Development Environment Configuration

CATSMAT Project – University of New England

Author: Derrick Hill

Last Modified: 30 July, 2017.

# General

The project is built on Mac OS X 10.10 using XCode 6.3.1 or above.

Alternatively, you can use CLion, in which case the build system is based on CMake.

If you are using CLion on Windows, you will need Cygwin with gcc.

# libMusicXML

## Version 1

We have previously used version 1 of libMusicXML, but this has been deprecated and removed from the system.

## Version 3

We use Verson 3 of MusicXML which is implemented in a framework called libMusicXML2. The source is contained within the build tree and you should not need to do anything except build the target. Both our XCode build system, and the CMake build system work for this.

# Boost Libraries

The project uses boost libraries, at time of writing we are on Version 1.58.0 (although I have upgraded to 1.64.0 on Windows, with no problems apparent). We use the filesystem and system libraries.

You need to download, build, and deploy the boost libraries as follows.

1. Download the latest boost libraries from <http://www.boost.org/>
2. Extract the package to some directory (call this BOOST\_DISTRO)
3. In a terminal window, change directory to $BOOST\_DISTRO and then run the following commands:

./bootstrap.sh --prefix=/usr/local/boost --with-libraries=filesystem,system

./b2 install

Note: if you do not have administrator/root access the second step will need to use the sudo command, i.e.

sudo ./b2 install

You may be prompted to enter your administrator password before installation proceeds.

At this point you have a new directory on your local filesystem called /usr/local/boost and under that you have include and lib.

## For XCode Users

Our XCode project file works with this installation of the boost libraries through the user defined Build Settings BOOST\_ROOT\_DIR, BOOST\_ROOT\_INC\_DIR, and BOOST\_ROOT\_LIB\_DIR.

I wouldn’t recommend doing this, but if you really need to install boost in any location other than /usr/local/boost, then you will need to do some things:

1. Rerun the bootstrap command described above with the new location as the –prefix argument and recompile through b2.
2. Change BOOST\_ROOT\_DIR in our XCode project to point to this different install directory (you probably don't have to change the other two settings, unless the structure of the file system installed by Boost changes)
3. In our XCode project, in the Project Navigator, click on the libraries, and in the File Inspector change the Absolute Path to point at the new location.

Don’t check your changes in – you will break everyone else who is using the default installation path.

Some information on adventures I had with boost can be found in VersionOne on Task TK-01003 - “DevEnv - Obtain code from GitHub, compile and run” and the information above is also contained within the VersionOne Task S-01024 - “Push changes for linking Boost libraries back to GitHub”. If you don’t need to read this, then don’t go there – the instructions above work.

## For WINDOWS Users

On Windows, you will be using the CMake build under CLion.

When you compile the Boost libraries, you *must* do this from a Cygwin terminal – do *not* run the Boost build from a Windows cmd prompt. If you run from Cygwin, then all the instructions and paths above will work.

On the Windows file system, the paths mentioned above will be relative to your Cygwin directory (by default you will end up with C:\cygwin64\usr\local\boost). You don’t have to worry about this – the CMake files in the CATSMAT project have the correct paths in them.

If you need to install Boost somewhere else (again, not recommended), then the following CMake files all have paths to the boost libraries in them, and you will have to modify these

* $/catsmat/catsmat/catsmat/cmake/CMakeCatsmatFiles.txt
* $/catsmat/catsmat/libIMUSANT\_Tests/cmake/CMakeLists.txt
* $/catsmat/catsmat/libIMUSANT/cmake/CMakeLists.txt

Note that I have experimented with installing the Boost libraries as part of the Cygwin install with no success. This may be my lack of understanding, and the approach may work for you, but you will probably have to modify the paths in the CMake files as described above to accommodate new install locations.

# Google Test Framework

For unit testing we are using Google Test, which is a variant of the \*Unit class of test utilities (e.g. NUnit, JUnit, etc).

Google Test is included in the build tree. You should not have to do anything except build the target.

## OLD Instruction

In the past, we asked you to install google test separately. This was a bit of a faff around and so I have included the instructions below, just in case you need to do this for any reason. The main point is that the compiler flags for the google test libraries need to match the compiler flags that you use to build your system under test.

