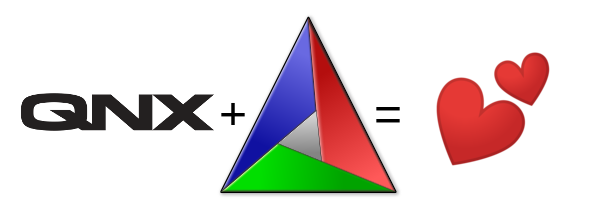
**[Cristian Adam](https://cristianadam.eu/)**

* [Twitter](https://twitter.com/cristianadam)
* [RSS](https://cristianadam.eu/atom.xml)
* [Blog](https://cristianadam.eu/)
* [Archives](https://cristianadam.eu/archives)

**A Better QNX CMake Toolchain File**

Dec 02, 2018



At the end of October 2018 on the Qt development mailing list it was [announced](https://lists.qt-project.org/pipermail/development/2018-October/034023.html) that CMake was chosen as the build system (generator) for building Qt6. That also meant that The Qt Company will gradually stop investing in their in house Qbs build system.

I personally think is a good idea to have major C++ projects like Boost ([July 2017 switch announcement!](https://lists.boost.org/boost-interest/2017/07/0162.php) ), LLVM/Clang, and now Qt to use CMake as their build system (generator). We C++ developers should work together in having a common build system.

There was a bit of email traffic on this topic. There was some skepticism of CMake being able to support specialized operating systems like QNX, so I pointed to an October 2017 blog entry of Doug Schaefer named [QNX CMake Toolchain File](http://cdtdoug.ca/2017/10/06/qnx-cmake-toolchain-file.html). There Doug Schaefer presents us with a minimal CMake Toolchain File.

Since I am lucky() to have a QNX 7.0 license I tried to compile and run the recently released CMake 3.13.0 for the QNX 7.0 x86\_64 target!

**Basic CMake Compilation**

The toolchain looks like this:

set(CMAKE\_SYSTEM\_NAME QNX)

set(arch gcc\_ntox86\_64)

set(ntoarch x86\_64)

set(QNX\_PROCESSOR x86\_64)

set(CMAKE\_C\_COMPILER qcc)

set(CMAKE\_C\_COMPILER\_TARGET ${arch})

set(CMAKE\_CXX\_COMPILER qcc -lang-c++)

set(CMAKE\_CXX\_COMPILER\_TARGET ${arch})

set(CMAKE\_ASM\_COMPILER qcc -V${arch})

set(CMAKE\_ASM\_DEFINE\_FLAG "-Wa,--defsym,")

set(CMAKE\_RANLIB $ENV{QNX\_HOST}/usr/bin/nto${ntoarch}-ranlib

CACHE PATH "QNX ranlib Program" FORCE)

set(CMAKE\_AR $ENV{QNX\_HOST}/usr/bin/nto${ntoarch}-ar

CACHE PATH "QNX qr Program" FORCE)

The build script looks like this:

#!/bin/bash

source $HOME/qnx700/qnxsdp-env.sh

cmake -B builddir -S cmake-3.13.0 -G Ninja \

-DCMAKE\_BUILD\_TYPE=Release \

-DCMAKE\_TOOLCHAIN\_FILE=`pwd`/qnx7.cmake \

-DCMAKE\_INSTALL\_PREFIX=`pwd`/installdir

cmake --build builddir --target install

Notice how I am using CMake 3.13.x’s -S and -B parameters! No more mkdir builddir && cd builddir commands anymore! Yeah! 

The configuration step had some problems because it uses try\_run and I was cross-compiling. Running the script a second time worked out fine.

The failed CMake configuration was due to:

CMake Error: TRY\_RUN() invoked in cross-compiling mode, please set the following cache variables appropriately:

KWSYS\_LFS\_WORKS (advanced)

KWSYS\_LFS\_WORKS\_\_TRYRUN\_OUTPUT (advanced)

CMake Error: TRY\_RUN() invoked in cross-compiling mode, please set the following cache variables appropriately:

HAVE\_POLL\_FINE\_EXITCODE (advanced)

HAVE\_POLL\_FINE\_EXITCODE\_\_TRYRUN\_OUTPUT (advanced)

CMake Error at CMakeLists.txt:2 (project):

The Ninja generator does not support Fortran using Ninja version

1.8.2

due to lack of required features. Kitware has implemented the required

features but as of this version of CMake they have not been integrated to

upstream ninja. Pending integration, Kitware maintains a branch at:

https://github.com/Kitware/ninja/tree/features-for-fortran#readme

with the required features. One may build ninja from that branch to get

support for Fortran.

