

How to Use the ThinkGear API in Xcode (Mac OS X)

The ThinkGear Communications Driver for Mac is deprecated and will no longer be updated.

Mental Effort and Familiarity are not included in the ThinkGear Communications Driver.

Introduction

Loadable modules (containing dynamic libraries or plugins) on Mac OS X are often packaged as bundles. These are generally analogous to `.dll` files in Windows or `.so` files in Linux and other *NIX platforms, though bundles also provide a richer set of functionality, e.g. facilities for loading non-executable assets such as localization strings or images.

Developers that want to integrate ThinkGear functionality into their OS X applications should utilize the `NSBundle` API in the Core Foundation framework to hook into `ThinkGear.bundle`. This document will describe the process of getting your Xcode project up and running with ThinkGear.

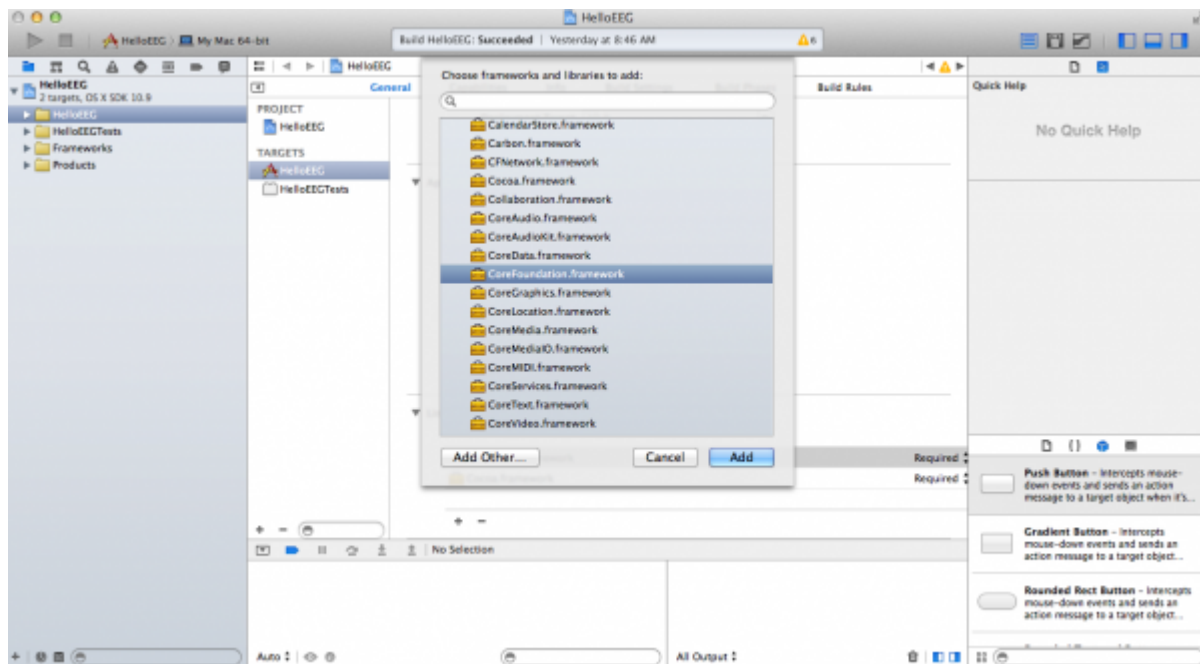
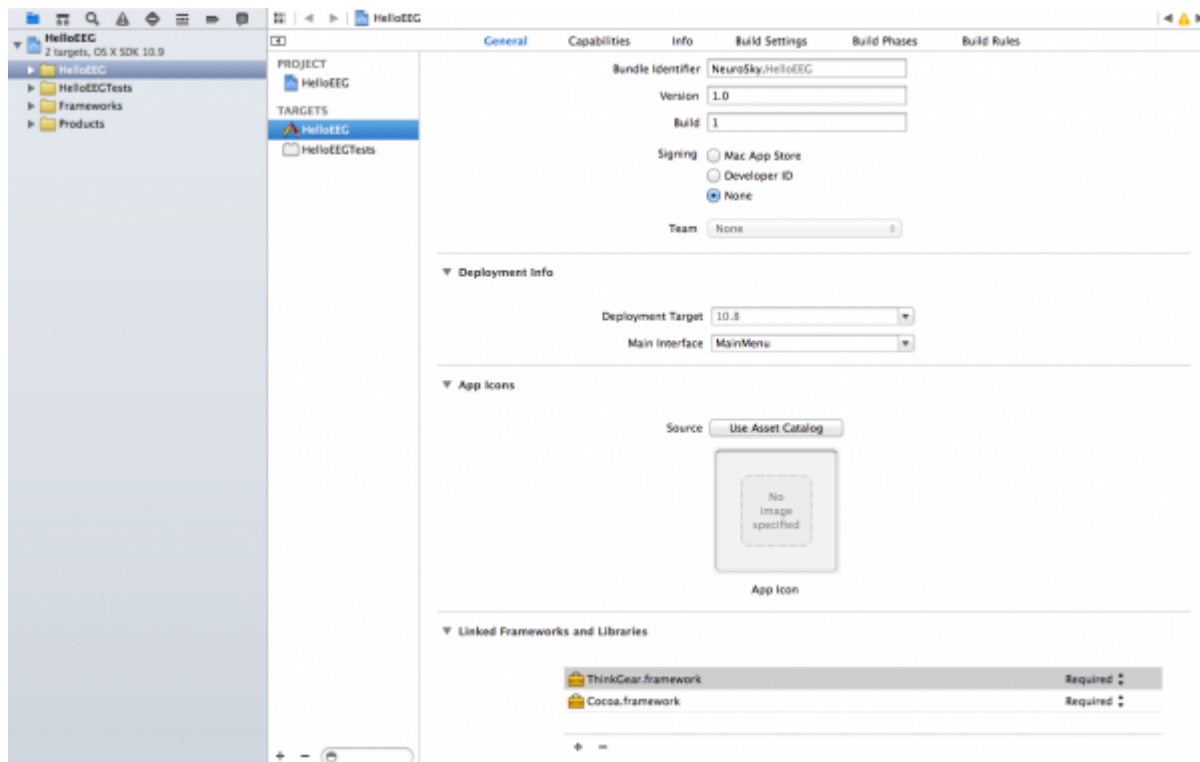
The `NSBundle` API in the Cocoa framework applies strictly to bundles containing Objective-C classes. Since ThinkGear is a C-only API, discussions of `NSBundle` are inappropriate in this context.

