Mac OSX SDL Setup & Maya Plugin Setup Using Xcode

By Thomas M Craigen Jr.

Introduction:

If you are developing on the OSX platform, whether it's maya, shake, after effects, OpenGL, SGL, or even strait Cocoa. You probably will be using XCode … In fact almost every book that I have referring to C++ development has recommended XCode not just because it's a good IDE but It's all 100% free. And registering for the Developers updates is free too! Not sure if Miscrosoft would even consider something to that extent.

Resources:

developer.apple.com

nehe.gamedev.net

http://www.libsdl.org

That Aside, lets get into setting up your environment.

The XCODE installation is located on one of your OSX disks. It's not installed by default. Just pop the disk in and follow the installation instructions for the Development Environment.

If you have XCODE installed it should be here:

HD>Developer>Applications>Xcode.app

If you can't find it, try searching for Xcode.app, if it's still not there try your installation again. Please don't fill the forums up with installation questions.

Next we need to download and install the SDL packages. For the OSX users you must download both the Developers and Runtime libraries. Be sure to install your Frameworks in the proper location. *This is not the case on the windows platform. If you don't have frameworks installed you will not have any success with SDL building.*

http://www.libsdl.org

SDL-devel (Project Builder + XCode)
SDL.dmg

So if you have SDL installed and XCODE installed you should be ready to start working in SDL on the OSX platform.

First, get familiar with XCODE by using the Default projects that are already provided as a template. This will help you in knowing how to setup your own projects later.

1. File > New Project

(fig.1)

Once you have your project Saved and named you can start integrating your code. I personally don't like Apples .m files. So I leave everything the way it is. Just dabble inside the SDL.m file later.

For now, lest create a new "cpp" file.

2. File > New File

Select "C++ File" under the Carbon heading.
This will just add your c++ file to your project.
Name it: Source.cpp

3. Implement your code for SDL in your Source.cpp

Implementation of the code for SDL is mostly the same. So you can use the sample code you find on the libsdl website or the Pong.zip source.cpp.
Usually the Source.cpp is the code that you want to implement all the features of your game instead of using the files that Xcode has setup for you.

Referencing External Source Files

When you get to calling resources such as bitmap images it's a little different than the windows implementation.

In windows you can refer to paths as if the folders are with the application.
Pong.exe would have folders associated with it like this.

OSX	WINDOWS
Pong.app	Pong.exe
Pong.app/Contents/Resources/pongBack.bmp	Data\pongBack.bmp
Pong.app/Contents/Resources/ball.bmp	Data\ball.bmp
Pong.app/Contents/Resources/paddle.bmp	Data\paddle.bmp

```
background = load_image("pong.app/Contents/Resources/pongBack.bmp");
ball = load_image("pong.app/Contents/Resources/ball.bmp");
p1 = load_image("pong.app/Contents/Resources/paddle.bmp");
p2 = load_image("pong.app/Contents/Resources/paddle.bmp");
```

It's a bit longer to type out on the Mac. But this is because everything for your game can be stored all in one file; the "pong.app" file in this example.

The dot app extension:
Basically the app extension is much like a zip file. You still can create the other folders outside your "app" that you have built but it's probably a good idea to keep things all packaged together for now in the "app" file.

MAYA API Implementation

Maya is a bit different than what you would expect. A lot of what you need for maya Plug-in development is contained in the Developers Kit (Dev Kit for short). You have to install this during you maya install.
Your Devkit should be here:
/Applications/Alias/Maya[version]/devkit/

You should make sure you are using the latest version release of maya along with the latest version release of Xcode. For beginners there is no need to start developing on an older version of Maya's API as it is even harder to get running than the current release. This will ensure that you have the least amount of problems to deal with.

Starting a maya project setup is a bit harder than you would think. So I would recommend copying a xcodeproj from the Developers kit and starting with that as your base code for implementation of your code.

http://www.davidgould.com/
http://www.cgtalk.com

I would also recommend the First and Second volumes of David Gould's books called "Complete Maya Programming vol1 and vol2". Much of the tools there should get you started when you are ready to start building maya plugins.

OSX is not really ready to be a full Maya API development environment. One main reason is because you can't unload and load plugins on the fly with Maya loaded. You have to close down maya and reload maya to see your new plugin changes.

I recommend implementing a script that launches maya once you have finished building your code.

This is the code I run once I have finished compiling my script. Your project in Xcode allows you to run scripts once the build is complete. This is a very powerful feature and I recommend using it.

In your project window you should see "Targets" and under that your application name you have chosen for you Maya API. Under that you should see "Run Script".

Targets>"Project Name">Run Script

Right Click: Run Script >> select Get Info

```
cp "$TARGET_BUILD_DIR/$EXECUTABLE_PATH" "$TARGET_BUILD_DIR/$PRODUCT_NAME.lib";
cp "$TARGET_BUILD_DIR/$PRODUCT_NAME.lib"
"$TARGET_BUILD_DIR/../$PRODUCT_NAME.lib"; .
/Applications/Alias/maya7.0/Maya.app/Contents/bin/MayaENV.sh;maya -command
"loadPlugin $PRODUCT_NAME;";
```

Once you have implemented the auto launch feature this comes in handy to actually launch maya then load your plugin. You can even submit more commands to setup your scene just as if you were doing it from the command line.

All your bash scripts and functions are also available through this run script feature. Again this is a very powerful feature of Xcode I recommend taking full advantage of it.

Customize Xcode:
I cannot stress this enough. By default Xcode seem to act like an old school style programming IDE. This is really not the case. If you check out the preferences you will find all kinds of “syntax” highlighting features.

Under “Code Sense” you will find features for “code” hinting. This becomes especially useful when learning C++ or even learning a new library or Framework such as SDL or OpenGL.

Framework’s:
Frameworks are just another name for Library. It’s a collection of resource that you will build with your project that will allow for you project to run properly on another machine without having to install the source code and header files on you end users machine.

This means, yes you can source all the frameworks, but it’s not practical and will make your application very large. I would only add the frameworks required to build your project.

Command Line Building:
Lucky for you and me, we may never need to do this on the Mac. Maybe if you need distribute your application across many platforms you might need to create a “Make file”. A “Make” is much like your Xcode project file. It holds all the information the compiler needs to link the external libraries, internal libraries, header files, and source files.

Many Linux and Unix websites will cover Make files for Linux and these files are very similar in structure on the Mac. I do recommend looking in the API documentation for special Make files OSX platforms use for building Maya Plugins. You can find this information in the same location as the Developers Sample’s that come with the API documentation.