# Configuring Eclipse for ROOT

## ...and maybe learning a little about GCC syntax, so things like this will be easier for you in the future.

Karl Kaess

University of Minnesota - Duluth

5 December 2013

# Disclaimer

This is a tutorial based on a single installation with minimal additional software. My setup includes:

- ROOT 5.34/11
- Eclipse 4.3.1
- Fedora 19 (Linux, for you PC/Mac folks.)

While I believe most of the steps here will generalize to most installations, I do not guarantee it. This lack of a guarantee is especially true for older versions of Eclipse and for systems running Windows.

# More Disclaimer (It's a long one!)

This tutorial is about configuring Eclipse to work with ROOT. It is not a tutorial on Eclipse or a tutorial on ROOT. I may throw a few bones to absolute beginners, but they will be few and far between.

For those of you in a terrible rush:

- If you're seriously not a beginner, have ROOT and Eclipse installed, but haven't gotten them to play nice together at all, start here.
- If you've managed to set up your include directory, but are still having trouble, start here. Based on what I've seen on RootTalk, this is where most people seem to get stuck. (This was the case for me, too, so no worries.)
- If you're in a terrible, terrible rush, and don't care about gaining any insight whatsoever, start here.

For the rest of you, it's not that long a read. Enjoy!

# Setup

If you're trying to make Eclipse work with ROOT, chances are you've already installed Eclipse and ROOT. If not, good places to start are,

- for ROOT: root.cern.ch
- for Eclipse: eclipse.org

If you have ROOT installed, open up a terminal window (command prompt, in Windows). Try typing *root-config --cflags* and *root-config --libs*. Let's take a look at what we get.

# A First Look at GCC Flags

Depending on your setup, you may get some different values from mine, but it seems like I have a good supply, so let's take a look.

```
[kaess@larissa ~]$ root-config --cflags
-pthread -m64 -I/home/kaess/root/include
[kaess@larissa ~]$ root-config --libs
-L/home/kaess/root/lib -lCore -lCint -lRIO -lNet -lHist -lGraf -lGraf3d -lGpad -lTree -lRint
-lPostscript -lMatrix -lPhysics -lMathCore -lThread -pthread -lm -ldl -rdynamic
```

There are definitely patterns and things we recognize from previous experience.

With --cflags, we see:

- a -I<*filepath*>
- a couple -<*harder to categorize flags*>

With --libs, we see:

- a -L<*filepath*>
- a bunch of -l<*characters or words*>
- a couple -<*harder to categorize flags*>

# A First Look at GCC Flags

Of course, if you want to know what all these flags mean, a good place to start would be the GCC manual, which you can find somewhere around here:

- gcc.gnu.org

However, I'm about to tell you where you can stick every single one of these things, so I would only recommend this if you have plenty of time on your hands.

Everybody else, we should go ahead and set up an Eclipse project.

# So Now We Have an Eclipse Project

So we've set up an Eclipse project. For the beginners out there:

- It should be a C++ project. ROOT is written in C++, so your project will be, too.
- If the words "make" and "target" mean nothing to you, use the Hello World template, and build off of that.

Now we should type up some syntactically (if not logically) correct C++ code using some of the the familiar ROOT classes...

# And Now We Have Some Non-working Code. Good!

Let's see what it looks like...

