

ThinkGear SDK for iOS: Development Guide

Introduction

This guide will teach you how to use NeuroSky's **ThinkGear SDK for iOS** to write iOS applications that can utilize bio-signal data from NeuroSky's ThinkGear family of bio-sensors (which includes the CardioChip family of products). This will enable your iOS apps to receive and use bio-signal data such as EEG and ECG/EKG acquired from NeuroSky's sensor hardware.

This guide (and the entire **ThinkGear SDK for iOS** for that matter) is intended for programmers who are already familiar with standard iOS development using Xcode and Apple's iOS SDK. If you are not already familiar with developing for iOS, please first visit Apple's web site for instruction and tools to develop iOS apps.

If you are already familiar with creating typical iOS apps, then the next step is to make sure you have downloaded NeuroSky's **ThinkGear SDK for iOS**. Chances are, if you're reading this document, then you already have it. If not, the SDK can be downloaded from <http://store.neurosky.com/products/developer-tools-3-iphone>.

ThinkGear SDK for iOS Contents

- ThinkGear SDK for iOS: Development Guide (this document ios_development_guide.pdf)
- ThinkGear SDK for iOS: API Reference (thinkgear_ios_api_reference.pdf)
- libTGAccessory.a library and headers
- ThinkGearTouch example project for iOS

You'll find the "API Reference" in the TG-SDK, the "libTGAccessory.a" in the `lib/` folder, and the "ThinkGearTouch example project" in the `Sample Project/ThinkGearTouch/` folder.

Supported ThinkGear Hardware

The ThinkGear SDK for iOS must be used with a ThinkGear-compatible hardware sensor device. The following ThinkGear-compatible hardware devices are currently supported:

- MindWave Mobile for use with an iOS device's built in Bluetooth
- CardioChip Starter Kit Unit

Important: Before using any iOS application that uses the TG-SDK for iOS, make sure you have paired the ThinkGear sensor hardware to your iOS device by carefully following the instructions in the User Manual that came with each ThinkGear hardware device!

SDK Bugs and Issues

The current iteration of the SDK has the following limitations:

- The SDK does not support Automatic Reference Counting in this release.

Your First Project: ThinkGearTouch

ThinkGearTouch is a sample project we've included in the **ThinkGear SDK for iOS** that demonstrates how to setup, connect, and handle data to a ThinkGear device. Add the project to your Xcode environment by following these steps:

1. In Xcode, select **File** **Open**
2. Browse in the TG-SDK to select the Sample Project/ThinkGearTouch directory
3. Click the Open button
4. Update the code signing options in the project target settings
5. Select **Product** **Run** to compile, link and start ThinkGearTouch in the Xcode emulator.

Note: This is an example application. It may not be completely compliant with Apple's guidelines for building deploy able applications.

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 iOS application.

Developing Your Own ThinkGear-enabled Apps for iOS

For most applications, using the ThinkGear iOS API is recommended. It reduces the complexity of managing ThinkGear accessory connections and handles parsing of the data stream from these ThinkGear accessories. To make a brainwave-sensing application, all you need to do is to import a library, add the requisite setup and teardown functions, and assign a delegate object to which accessory event notifications will be dispatched.

Some limitations of the ThinkGear for iOS API include:

- Can only communicate with one attached ThinkGear-enabled accessory
- Depending on the value of the user-configured event dispatch interval, some data received from the headset may be discarded

The "ThinkGear SDK for iOS: API Reference" contains descriptions of the classes and protocols available in the ThinkGear iOS API.

The ThinkGear iOS SDK also includes the `ThinkGearTouch` sample project, which is a simple `UITableView`-based iOS application that displays the data coming from a ThinkGear hardware module.

Configuring Your Environment

In order for you app to communicate with any ThinkGear hardware module, you must include include the `UISupportedExternalAccessoryProtocols` or `Supported external accessory protocols` key in your app's `Info.plist` file. This key contains an array of strings that identify the communications protocols that your app supports. Add `com.neurosky.thinkgear` to the list of supported external accessory protocols.

Copy the following directories from the `src/lib` directory in the ThinkGear SDK for iOS into the `Libraries` group in your project:

- `libTGAccessory.a`
- `TGAccessoryDelegate.h`
- `TGAccessoryManager.h`

Your project window should now look similar to this:

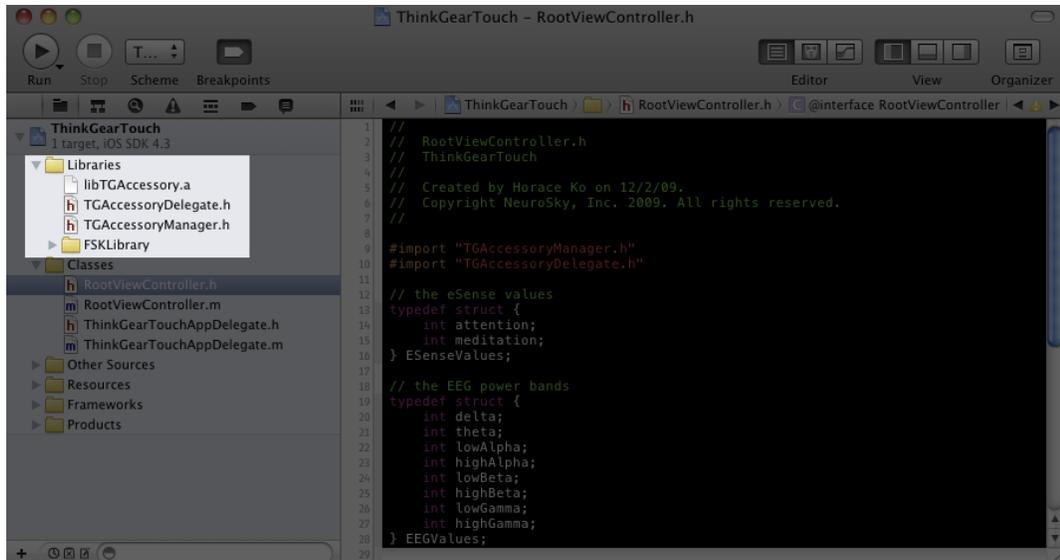


Figure 1: Xcode project window with the ThinkGear iOS library

Next, add the `Accelerate` and the `ExternalAccessory` frameworks to the project.

1. Navigate to your project settings
2. Select your target
3. Select Build Phases
4. Expand **Link Binary With Libraries**
5. Click on + and select `Accelerate.framework` and `ExternalAccessory.framework` and click **Add**

Your project window should now look similar to this:

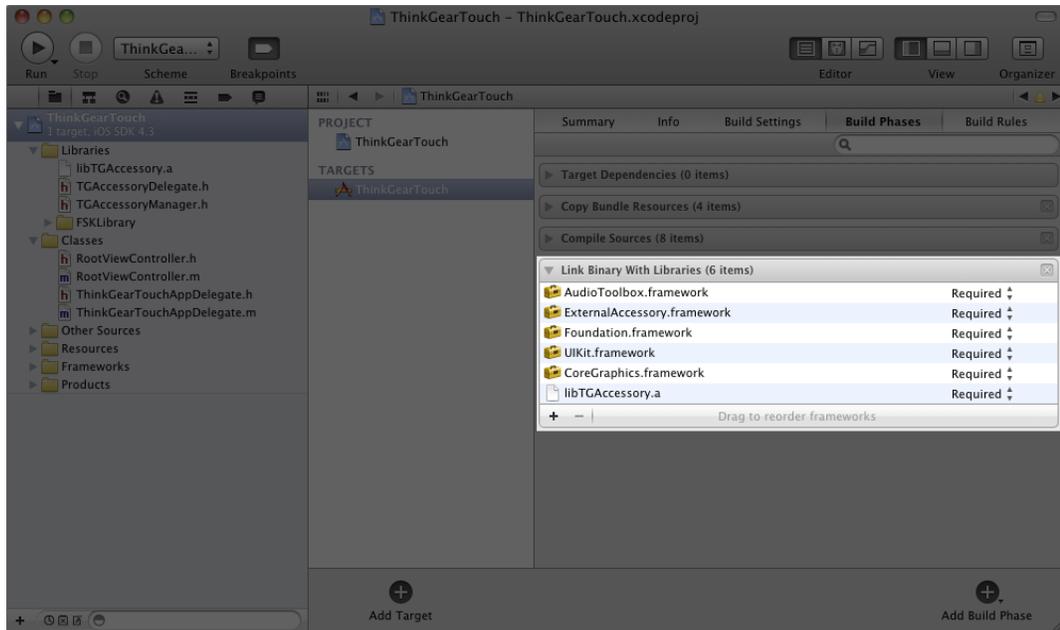


Figure 2: Add frameworks to project

Then, import the appropriate header files (`TGAccessoryManager.h` and `TGAccessoryDelegate.h`) into the requisite classes.

Setting Up the `TGAccessoryManager`

Setting up the `TGAccessoryManager` should be performed as early as necessary. Typically, this would be in the `applicationDidFinishLaunching:` method in the application delegate class. Simply add the following two lines to that method:

```
[[TGAccessoryManager sharedTGAccessoryManager] setupManagerWithInterval:0.05];
[[TGAccessoryManager sharedTGAccessoryManager] setDelegate:(RootViewController
*)[[navigationController viewControllers] objectAtIndex:0]];
```

This sets up the `TGAccessoryManager` instance to dispatch `dataReceived:` notifications every 0.05s, or roughly 20 times per second. The delegate can be set to any class that implements the `TGAccessoryDelegate` protocol — in this case, it's an instance of `RootViewController`.