Setting up Xcode

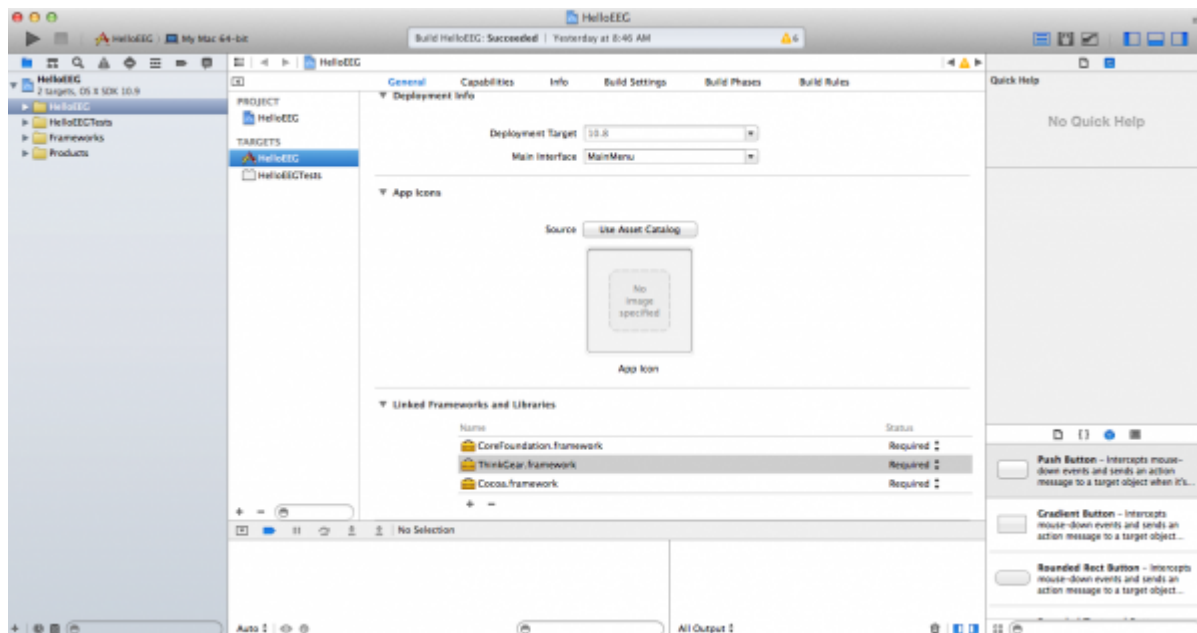
The only requirement for loading `ThinkGear.bundle` is that the Core Foundation framework be included in your project's list of external frameworks and libraries. This can be done by clicking on "HelloEEG" under "TARGETS" section in your Xcode project window as shown in the following image.