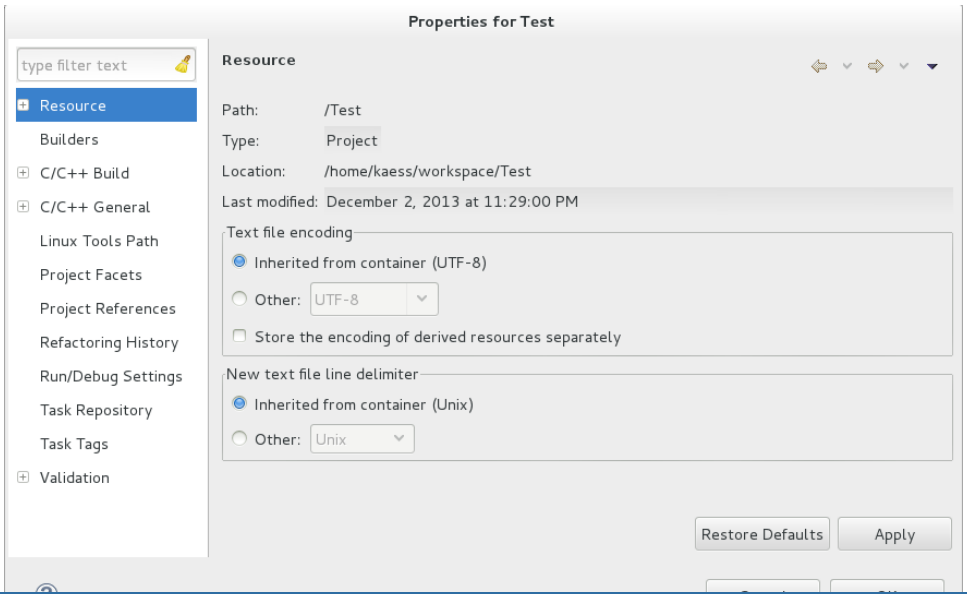
```
//============================================================================
// Name        : Test.cpp
// Author      : Karl Kaess
// Version     : 0.1 alpha
// Copyright   : 2013. No, not really.
// Description : It's a test, silly.
//============================================================================

#include <TFile.h>
#include <TH1.h>
#include <TString.h>
#include <iostream>
using namespace std;


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

    for (int i=0;i<10;i++) {

        TString ts = TString("data");
        ts+=i;
        ts+=".root";

        TFile * tfile = TFile::Open(ts.Data());
        TH1 * th1 = (TH1 *) tfile->Get("data");

        cout<<th1->Integral()<<endl;

        //Delete everything, because I'm not sure what ROOT does behind the scenes!
        delete th1;//This may be superfluous. If so, it may be dangerous, in some rare cases.
        tfile->Close();
        delete tfile;
    }

    return 0;
}
```

Method 'Close' could not be resolved

This should open files with a sequential naming pattern (located in the project's root directory), extract an identically named TH1 from each, calculate its integral, and print it out. But Eclipse doesn't recognize any of the ROOT classes! Something must be done.