Before the application quits, teardown of the `TGAccessoryManager` instance should be performed. This should be performed as late as necessary, typically in the `applicationWillTerminate:` method in the application delegate class. The following code should be added to that method:

```
[[TGAccessoryManager sharedTGAccessoryManager] teardownManager];
```

Handling Data Receipt

Since the delegate object was set to be a `RootViewController` instance, we have to edit its class definition to indicate support of the `TGAccessoryDelegate` protocol. In the sample project file, the class definition in `RootViewController.h` looks similar to the following:

```
@interface RootViewController : UITableViewController
```

Simply modify the definition in the following way:

```
@interface RootViewController : UITableViewController <TGAccessoryDelegate>
```

As a requisite of supporting the `TGAccessoryDelegate` protocol, the `dataReceived:` method must be implemented. In the header (`.h`) file, add the following method definition:

```
- (void) dataReceived: (NSDictionary *) data;
```

And in the implementation (`.m`) file, implement the method. A few `NSLog` calls are provided as a trivial example of accessing the `data` parameter. Check the "ThinkGear SDK for iOS: API Reference" for a full list of the supported keys.

```
- (void) dataReceived: (NSDictionary *) data {
    NSLog(@"Data received!");
    NSLog(@"Raw: %d", [[data valueForKey:@"raw"] intValue]);
    NSLog(@"Attention: %d", [[data valueForKey:@"eSenseAttention"] intValue]);
}
```

Handling Accessory Connection and Disconnection

The `TGAccessoryDelegate` protocol also specifies two methods for the delegate object to handle accessory connection and disconnection — `accessoryDidConnect:` and `accessoryDidDisconnect:`. Add the following method definitions to the header file:

```
- (void) accessoryDidConnect: (EAAccessory *) accessory;
- (void) accessoryDidDisconnect;
```

In the implementation file, implement these methods:

```
- (void) accessoryDidConnect: (EAAccessory *) accessory {
    NSLog(@"%@ was connected to this device.", [accessory name]);
}

- (void) accessoryDidDisconnect {
    NSLog(@"An accessory was disconnected.");
}
```

Starting the Data Stream

When your application is ready to receive the headset data, call the `startStream` method in `TGAccessoryManager`. In the sample project, this is done in the `viewWillAppear:` method. It is advisable to check whether an accessory was found by the `TGAccessoryManager` before starting the data stream:

```
if([[TGAccessoryManager sharedTGAccessoryManager] accessory] != nil)
    [[TGAccessoryManager sharedTGAccessoryManager] startStream];
```

You will also need a matching call to `stopStream` in the `viewWillDisappear:` method. Again, it is advisable to make sure that a data stream is connected and active before closing it:

```
if([[TGAccessoryManager sharedTGAccessoryManager] connected])
    [[TGAccessoryManager sharedTGAccessoryManager] stopStream];
```

Application lifecycle

On devices that support multitasking, your application should expect the following behavior:

- Upon entering the background, `accessoryDidDisconnect:` will be called.
- Upon returning from the background, `accessoryDidConnect:` will be called.

Your application **must** restart the data stream when resuming from the background. For example:

```
- (void) accessoryDidConnect: (EAAccessory *) accessory {
    [[TGAccessoryManager sharedTGAccessoryManager] startStream];
}
```

Before your application is terminated, you **must** stop the manager if you have not done so already.

```
- (void) applicationWillTerminate: (UIApplication *) application {
    [[TGAccessoryManager sharedTGAccessoryManager] teardownManager];
}
```

Log messages

The `TGAccessory` library will emit some debug messages through `NSLog()` to help you develop and debug your application. These messages will be prefixed with "TGAccessory:".

Further Considerations

- The application should not expect there to be a ThinkGear accessory attached to the iOS-based device on startup. As such, it should handle that case accordingly (e.g. by displaying a static splash screen prompting the user to connect a ThinkGear accessory).
- Provide a consistent user experience by adhering to the guidelines set by the [NeuroSky Developer Application Standards](#) document.

References

- [Communicating with External Accessories](#) (Apple documentation)
- [EAAccessoryManager Class Reference](#)
- [EAAccessory Class Reference](#)
- [EASession Class Reference](#)

Corporate Address

NeuroSky, Inc.
125 S. Market St., Ste. 900
San Jose, CA 95113
United States
(408) 600-0129

Questions/Support: <http://support.neurosky.com>
or email: support@neurosky.com

Community Forum: <http://developer.neurosky.com/forum>

Information in this document is subject to change without notice.

Reproduction in any manner whatsoever without the written permission of NeuroSky Inc. is strictly forbidden. Trademarks used in this text: eSense™, ThinkGear™, Mind-Kit™, NeuroBoy™ and NeuroSky® are trademarks of NeuroSky, Inc.

Disclaimer: The information in this document is provided in connection with NeuroSky products. No license, express or implied, by estoppels or otherwise, to any intellectual property rights is granted by this document or in connection with the sale of NeuroSky products. NeuroSky assumes no liability whatsoever and disclaims any express, implied or statutory warranty relating to its products including, but not limited to, the implied warranty of merchantability, fitness for a particular purpose, or non-infringement. In no even shall NeuroSky be liable for any direct, indirect, consequential, punitive, special or incidental damages (including, without limitation, damages for loss of profits, business interruption, or loss of information) arising out of the use of inability to use this document, even if NeuroSky has been advised of the possibility of such damages. NeuroSky makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. NeuroSky does not make any commitment to update the information contained herein. NeuroSky's products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.