Under the general tab scroll down to **Linked Frameworks and Libraries**, then click on the triangle to expand it. Next, click the next "+" button and Look for the `CoreFoundation.framework`, and then click **Add**.

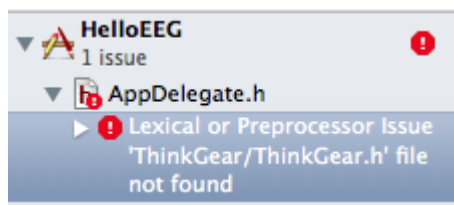
Last update: 2014/06/26 22:23
how_to_use_the_thinkgear_api_in_xcode_mac_os_x http://developer.neurosky.com/docs/doku.php?id=how_to_use_the_thinkgear_api_in_xcode_mac_os_x



The image below shows what your project window should look like once the Core Foundation framework has been added.

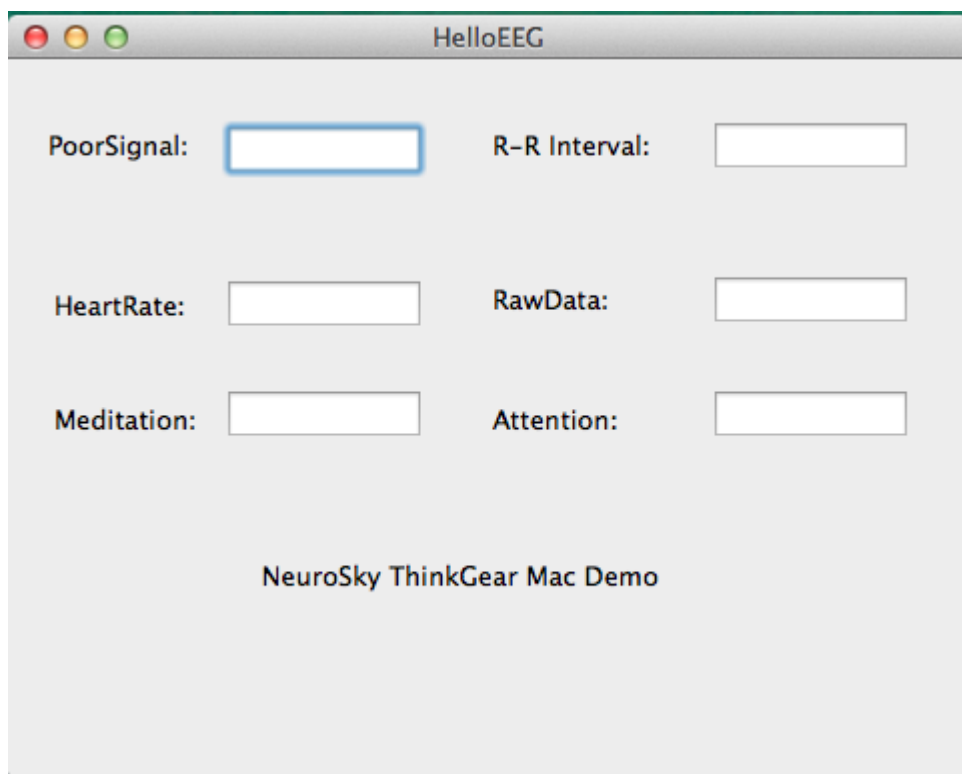


Note: If HelloEEG does not compile and you see the following error, try deleting and re-adding ThinkGear.framework under Frameworks and Libraries from the menu shown below.