CMake Error: CMAKE\_Fortran\_COMPILER not set, after EnableLanguage

The compilation fails at some point because libuv doesn’t have QNX support. The following patch gets things working!

diff --git a/Utilities/cmlibuv/CMakeLists.txt b/Utilities/cmlibuv/CMakeLists.txt

index a503041be..364a1e75e 100644

--- a/Utilities/cmlibuv/CMakeLists.txt

+++ b/Utilities/cmlibuv/CMakeLists.txt

@@ -135,6 +135,23 @@ if(CMAKE\_SYSTEM\_NAME STREQUAL "AIX")

)

endif()

+if(CMAKE\_SYSTEM\_NAME STREQUAL "QNX")

+ list(APPEND uv\_libraries

+ )

+ list(APPEND uv\_headers

+ include/uv-posix.h

+ )

+ list(APPEND uv\_defines

+ )

+ list(APPEND uv\_sources

+ src/unix/bsd-ifaddrs.c

+ src/unix/no-fsevents.c

+ src/unix/no-proctitle.c

+ src/unix/posix-hrtime.c

+ src/unix/posix-poll.c

+ )

+endif()

+

if(CMAKE\_SYSTEM\_NAME MATCHES "CYGWIN")

list(APPEND uv\_libraries

)

diff --git a/Utilities/cmlibuv/include/uv-unix.h b/Utilities/cmlibuv/include/uv-unix.h

index 455674d1a..10389f474 100644

--- a/Utilities/cmlibuv/include/uv-unix.h

+++ b/Utilities/cmlibuv/include/uv-unix.h

@@ -66,6 +66,8 @@

# include "uv-bsd.h"

#elif defined(\_\_CYGWIN\_\_) || defined(\_\_MSYS\_\_)

# include "uv-posix.h"

+#elif defined(\_\_QNXNTO\_\_)

+# include "uv-posix.h"

#endif

#ifndef PTHREAD\_BARRIER\_SERIAL\_THREAD

cmake, cpack, and ctest compiled and installed just fine! So… we’re done, right? 

**CMake-GUI Building**

What happens if we want to build cmake-gui? CMake is shipping cmake-gui for Windows/Mac/Linux as a GUI application statically linked to Qt.

So I went and compiled Qt 5.11.2 statically for QNX x86\_64 with this script:

#!/bin/bash

source $HOME/qnx700/qnxsdp-env.sh

../qt-everywhere-src-5.11.2/configure \

-xplatform qnx-x86-64-qcc \

-release \

-static \

-no-fontconfig \

-no-icu \

--freetype=qt \

-ccache \

-nomake examples \

-nomake tests \

-skip qtwebengine \

-prefix /opt/qt5 \

-opensource

The magic part above is the -xplatform qnx-x86-64-qcc. I don’t build icu, fontconfig, because the QNX 7.0 VMware image doesn’t provide them, and I felt that it defeated my goal to deploy \*.so files, hack LD\_LIBRARY\_PATH, and so on. I just wanted to run ./cmake-gui.

The toolchain does have libicu, which is quite a monster (31.4M!):

$ du -h /home/cadam/qnx700/target/qnx7/x86\_64/usr/lib/libicu\*

0 /home/cadam/qnx700/target/qnx7/x86\_64/usr/lib/libicudata.so

0 /home/cadam/qnx700/target/qnx7/x86\_64/usr/lib/libicudata.so.58

26M /home/cadam/qnx700/target/qnx7/x86\_64/usr/lib/libicudata.so.58.1

0 /home/cadam/qnx700/target/qnx7/x86\_64/usr/lib/libicui18n.so

0 /home/cadam/qnx700/target/qnx7/x86\_64/usr/lib/libicui18n.so.58

3.3M /home/cadam/qnx700/target/qnx7/x86\_64/usr/lib/libicui18n.so.58.1

0 /home/cadam/qnx700/target/qnx7/x86\_64/usr/lib/libicuuc.so

0 /home/cadam/qnx700/target/qnx7/x86\_64/usr/lib/libicuuc.so.58

2.1M /home/cadam/qnx700/target/qnx7/x86\_64/usr/lib/libicuuc.so.58.1

My CMake build script would change to:

#!/bin/bash

source $HOME/qnx700/qnxsdp-env.sh

cmake -B builddir -S cmake-3.13.0 -G Ninja \

-DCMAKE\_BUILD\_TYPE=Release \

-DCMAKE\_TOOLCHAIN\_FILE=`pwd`/qnx7.cmake \

-DCMAKE\_INSTALL\_PREFIX=`pwd`/installdir \

-DBUILD\_QtDialog=ON \

-DCMAKE\_PREFIX\_PATH=`pwd`/qt5/lib/cmake

cmake --build builddir --target install

Unfortunately CMake configure step stops with the following error:

CMake Error in Source/QtDialog/CMakeLists.txt:

No known features for CXX compiler

"QCC"

version 5.4.0.

