC pi@<pi-ip>:/usr/local/lib sysroot/usr/local
eval $RSYNC pi@<pi-ip>:/usr/local/include sysroot/usr/local
eval $RSYNC pi@<pi-ip>:/etc          sysroot
```

4.3 Fix absolute symlinks

The Pi's symlinks are absolute (they point at the Pi's `/`). Rewrite them to be sysroot-relative so they resolve correctly on the host.

```
python3 sysroot-relativelinks.py sysroot
```

(`sysroot-relativelinks.py` is a ~30-line script that walks the tree and rewrites absolute symlinks to relative ones. A copy ships with the `riscv-poky` project on GitHub if you don't have it.)

5. Part 3 — The toolchain file (toolchain.cmake)

This is the CMake cross-toolchain that points at the cross-compiler and the sysroot. Place it somewhere permanent and adjust `TARGET_SYSROOT` to your path.

```
cmake_minimum_required(VERSION 3.18)
include_guard(GLOBAL)

set(CMAKE_SYSTEM_NAME Linux)
set(CMAKE_SYSTEM_PROCESSOR arm)

set(TARGET_SYSROOT /path/to/bookworm/sysroot)
set(TARGET_ARCHITECTURE aarch64-linux-gnu)
set(CMAKE_SYSROOT ${TARGET_SYSROOT})

set(ENV{PKG_CONFIG_PATH} $PKG_CONFIG_PATH:${CMAKE_SYSROOT}/usr/lib/${TARGET_ARCHITECTURE}/pkgconfig)
set(ENV{PKG_CONFIG_LIBDIR} ${CMAKE_SYSROOT}/usr/lib/pkgconfig:${CMAKE_SYSROOT}/usr/share/pkgconfig:${CMAKE_SYSROOT}/usr/lib/${TARGET_ARCHITECTURE}/pkgconfig:${CMAKE_SYSROOT}/usr/lib/pkgconfig)
set(ENV{PKG_CONFIG_SYSROOT_DIR} ${CMAKE_SYSROOT})

set(CMAKE_C_COMPILER /opt/cross-pi-gcc/bin/${TARGET_ARCHITECTURE}-gcc)
set(CMAKE_CXX_COMPILER /opt/cross-pi-gcc/bin/${TARGET_ARCHITECTURE}-g++)

set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -isystem=/usr/include -isystem=/usr/local/include -isystem=${CMAKE_SYSROOT}/usr/include/${TARGET_ARCHITECTURE}")
set(CMAKE_CXX_FLAGS "${CMAKE_C_FLAGS}")

set(QT_COMPILER_FLAGS "-march=armv8-a")
```

```

set(QT_COMPILER_FLAGS_RELEASE "-O2 -pipe")
set(QT_LINKER_FLAGS "-Wl,-01 -Wl,--hash-style=gnu -Wl,--as-needed -Wl,-rpath-
link=${TARGET_SYSR00T}/usr/lib/${TARGET_ARCHITECTURE}
-Wl,-rpath-link=/opt/Trolltech/Qt6/qt-raspi/lib")

set(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER)
set(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
set(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)
set(CMAKE_FIND_ROOT_PATH_MODE_PACKAGE ONLY)
set(CMAKE_INSTALL_RPATH_USE_LINK_PATH TRUE)
set(CMAKE_BUILD_RPATH ${TARGET_SYSR00T})

include(CMakeInitializeConfigs)

function(cmake_initialize_per_config_variable _PREFIX _DOCSTRING)
    if (_PREFIX MATCHES "CMAKE_(C|CXX|ASM)_FLAGS")
        set(CMAKE_${CMAKE_MATCH_1}_FLAGS_INIT "${QT_COMPILER_FLAGS}")
        foreach (config DEBUG RELEASE MINSIZEREL RELWITHDEBINFO)
            if (DEFINED QT_COMPILER_FLAGS_${config})
                set(CMAKE_${CMAKE_MATCH_1}_FLAGS_${config}_INIT "$
${QT_COMPILER_FLAGS_${config}}")
            endif()
        endforeach()
    endif()

    if (_PREFIX MATCHES "CMAKE_(SHARED|MODULE|EXE)_LINKER_FLAGS")
        foreach (config SHARED MODULE EXE)
            set(CMAKE_${config}_LINKER_FLAGS_INIT "${QT_LINKER_FLAGS}")
        endforeach()
    endif()