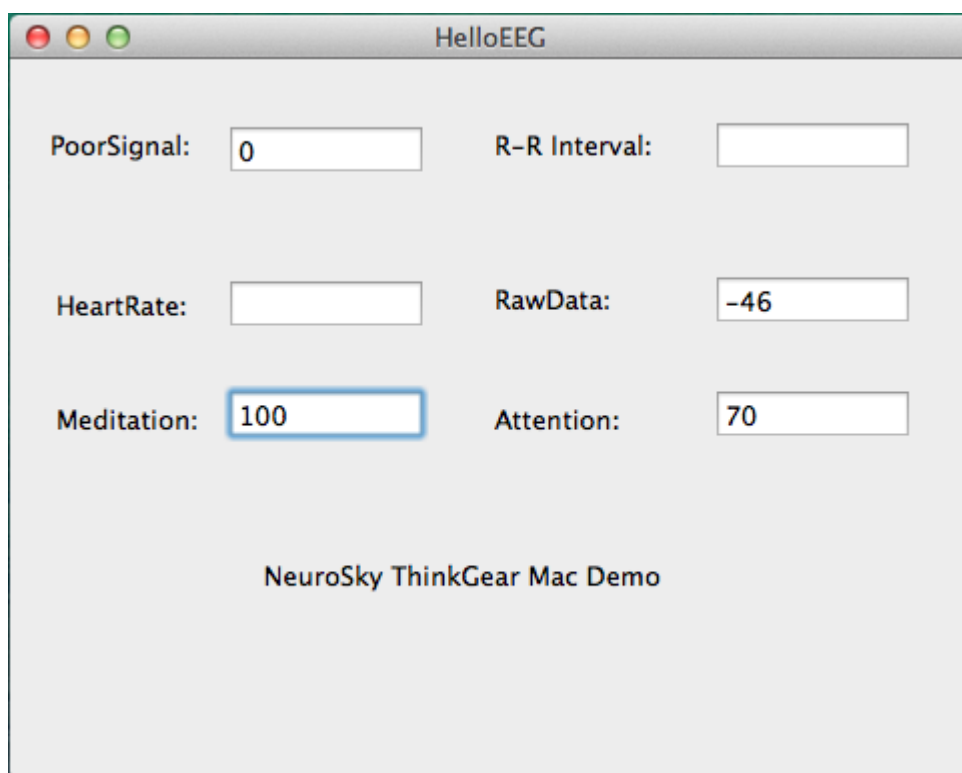
1. Click **ThinkGear.framework** and then click the - button.
2. Click the + button and then click **Add Other**
3. Find and add **ThinkGear.framework** in **TG-SDK -> TG-SDK For Mac -> lib -> ThinkGear.framework**

If HelloEEG is run without connecting the ThinkGear-enabled headset, the application should look like this:



The screenshot shows a Mac OS X window titled "HelloEEG". Inside the window, there are six text input fields arranged in a 3x2 grid. The labels for the fields are "PoorSignal:", "R-R Interval:", "HeartRate:", "RawData:", "Meditation:", and "Attention:". All input fields are currently empty. Below the grid of fields, the text "NeuroSky ThinkGear Mac Demo" is centered.

Once the headset is connected, run the application once again. The application should now look like this:



The screenshot shows the same "HelloEEG" window, but now with data populated in the input fields. The "PoorSignal:" field contains the value "0", the "Meditation:" field contains "100", and the "Attention:" field contains "70". The "R-R Interval:", "HeartRate:", and "RawData:" fields remain empty. The "RawData:" field also displays the value "-46" below the input field. The text "NeuroSky ThinkGear Mac Demo" is still centered at the bottom.

If the Poor Signal data reads 0, Meditation and Attention data should be fluctuating every second. If Poor Signal reads something other than 0, re-adjust the headset accordingly until Poor Signal data goes down to 0.

At this point, you should be able to browse the code, make modifications, compile, and deploy the app to your device or emulator just like any typical OS X application.

