



## Cmake Build Framework – Part 2

# C++



Algorithmic Trading & Quant Research Hub



## C++ Set-Up for Algo Quant Trading

By Nicholas Burgess

## C++ Set-Up for Algo Quant Trading



### ➤ Part 1 – Visual Studio for Windows

- Online C++ Emulators & Code Snippets
- Visual Studio Projects & Solutions
- C++ Building, Compilation & Linking

### ➤ Part 2 – CMake for Cross-platform Builds

- The CMake Build System
- How to use CMake
- Build Environments & Compilers



### Example: Visual Studio & CMake

<https://github.com/nburgessx/QuantResearch/tree/main/CMake%20Examples>

Algorithmic Trading & Quant Research Hub



# C++



Algorithmic Trading & Quant Research Hub



CMake Build Framework – Part 2





# CMake Cross-Platform Build System

## CMake – What it is and what it does

- A **cross-platform** build system – not a compiler
- It uses platform-independent configuration files, **CMakeLists.txt**
- Generates native build files e.g. Visual Studio Solutions, Linux Make files, Ninja files, macOS Xcode projects
- Available as part of Visual Studio, see Tools -> Command line -> Developer Command Prompt

## How to generate a solution File for Visual Studio?

- Create the necessary CMakeLists.txt files
- Open Visual Studio command line and type:

```
cmake -G "Visual Studio 17 2022" <path-to-project-root>
```



# CMake Config Files – CMakeLists.txt

## CMake Commands for CMakeLists.txt

### ➤ Solution Config File

- Name the solution file ([project](#))
- Specify what projects to include ([add\\_subdirectory](#))

### ➤ Project Config Files

- Name the project ([project](#))
- Specify the project type and list the headers and source files to include ([add\\_library](#) | [add\\_executable](#))
- Provide the path to the include folder(s) and header files ([target\\_include\\_directories](#))
- List any dependency projects to include ([target\\_link\\_libraries](#))



# Example: Create Solution File

- Consider a simple C++ maths library where the main project is called **Analytics** that depends on two projects named **Addition** and **Subtraction**. The folder structure looks as follows,

## MathLibrary (Root Folder)

CMakeLists.txt

## Analytics

CMakeLists.txt | Main.cpp

## Addition

CMakeLists.txt | Add.h | Add.cpp

## Subtraction

CMakeLists.txt | Subtract.h | Subtract.cpp

- The solution root folder and each project folder requires a **CMakeLists.txt** config file
- The config file defines the **project type** and specifies the **include paths** and **project dependencies**

C++

C++

C++

C++

# Solution Config File, CMakeLists.txt

```
1 cmake_minimum_required(VERSION 3.20)
2
3 project(MathLibrary LANGUAGES CXX)
4
5 # ---- Language standard ----
6 set(CMAKE_CXX_STANDARD 17)
7 set(CMAKE_CXX_STANDARD_REQUIRED ON)
8
9 # ---- Targets ----
10 add_subdirectory(Addition)
11 add_subdirectory(Subtraction)
12 add_subdirectory(Analytics)
```

- `project` – Name of the solution file
- `add_subdirectory` – List project folders to include



C++

C++

C++

C++

# Main Project Config File, CMakeLists.txt

```
1  add_executable (Analytics
2      main.cpp
3  )
4
5  target_link_libraries (Analytics
6      PRIVATE
7          Addition
8          Subtraction
9  )
```

## ➤ `add_executable`

- Creates project that outputs an executable called Analytics.
- List all the .h and .cpp files to include.

## ➤ `target_link_libraries`

- List the project name then the dependency projects to include
- Here we add the addition and subtraction projects to the analytics project



# Dependency Project Config File, CMakeLists.txt

```
1  add_library(Addition STATIC
2      add.h
3      add.cpp
4  )
5
6  target_include_directories(Addition
7      PUBLIC
8          $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}>
9  )
```

## ➤ add\_library

- Creates a project named Addition. Use **STATIC** to generate a .lib and **SHARED** to generate a .dll
- List all the .h and .cpp files to include.

## ➤ target\_include\_directories

- List the include directories for the Addition project
- **`$(CMAKE_CURRENT_SOURCE_DIR)`** means use the current folder

C++

C++

C++

C++

# Generating the Visual Studio Solution File

How to generate the Visual Studio solution and project files?

