Cloning and building the Orx library on the Mac with gmake

In the last article, I showed you how to get your Mac ready to develop Orx by downloading the *Xcode Commandline Developer Tools* which gives you `git`, `gcc` and a MacOSX SDK.

`git` is the tool that we will use to clone (or download) the latest repo of the Orx library.

Getting Orx

- Open a Terminal
- cd to a folder where you would like to download Orx to. Suggestion: `~/Documents`

```
$ git clone https://github.com/orx/orx.git
```

This will show the cloning process to your hard drive:

```
Cloning into 'orx'...
remote: Counting objects: 73959, done.
remote: Total 73959 (delta 0), reused 0 (delta 0), pack-reused 73959
Receiving objects: 100% (73959/73959), 58.93 MiB | 87.00 KiB/s, done.
Resolving deltas: 100% (54159/54159), done.
~Documents$
```

The Orx library source code is now on your Mac.

The next step is to run the setup script which will do three things for you automatically:

- Download all the external dependencies that Orx requires.
- Set a special $ORX environment variable
- Create projects for the more popular Mac IDEs so you can build the Orx library.

```
~Documents$ cd orx/
orx$ ./setup.sh
```

You will get the following output:

```
== Checking version: [ extern/ ]
== [ d99b8c61d781 ] needed, current [ ]
== [ d99b8c61d781 ] not in cache
== Fetching [ https://bitbucket.org/orx/orx-extern/get/d99b8c61d781.zip ]
```

```
== Please wait!
== [ d99b8c61d781 ] cached!
== Decompressing [ cache/d99b8c61d781.zip ] => [ extern/ ]
== [ d99b8c61d781 ] installed!
== Copying [ premake4 ] to [ code/build ]
== Copying [ premake4 ] to [ tutorial/build ]
== Copying [ premake4 ] to [ tools/orxFontGen/build ]
== Copying [ premake4 ] to [ tools/orxCrypt/build ]
```

The external dependencies are downloaded and unpacked into your orx/extern folder. `premake4` is copied to several folders which will be explained in a moment.

Next, the script sets a very handy environment variable ($ORX) on your system. This will have many uses for when you come to create your own game projects, but this is covered in the next article. But you will see output of:

```
== Setting environment: [ ORX = /Users/Someone/Documents/orx/code ]
```

The third step of the script is to create all the Projects for many IDEs availble for the Mac. `premake4` is the tool that the script uses to create them, and the output will look like:

```
== Generating build files for [ mac ]
== Generating [ gmake ]
Building configurations...
Running action 'gmake'...
Generating mac/gmake/Makefile...
Generating mac/gmake/orx.make...
Generating mac/gmake/orxLIB.make...
Generating mac/gmake/Bounce.make...
Done.
Building configurations...
Running action 'gmake'...
Generating mac/gmake/Makefile...
Generating mac/gmake/01_Object.make...
    ... >8 snip!...
Generating mac/gmake/12_Lighting.make...
Done.
Building configurations...
Running action 'gmake'...
Generating mac/gmake/Makefile...
Generating mac/gmake/orxFontGen.make...
Done.
Building configurations...
Running action 'gmake'...
Generating mac/gmake/Makefile...
Generating mac/gmake/orxCrypt.make...
Done.
```

```
== Generating [ codelite ]
Building configurations...
Running action 'codelite'...
Generating mac/codelite/orx.workspace...
Generating mac/codelite/orx.project...
Generating mac/codelite/orxLIB.project...
   ...etc...
Done.
== You can now build orx in [ code/build/mac ]
== Installing git hook [ post-checkout ]
== Installing git hook [ post-merge ]

== IMPORTANT - New environment detected: Please logout/login to refresh your
environment if you're using an IDE.

== [ 0:14:58 ] Setup successful!
```

We now have everything in place to actually build the Orx library.

> Please note the message above that says: `== IMPORTANT - New environment detected: Please logout/login to refresh your environment if you're using an IDE.` Just close your Terminal and open a new one so that Orx settings are available during Terminal sessions.

Building the Orx Library with gmake

Notice in the output above, Solution Projects have been created for gmake, Codelite, Code::Blocks and Xcode.

> If you would rather compile using Xcode than gmake, follow this tutorial: Building the Orx Library with Xcode on Mac OS X

> If you would rather compile using Codelite than gmake, follow this tutorial: Building the Orx library on a Mac with Codelite

As we are setting up without Xcode, let's use gmake as our method to build Orx as it is available to everyone (if you followed the first article).