As it turns out I hit [QTBUG-54666: CMake fails to configure Android build](https://bugreports.qt.io/browse/QTBUG-54666)!

The CMake package files that Qt provides require some C++ compiler features to be present.

So what did CMake detect using our QNX toolchain? Let’s just take a peak!

$ cat ./builddir/CMakeFiles/3.13.0/CMakeCXXCompiler.cmake | grep -i compile\_features

set(CMAKE\_CXX\_COMPILE\_FEATURES "")

set(CMAKE\_CXX98\_COMPILE\_FEATURES "")

set(CMAKE\_CXX11\_COMPILE\_FEATURES "")

set(CMAKE\_CXX14\_COMPILE\_FEATURES "")

set(CMAKE\_CXX17\_COMPILE\_FEATURES "")

set(CMAKE\_CXX20\_COMPILE\_FEATURES "")

The above bug report has some workarounds for this problem, but what if we fixed this? QNX has QCC as a compiler wrapper around GCC, so what if I used GCC directly?

I came up with this small QNX toolchain:

set(CMAKE\_SYSTEM\_NAME QNX)

set(arch ntox86\_64)

set(QNX\_PROCESSOR x86\_64)

set(CMAKE\_C\_COMPILER $ENV{QNX\_HOST}/usr/bin/${arch}-gcc)

set(CMAKE\_C\_COMPILER\_TARGET ${arch})

set(CMAKE\_CXX\_COMPILER $ENV{QNX\_HOST}/usr/bin/${arch}-g++)

set(CMAKE\_CXX\_COMPILER\_TARGET ${arch})

Now I was able to compile, but not to link. Oh no! 

**CMake-GUI Linking and Deployment**

I had a look at what CMake was doing for Windows and came up with similar approach for QNX.

I needed to apply the following patch:

diff -Naur cmake-3.13.0-vanilla/Source/QtDialog/CMakeLists.txt cmake-3.13.0/Source/QtDialog/CMakeLists.txt

--- cmake-3.13.0-vanilla/Source/QtDialog/CMakeLists.txt 2018-11-20 15:49:09.000000000 +0100

+++ cmake-3.13.0/Source/QtDialog/CMakeLists.txt 2018-12-01 01:27:01.203767827 +0100

@@ -39,6 +39,12 @@

PROPERTY COMPILE\_DEFINITIONS USE\_QWindowsIntegrationPlugin)

endif()

+ if(CMake\_QT\_STATIC\_QQnxIntegrationPlugin\_LIBRARIES)

+ list(APPEND CMake\_QT\_LIBRARIES ${CMake\_QT\_STATIC\_QQnxIntegrationPlugin\_LIBRARIES})

+ set\_property(SOURCE CMakeSetup.cxx

+ PROPERTY COMPILE\_DEFINITIONS USE\_QQnxIntegrationPlugin)

+ endif()

+

# We need to install platform plugin and add qt.conf for Qt5 on Mac and Windows.

# FIXME: This should be part of Qt5 CMake scripts, but unfortunately

# Qt5 support is missing there.

diff -Naur cmake-3.13.0-vanilla/Source/QtDialog/CMakeSetup.cxx cmake-3.13.0/Source/QtDialog/CMakeSetup.cxx

--- cmake-3.13.0-vanilla/Source/QtDialog/CMakeSetup.cxx 2018-11-20 15:49:09.000000000 +0100

+++ cmake-3.13.0/Source/QtDialog/CMakeSetup.cxx 2018-12-01 14:28:09.902659579 +0100

@@ -49,6 +49,11 @@

Q\_IMPORT\_PLUGIN(QWindowsIntegrationPlugin);

#endif

+#if defined(USE\_QQnxIntegrationPlugin)

+Q\_IMPORT\_PLUGIN(QQnxIntegrationPlugin);

+#endif

+

+

int main(int argc, char\*\* argv)

{

cmsys::Encoding::CommandLineArguments encoding\_args =

Now the build script looks like this:

#!/bin/bash

source $HOME/qnx700/qnxsdp-env.sh

# CMake ; separated list

LIBS="`pwd`/qt5/plugins/platforms/libqqnx.a;"

LIBS+="`pwd`/qt5/lib/libQt5EglSupport.a;"

LIBS+="`pwd`/qt5/lib/libQt5EventDispatcherSupport.a;"

LIBS+="`pwd`/qt5/lib/libQt5FontDatabaseSupport.a;"

LIBS+="`pwd`/qt5/lib/libQt5Core.a;"

LIBS+="`pwd`/qt5/lib/libQt5Gui.a;"

LIBS+="`pwd`/qt5/lib/libqtharfbuzz.a;"

LIBS+="`pwd`/qt5/lib/libqtpcre2.a;"

LIBS+="`pwd`/qt5/lib/libqtfreetype.a;"

LIBS+="-lpng16;-lz;-lslog2;"

LIBS+="-lscreen;-lpps;-lEGL;-lGLESv2"

cmake -B builddir -S cmake-3.13.0 -G Ninja \

-DCMAKE\_BUILD\_TYPE=Release \

-DCMAKE\_TOOLCHAIN\_FILE=`pwd`/qnx7\_gcc.cmake \

-DCMAKE\_INSTALL\_PREFIX=`pwd`/installdir \

-DBUILD\_QtDialog=ON \

-DCMake\_QT\_STATIC\_QQnxIntegrationPlugin\_LIBRARIES="$LIBS" \

-DCMAKE\_PREFIX\_PATH=`pwd`/qt5/lib/cmake

cmake --build builddir --target install

That looks pretty scary! That’s because Qt’s CMake files do not track dependencies when built in static mode.

This is being tracked and hopefully soon fixed, as seen here: [QTBUG-38913: Can’t link against static Qt5 (missing usage requirements for static libs wrt harfbuzz/glib/others)](https://bugreports.qt.io/browse/QTBUG-38913).

Until Qt fixes their CMake files, we could just do the following:

diff -Naur qt5-vanilla/lib/cmake/Qt5Core/Qt5CoreConfig.cmake qt5/lib/cmake/Qt5Core/Qt5CoreConfig.cmake

--- qt5-vanilla/lib/cmake/Qt5Core/Qt5CoreConfig.cmake 2018-12-01 12:37:49.000000000 +0100

+++ qt5/lib/cmake/Qt5Core/Qt5CoreConfig.cmake 2018-12-01 22:11:51.552732104 +0100

@@ -111,8 +111,16 @@

list(REMOVE\_DUPLICATES Qt5Core\_COMPILE\_DEFINITIONS)

list(REMOVE\_DUPLICATES Qt5Core\_EXECUTABLE\_COMPILE\_FLAGS)

- set(\_Qt5Core\_LIB\_DEPENDENCIES "")

-

+ set(\_Qt5Core\_LIB\_DEPENDENCIES

+ ${\_qt5Core\_install\_prefix}/lib/libQt5EglSupport.a

+ ${\_qt5Core\_install\_prefix}/lib/libQt5EventDispatcherSupport.a

+ ${\_qt5Core\_install\_prefix}/lib/libQt5FontDatabaseSupport.a

+ ${\_qt5Core\_install\_prefix}/lib/libqtharfbuzz.a

+ ${\_qt5Core\_install\_prefix}/lib/libqtpcre2.a

+ ${\_qt5Core\_install\_prefix}/lib/libqtfreetype.a

+ -lpng16 -lz -lslog2

+ -lscreen -lpps -lEGL -lGLESv2

+ )

add\_library(Qt5::Core STATIC IMPORTED)

set\_property(TARGET Qt5::Core PROPERTY IMPORTED\_LINK\_INTERFACE\_LANGUAGES CXX)

Now the build script looks like this:

#!/bin/bash

source $HOME/qnx700/qnxsdp-env.sh

LIBS="`pwd`/qt5/plugins/platforms/libqqnx.a"

cmake -B builddir -S cmake-3.13.0 -G Ninja \

-DCMAKE\_BUILD\_TYPE=Release \

-DCMAKE\_TOOLCHAIN\_FILE=`pwd`/qnx7\_gcc.cmake \

-DCMAKE\_INSTALL\_PREFIX=`pwd`/installdir \

-DBUILD\_QtDialog=ON \

-DCMake\_QT\_STATIC\_QQnxIntegrationPlugin\_LIBRARIES="$LIBS" \

-DCMAKE\_PREFIX\_PATH=`pwd`/qt5/lib/cmake

cmake --build builddir --target install

That’s more like it!

cmake-gui builds and links fine now. In order to run it on the VM, I need to have sftp / ssh access. This is done by running vi /etc/ssh/sshd\_config and change # PermitRootLogin no to PermitRootLogin yes.