Importing ThinkGear Functions

At the top of your header or implementation, you should include the Core Foundation library:

```
#include <CoreFoundation/CoreFoundation.h>
```

Before importing the functions, a bundle reference (CFBundleRef) must first be created for the bundle. This is constructed from a path describing the location of the bundle, which is encapsulated in a CFURLRef object. Let's first declare these objects.

```
CFURLRef bundleURL;  
CFBundleRef thinkGearBundle;
```

And now, to initialize them:

```
bundleURL = CFURLCreateWithFileSystemPath(kCFAllocatorDefault,  
                                          CFSTR("ThinkGear.bundle"),  
                                          kCFURLPOSIXPathStyle,  
                                          true);  
  
thinkGearBundle = CFBundleCreate(kCFAllocatorDefault, bundleURL);
```

CFBundleCreate returns NULL if the bundleURL points to an invalid bundle, so it's a good idea to check that for validity before continuing. Note that the path above is a relative path, so the executable will need the bundle to be located in the same directory. Apple provides documentation on [different ways of locating bundles](#)

We then need to declare some function pointers that reference the functions inside ThinkGear.bundle. It is recommended to use the same naming scheme for the functions as is used in the API. A few examples are provided below for clarity. Refer to ThinkGear.h (provided in the ThinkGear SDK) for the function prototypes.

```
int (*TG_GetDV)() = NULL; // TG_GetDriverVersion  
int (*TG_GetNCId)() = NULL; // TG_GetNewConnectionId  
int (*TG_Connect)(int, const char *, int, int) = NULL;
```

Finally, we'll want to create the references to the ThinkGear functions. This is done using the CFBundleGetFunctionPointerForName function, which takes the bundle reference as one of its parameters. This should be done for any ThinkGear functions that you plan on using in your application.

```
TG_GetDV = (void*)CFBundleGetFunctionPointerForName(thinkGearBundle,  
                                                    CFSTR(  
"TG_GetDriverVersion"));  
TG_GetNCId = (void*)CFBundleGetFunctionPointerForName(thinkGearBundle,  
                                                    CFSTR(  
"TG_GetNewConnectionId"));
```

```
TG_Connect = (void*)CFBundleGetFunctionPointerForName(thinkGearBundle,  
                                                       CFSTR("TG_Connect"));
```

Before using these imported functions, it is prudent to check that they were successfully imported.

```
if(!TG_Connect)  
    return -1;
```

Before your application quits (or when you're done using the functions), you'll need to release the allocated Core Foundation objects; namely, the CFURLRef and CFBundleRef objects. This is effectively equivalent to an object destructor.

```
CFRelease(bundleURL);  
CFRelease(thinkGearBundle);
```

Using Imported ThinkGear Functions

The imported functions can be used as if they were normally declared and implemented in your code, e.g.

```
int retVal = TG_Connect(connectionID, "/dev/tty.MindsetMSEM1-DevB-1", 9600,  
0);  
printf("TG_Connect returned: %d\n", retVal);
```

Conclusion

By reading this document, you have familiarized yourself on how to integrate the ThinkGear library into your OS X application. A sample Xcode project, implementing a simple command-line data streamer for the headset, is included in the MindKit SDK.

References

- [Sample Xcode project with source code](#)
- [Apple CFBundle Reference](#)
- [ThinkGear API and Reference Manual](#)
- [ThinkGear API MacOSX Example](#)

From:

<http://developer.neurosky.com/docs/> - NeuroSky Developer - Docs

Permanent link:

http://developer.neurosky.com/docs/doku.php?id=how_to_use_the_thinkgear_api_in_xcode_mac_os_x

Last update: **2014/06/26 22:23**



Warnings and Disclaimer of Liability

THE ALGORITHMS MUST NOT BE USED FOR ANY ILLEGAL USE, OR AS COMPONENTS IN LIFE SUPPORT OR SAFETY DEVICES OR SYSTEMS, OR MILITARY OR NUCLEAR APPLICATIONS, OR FOR ANY OTHER APPLICATION IN WHICH THE FAILURE OF THE ALGORITHMS COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR. YOUR USE OF THE SOFTWARE DEVELOPMENT KIT, THE ALGORITHMS AND ANY OTHER NEUROSKY PRODUCTS OR SERVICES IS "AS-IS," AND NEUROSKY DOES NOT MAKE, AND HEREBY DISCLAIMS, ANY AND ALL OTHER EXPRESS AND IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, AND ANY WARRANTIES ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

IN NO EVENT SHALL NEUROSKY BE LIABLE FOR ANY SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, INCLUDING BUT NOT LIMITED TO LOSS OF PROFITS OR INCOME, WHETHER OR NOT NEUROSKY HAD KNOWLEDGE, THAT SUCH DAMAGES MIGHT BE INCURRED.