You need to download, compile, and install the Google test framework. You obtain the download from here:

<https://code.google.com/p/googletest/>

At time of writing the latest version is 1.7.0 and that is what we are using. The instructions here are generally useful:

<https://code.google.com/p/googletest/wiki/V1_7_XcodeGuide>

The steps are as follows:

1. Download the package and extract it somewhere on your hard drive.
2. Navigate to the XCode directory within the extracted package, and open the XCode project file that you will find there.
3. Make sure that the following build settings in the gtest project match those that we use in our CATSMAT project, otherwise you will get link errors:

*Note that a these settings must be the same in both the gtest project, and also in the CATSMAT project. You should check these in both places.*

*Apple LLVM 6.1 Language C++*

C++ Standard Library = libc++

*Architectures*

Architecture = Standard Architecture (64-bit Intel)

Base SDK = Latest OS X

*Deployment*

OS X Deployment Target = OS X 10.7

1. Build the gtest.framework target. The gtest build produces gtest.framework, and you need to deploy this to the correct place on your filesystem, otherwise our code will not link against it properly.
   1. In the Project Navigator window in XCode, under the Products folder, right click the gtest.framework, and select “Show in Finder”
   2. Copy the gtest.framework directory and paste it into /Library/Frameworks (you need local admin rights to change the contents of /Library/Frameworks).

Note that /Libraries/Frameworks is the default location on OS X. I tried to put this framework somewhere else using the Build Setting "Framework Search Paths” but I couldn’t get the system to find the framework at run time (I was getting “Image not found” errors). More work might resolve this, but for the moment the benefit seems small.

From this point, any existing unit tests within our IMUSANT project should work. If you are adding new unit test directories you will need to proceed as follows:

1. Create a new folder in the project for the tests you are going to add (for example, if you are testing CATSMAT, then call this folder CATSMAT\_Tests).
2. Create a new build target for the new tests. Edit the following build settings:

* Header Search Path  
  Add /Library/Frameworks/gtest.framework/Headers
* Build Phase  
  Add gtest.framework into the “Link Binary with Libraries” Build Phase.

1. Add a main() into the unit test folder.

* Note that you could instead link against gtest\_main in which case gtest will provide the main() for you, but I haven’t done this yet. See the gtest documentation refered to above.
* Note also that the signature of your main might include a const in front of char \* argv[] folder:

int main(int argc, const char \* argv[])

When you call InitGoogleTest() this will cause a “No matching function call” error at build time. You need to remove the const from the signature of main().

# Building Using CMake under CLion

You can use CLion as an IDE on Windows or Mac if you want to. The build system for this is CMake, and appropriate CMake files are included in the source tree.

## Setting up Your Environment

### Install Cygwin – Windows Only

On Windows, Cygwin is a prerequisite for the CLion installation that we are using. It gives you your C++ compiler as well as the foundations of the build system.

Install Cygwin x64 with packages: GCC/G++, make, gdb 7.8+

These prerequisites are defined here:

<https://www.jetbrains.com/help/clion/2017.1/requirements-for-clion.html#d75987e123>

There’s a better description here, particularly with respect to the Cygwin packages that you need to install:

<https://medium.com/@RobertSimoes/installing-clion-on-windows-for-c-c-newbies-like-me-4a346aaf9557>

### Install CLion

Download CLion from <https://www.jetbrains.com/clion/> and install it.

If you need an academic license for CLion then go here to get it: <https://www.jetbrains.com/student/>. You will need an email address from a recognized educational institution (I think this just means that it ends in .edu).

CMake is bundled with CLion.

