**Get the latest gcc**

Modern GCC can be chased by a single PPA.  
By the way, you can also get the standard library for C++20 automatically by dependency.

sudo add-apt-repository ppa:ubuntu-toolchain-r/test

sudo apt update

sudo apt install -y g++-9

The version of g++ to be installed here is better to keep it up to date at that time. I am writing from now on on on the premise of g++-9, but please read it as appropriate

**gcc configuration**

Define the environment variables CXX and CC to make g++-9 easier to use by default.

echo "export CXX='g++-9'" >> ~/.bashrc

echo "export CC='gcc-9'" >> ~/.bashrc

source ~/.bashrc

### gcc installation verification method

g++-9 --version Something to be displayed at: If it fails, it says the command cannot be found.

### cmake

<https://cmake.org/install/> was helpful.

First, download and unzip CMake's source **Unix/Linux Source (has\n line feeds)** from [the https://cmake.org/download/](https://cmake.org/download/). Please change the part related to the CMake Latest version as appropriate with reference to [https://cmake.org/download/.](https://cmake.org/download/)

$ wget https://github.com/Kitware/CMake/releases/download/v3.17.1/cmake-3.17.1.tar.gz

$ tar zxvf cmake-3.17.1.tar.gz

Next, build the downloaded source.

$ cd cmake-3.17.1/

$ ./bootstrap

$ make

$ sudo make install

# Passing CMake

$ echo 'export PATH=$HOME/cmake-3.17.1/bin/:$PATH' >> ~/.bashrc

$ source ~/.bashrc

The path variable is wherever you put it.

# Checking CMake Version

$ cmake --version

cmake version 3.17.1

CMake suite maintained and supported by Kitware (kitware.com/cmake).

Examine the code for CMake's FindBoostModule that you downloaded earlier

# cd cmake

grep -n -A 2 \_Boost\_KNOWN\_VERSIONS Modules/FindBoost.cmake

As a result, for example, the following result will come back.

> grep -n -A 2 \_Boost\_KNOWN\_VERSIONS Modules/FindBoost.cmake

1431: set(\_Boost\_KNOWN\_VERSIONS ${Boost\_ADDITIONAL\_VERSIONS}

1432- "1.72.0" "1.72" "1.71.0" "1.71" "1.70.0" "1.70" "1.69.0" "1.69"

1433- "1.68.0" "1.68" "1.67.0" "1.67" "1.66.0" "1.66" "1.65.1" "1.65.0" "1.65"

--

1447: foreach(version ${\_Boost\_KNOWN\_VERSIONS})

1448- if(NOT "${version}" VERSION\_LESS "${Boost\_FIND\_VERSION}")

1449- # This version is high enough.

--

1459: set(\_boost\_TEST\_VERSIONS "${\_Boost\_KNOWN\_VERSIONS}")

1460- endif()

1461-endif()

Usually, the first version is the highest corresponding version. For this reason, we will use this version in the following explanations, but please read it as appropriate.1.72.0

### Download and install Boost

Boost is compiled from the Git repository and used, and if I may say so first,  
it takes a lot of time to download and compile.

git clone https://github.com/boostorg/boost.git

Unlike other software, the version of Boost is not always up to date, switch to the latest version of CMake-compatible that we investigated earlier.

cd boost

git checkout boost-1.72.0 # Please replace with your latest version

git submodule update --init

./bootstrap.sh

./b2 toolset=gcc-9 --prefix=/usr/local -j5

sudo ./b2 install toolset=gcc-9 --prefix=/usr/local -j5

At compile time it is specified and handled by the latest compiler  
CXX or CC does not apply, so it must be applied explicitly There are a lot of warnings in the middle,  
but std::auto\_ptr and unused value are just making a fuss. Don't bother with Boost as it comes with code for every versiontoolset=gcc-9

Argument 5 of the j option is the number of parallel operations. If you make the number of logical cores of the CPU +1 and do no other work at all, it will compile with all your might. Since the speed is different, it is good to properly investigate and specify the number of logical cores.