After deployment and running /etc/graphics-startup.sh I was able to run /root/installdir/bin/cmake-gui, but then got these nice warnings:

QFontDatabase: Cannot find font directory /opt/qt5/lib/fonts.

Note that Qt no longer ships fonts. Deploy some (from http://dejavu-fonts.org for example) or switch to fontconfig.

This can be fixed in two ways, either set QT\_QPA\_FONTDIR environment variable to /usr/share/fonts, or create a symlink like:

# mkdir -p /opt/qt5/lib

# ln -s /usr/share/fonts /opt/qt5/lib/fonts

And now I can present this beautiful screenshot: 

**Let me see you stripped!**

CMake when it uses GCC/Clang it builds binaries unstripped, with debug information. Let’s see how big the above resulted binaries are:

$ du -ah installdir/bin/

8.0M installdir/bin/cmake

26M installdir/bin/cmake-gui

8.2M installdir/bin/cpack

9.2M installdir/bin/ctest

52M installdir/bin/

CMake does have in the CMakeCache.txt an entry called CMAKE\_STRIP, which is the case of the original QNX toolchain is set to /usr/bin/strip, because CMake’s share/cmake-3.13/Modules/CMakeFindBinUtils.cmake has a bug for QNX, it can’t determine the ${\_CMAKE\_TOOLCHAIN\_PREFIX} variable!

This is the reason why the original QNX toolchain had entries to ar and ranlib utilities.

My toolchain simply works, because the GNU GCC detection mechanism of ${\_CMAKE\_TOOLCHAIN\_PREFIX} still applies!

But how can we use CMAKE\_STRIP? Well, CMake has an **undocumented target** named install/strip!

The build script looks like this now:

#!/bin/bash

source $HOME/qnx700/qnxsdp-env.sh

LIBS="`pwd`/qt5/plugins/platforms/libqqnx.a"

cmake -B builddir -S cmake-3.13.0 -G Ninja \

-DCMAKE\_BUILD\_TYPE=Release \

-DCMAKE\_TOOLCHAIN\_FILE=`pwd`/qnx7\_gcc.cmake \

-DCMAKE\_INSTALL\_PREFIX=`pwd`/installdir \

-DBUILD\_QtDialog=ON \

-DCMake\_QT\_STATIC\_QQnxIntegrationPlugin\_LIBRARIES="$LIBS" \

-DCMAKE\_PREFIX\_PATH=`pwd`/qt5/lib/cmake

cmake --build builddir --target install/strip

How big are the binaries now?

$ du -ah installdir/bin

6.7M installdir/bin/cmake

23M installdir/bin/cmake-gui

7.0M installdir/bin/cpack

7.9M installdir/bin/ctest

44M installdir/bin

That’s like **~15%** binary size decrease!

**Interprocedural optimization (IPO)**

CMake starting with version 3.9 has support for [Interprocedural optimization (IPO)](https://cmake.org/cmake/help/v3.9/module/CheckIPOSupported.html) for GCC and Clang compilers.

If we have a look at share/cmake-3.13/Modules/Compiler/QCC.cmake we could find:

set(\_CMAKE\_${lang}\_IPO\_SUPPORTED\_BY\_CMAKE NO)

set(\_CMAKE\_${lang}\_IPO\_MAY\_BE\_SUPPORTED\_BY\_COMPILER NO)

The original QNX toolchain is not use for us. But my toolchain is GCC based, which should just work.

CMake 3.13.0 source code doesn’t have support for building with IPO, but applying the following patch enables it:

diff -Naur cmake-3.13.0-vanilla/CMakeLists.txt cmake-3.13.0/CMakeLists.txt

--- cmake-3.13.0-vanilla/CMakeLists.txt 2018-11-20 15:49:09.000000000 +0100

+++ cmake-3.13.0/CMakeLists.txt 2018-11-24 22:53:52.323711946 +0100

@@ -117,6 +117,19 @@

"Build CMake Developer Reference" OFF)

mark\_as\_advanced(CMake\_BUILD\_DEVELOPER\_REFERENCE)

+# option to build using interprocedural optimizations (IPO/LTO)

+if (NOT CMAKE\_VERSION VERSION\_LESS 3.12.2)