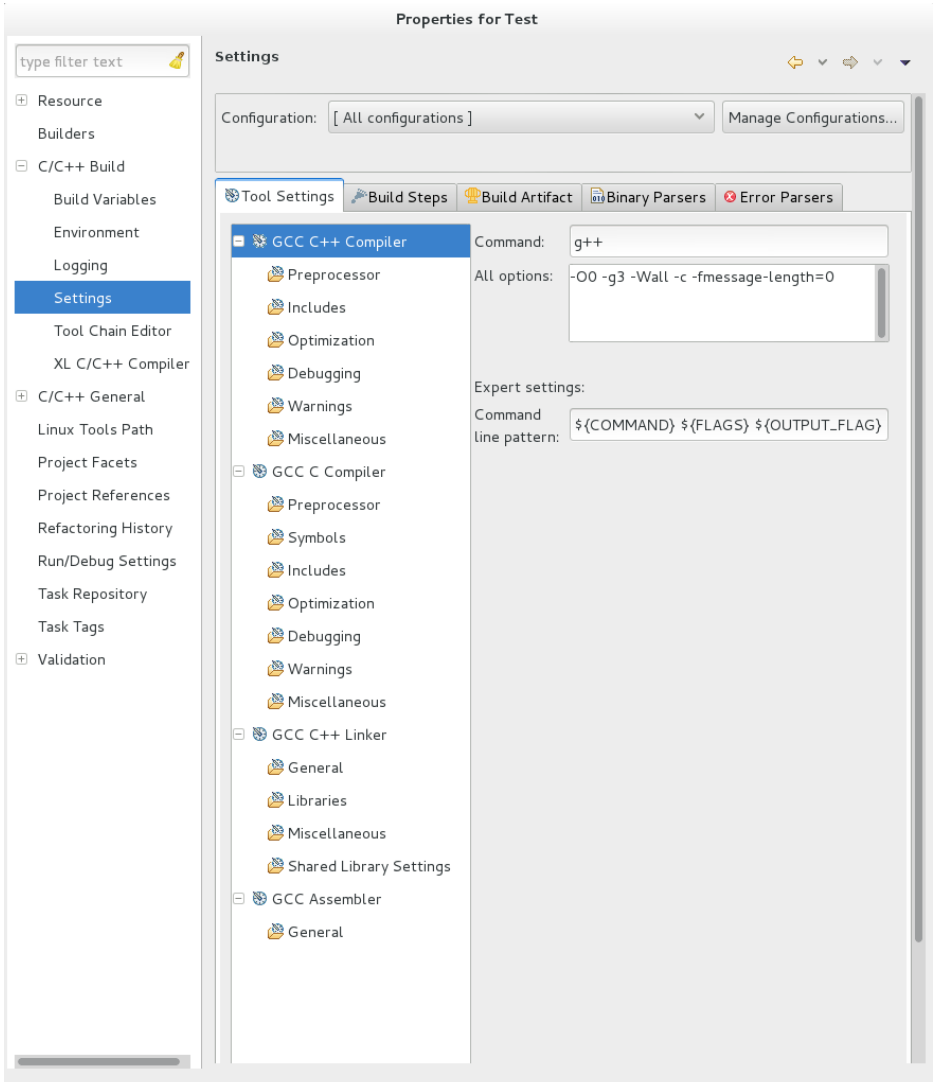
# Finding the Settings

Start by right-clicking our project folder in the Project Explorer, and selecting "Properties" from the extremely long pop-up menu. (It's at the bottom.) Alternately, this option can be found in the File menu, but make sure you have the correct project selected in the Project Explorer, or you'll be configuring another project. You should get a dialog box that looks something like this:



# Finding the Settings

Now that you have the Properties dialog open, expand the "C/C++ Build" option on the left-hand side, and select "Settings". You should have something that looks like the picture on the right. Now, unless you know why and how you want to vary the settings between configurations, you should set the "Configuration:" option at the top to *[all configurations]*.



# The Compiler

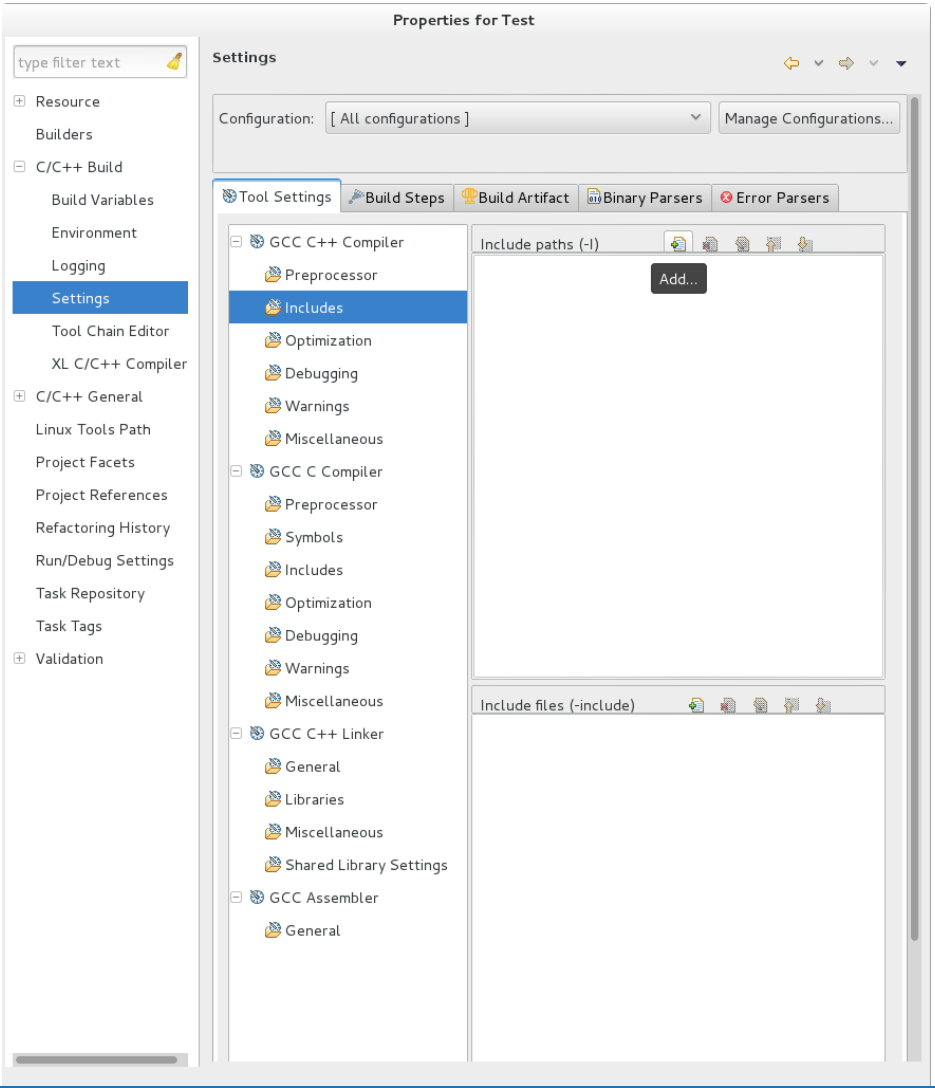
One could imagine that "cflags" might stand for "compiler flags", and that the *root-config --cflags* command might output a list of them. It does, more or less, depending on what side of the compiler nomenclature wars you fought on back in the 70s. Me? I was a conscientious objector, on the grounds of not having been born yet. Am I making this war up, then? Maybe. Let's look at the flags:

- -I<*filepath*>: this is an include directory.
- -pthread and -m64: These are regular old flags. Optimization flags? Miscellaneous flags? Does it matter? Not that I know of.

Now let's figure out where to put them.

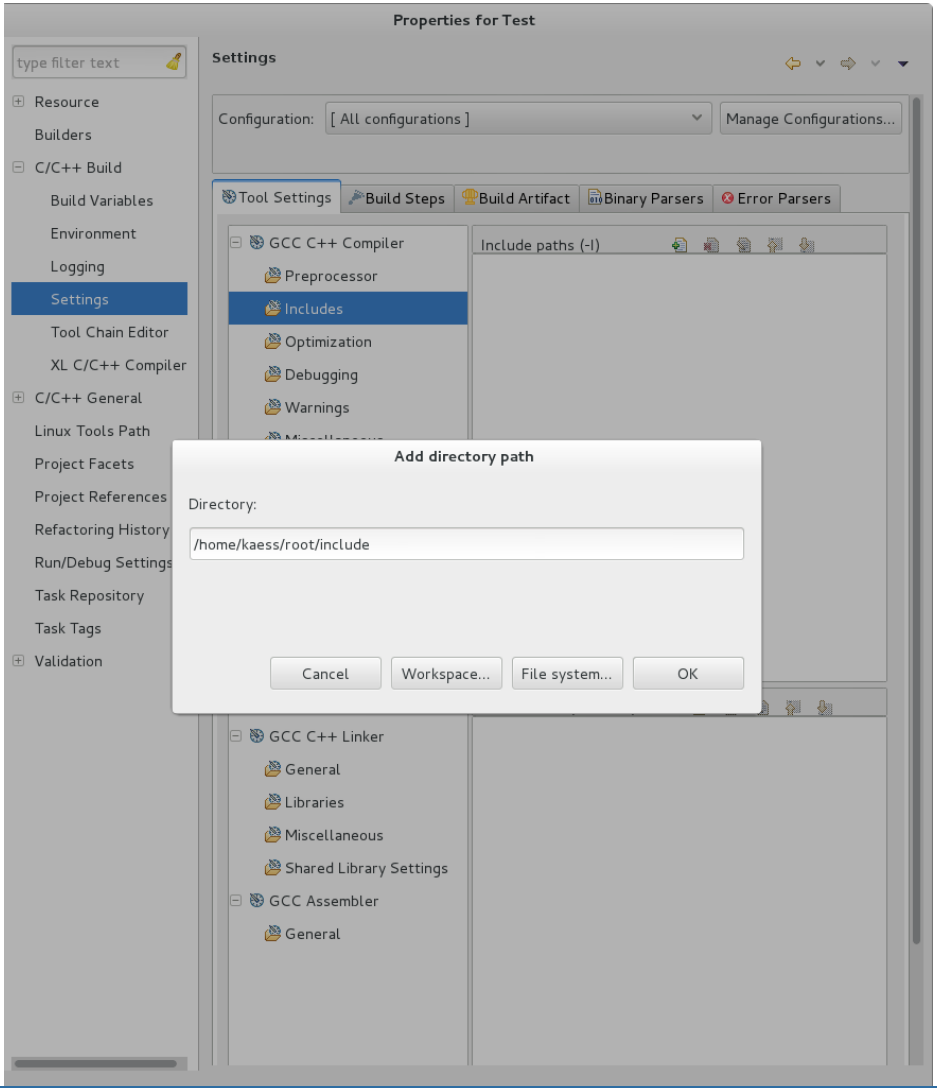
# The Compiler

Under “Tool Settings”, there is a selection tree that includes “C++ Compiler” and “C++ Linker” sub-trees. Expand the “C++ Compiler” sub-tree, and select the “Includes” option. On the right, you should see a box labeled “Include paths (-I)”. This is where any -I flags go. Click the small “+” icon to the right.