    _cmake_initialize_per_config_variable(${ARGV})
endfunction()

set(XCB_PATH_VARIABLE ${TARGET_SYSR00T})
set(GL_INC_DIR ${TARGET_SYSR00T}/usr/include)
set(GL_LIB_DIR ${TARGET_SYSR00T}:${TARGET_SYSR00T}/usr/lib/$
{TARGET_ARCHITECTURE}/${TARGET_SYSR00T}/usr:${TARGET_SYSR00T}/usr/lib)

set(EGL_INCLUDE_DIR ${GL_INC_DIR})
set(EGL_LIBRARY
${XCB_PATH_VARIABLE}/usr/lib/${TARGET_ARCHITECTURE}/libEGL.so)
set(OPENGGL_INCLUDE_DIR ${GL_INC_DIR})
set(OPENGGL_opengl_LIBRARY ${XCB_PATH_VARIABLE}/usr/lib/$
{TARGET_ARCHITECTURE}/libOpenGL.so)
set(GLESv2_INCLUDE_DIR ${GL_INC_DIR})
set(GLESv2_LIBRARY
${XCB_PATH_VARIABLE}/usr/lib/${TARGET_ARCHITECTURE}/libGLESv2.so)
set(gbm_INCLUDE_DIR ${GL_INC_DIR})
set(gbm_LIBRARY
${XCB_PATH_VARIABLE}/usr/lib/${TARGET_ARCHITECTURE}/libgbm.so)
set(Libdrm_INCLUDE_DIR ${GL_INC_DIR})
set(Libdrm_LIBRARY

```

```

${XCB_PATH_VARIABLE}/usr/lib/${TARGET_ARCHITECTURE}/libdrm.so)
set(XCB_XCB_INCLUDE_DIR ${GL_INC_DIR})
set(XCB_XCB_LIBRARY
${XCB_PATH_VARIABLE}/usr/lib/${TARGET_ARCHITECTURE}/libxcb.so)

# Help CMake's FindCups locate CUPS in the sysroot (needed by
Qt6PrintSupport).
set(CUPS_INCLUDE_DIR ${CMAKE_SYSROOT}/usr/include)
set(CUPS_LIBRARIES
${CMAKE_SYSROOT}/usr/lib/${TARGET_ARCHITECTURE}/libcups.so)

list(APPEND CMAKE_LIBRARY_PATH ${CMAKE_SYSROOT}/usr/lib/$
{TARGET_ARCHITECTURE})
list(APPEND CMAKE_PREFIX_PATH "/usr/lib/${TARGET_ARCHITECTURE}/cmake")

```

Note the explicit **CUPS** hints at the bottom — these were added because **Qt6PrintSupport** could not locate CUPS inside the sysroot otherwise. The GL/EGL/GBM **set(...)** blocks near the bottom similarly pin those libraries into the sysroot.

6. Part 4 — Get the Qt source

Clone the Qt supermodule and check out Qt 6.8 LTS. (The repository is historically named **qt5** even for Qt 6 — it is the supermodule that references every Qt submodule.)