+ option(CMake\_BUILD\_LTO "Compile CMake with link-time optimization if supported" OFF)

+ if(CMake\_BUILD\_LTO)

+ cmake\_policy(SET CMP0069 NEW)

+ include(CheckIPOSupported)

+ check\_ipo\_supported(RESULT HAVE\_IPO)

+ if(HAVE\_IPO)

+ set(CMAKE\_INTERPROCEDURAL\_OPTIMIZATION TRUE)

+ endif()

+ endif()

+endif()

+

#-----------------------------------------------------------------------

# a macro to deal with system libraries, implemented as a macro

# simply to improve readability of the main script

diff -Naur cmake-3.13.0-vanilla/Source/kwsys/CMakeLists.txt cmake-3.13.0/Source/kwsys/CMakeLists.txt

--- cmake-3.13.0-vanilla/Source/kwsys/CMakeLists.txt 2018-11-20 15:49:09.000000000 +0100

+++ cmake-3.13.0/Source/kwsys/CMakeLists.txt 2018-11-24 22:53:26.555712768 +0100

@@ -90,6 +90,7 @@

CMP0048 # CMake 3.0, Let the project command manage version variables.

CMP0056 # CMake 3.2, Honor link flags in try\_compile() source-file signature.

CMP0063 # CMake 3.3, Honor visibility properties for all target types.

+ CMP0069 # CMake 3.9, INTERPROCEDURAL\_OPTIMIZATION is enforced when enabled.

)

IF(POLICY ${p})

CMAKE\_POLICY(SET ${p} NEW)

diff -Naur cmake-3.13.0-vanilla/Utilities/cmcurl/CMakeLists.txt cmake-3.13.0/Utilities/cmcurl/CMakeLists.txt

--- cmake-3.13.0-vanilla/Utilities/cmcurl/CMakeLists.txt 2018-11-20 15:49:11.000000000 +0100

+++ cmake-3.13.0/Utilities/cmcurl/CMakeLists.txt 2018-11-24 22:57:54.247704229 +0100

@@ -132,6 +132,8 @@

project(CURL C)

+cmake\_policy(SET CMP0069 NEW)

+

if(0) # This code not needed for building within CMake.

message(WARNING "the curl cmake build system is poorly maintained. Be aware")

endif()

Now the build script looks like this:

#!/bin/bash

source $HOME/qnx700/qnxsdp-env.sh

LIBS="`pwd`/qt5/plugins/platforms/libqqnx.a"

cmake -B builddir -S cmake-3.13.0 -G Ninja \

-DCMAKE\_BUILD\_TYPE=Release \

-DCMAKE\_TOOLCHAIN\_FILE=`pwd`/qnx7\_gcc.cmake \

-DCMAKE\_INSTALL\_PREFIX=`pwd`/installdir \

-DBUILD\_QtDialog=ON \

-DCMake\_QT\_STATIC\_QQnxIntegrationPlugin\_LIBRARIES="$LIBS" \

-DCMAKE\_PREFIX\_PATH=`pwd`/qt5/lib/cmake \

-DCMake\_BUILD\_LTO=ON

cmake --build builddir --target install/strip

The binary sizes are:

$ du -ah installdir/bin/

5.5M installdir/bin/cmake

21M installdir/bin/cmake-gui

5.3M installdir/bin/cpack

6.2M installdir/bin/ctest

37M installdir/bin/

That’s like **~16%** binary size decrease!

**Fancy Debug Build Flags**

In the article [Improving C++ Builds with Split DWARF](http://www.productive-cpp.com/improving-cpp-builds-with-split-dwarf/) we learn about -gsplit-dwarf compilation flag which speeds up compilation times in Debug mode.

Unfortunately QCC compiler wrapper doesn’t forward this flag to GCC , fortunately my toolchain file makes this possible, since we’re using directly the GCC compiler!

Let’s build a normal Debug build with this script:

#!/bin/bash

source $HOME/qnx700/qnxsdp-env.sh

LIBS="`pwd`/qt5/plugins/platforms/libqqnx.a"

cmake -B builddir -S cmake-3.13.0 -G Ninja \

-DCMAKE\_BUILD\_TYPE=Debug \

-DCMAKE\_TOOLCHAIN\_FILE=`pwd`/qnx7\_gcc.cmake \

-DCMAKE\_INSTALL\_PREFIX=`pwd`/installdir \

-DBUILD\_QtDialog=ON \

-DCMake\_QT\_STATIC\_QQnxIntegrationPlugin\_LIBRARIES="$LIBS" \

-DCMAKE\_PREFIX\_PATH=`pwd`/qt5/lib/cmake

cmake --build builddir --target install

The builddir and installdir sizes were:

$ du -sh builddir

2.6G builddir

$ du -sh installdir

686M installdir

Now let’s enable all the fancy debug build flags:

#!/bin/bash

source $HOME/qnx700/qnxsdp-env.sh

export CFLAGS="-gsplit-dwarf -fuse-ld=gold"

export CXXFLAGS=$CFLAGS

export LDFLAGS=-Wl,--gdb-index

LIBS="`pwd`/qt5/plugins/platforms/libqqnx.a"

cmake -B builddir -S cmake-3.13.0 -G Ninja \

-DCMAKE\_BUILD\_TYPE=Debug \

-DCMAKE\_TOOLCHAIN\_FILE=`pwd`/qnx7\_gcc.cmake \

-DCMAKE\_INSTALL\_PREFIX=`pwd`/installdir \

-DBUILD\_QtDialog=ON \

-DCMake\_QT\_STATIC\_QQnxIntegrationPlugin\_LIBRARIES="$LIBS" \

-DCMAKE\_PREFIX\_PATH=`pwd`/qt5/lib/cmake

cmake --build builddir --target install

I searched after the presence of dwo files. This works starting with QNX 7.0!

The builddir and installdir sizes have become:

$ find builddir -type f -name "\*.dwo" | wc

826 826 62932

$ du -sh builddir

1.3G builddir

$ du -sh installdir

187M installdir

That’s like **50%** size reduction for builddir, and **72%** size reduction for installdir!

**Need for speed**

Remember how QCC is a wrapper for GCC, when coupled with ccache you should make sure that you have direct hits and not preprocessed ones!

This can lead to 30% speed degradation, depending on your QNX toolchain and ccache usage.

This version of the build script enables ccache, and since we use GCC directly we should mostly get direct hits:

#!/bin/bash

source $HOME/qnx700/qnxsdp-env.sh

LIBS="`pwd`/qt5/plugins/platforms/libqqnx.a"

cmake -B builddir -S cmake-3.13.0 -G Ninja \

-DCMAKE\_BUILD\_TYPE=Release \

-DCMAKE\_TOOLCHAIN\_FILE=`pwd`/qnx7\_gcc.cmake \

-DCMAKE\_INSTALL\_PREFIX=`pwd`/installdir \

-DCMAKE\_CXX\_COMPILER\_LAUNCHER=ccache \

-DCMAKE\_C\_COMPILER\_LAUNCHER=ccache \

-DBUILD\_QtDialog=ON \

-DCMake\_QT\_STATIC\_QQnxIntegrationPlugin\_LIBRARIES="$LIBS" \

-DCMAKE\_PREFIX\_PATH=`pwd`/qt5/lib/cmake

cmake --build builddir --target install/strip

The statistics were:

$ ccache -s

cache directory /home/cadam/.ccache

primary config /home/cadam/.ccache/ccache.conf

secondary config (readonly) /etc/ccache.conf

stats zero time Sun Dec 2 00:05:30 2018

cache hit (direct) 820

cache hit (preprocessed) 4

cache miss 0

cache hit rate 100.00 %

cleanups performed 0

files in cache 21883

cache size 671.5 MB

max cache size 50.0 GB

I don’t know why I had 4 preprocessed cache hits 

**Qt Creator Integration**

Qt Creator has had some problems with the QNX / CMake integration. For example these bugs:

* [QTCREATORBUG-20386: QtCreator doesn’t not allow selection of the QNX C compiler (qcc)](https://bugreports.qt.io/browse/QTCREATORBUG-20386)
* [QTCREATORBUG-20392: CMake: Qt Creator is unable to configure and build a project with CMAKE\_TOOLCHAIN\_FILE](https://bugreports.qt.io/browse/QTCREATORBUG-20392)

The good news is that with the GCC toolchain file these bugs are no longer reproduceable! 

**Devil is in the details**

If we have a closer look at what CMake compiler detection cmake file (builddir/CMakeFiles/3.13.0/CMakeCXXCompiler.cmake) contains for the CMAKE\_CXX\_IMPLICIT\_LINK\_LIBRARIES, we can see that there is a difference between the original QNX toolchain file and my own. It’s mainly about libgcc.a.