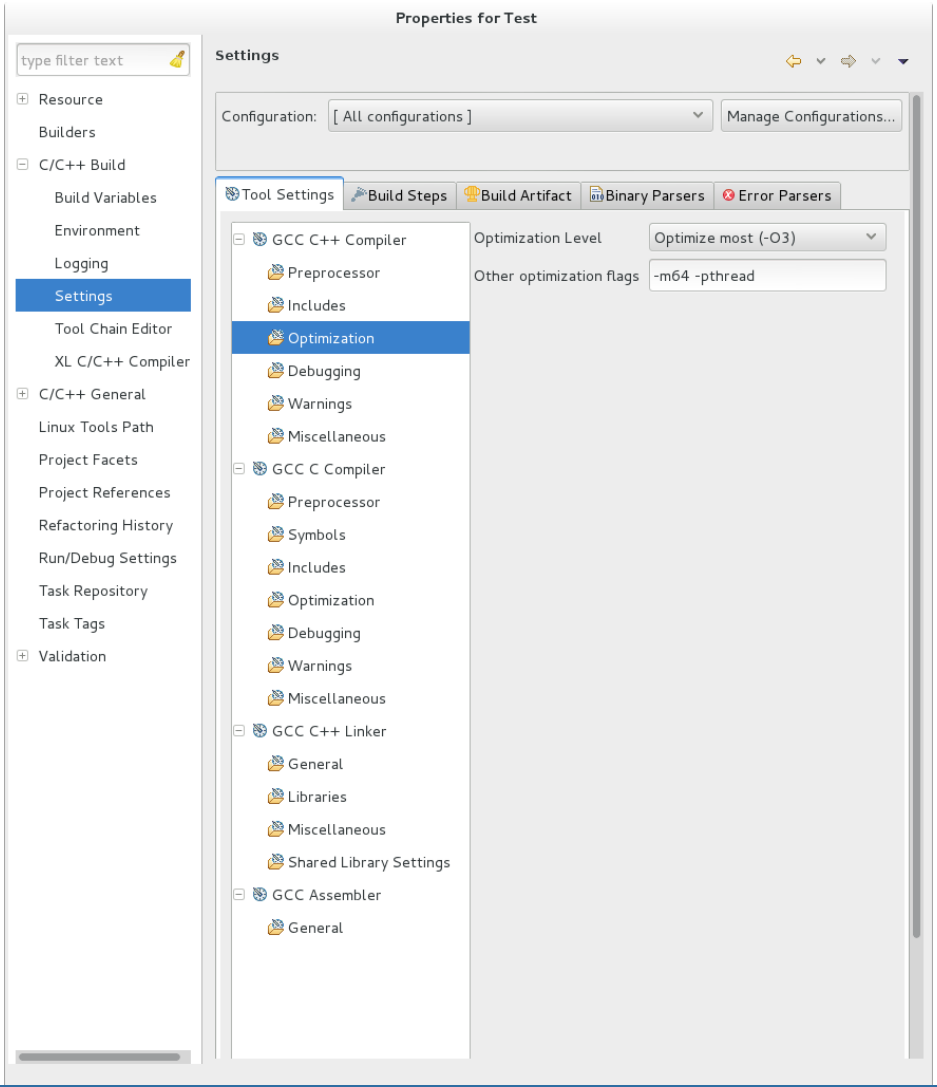
# The Compiler

On the dialog that appears, enter the filepath that appeared in -I<*filepath*>. This should appear in the "Include paths" box, and will let Eclipse find the ROOT classes. This won't get your program to run, but the next error you get will mention parts of ROOT that you probably have never heard of. We'll fix that by setting up the linker, but first, let's take care of those other flags.



# The Compiler

Next, select “Optimization”. Now is a good time to set your optimization level to -O3. (Beginners: This can make debugging harder, so here’s one place where you might set up your “Debug” and “Release” configurations differently.) In addition, here’s where I put my extra flags. You can also file them under “Miscellaneous”.



# The Linker

One could imagine that "libs" might stand for "linker flags". No, wait...

Actually, the option *root-config --ldflags* lists linker flags, the output of which I would recommend including in your configuration steps (I get -m64). I've never had a problem skipping it, but don't count on it!