```

git clone https://codereview.qt-project.org/qt/qt5 qt6
cd qt6
git checkout v6.8.3
perl init-repository -f    # pulls all submodules (qtbases, qtwebengine, ...)

```

7. Part 5 — The QtWebEngine fixes (what makes it build)

QtWebEngine (Chromium) is the part that needs manual intervention. There are **three fixes**, plus one build instruction. These are the exact changes that were required to get WebEngine to cross-compile for arm64 against the bookworm sysroot.

7.1 Fix 1 — the toolchain limits.h

File:

/opt/cross-pi-gcc/lib/gcc/aarch64-linux-gnu/12.4.0/include-fixed/limits.h

Symptom: Chromium/V8 fails to compile with missing **MB_LEN_MAX**, **SSIZE_MAX**, and other limits macros.

Root cause: the GCC-generated "fixed" limits.h in this toolchain has a non-standard include guard and does not include **<syslimits.h>**, so glibc's real limits.h is never pulled in correctly. **MB_LEN_MAX** also defaults to 1 instead of 16.

Changes:

8. Change the include guard `_LIMITS_H__` → `_GCC_LIMITS_H_`.
9. Add `#include <syslimits.h>` (just before the final `#endif`).
10. Set `MB_LEN_MAX` to 16 (matching the bookworm glibc).

Equivalently, with `sed`:

```
TL=/opt/cross-pi-gcc/lib/gcc/aarch64-linux-gnu/12.4.0/include-fixed/limits.h
sed -i \
  -e 's/_LIMITS_H__/_GCC_LIMITS_H_/g' \
  -e 's/#define[[:space:]]*MB_LEN_MAX[[:space:]]*[0-9]*/#define MB_LEN_MAX
16/' "$TL"
grep -q '#include <syslimits.h>' "$TL" || \
  sed -i 's|^#endif /\* _GCC_LIMITS_H_ */|#include <syslimits.h>\n#endif /*
_GCC_LIMITS_H_ */|' "$TL"
```

Important: this fix lives in the `*toolchain*`, not the Qt source, so it is **lost if you rebuild** ``/opt/cross-pi-gcc`` — re-apply it after any toolchain rebuild. The fix only makes the fixed header behave like a normal GCC `limits.h`; it is safe for other projects.

7.2 Fix 2 — `webengine_undef_math.h` (new file)

File (new): `qt6/qtwebengine/src/core/webengine_undef_math.h`

Symptom: compiling ANGLE fails with `'isnan' is not a member of 'sh::BuiltInName'` and the same for `isinf`.

Root cause: bookworm's glibc `<math.h>` defines `isnan()` and `isinf()` as macros. ANGLE's generated `SymbolTable_*_autogen.cpp` files use the literal names `isnan` and `isinf` as identifiers inside `namespace BuiltInName`, so the preprocessor expands them into invalid built-in calls.

Fix: create a tiny forced-include header that includes `<cmath>`. The C++ standard library's `<cmath>` pulls in `<math.h>` and then explicitly `#undefs` `isinf/isnan`; because `<math.h>` is then guarded, the macros stay undefined for the rest of the translation unit. The header also re-exports the `std::` classification names into the global namespace so normal callers keep working.

Full contents:

```
// Helper header for the Qt WebEngine / ANGLE cross build.
// glibc's <math.h> defines isnan() and isinf() as macros. ANGLE's
// generated SymbolTable_*_autogen.cpp files use the literal names
// "isnan" and "isinf" as identifiers inside namespace BuiltInName,
// which breaks when those macros are visible.
//
// Including <cmath> early pulls in <math.h> and then undefines the
// isnan/isinf macros (libstdc++/libc++ both do this for C++). After
// that the macro names stay undefined for the rest of the translation
```

```
// unit because <math.h> is guarded by its include guard.

#ifndef WEBENGINE_UNDEF_MATH_H
#define WEBENGINE_UNDEF_MATH_H

#ifdef __cplusplus
#include <cmath>
using std::fpclassify;
using std::isfinite;
using std::isgreater;
using std::isgreaterequal;
using std::isinf;
using std::isless;
using std::islessequal;
using std::islessgreater;
using std::isnan;
using std::isnormal;
using std::isunordered;
using std::signbit;
#endif // __cplusplus

#endif // WEBENGINE_UNDEF_MATH_H
```

7.3 Fix 3 — BUILD.toolchain.gn.in (force-include the header)

File: `qt6/qtwebengine/src/host/BUILD.toolchain.gn.in`

Edit the `extra_cppflags` line to force-include the new header. **Before:**