- After creating the necessary CMakeLists.txt configuration files
- Open Visual Studio command prompt and navigate to the solution root folder
- Type **mkdir build** to create a folder called 'build'
- Navigate to the build folder **cd build**

```
cmake -G "Visual Studio 17 2022" <path-to-project-root>
```

- To generate the solution file type: **cmake -G "Visual Studio 17 2022" ..**
- Note ".." means the root project is up one folder level

How to generate the native build projects for Linux, macOS and other platforms and compilers?

- Change the name of the compiler from **"Visual Studio 17 2022"** to the compiler of your choice
- Examples: For Linux **"Unix Makefiles"** or **"Ninja"** and for macOS use **"Xcode"**

# C++



Algorithmic Trading & Quant Research Hub



CMake Resources



# Getting Started with CMake

## Professional C++ with CMake

- Outlines how professional Quants use CMake
- Includes canonical stylized working examples
- Intentionally simple and easy to follow

AlgoQuantHub Weekly Deep Dive

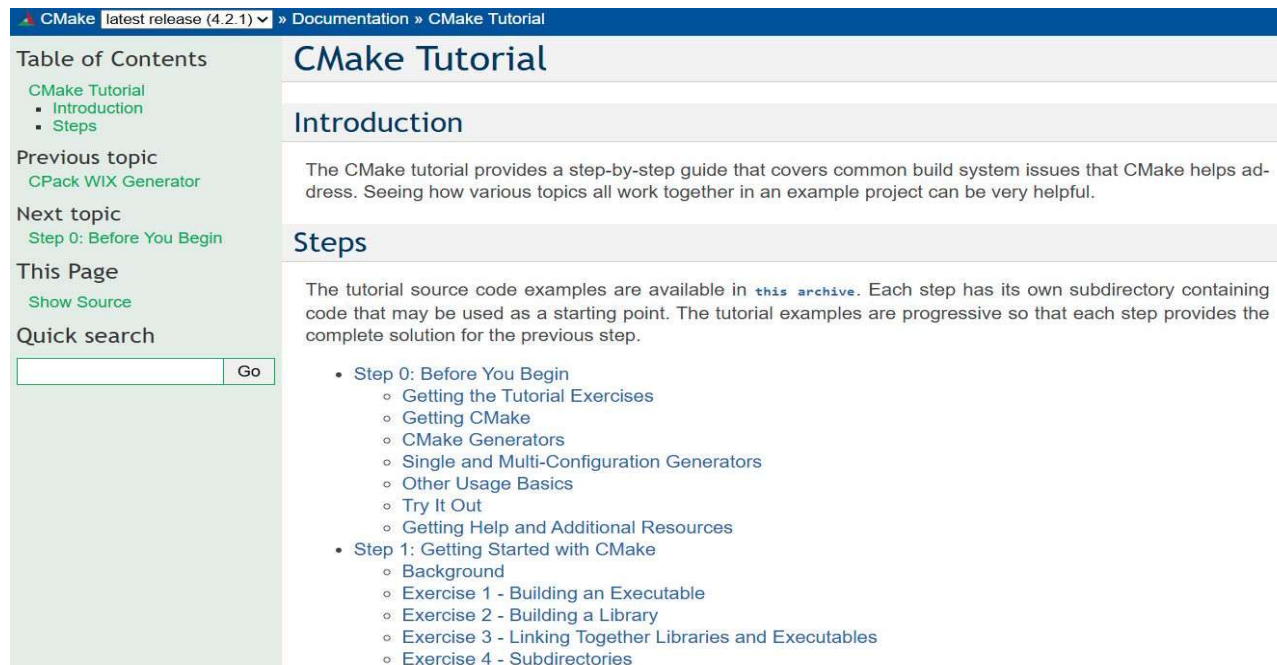


**Professional C++ with CMake for  
Quants & Algo Trading**

Link: <https://algoquanthub.beehiiv.com/p/professional-c-with-cmake-for-quants-algo-trading>

Examples: <https://github.com/nburgessx/QuantResearch/tree/main/CMake%20Examples>

# CMake Tutorial – cmake.org



The screenshot shows the CMake Tutorial page on the cmake.org website. The page has a blue header with the CMake logo and the text "latest release (4.2.1)". The main content area is titled "CMake Tutorial" and "Introduction". The left sidebar contains a "Table of Contents" with links to "CMake Tutorial", "Introduction", and "Steps". Below this are links for "Previous topic" (CPack WIX Generator), "Next topic" (Step 0: Before You Begin), "This Page" (Show Source), and a "Quick search" box. The main content area includes an "Introduction" paragraph and a "Steps" section with a list of links for Step 0 and Step 1.

