Development Environment Configuration

CATSMAT Project – University of New England

Author: Derrick Hill

Last Modified: 5 September, 2015.

# General

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

# libMusicXML

## Version 1

We use version 1 of libMusicXML, but at the time of writing the required files from that library are included within our project and build out of XCode. You don’t need to do anything for libMusicXML.

## Version 3

We also use Verson 3 of MusicXML which is implemented in a framework called libMusicXML2. You need to download this file from sourceforge at this address:

* <http://sourceforge.net/projects/libmusicxml/files/libmusicxml-macosx/>

Note that this is the framework for OS X – if you are on another platform these instructions might be different.

The download gives you a file called libmusicxml-3.00-macosx.tgz. Unpack this somewhere and copy the libmusicxml2.framework to /Library/Frameworks on your machine.

# Boost Libraries

The project uses boost libraries, at time of writing we are on Version 1.58.0. 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 will 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.

If you 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.

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”.

## For WINDOWS Users

Using the CMake build under CLion.

When you compile the Boost libraries, you *must* do this from a Cygwin terminal. If you do that, then all the instructions and paths at the beginning of this section will work.

The paths mentioned above will be relative you 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 have the correct paths in them.

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

# 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).

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

## Setting up Your Environment

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.

You need certain prerequisites to build a system with CLion, including a C++ compiler. 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:

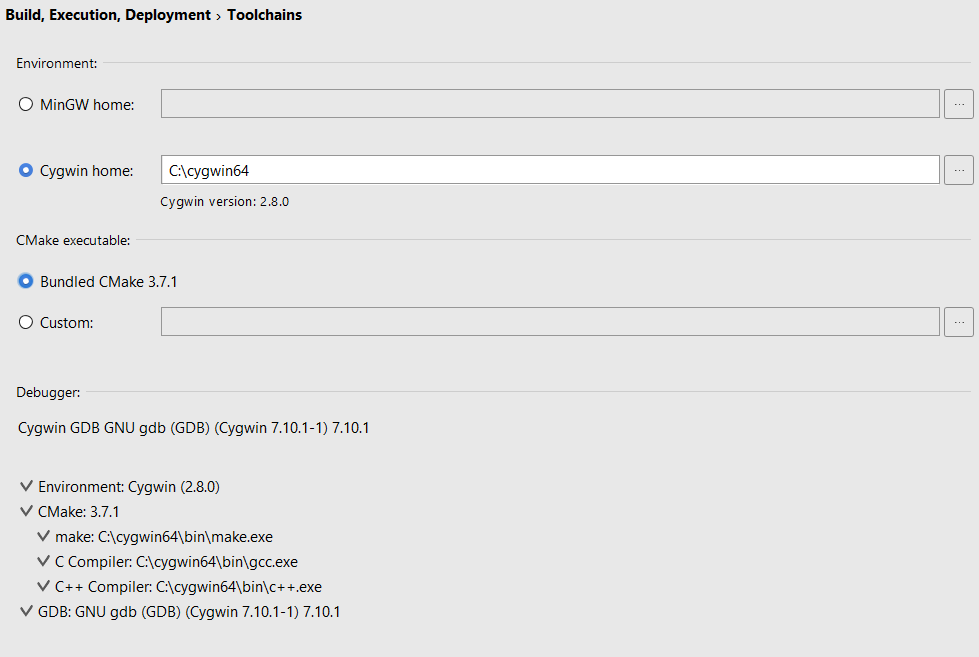
<https://medium.com/@RobertSimoes/installing-clion-on-windows-for-c-c-newbies-like-me-4a346aaf9557>

I chose to install Cygwin rather than MinGW.

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:



Import the source into CLion. In CLion select

File | Import Project

Select the root of your CATSMAT project (i.e. $\CATSMAT)

Select “OK”

Don’t include files in the test\_files folder.

At this point the Run | Build menu item should become available to you (but selecting it will lead to errors in the CMake Debug screen).

## CMake Files

CMake is driven by CMakeLists.txt file that can live in any directory to govern the build in that part of the tree.

If you imported the project as described above, then CLion created this file for you:

C:\Development\CATSMAT\CATSMAT\CMakeLists.txt

Unfortunately, it’s no good as it stands.

### Building libMusicXMLv3

To build libMusicXMLv3 I changed this file to be as follows:

cmake\_minimum\_required(**VERSION 3.7**)  
project(**CATSMAT**)  
  
add\_subdirectory (**catsmat/libMusicXMLv3/libmusicxml-3.00-src/cmake**)

I commented out all subsequent generated commands.