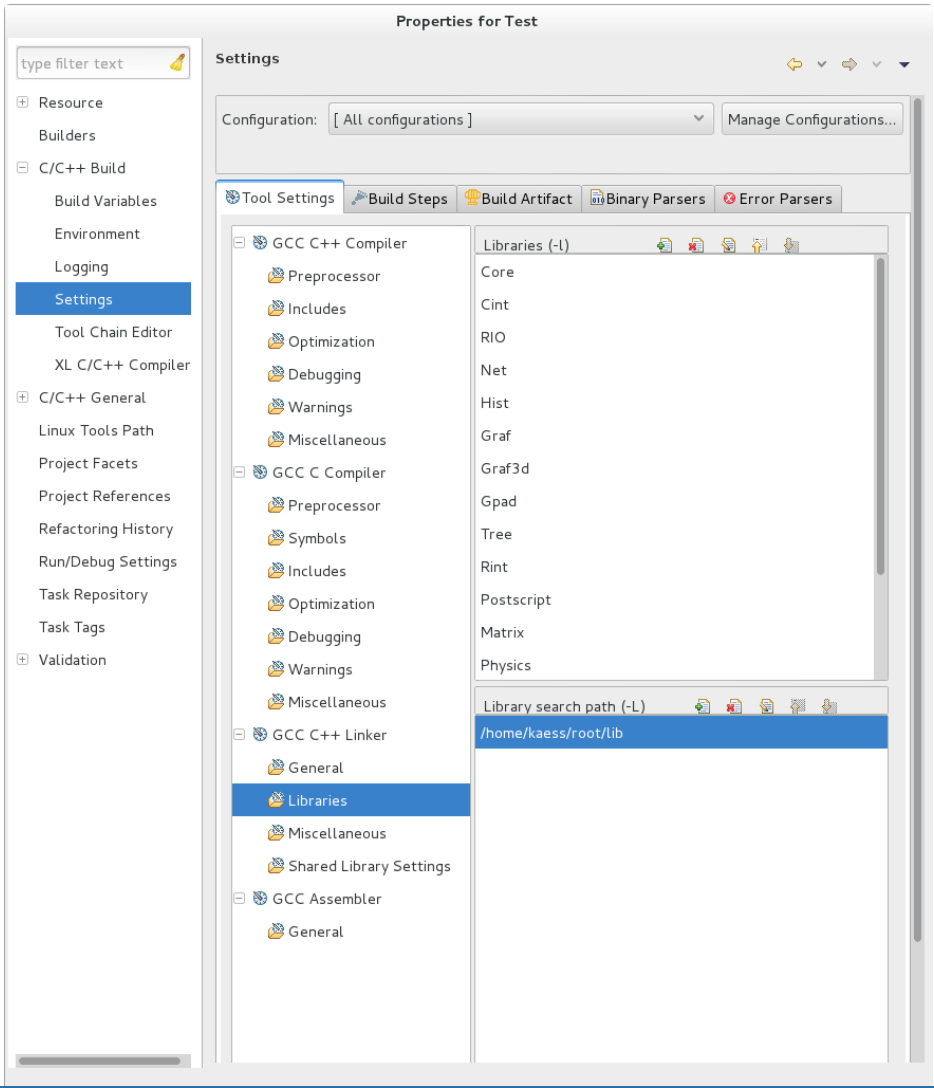
So,"libs" stands for libraries. Now let's look at the flags:

- -L<*filepath*>: this is a library search directory.
- -l<*characters or words*>: these are the libraries themselves.
- -pthread and -rdynamic: These are regular old flags. Fortunately, there's only one place to put them, so I don't have to worry about misclassifying them. (Yay!)