It is mandatory to enter in the middle. If you do, update using the git submodule appropriately, or delete the boost directory and redo this procedure.git submodule

## Operation check

Create a directory of your choice and try compilation.  
First, create CMakeLists.txt and main .cpp in the directory.

- CMakeLists.txt

- main.cpp

CMakeLists.txt

cmake\_minimum\_required(VERSION 3.16)

project(test)

find\_package(Boost REQUIRED COMPONENTS system)

find\_package(Threads REQUIRED)

include\_directories(Boost::boost)

add\_executable(main main.cpp)

target\_link\_libraries(main Boost::system Threads::Threads)

set(TARGETS main)

set\_property(TARGET ${TARGETS} PROPERTY CXX\_STANDARD 20)

set\_property(TARGET ${TARGETS} PROPERTY CXX\_STANDARD\_REQUIRED ON)

set\_property(TARGET ${TARGETS} PROPERTY CXX\_EXTENSION OFF)

main.cpp

#include <iostream>

#include <string\_view>

#include <boost/asio.hpp>

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

{

namespace asio = boost::asio;

using asio::ip::tcp;

asio::io\_service io\_service;

tcp::socket socket(io\_service);

boost::system::error\_code error;

socket.connect(tcp::endpoint(asio::ip::address::from\_string("127.0.0.1"), 31400), error);

if (error)

std::cout << "connect failed : " << error.message() << std::endl;

else

std::cout << "connected" << std::endl;

}

Once you have the file ready, compile it with cmake.

cmake .

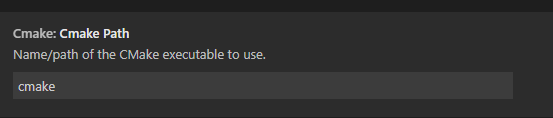
make

If you can compile it safely, all installation and configuration are finished.

#include <string\_view>If you spit out an error, the g++-9 setting is wrong.

If Cmake finds the boost library, but is spitting warn, your CMake is not compatible with the latest Boost. Let's deal with it by lowering the version of Boost and reinstalling It  
is also possible that the CMake you used was not the latest one. Check withcmake –version

1. resetting the extension (CTRL+SHIFT+P then "CMake: Reset CMake Tools Extension State (For troubleshooting)"
2. checking that cmake is in my Path
3. just putting "cmake" (without the brackets) in the extension settings GUI

[](https://user-images.githubusercontent.com/16573776/106479355-8a279580-64aa-11eb-903a-cf60e4111744.png)

## Operation check

Portable code library POCO is an extension to boost and has been used for some of the functionality.

I’m not sure how much extra memory it takes and if we should remove it, it was preferred for portability and but it may not be necessary.

Download the latest tar file from poco site currently poco-1.12.2.tar

Then follow the instructions below

./configure – sets it to Linux

1. create a cmake-build directory (e.g. in the POCO root directory):

$ mkdir cmake-build

2. and run CMake from there:

$ cd cmake-build

$ cmake ..

$ make -s -j

The full length version also needs this

sudo apt install openssl-dev libssl-dev libapr1-dev libaprutil1-dev apache2-dev libmysqlclient-dev libpq-dev unixodbc-dev

I have only used the cutdown version this didn’t seem to work

So I repeated this with the version **poco-1.10.1-all**

This is the guidelines listed

$ gunzip poco-X.Y.tar.gz

$ tar -xf poco-X.Y.tar

$ cd poco-X.Y

$ ./configure 🡨-- sets it to Linux when Ran (important I think)

$ gmake -s or make

But I again found it didn’t work right when it compiled.

So I did this I

$ mkdir cmake-build

$ cmake ..

$ make

$ LD\_LIBRARY\_PATH=/usr/local/lib/

$ export LD\_LIBRARY\_PATH

$ echo $LD\_LIBRARY\_PATH

To install

Type at the prompt

$> sudo make install

add the following to ~/.bashrc to make your changes permanent

by typing

$>vi ~/.bashrc

export LD\_LIBRARY\_PATH=/usr/local/lib/

If you don’t have this right this will not work correctly so run it to test the pathe variable.