If you now select Run | Build in CLion this runs successfully and seems to work. It builds the library into:

$CATSMAT\cmake-build-debug\catsmat\libMusicXMLv3\libmusicxml-3.00-src\cmake

**Note**: There appears to be a dependency on Cygwin built in if you use the Cygwin toolchain. I built a toy executable and attempted to run it, but received the error message *System Error "The program can't start because cygwin1.dll is missing from your computer."* The solution, apparently, is to add the path to the dll into $PATH, but this implies that the user must have Cygwin installed, which is not good.

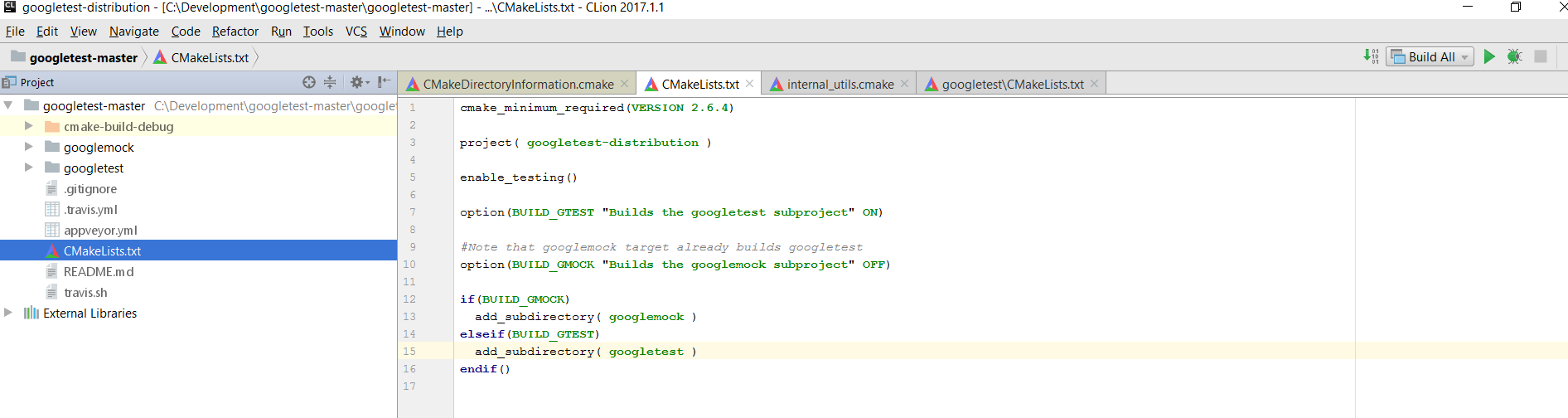
### Building IMUSANT\_Test

To build IMUSANT\_Tests you need to have the Google Test static libraries available to you.

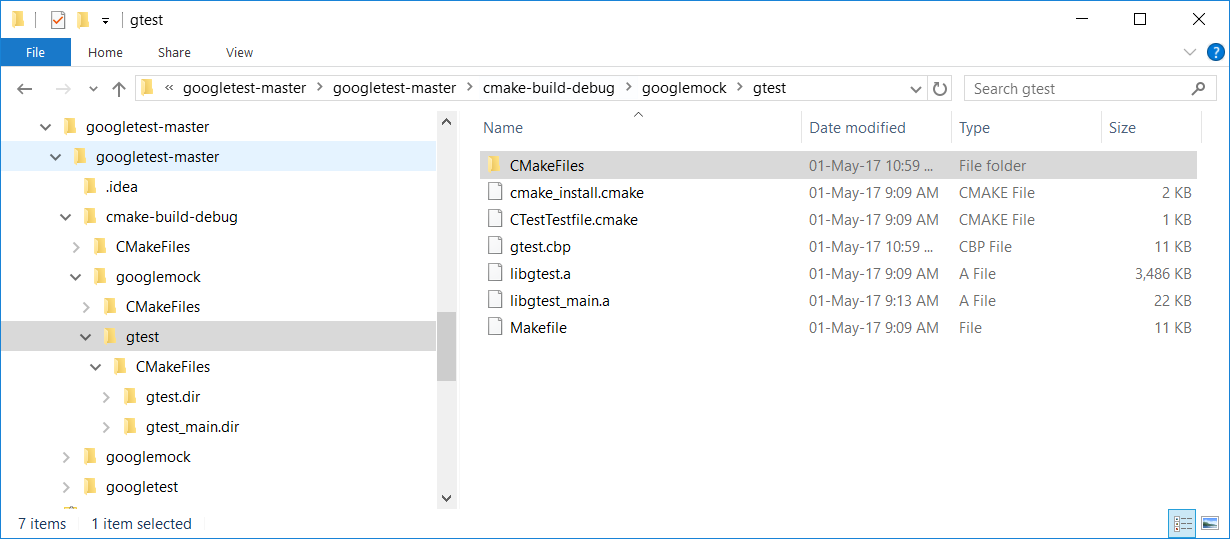
Download the google test source tree from GitHub. Open it in CLion (Google test has a cmake build system).