```
extra_cppflags = "-isystem=/usr/include -isystem=/usr/include/aarch64-linux-
gnu -Uisnan -Uisinf"
```

After:

```
extra_cppflags = "-isystem=/usr/include -isystem=/usr/include/aarch64-linux-
gnu -include @WEBENGINE_ROOT_SOURCE_DIR@/src/core/webengine_undef_math.h"
```

(`-Uisnan -Uisinf` alone does **not** work: `-U` only undefines before preprocessing starts, and transitively-included headers redefine the macros again.)

7.4 Qt 6.8 build flow — drive it with `cmake --build`

In Qt 6.8, `gn gen` (which generates Chromium's build files) runs during the `cmake build` phase, not during configure. So you must build with `cmake --build .` — it fires `gn gen` first, then compiles. The old 6.7.x manual step (`cd src/core/Release/arm; ninja QtWebEngineCore`) no longer works: it skips `cmake --build`, so `build.ninja` is never generated and `ninja` fails with `loading 'build.ninja': No such file or directory`.

(The earlier "build only QtWebEngineCore" workaround is obsolete in 6.8 — the host build proves the default `cmake --build` target now builds cleanly.)

8. Part 6 — Build the Qt modules for the host (x86_64)

Build the **host** Qt first: its **qt-configure-module** and host tools (moc/rcc/uic) are what the cross build reuses. Everything installs into **/opt/Trolltech/Qt6/qt-host**.

The full module set (in build order — qtbases first, then the rest; this order respects inter-module dependencies):

```
qtbases      # always first
qtimageformats
qtlanguageserver
qtshadertools
qtsvg
qtserialbus
qtdeclarative
qtwebsockets
qtwebchannel
qttools
qtserialport
qtpositioning
qt5compat
qtwayland
```

(QtWebEngine is built separately in Part 8 — it has its own fixes and is huge.)

8.1 Build qtbases (host)

```
mkdir -p build/host/qtbases && cd build/host/qtbases
cmake /path/to/qt6/qtbases -GNinja \
  -DINPUT_opengl=es2 -DCMAKE_BUILD_TYPE=Release \
  -DQT_BUILD_EXAMPLES=OFF -DQT_BUILD_TESTS=OFF \
  -DCMAKE_INSTALL_PREFIX=/opt/Trolltech/Qt6/qt-host \
  -DQT_FEATURE_xcb=ON -DFEATURE_xcb_xlib=ON -DQT_FEATURE_xlib=ON
cmake --build . --parallel 8
cmake --install .
```

8.2 Build each remaining module (host)

For every module in the list above (except qtbases), use a fresh build directory and the host Qt's **qt-configure-module**:

```
mkdir -p build/host/<module> && cd build/host/<module>
/opt/Trolltech/Qt6/qt-host/bin/qt-configure-module /path/to/qt6/<module>
cmake --build . --parallel 8
cmake --install .
```

Repeat for **qtimageformats**, **qtlanguageserver**, **qtshadertools**, **qtsvg**, **qtserialbus**, **qtdeclarative**, **qtwebsockets**, **qtwebchannel**, **qttools**, **qtserialport**, **qtpositioning**, **qt5compat**, **qtwayland** — in that order.

9. Part 7 — Build the Qt modules for the target (arm64)

Now cross-compile the same modules for arm64, installing into `/opt/Trolltech/Qt6/qt-raspi`. Each target configure points at the toolchain file, the host Qt, and the sysroot.

9.1 Build qtbase (target)