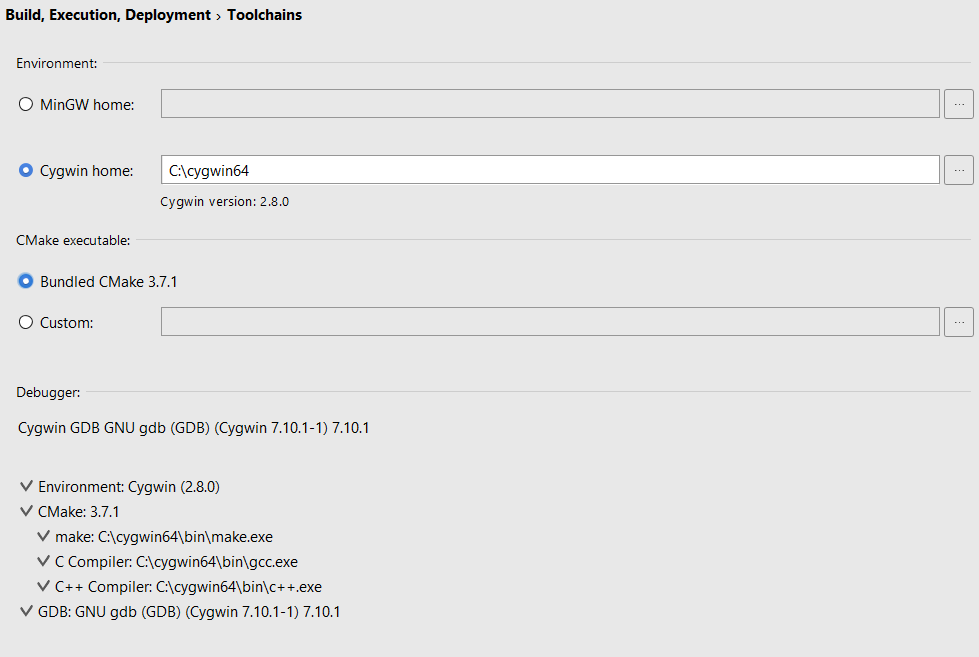
### Configure CLion on Windows

On Windows Only…

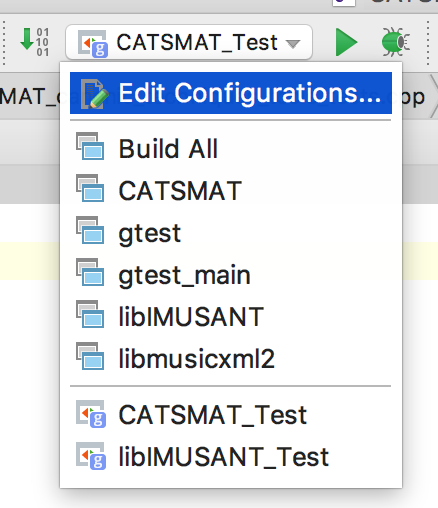
In CLion go to:

File | Settings | “Build, Execution, Deployment” | Toolchains

Select the Cygwin Home radio button and then select the root directory of your Cygwin install (by default this will be C:\cygwin64). The details highlighted below will update and should look like this:



At this point you should be able to open the CATSMAT project. The build targets should be available in the CLion menu bar:



## CMake Files

CMake is driven by CMakeLists.txt files. We have them

* In the root directory to drive the complete build
* In a cmake folder in each of the projects
  + libIMUSANT
  + libIMUSANT\_Tests
  + catsmat
  + CATSMAT\_Tests
* In the google test directory (provided by the google test project- not by us)
* In the libMusicXMLv3 directory (provided by the library authors – not by us).

In addition, in the catsmat cmake directory there is a file called CMakeCatsmatFiles.txt. This file contains definitions for the variables that you need to identify the source and header files that make up the catsmat build. We have put these into a separate file so that it can be included by the CATSMAT\_Test build which must also build the catsmat files into an executable. In future it would be good to structure CATSMAT as a library and create a separate application – that wat the test harness could link against the library in the same way that we have done in libIMUSANT.

# Modifying the CMake Build

If you want to add source files then you do this by adding to file onto the file system in the relevant directory (using CLion), and then creating new entries in the relevant cmake files.

|  |  |
| --- | --- |
| **Project** | **CMake File** |
| Catsmat | $/catsmat/catsmat/cmake/CMakeCatsmatFiles.txt |
| Catsmat\_Test | $/catsmat/catsmat/cmake/CMakeCatsmatFiles.txt |
| libIMUSANT | $/catsmat/catsmat/libIMUSANT/cmake/CMakeLists.txt |
| libIMUSANT\_Test | $/catsmat/catsmat/libIMUSANT\_Tests/cmake/CMakeLists.txt |

Note: For libIMUSANT, the cmake file uses wildcards to identify source files. If you add a file, cmake does not know that it should regenerate its makefiles, so if you add files into the libIMUSANT directories you should run this from the CLion main menu:

Tools | CMake | Regenerate

The same applies to header files everywhere.