CardioChip Developer Guide for iOS

Features

- Develop iOS applications that utilize ThinkGear technology
- Downloadable ThinkGear-enabled sample iOS project with full sample code
- Uses the iOS External Accessories API (available in iPhone OS 3.0+) or the ThinkGear iOS API

Introduction

This application note will walk you through the process of creating a ThinkGear-capable iOS application.

This application note documents the usage of two different APIs to connect to a MindWave Mobile or other compatible 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.
- There are some static analyzer warnings in the FSKLibrary, they are safe to ignore.

Hardware

The ThinkGear iOS API supports the following hardware:

- The MindWave Mobile which connects though an iOS device's built in Bluetooth
- The CardioChip Starter Kit

Using the ThinkGear iOS API

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 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_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 (contained in `src/`), which is a simple `UITableView`-based iOS application that displays the data coming from a MindSet headset.

Configuring Your Environment

In order for your app to communicate with the MindWave Mobile, you must 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 iOS SDK into the `Libraries` group in your project:

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

Your project window should now look similar to this:

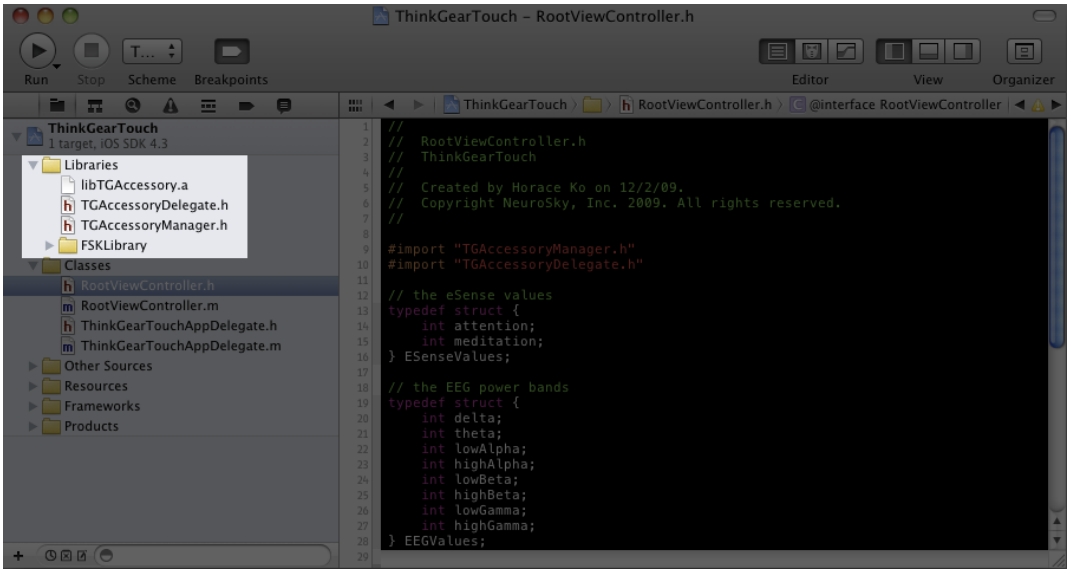

Figure 1: Xcode project window with the ThinkGear iOS library

Next, add the AudioToolbox 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 `AudioToolbox.framework` and `ExternalAccessory.framework` and click **Add**

Your project window should now look similar to this:

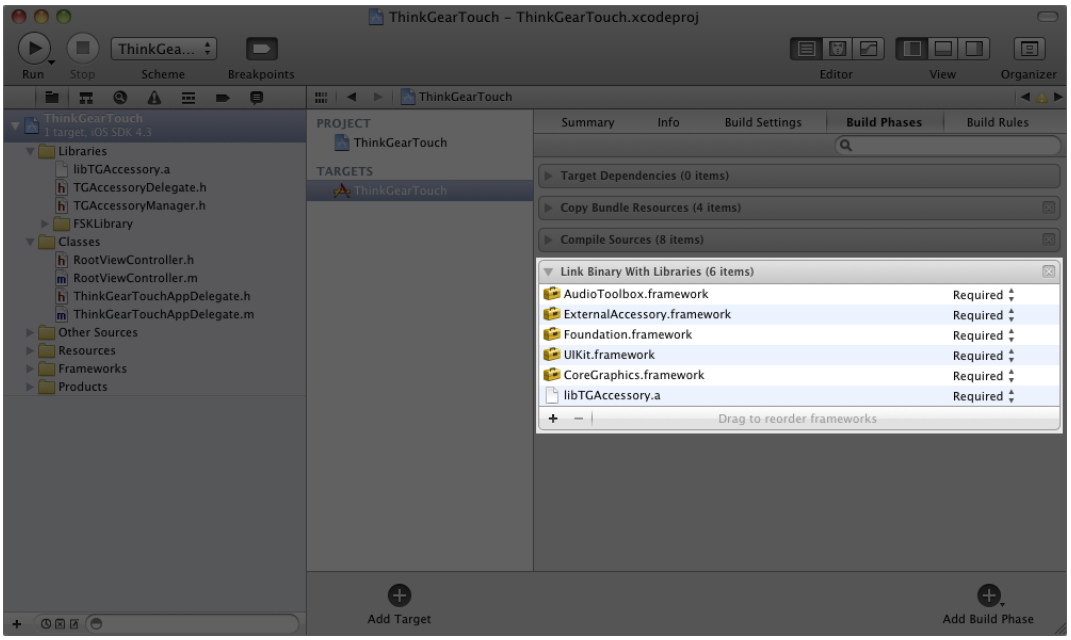

Figure 2: Add frameworks to project

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 `dataRecieved:` 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_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 Accesssory 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] startStream];
```

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 consistant user experience by adhering to the guidelines set by the App 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

NeuroSky
Brain-Computer Interface Technologies