# The Linker

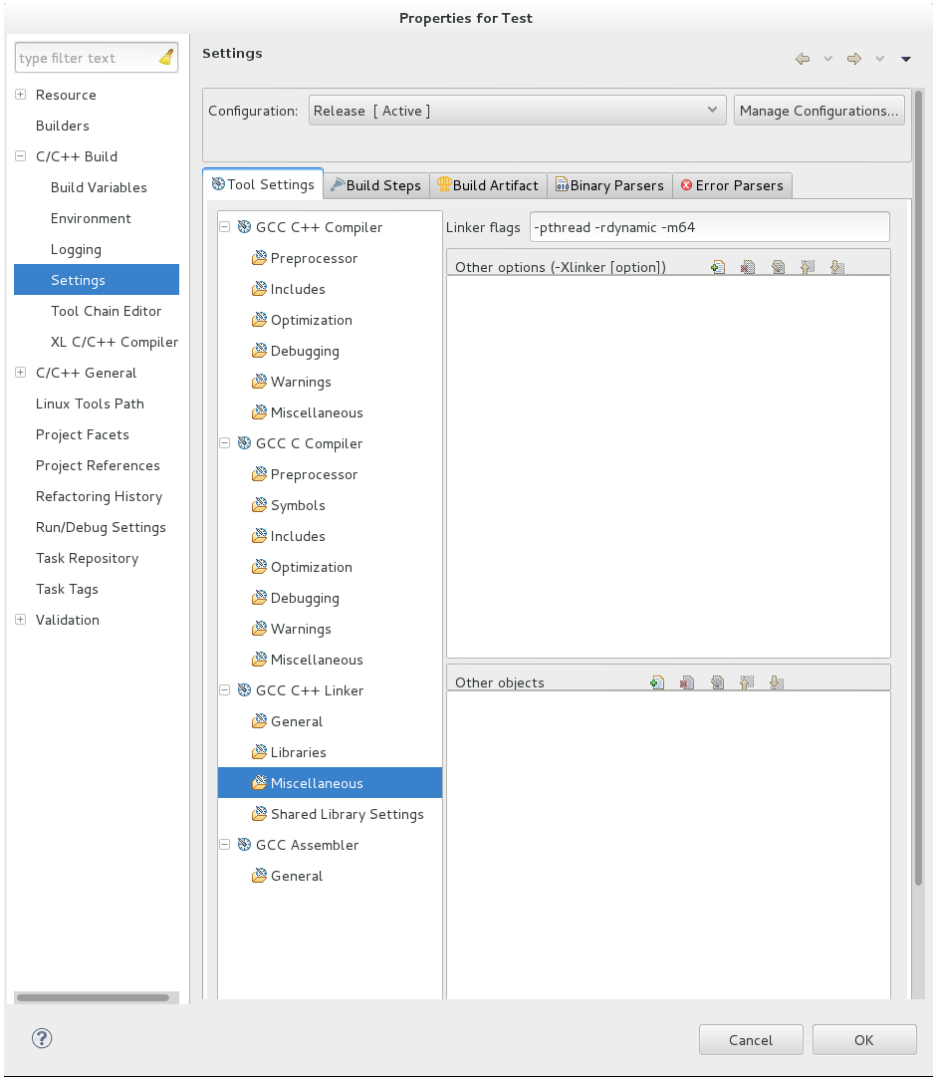
*Also include “Gui” above “Core”*

Next, expand the “C++ Linker” sub-tree, and select “Libraries”. Enter the filepath from -L<*filepath*> in the “Library search path (-L)” box similarly to how we entered the Includes path. Then, add each of the libraries to the “Libraries (-l)” box. The ordering matters, since later libraries can have dependencies in earlier ones, but not vice versa. When you’re done, it should look something like this.



# The Linker

Finally, select "Miscellaneous" in the linker sub-tree. Enter the remaining flags next to "Linker flags". And we're all out of flags, which means we're done!



# The Cheat

Instead of setting this all up, you can enter `` `root-config --cflags` `` and `` `root-config --libs --ldflags` `` in the miscellaneous compiler and linker flags settings, respectively. I doubt this works in Windows, but it does in Linux, and hopefully does in OS X. (OS X is Unix-based!)