CMake latest release (4.2.1) » Documentation » CMake Tutorial

## CMake Tutorial

### Introduction

The CMake tutorial provides a step-by-step guide that covers common build system issues that CMake helps address. Seeing how various topics all work together in an example project can be very helpful.

### Steps

The tutorial source code examples are available in [this archive](#). Each step has its own subdirectory containing code that may be used as a starting point. The tutorial examples are progressive so that each step provides the complete solution for the previous step.

- Step 0: Before You Begin
  - Getting the Tutorial Exercises
  - Getting CMake
  - CMake Generators
  - Single and Multi-Configuration Generators
  - Other Usage Basics
  - Try It Out
  - Getting Help and Additional Resources
- Step 1: Getting Started with CMake
  - Background
  - Exercise 1 - Building an Executable
  - Exercise 2 - Building a Library
  - Exercise 3 - Linking Together Libraries and Executables
  - Exercise 4 - Subdirectories

## ➤ CMake Tutorial – cmake.org

- Provides a step-by-step guides and tutorials on how to use CMake
- Link: <https://cmake.org/cmake/help/book/mastering-cmake/cmake/Help/guide/tutorial/index.html>



C++

C++

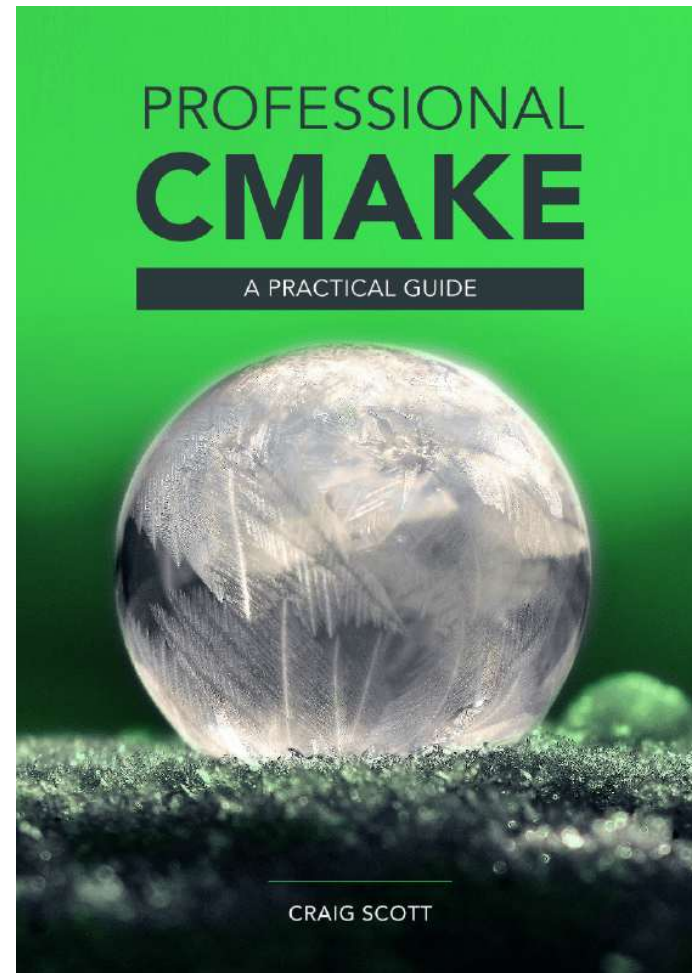
C++

C++

# Professional CMake

## ➤ Professional CMake – A Practical Guide

- Free Book
- By Craig Scott
- <https://crascit.com/professional-cmake/>



# C++



Algorithmic Trading & Quant Research Hub



[More Info ...](#)







Subscribe to my Quant Newsletter

<https://algoquanthub.beehiiv.com/subscribe>

Subscribe to my Quant YouTube Channel

<https://www.youtube.com/@algoquanthub>

Algo Trading & Quant Store

<https://payhip.com/AlgoQuantHub>

Follow me on LinkedIn

<https://www.linkedin.com/in/nburgessx>



Have questions or want further info?

## Contact

LinkedIn: [www.linkedin.com/in/nburgessx](https://www.linkedin.com/in/nburgessx)