/usr/local/bin/cpspc

This shows the libraries you installed

anthony@Ubunt-18-dev1:~/poco/poco-1.12.2/cmake-build/poco\_test$ ls /usr/local/lib/libPocoF\*

/usr/local/lib/libPocoFoundationd.so

/usr/local/lib/libPocoFoundationd.so.71

/usr/local/lib/libPocoFoundation.so

/usr/local/lib/libPocoFoundation.so.71

/usr/local/lib/libPocoFoundation.so.92

To test this use this code to check the running of the library do the following.

$ mkdir poco\_test\_project

$ cd poco\_test\_project

$ vi sample.cpp

#include <iostream>

#include <Poco/RegularExpression.h>

int main() {

Poco::RegularExpression regexp("^[a-zA-Z]+");

std::string buf;

regexp.extract("ABC123", buf);

std::cout << buf << std::endl; //=> ABC

return 0;

}

-----------------------------------------------------------------------------

$ vi CMakeLists.txt

cmake\_minimum\_required (VERSION 3.17)

project (event\_demo)

enable\_language(C)

enable\_language(CXX)

# Always include the source and build directories in the include path.

set(CMAKE\_INCLUDE\_CURRENT\_DIR ON)

# Set the output folder where your program will be created

set(CMAKE\_BINARY\_DIR ${CMAKE\_SOURCE\_DIR}/bin)

set(EXECUTABLE\_OUTPUT\_PATH ${CMAKE\_BINARY\_DIR})

set(LIBRARY\_OUTPUT\_PATH ${CMAKE\_BINARY\_DIR})

# Find Poco package 1.8.1

find\_package(Poco REQUIRED COMPONENTS Foundation Util Net XML JSON)

# no Poco\_INCLUDE\_DIRS, we have to set by hand

if(MSVC) # WIN32

SET(Poco\_INCLUDE\_DIRS "C:/Program Files/Poco/include")

else()

SET(Poco\_INCLUDE\_DIRS "/usr/local/include/Poco")

endif(MSVC)

MESSAGE( [Main] " Poco\_INCLUDE\_DIRS = ${Poco\_INCLUDE\_DIRS}")

MESSAGE( [Main] " Poco\_LIBRARIES = ${Poco\_LIBRARIES}")

# The following folder will be included

include\_directories(

${MY\_SRC\_INCLUDE}

${Poco\_INCLUDE\_DIRS}

)

link\_directories(${CMAKE\_BINARY\_DIR})

add\_executable(sample sample.cpp)

target\_link\_libraries(sample ${Poco\_LIBRARIES})

---------------------------------------------------------------------------

$ cmake .

$ make

$ cd bin

$ ./sample

ABC

g++-8 -I/usr/local/include -L/usr/local/lib -lPocoFoundation -lPocoXML -lPocoUtil -std=c++17 poco\_log3.cpp

the examples have been put into here and compiled okay using the above cmake file template.

/home/anthony/poco/poco-1.12.2/cmake-build/poco\_examples

The camera library for version 1.05 x86 linux has been loaded and modified to be @

/home/anthony/latest/Airobot-Dynamics-UK/DronePayloadManager/libs/SonySDK/build

**How to install UHubCtrl to manage the usb power to the sony camera.**

cd /home/anthony

open browser and go to url

<https://github.com/mvp/uhubctl>

then follow the instructions as per the webpage

while in uhubctl directory

vi reset\_usb.py and insert the follwoing code press i <paste> then :w :q

# simple python to reset the usb power when requested by the poco lib example

import time

import os

p = os.popen('sudo /usr/sbin/uhubctl -l 1-1 -a 0')

time.sleep(5)

p = os.popen('sudo /usr/sbin/uhubctl -l 1-1 -a 1')

This will be called from the main C program.