- cd into the build folder:

```
$ cd ~/Documents/orx/code/build/mac/gmake/
```

Then build the **Debug** version of the Orx library with:

```
$ make config=debug64
```

The output will look like this:

```
==== Building orxLIB (debug64) ====
Creating obj/x64/Debug/orxLIB
orxPlugin_EmbeddedList.cpp
orxAnim.c
orxAnimPointer.c
orxAnimSet.c
   ... >8 snip ....
orxString.c
orxTree.c
Linking orxLIB
ld: warning: directory not found for option '-L/usr/lib64'
Running post-build commands
cp -f ../../../lib/dynamic/liborx*.dylib ../../../bin
==== Building orx (debug64) ====
Creating obj/x64/Debug/orx
orxMain.c
Linking orx
ld: warning: directory not found for option '-L/usr/lib64'
==== Building Bounce (debug64) ====
Creating obj/x64/Debug/Bounce
orxBounce.c
Linking Bounce
ld: warning: directory not found for option '-L/usr/lib64'
```

Next, create the **Profile** version of the library with:

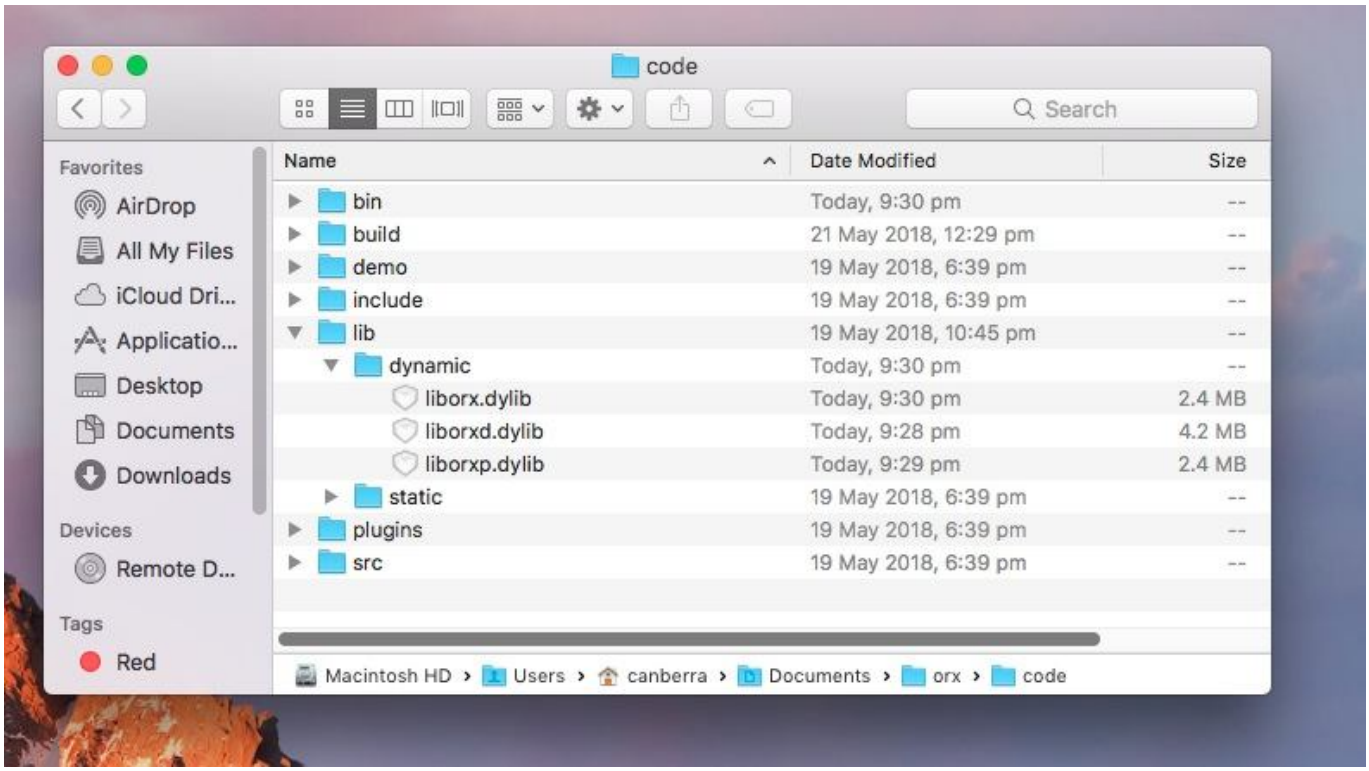
```
$ make config=profile64
```

And finally, the **Release** version of the library with:

```
$ make config=release64
```

Finding our Orx libraries

These are saved to the `orx/code/libs/dynamic` folder. Use finder to take a look at the output:



You can see the `liborx.dylib`, `liborxd.dylib` and the `liborxp.dylib` versions.

Well done. You have compiled the Orx library which can now be used in your own game projects.

Now we are ready to make a game project of our own.

> Just a little note on the $ORX environment variable that the `setup.sh` script created for you: if you open the `.bashrc` and `.profile` files in your ~ folder, you will see the ORX variable set there for you.