In CLion edit the CMakeLists.txt file in the root directory to set the BUILD\_GTEST option to ON and the BUILD\_GMOCK option to OFF.

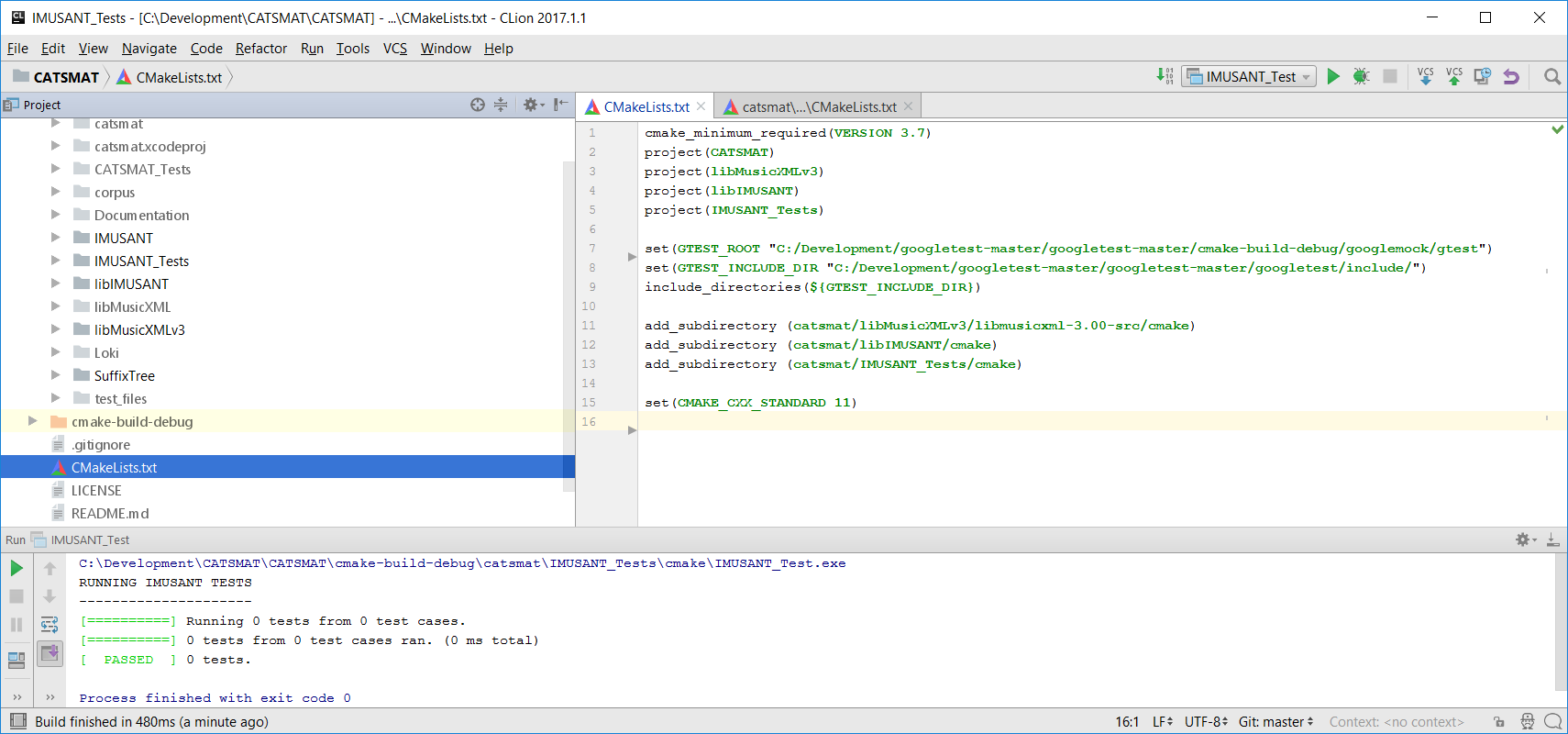
In CLion select Build All, and then, from the menu Run | Build:



This will create the libraries:



In the root CMakeLists.txt file, set the path to these libraries:



It should build and run.