```
mkdir -p build/target/qtbase && cd build/target/qtbase
cmake /path/to/qt6/qtbase -GNinja \
  -DINPUT_opengl=es2 -DCMAKE_BUILD_TYPE=Release \
  -DCMAKE_TOOLCHAIN_FILE=/path/to/toolchain.cmake \
  -DQT_HOST_PATH=/opt/Trolltech/Qt6/qt-host \
  -DCMAKE_STAGING_PREFIX=/opt/Trolltech/Qt6/qt-raspi \
  -DCMAKE_INSTALL_PREFIX=/usr/local/qt6 \
  -DQT_FEATURE_xcb=ON -DFEATURE_xcb_xlib=ON -DQT_FEATURE_xlib=ON
cmake --build . --parallel 8
cmake --install .
```

`CMAKE_STAGING_PREFIX` is where the files land on the host build machine;

`CMAKE_INSTALL_PREFIX=/usr/local/qt6` is the runtime path baked into the binaries, matching where they will live on the Pi.

9.2 Build each remaining module (target)

```
mkdir -p build/target/<module> && cd build/target/<module>
/opt/Trolltech/Qt6/qt-raspi/bin/qt-configure-module /path/to/qt6/<module>
cmake --build . --parallel 8
cmake --install .
```

Same module list and order as the host pass (8.2).

10. Part 8 — Build QtWebEngine (host + target)

Apply the Part 5 fixes **before** this step. Then build WebEngine for the host:

```
mkdir -p module.host && cd module.host
/opt/Trolltech/Qt6/qt-host/bin/qt-configure-module /path/to/qt6/qtwebengine
cmake --build . --parallel 8
cmake --install .
```

Then cross-build it for the Pi.

Qt 6.8 flag rename: the old `-no-webengine-*` configure options became `*features*`, so they are now passed as `-no-feature-*` (e.g. `-no-qtpdf-build` → `-no-feature-qtpdf-build`). They disable the features that need extra dependencies (PDF, WebRTC, printing, spellchecker, Pepper, proprietary codecs, Kerberos, extensions):

```

mkdir -p module.raspi && cd module.raspi
/opt/Trolltech/Qt6/qt-raspi/bin/qt-configure-module
/path/to/qt6/qtwebengine \
  -no-feature-qtpdf-build \
  -no-feature-webengine-webrtc \
  -no-feature-webengine-printing-and-pdf \
  -no-feature-webengine-spellchecker \
  -no-feature-webengine-pepper-plugins \
  -no-feature-webengine-proprietary-codecs \
  -no-feature-webengine-kerberos \
  -no-feature-webengine-extensions

cmake --build . --parallel $(nproc)  # triggers gn gen, then builds
QtWebEngineCore
cmake --install .

```

Qt 6.8 build flow: we build the module with `cmake --build .` (which fires `gn gen` first), not the old manual `cd src/core/Release/arm; ninja QtWebEngineCore` step. See 7.4.

11. Part 9 — Deploy to the Pi and verify

11.1 Deploy

Copy the cross-built Qt (`qt-raspi`) to the Pi and refresh the linker cache:

```

rsync -az --info=progress2 --rsync-path="sudo rsync" \
  /opt/Trolltech/Qt6/qt-raspi/ pi@<pi-ip>:/usr/local/qt6
ssh pi@<pi-ip> 'sudo ldconfig'

```

11.2 Verify the binaries are valid arm64 with a matching glibc

```

file <binary>
# expect: ... ARM aarch64, ... ELF

readelf -l <binary> | grep interpreter
# expect: /lib/ld-linux-aarch64.so.1

readelf --version-info <binary> | grep -o 'GLIBC_2\.[0-9]*' | sort -Vu
# expect: only GLIBC_2.36 and lower – this is the check that catches an
#          accidental glibc-version mismatch in the toolchain.

```

Appendix A — Full fix reference

A.1 Toolchain limits.h (before → after)

The three edits, shown as the exact strings:

Change	Before	After
Include guard	<code>`_LIMITS_H__`</code>	<code>`_GCC_LIMITS_H_`</code>
syslimits include	(absent)	<code>`#include <syslimits.h>`</code>
MB_LEN_MAX	<code>`#define MB_LEN_MAX 1`</code>	<code>`#define MB_LEN_MAX 16`</code>

A.2 webengine_undef_math.h

The complete new file (also shown in 7.2). Guarded with `#ifdef __cplusplus` because `<cmath>` is C++-only and some C translation units share the same flags.