Luckily CMake can be configured to adjust to this, and my toolchain file is a bit more complicated 

set(CMAKE\_SYSTEM\_NAME QNX)

set(arch ntox86\_64)

set(QNX\_PROCESSOR x86\_64)

set(CMAKE\_C\_COMPILER $ENV{QNX\_HOST}/usr/bin/${arch}-gcc)

set(CMAKE\_C\_COMPILER\_TARGET ${arch})

set(CMAKE\_CXX\_COMPILER $ENV{QNX\_HOST}/usr/bin/${arch}-g++)

set(CMAKE\_CXX\_COMPILER\_TARGET ${arch})

file(GLOB\_RECURSE libgcc\_a

"$ENV{QNX\_HOST}/usr/lib/gcc/${QNX\_PROCESSOR}\*/\*/pic/libgcc.a")

set(CMAKE\_C\_STANDARD\_LIBRARIES\_INIT

"${libgcc\_a} -lc -Bstatic -lcS ${libgcc\_a}")

set(CMAKE\_CXX\_STANDARD\_LIBRARIES\_INIT

"-lc++ -lm ${CMAKE\_C\_STANDARD\_LIBRARIES\_INIT}")

set(CMAKE\_EXE\_LINKER\_FLAGS\_INIT "-nodefaultlibs")

set(CMAKE\_SHARED\_LINKER\_FLAGS\_INIT "-nodefaultlibs")

set(CMAKE\_MODULE\_LINKER\_FLAGS\_INIT "-nodefaultlibs")

If you want an ARM 64 version, just change these two lines:

set(arch ntoaarch64)

set(QNX\_PROCESSOR aarch64)

I hope you have enjoyed this C++ compilation ride in the world of the exotic operating system that is QNX!

Posted by Cristian Adam Dec 02, 2018 [c++](https://cristianadam.eu/categories/c-plus-plus/), [cmake](https://cristianadam.eu/categories/cmake/)

[« Speeding up CMake](https://cristianadam.eu/20170709/speeding-up-cmake/) [Modifying the default CMake build types »](https://cristianadam.eu/20190223/modifying-the-default-cmake-build-types/)

**Comments**

**Recent Posts**

* [Speeding up C++ GitHub Actions using ccache](https://cristianadam.eu/20200113/speeding-up-c-plus-plus-github-actions-using-ccache/)
* [Using GitHub Actions with C++ and CMake](https://cristianadam.eu/20191222/using-github-actions-with-c-plus-plus-and-cmake/)
* [Building multiple configurations with CMake in one go!](https://cristianadam.eu/20191012/building-multiple-configurations-with-cmake-in-one-go/)
* [Bundling together static libraries with CMake](https://cristianadam.eu/20190501/bundling-together-static-libraries-with-cmake/)
* [Speeding up libclang on Windows](https://cristianadam.eu/20190318/speeding-up-libclang-on-windows/)
* [Modifying the default CMake build types](https://cristianadam.eu/20190223/modifying-the-default-cmake-build-types/)
* [A Better QNX CMake Toolchain File](https://cristianadam.eu/20181202/a-better-qnx-cmake-toolchain-file/)
* [Speeding up CMake](https://cristianadam.eu/20170709/speeding-up-cmake/)
* [Qt Creator, Ubuntu, and VirtualBox](https://cristianadam.eu/20170321/qt-creator-ubuntu-virtualbox/)
* [NullPointerException in C++](https://cristianadam.eu/20160914/nullpointerexception-in-c-plus-plus/)
* [C++ I/O Benchmark](https://cristianadam.eu/20160410/c-plus-plus-i-slash-o-benchmark/)
* [Introducing C++ experimental io2d](https://cristianadam.eu/20160228/introducing-c-plus-plus-experimental-io2d/)
* [Speeding up libclang on Windows](https://cristianadam.eu/20160104/speeding-up-libclang-on-windows/)
* [QtCreator and Google Test](https://cristianadam.eu/20151213/qtcreator-and-google-test/)
* [Total Commander and SFTP](https://cristianadam.eu/20150927/total-commander-and-sftp/)
* [Octopress on Windows](https://cristianadam.eu/20150719/octopress-on-windows/)
* [Grim Fandango Remastered](https://cristianadam.eu/20150514/grim-fandango-remastered/)
* [From Blogger to Octopress](https://cristianadam.eu/20150510/from-blogger-to-octopress/)

**Tip Jar**

If you like this blog, [leave a tip!](https://cristianadam.eu/tip-jar/index.html)

Copyright © 2020 - Cristian Adam - Powered by [Octopress](http://octopress.org) | This work is licensed under [CC BY 4.0](http://creativecommons.org/licenses/by/4.0/)