Building *THE* libraries

Prepare for the journey

Mac

This is the easiest platform. Congratulations! Throughout this tutorial I build everything in xcode. I try to explain the differences between 64 bit and 32 bit builds. Note, through this document I will address them separately. If you would like a universal binary, most of the time they can be built together.

CRCNS Environment Variable

Many of the tools reference a CRCNS environment variable. Cmake uses it to search local paths to find all of the code, include files, and prebuilt libraries. To set this up, from the command line type:
export CRCNS=/your/crcns/directory/here

TODO: Permanent setting up

This directory should look as follows:
trunk (or alternate name depending on checkout)
- bin
- code
- docs
- include
- lib

Libraries for *THE,ir-tools, and ir-tweak*

Library	**Pre-built**	**Required**
Cg	Yes	Yes
Glew	Yes	Yes
itk	No	Yes
Boost	Yes (Only necessary libraries)	Yes
Qt	No	Yes
FLTK	No	No

Building Qt

Version: 4.5

Building Qt is fairly simple for mac. It does take a fair bit of time though, so be warned. First go to qt's website. Click LGPL / Free Downloads (unless by this time we're using a licensed version of qt). Download Qt: Framework Only. The installer is nice, but we need to install these things on our own. Unzip it and place it somewhere nice.

To install 32 bit:

```
./configure -prefix ${PWD} -no-glib -qt-libjpeg -qt-libmng -qt-libpng -qt-gif -qt-zlib -static -release
-make libs -make tools
 make
make install
```

To install 64 bit:

```
./configure -arch x86_64 -prefix ${PWD} -no-glib -qt-libjpeg -qt-libmng -qt-libpng -qt-gif -qt-zlib
-static -release -make libs -make tools
```

Note: The only difference here is in passing the x86_64 flag. For most libraries and tools on the mac setting x86_64 will give you a pure 64 bit binary. For a universal binary you can pass multiple architecture flags.

Congratulations! One tool down. There shouldn't be anything left to do.

Building Qt Projects

Unfortunately not all of the projects use cmake (though they really all should). The common alternative is using Qt projects. Qt projects are all contained in a .pro file. Below is an example of the current version of iris.pro

```
CONFIG +=   console
HEADERS =   IrisGui.h
SOURCES =   IrisGui.cpp iris.cpp
FORMS =   iris.ui

target.path =   myqtapp
sources.files =   $$SOURCES $$HEADERS $$RESOURCES $$FORMS *.pro
sources.path = .
INSTALLS = target  sources
TEMPLATE = app
```

You'll notice its a fairly simple file that contains information about how it needs to be built. To make a Visual Studio project out of this .pro file you can use qmake. Since we just built this, let's use it!

Building iris

1. Open up the command line.
2. Navigate to %CRCNS%\code\iris
3. *Create a folder for your project such as "vs"*
4. Use the command:
   ```
   qmake -t vcapp ..\iris.pro
   ```

Step three is italicized since its optional. Visual studio creates a fair bit of temporary data, so its nice to have that data in a separate directory. You're welcome not to. If you don't ../iris.pro will be changed to iris.pro.

Now using Visual Studio you can open the iris Visual Studio Project file. If you build Qt correctly, this will now build without a problem. Congratulations! You're one step further to getting *THE* built.

Cmake

Cmake is a cross platform tool for building. It allows you to pick your compiler and it creates a project file for that specific compiler. We will be picking xcode.

It can be downloaded here: http://www.cmake.org/

There are no specific needs for this tool. It should install and work great.

Building itk

WARNING:
The Insight Toolkit has been patched by Paul. Because of this it is NOT wise to use the most up to date version of the Insight Toolkit. I have tried the latest itk and came across build errors
Instead use this one: http://superb-west.dl.sourceforge.net/sourceforge/itk/InsightToolkit-3.2.0.tar.gz

Download the above link and place in $CRCNS\bin. I suggest renaming it to itk. As of this writing the cmake files specifically look for an itk folder and an itkbin folder. I recommend naming them that way.

The good news is itk comes with a Cmake project file. But before we get started we need to patch itk.

Patching itk

Copy $CRCNS\include\itk-patch to $CRCNS\itk\

That's it. If you're using a newer version doing the above will stomp newer code changes. This can cause it to fail building. If you need a newer version I recommend trying to merge the changes instead. You may have some success.

Building itk

1. Run Cmake
2. Point binaries to $CRCNS\bin\itk
3. Point source to $CRCNS\itk

Turn off...
BUILD_EXAMPLES
BUILD_SHARED_LIBS – *This is important since we want static libraries. It won't build them with this turned on.*
BUILD_TESTING

For 64 bit:
Change CMAKE_OSX_ARCHITECTURES to x86_64

Testing and examples are not needed.

Create a new project. Select xcode.

Configuring will take a while. You should have no errors. You can then click okay. You will have a new project built in: $CRCNS\bin\itkbin. There should be an xcode project file named ITK.xcodeproj.

Double check to make sure that Release and x86_64 are selected. Click build and wait patiently. If you do not subtract the examples and tests this build will take a very very long time. But since I'm assuming you did, its not too terrible long.

Congratulations! Two tools down. And the long ones at that.

Building *THE*

Requirements:

Building *ir-tools*

Requirements:

Building ir-tweak

Requirements:

Comprehensive

Boost

Download boost from their site.

TODO: Make better

TODO: use bjam

TODO: 64-bit flags

FFTW

Version: 3.2

Cmake searches for both thread enabled and thread disabled libraries. I don't find any reason to build the libraries twice.

If you decide not to build the libraries twice you'll either have to, point FFTW_THREADS_LIBRARY to the same library as FFTW_LIBRARY. Or you can just copy the library twice and append _threads to the end of the second.

You'll need to run both of these. The first is without float support:

```
configure --prefix=$PWD --with-our-malloc16 --enable-static --disable-shared --enable-threads
--with-combined-threads –enable-sse2
make
```

The second with float enabled:

```
configure --prefix=$PWD --with-our-malloc16 --enable-static --disable-shared --enable-threads
--with-combined-threads –enable-float
make
```

It should append a f to the libraries to differentiate them. They are now located in a hidden folder .libs.

They can be copied to $CRCNS/lib/MacOS/fftw and cmake will search for them automatically.

For 64 bit:

fftw recognizes cflags. Add CFLAGS="-m64" to either of the above.

glew

Glew is checked into the code directory. This can be built using cmake.

For 64 bit:

Make sure that CMAKE_OSX_ARCHITECTURES is x86_64

Once built libGLEW.a can be copied into $CRCNS/lib/MacOS/glew. This makes it easier for other tools to find it.

CG

Please just install this. It can be downloaded from NVIDIA's website, and their installer puts it where you need it. For ir-tweak you'll need to copy it over into the bundle, which is explained in ir-tweak.