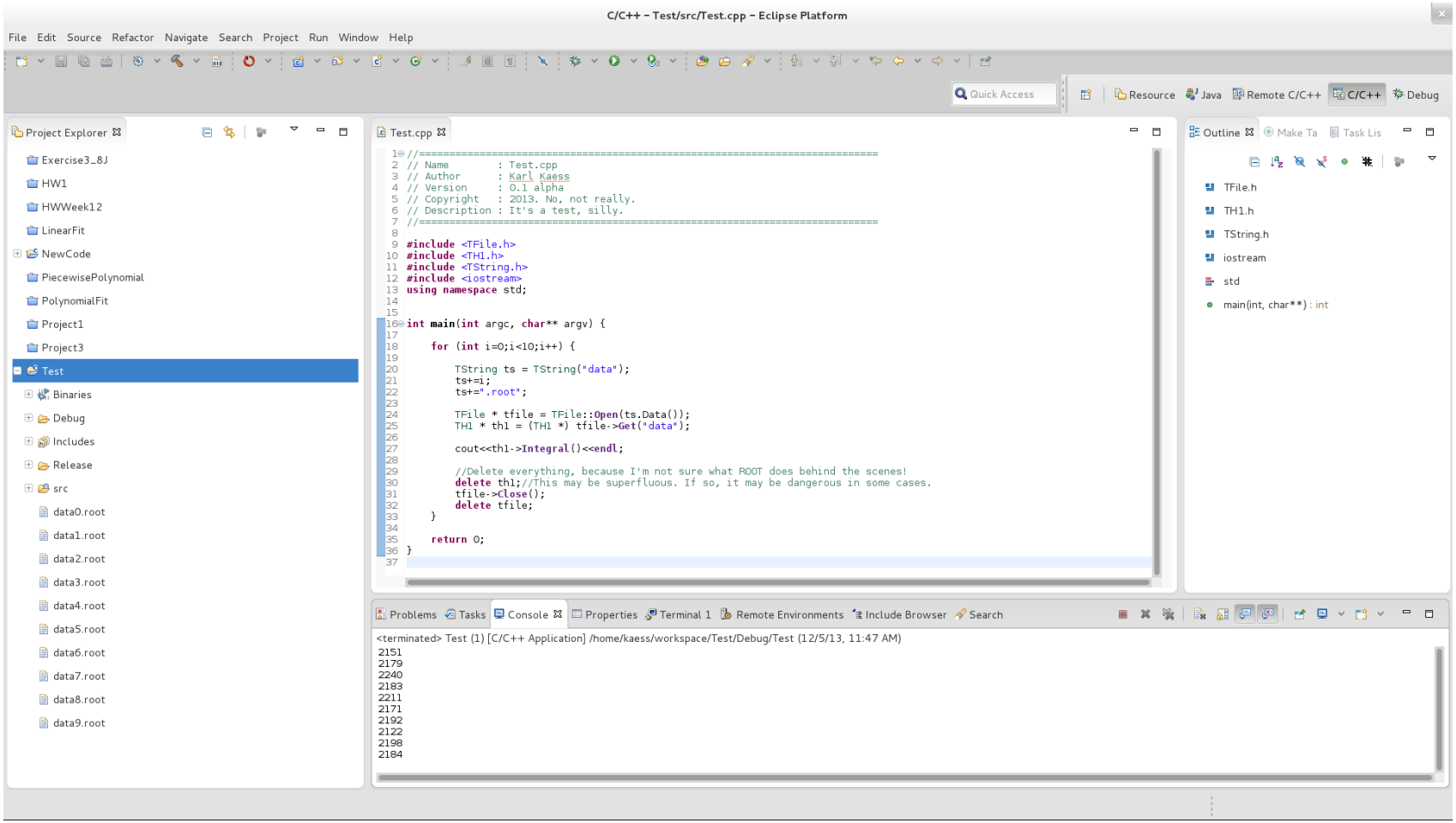
You should probably still set up your includes directory settings, so you can benefit from some of Eclipse's nicer features. For those of you who skipped straight here, I explain how to do this here.

# Building Your Code

Now, let's try building the code. In the console output, we can see all the things we put in. The first call to g++ compiles our code, and the second links it.

```
13:11:08 **** Incremental Build of configuration Release for project Test ****
make all
Building file: ../src/Test.cpp
Invoking: GCC C++ Compiler
g++ -I/home/kaess/root/include -O3 -m64 -pthread -Wall -c -fmessage-length=0 -MMD
-MP -MF"src/Test.d" -MT"src/Test.d" -o "src/Test.o" "../src/Test.cpp"
Finished building: ../src/Test.cpp

Building target: Test
Invoking: GCC C++ Linker
g++ -L/home/kaess/root/lib -pthread -rdynamic -m64 -o "Test"  ./src/Test.o
-lCore -lCint -lRIO -lNet -lHist -lGraf -lGraf3d -lGpad -lTree -lRint -lPostscript
-lMatrix -lPhysics -lMathCore -lThread -lm -ldl
Finished building target: Test


13:11:10 Build Finished (took 1s.776ms)
```

# Running Your Code




It's alive! And now you can move on with your life, instead of banging your head against the wall trying to configure Eclipse.