Nick Cullen : Dev Blog

All things programming

[Raspberry Pi C ++] Using CMake

Previous Post: Setting Up The Compiler

Next Post: Compiling and Linking WiringPi for GPIO Access

In a previous post I talked about setting up a compiler on the Raspberry Pi and using g++ on the command line. This is all well and good as a small example with one file, but when you (rightly so) have your code broken up into many files, compiling them via the command line is no longer viable.

To get around this problem, programmers using GCC would create files known as

"makefiles".

Makefiles

Makefiles are GCCs way of passing to the compiler what source files need to be compiled and what they should be compiled into. A downside to using makefiles is that the syntax for them can get extremely cryptic and hard to read/understand when your code base grows.

When you start creating makefiles, you are creating what is known as a "**build system**" for your project. The more code files you have to manage, the more fundamental your build system becomes.

Different platforms/compilers have different ways of managing build systems for C++ projects. For example, the GCC default is with makefiles whereas Windows uses its own Visual Studio project solution files to manage build types. Technically, there are Windows ports of GCC compilers included in software such as MinGW or Cygwin but this isn't the norm with regards to Windows programming.

This means that if you want to compile your code on multiple platforms you will have to create and maintain multiple build systems for your project.

For these reasons, programmers tend to look elsewhere for managing their build systems. One such tool is CMake

CMake

Note: I just want to make it clear here that CMake does **not** compile your code! CMake is one level higher than a native build system; infact, it abstracts the underlying build system completely with it's own syntax and using what it calls "**Generators**".

For example, you create your CMake files (described later) as you would your makefiles / Visual Studio files, but you will use a **CMake generator** to *generate* your build system files which can range from Makefiles to Visual Stu-

dio and Xcode projects. **You will then use this generated build system to *compile* your code.** Here is a list of CMake generators readily available.

It's best to show the process with an example.

CMake Example on Raspberry Pi C ++

Installing CMake

Installing CMake on your Raspberry Pi is made easy by using the apt package manager.

First thing you want to do is SSH into your Pi and at the terminal/command line enter the following

```
> sudo apt-get install cmake
```

After a few seconds you should be able to enter the "cmake" command in the terminal and you will see a list of generators available.

Using CMake

For CMake to work, there should be a file named "CMakeLists.txt" in the root directory of your C++ project. In this example, I will have one CMakeLists.txt file and one Hello.cpp file from the previous tutorial.

Your folder structure should look like so:

```
- Root Folder/
| - CMakeLists.txt
| - Hello.cpp
```

To create the files enter the following command


```
> touch <filename.ext>
```

The contents for each file is as follows (notice each CMake command has a comment describing what it is doing)

```cpp
#include <stdio.h>

int main(int argc, char** argv)
{
        printf("Hello, World!\n");
        return 0;
}
```

Hello.cpp

```
# Minimum CMake version required to generate
# our build system
cmake_minimum_required(VERSION 3.0)

# Name of our Project
project(hello)

# add_executable creates an executable with
# The given name. In our case it is "Hello"
# Source files are given as parameters. In our
# Case we only have one source file Hello.cpp
add_executable(Hello Hello.cpp)
```

CMakeLists.txt

Generating Our Makefiles

Now that we have our CMake file setup with our single source file Hello.cpp, we next need to *generate* the build system files. CMake will generate quite a lot of files

and folders, so it is best not to run the CMake command in the project root directory. Instead we will create a sub folder called “Build” and run the generation in here.

```
> mkdir Build && cd Build
```

This will make the directory called “Build” and also change into the directory. This is how your folder structure should look now:

```
- Root Folder/
| - CMakeLists.txt
| - Hello.cpp
| - Build/
```

To run the CMake Generator we need to use the “cmake” command and tell it where the root CMakeLists.txt file is. As we are in a sub directory we will use the parent directory alias “..”. If no generator is specified it will default to the platforms native build system (in the Pi’s case, this will be GCC makefiles)

```
> cmake ..
```

Once complete you can list the contents of the directory (using the “ls” command) and notice there is now a “MakeFile” present!

Compiling the Code

This bit is easy now that CMake has generated our MakeFile for us!

Simply enter the following into the terminal (make sure you are still in the Build directory)

```
> make
```

Provided you copied the code as is above, you should see the line “Built target

Hello”. To execute the application enter the following:

```
./Hello
```

You should see the line “Hello, World!”

Previous Post: Setting Up The Compiler

Next Post: Compiling and Linking WiringPi for GPIO Access

[Raspberry Pi C ++] GPIO Access : Compiling, Linking & Using WiringPi
18th July 2017
In "Raspberry Pi Tutorials"

[Raspberry Pi C ++] Setting Up A Compiler (g++)
11th July 2017
In "Raspberry Pi Tutorials"

[Raspberry Pi C ++] The Beginnings, A Very Brief Overview
8th July 2017
In "Raspberry Pi Tutorials"

PUBLISHED BY

Nick
Nick Cullen is a software developer living in South Wales, UK. He is primarily focused around coding in C++ and C# and loves tinkering with new programming languages and technologies. A key technological interest of his is Raspberry Pi development, which he has helped pioneer a unique product commercially using a Pi and programming the software in C++. View all posts by Nick →

15th July 2017 Nick Raspberry Pi Tutorials C++, cmake, raspberry pi, tutorial