# Visual Studio Code Installation

Please follow the official doc at <https://code.visualstudio.com/docs/setup/windows>. When it’s done, open Visual Studio Code and install CMake Tools extension.

Note: install C/C++ extension

# Build Tools Installation

We will be using two compilers: for Beaglebone and for Windows host.

Beaglebone compiler can be downloaded at <https://gnutoolchains.com/beaglebone/>. You will need GCC 8.3.0 for current Beaglebone image. Download and install it with default settings.

We will use MSYS2 project to install Windows compiler. Download it at <https://www.msys2.org/>. Follow the installation procedure described there. Only the packages update procedure is different for new installers. It is decribed at <https://www.msys2.org/wiki/MSYS2-installation/>, but it’s really a minor difference.

Once you have MSYS2 in place, open MSYS2 MSYS command prompt and install the needed packages by issuing this command: **pacman -S rsync openssh mingw-w64-x86\_64-toolchain mingw-w64-x86\_64-cmake**. Rsync is needed to download sysroot from Beaglebone itself, and openssh will be called by rsync to perform download task. Also, it is used to upload the built executables to Beaglebone, but if you’re using some other means of uploading files to Beaglebone, you might not need openssh.

Note: Add “C:\msys64\mingw64\bin” to user environment variable as below. Please use respective installed msys64 path



Note: install ninja build to generate make files for CppUTest. Open MSYS2 MSYS command prompt and issue command as below.

**pacman -S mingw-w64-x86\_64-ninja**

It should be done because the default generator for MinGW CMake is Ninja. Alternatively, when installing CppUTest, you may specify another generator: **cmake -G ‘MinGW Makefiles’ ..**

# Downloading sysroot from Beaglebone

This step should be performed only once in the beginning and then every time the libraries change.

If there is a specific library you’d like to use for your program, you’ll need to install it to Beaglebone first. Then you’ll need to copy the libraries to your development machine: open MSYS2 MINGW64 command prompt, create an empty folder somewhere and issue the following command:

**rsync -avHAXRL --numeric-ids --info=progress2 <username>@<ip-address>:/{lib,usr} <rootfs-path>**

Here, username and IP address are those of Beaglebone Black you’re copying sysroot from, and rootfs path is the path to that empty folder you’ve created previously. Rsync will copy all libraries and header files from Beaglebone there.

NB: for correct CMake operation later on, rootfs path must not contain spaces!

After the files are copied, create **etc** subdirectory in your rootfs path and copy **/etc/ld.so.conf** file from the BBB. Edit it and instead of ***include /etc/ld.so.conf.d/\*.conf*** paste the contents of all **.conf** files inside **/etc/ld.so.conf** directory. For example, my file now looks like this:

# Multiarch support

/usr/local/lib/arm-linux-gnueabihf

/lib/arm-linux-gnueabihf

/usr/lib/arm-linux-gnueabihf

/usr/lib/arm-linux-gnueabihf/libfakeroot

# libc default configuration

/usr/local/lib

This file lists all paths where linker should look for the libraries to be linked with your program.

# CppUTest installation

Download release 3.8 at <https://cpputest.github.io/> and unzip it to some folder. Open the MSYS2 MINGW64 command prompt and head over to that folder. Please note the difference in path notation: the path “C:\some\_folder” is written in MSYS2 MINGW64 command prompt as “/c/some\_folder”. Other than that, it is based on Cygwin, and has many POSIX commands which may be familiar by Linux environment.

To build CppUTest, issue the following commands in MSYS2 MINGW64 command prompt:

**cd cpputest\_build**

**cmake ..**

**mingw32-make** (note: I have used **cmake --build** instead of this line - it calls whichever build tool is configured, be it mingw32-make, ninja or some other utility.)

When the build is done, you will find compiled libraries in src subfolder.

# Sample helloworld project

Note: Install git for windows from <https://git-scm.com/download/win>

To enable cross-compiling, you will need a toolchain file for CMake. Download it from <https://github.com/robamu-org/beaglebone-crosscompiling>, also please take a look at issues in that repository for some caveats and fixes.

After you open the project in VSCode, please adjust the settings of CMake Tools extension:

* Set generator to “MinGW Makefiles”.
* Set configure settings: add the variable named CMAKE\_MAKE\_PROGRAM to “C:/msys64/mingw64/bin/mingw32-make” or whatever location of mingw32-make utility.
* Set environment: add two environment variables – CPPUTEST\_HOME with path to the folder with CppUTest 3.8, and LINUX\_ROOTFS with path to the sysroot of the Beaglebone.

Note: settings.json should look below after adding about settings of CMake tools extension.

{

    "cmake.configureSettings": {

        "CMAKE\_MAKE\_PROGRAM" : "C:/msys64/mingw64/bin/mingw32-make.exe"

    },

    "cmake.generator": "MinGW Makefiles",

    "cmake.environment": {

        "CPPUTEST\_HOME": "C:/projects/cpputest-3.8",

        "LINUX\_ROOTFS": "C:/projects/BBBrootfs"

    },

    "cmake.cmakePath": "C:/msys64/mingw64/bin/cmake.exe"

}

* Note - Add cmake-kits.json file and add content as shown in fig1 to it. Please provide respective paths from the installed directory in your machine.

**Fig.1. Visual Studio Code with opened helloworld project.**

Note: restart vs code to take above environment changes into effect.

To build the project, please note there’s selection of CMake build type and kit in the blue toolbar at the bottom of the window. You can choose Debug, Release and Test build types, and switch kits between “SysProgs Beaglebone” for cross-compiling and “MinGW GCC 10.3.0” for host compiling. And there’s “Build” button to build the project with the chosen build type and kit.