```
#ifndef WEBENGINE_UNDEF_MATH_H
#define WEBENGINE_UNDEF_MATH_H

#ifdef __cplusplus
#include <cmath>
using std::fpclassify;
using std::isfinite;
using std::isgreater;
using std::isgreaterequal;
using std::isinf;
using std::isless;
using std::islessequal;
using std::islessgreater;
using std::isnan;
using std::isnormal;
using std::isunordered;
using std::signbit;
#endif // __cplusplus

#endif // WEBENGINE_UNDEF_MATH_H
```

A.3 BUILD.toolchain.gn.in — the single line that changes

```
# BEFORE
extra_cppflags = "-isystem=/usr/include -isystem=/usr/include/aarch64-linux-gnu -Uisnan -Uisinf"

# AFTER
extra_cppflags = "-isystem=/usr/include -isystem=/usr/include/aarch64-linux-gnu -include @WEBENGINE_ROOT_SOURCE_DIR@/src/core/webengine_undef_math.h"
```

Appendix B — Qt Creator kit setup (optional)

To use the cross-built Qt from Qt Creator for your own projects:

11. **Compilers:** add C and C++ pointing at `/opt/cross-pi-gcc/bin/aarch64-linux-gnu-gcc / -g++`.
12. **Qt version:** add the arm64 Qt at `/opt/Trolltech/Qt6/qt-raspi`.
13. **Kit:** device type **Generic Linux**, sysroot = your `sysroot/` directory, with the compiler and Qt version above.
14. In the kit's **CMake Configuration**, add
`-DCMAKE_TOOLCHAIN_FILE=/path/to/toolchain.cmake`.
15. Disable the maintenance-tool provider (it breaks Qt6 component lookup):

```
-DQT_CREATOR_ENABLE_MAINTENANCE_TOOL_PROVIDER:BOOL=OFF  
-DQT_CREATOR_ENABLE_PACKAGE_MANAGER_SETUP:BOOL=OFF
```

16. If `Qt6WebEngineWidgets` fails under the umbrella `find_package(Qt6 COMPONENTS ...)`, call it directly:

```
find_package(Qt6WebEngineWidgets)
```

17. On the Pi, ensure the runtime library path is known:

```
echo '/usr/local/qt6/lib' | sudo tee /etc/ld.so.conf.d/qt6.conf && sudo  
ldconfig
```

Share this guide

Here is a ready-to-paste announcement for forums, Hacker News, or social media. Lead with the three QtWebEngine fixes — that is the part people are stuck on.

Cross-compiling Qt 6.8 + QtWebEngine for Raspberry Pi 64-bit (bookworm)

I've written a complete, step-by-step guide to cross-compiling Qt 6.8 LTS – including the notoriously difficult QtWebEngine – for a 64-bit Raspberry Pi running Raspberry Pi OS bookworm, from an x86_64 Linux host.

Inside you'll find:

- Building the aarch64 cross-compiler (GCC 12.4.0 / glibc 2.36) from source.
- Preparing the Raspberry Pi sysroot and the CMake toolchain file.
- The three QtWebEngine arm64 show-stoppers nobody documents, and the fixes:
 1. the GCC include-fixed limits.h fix,
 2. the ANGLE isnan/isinf macro-collision workaround,
 3. the Qt 6.8 flag rename (-no-* → -no-feature-*) + cmake --build flow.
- Building every Qt module for host and target, in dependency order.

To learn how to do it, go to: www.vremsoft.com

Tested on Raspberry Pi OS bookworm (64-bit).

– Ed Chapman

(This is for Linux users. If you ask me how to do it on Windows, I won't answer – not because I'm rude, but because I don't know a thing about Windows.

I'm old, and I haven't got the brain cells left to leave the Unix/Linux world.)