Overview

During this lab, you will work with a partner to create two applications: one app that sends a Bluetooth signal when the accelerometer detects a shake, and one that receives Bluetooth messages and plays a sound. In the first two parts, you will ensure you can get the accelerometer and sound working independent of using Bluetooth.

- Part 1: Introduction to Accelerometer
- Part 2: Introduction to Sounds
- Part 3: Using Bluetooth
- Part 4: Guitar App Project

Special Due Date

This lab has a two-week deadline to ensure you and your partner have enough time to work on the project. Be sure to have it turned in by the end of your lab session in two weeks, either **October 9th, October 11th, or October 12th**.

There will be another lab started next week with the same due date as this lab.

Submission

Lab evaluation form
Online Lab feedback form

Prelab

First, read through the lab manual. Next, answer the questions on the lab evaluation form by reviewing Apple's Documentation, this lab manual, or using Google.

Topics to look up:

Classes: UIAccelerometer (singleton), UIAcceleration, UIAccelerationValue (wrapper on double), NSBundle
Delegate Protocol: UIAccelerometerDelegate
Multimedia Programming

Part 1: Introduction to Accelerometer

Step 1) Ensure you have a partner for this lab (you will need two devices).

Step 2) Create a new project called **AccelSender** using the Single View Application template.

There are three main methods for determining a "shake" event: shaking-motion events, Core Motion, and interpreting the raw accelerometer date. This tutorial describes the last approach.

Step 3) Read the description on protocols and delegates below:

Delegation is where one class "delegates" another class in order to pass off work to it. The class passing off work defines a protocol; the delegate object then implements the protocol's methods. Simple event handling, like you have done with IBActions, works fine for delegating simple event, such as button presses, to the ViewController. Delegate protocols allow developers to define more complicated interfaces to handle more complicated events.

In this part, we want one of our view controllers to respond to acceleration events. Adding <UIAccelerometerDelegate> to the class definition imports the Delegate protocol and tells the compiler you will implement the protocol. When you add a delegate protocol in this way, your class is given a list of optional and mandatory methods to implement. For UIAccelerometerDelegate, there is one optional method to implement, ***accelerometer:didAccelerate:****. Using delegates is* *a way to ensure that an object to which you delegate work has a standardized set of methods that the delegating object can send messages.*

Your view controller could be the delegate object for multiple objects. For such cases, the delegate definitions are separated by commas. For example:

@interface ViewController : UIViewController <UITextFieldDelegate, UIAccelerometerDelegate>

Step 4) Add the delegate protocol for UIAccelerometerDelegate to your view controller's header file.

@interface ViewController : UIViewController ***<UIAccelerometerDelegate>***

Step 5) Register your view controller to be the delegate object for acceleration events. To do this, uncomment the viewDidLoad method of your view controller and add the following line of code:

```
- (void)viewDidLoad {
    [super viewDidLoad];

    [UIAccelerometer sharedAccelerometer].delegate = self;  // tell the singleton instance of UIAccelerometer
                                                            // that this view controller (self) wants to be
                                                            // delegated work
}
```

Step 6) Implement the delegate protocol by adding an implementation of the **accelerometer:didAccelerate:** method.

```
- (void)accelerometer:(UIAccelerometer *)accelerometer didAccelerate:(UIAcceleration *)acceleration {
    //NSLog(@"Acceleration Event (x:%f, y:%f, z:%f)", acceleration.x, acceleration.y, acceleration.z);

    // TODO - Implement Step 8 Here
}
```

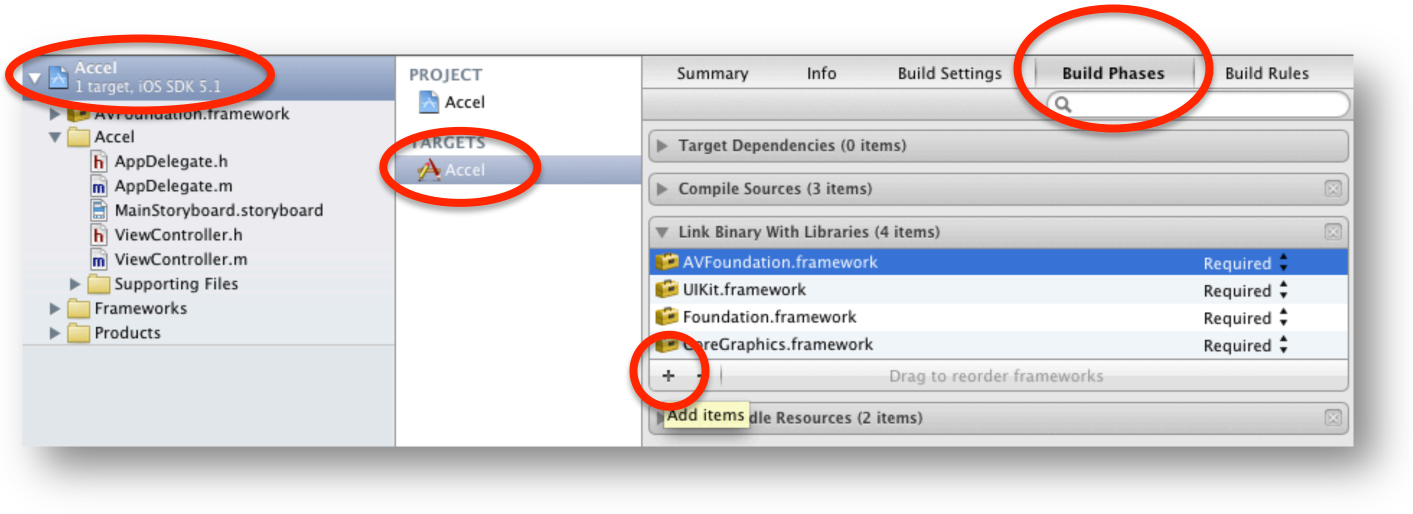
Step 7) Implement logic inside the **accelerometer:didAccelerate:** method that will detect when the iPod Touch is strummed. You will use this logic later in Part 4.

One of the simplest implementations might store the last acceleration and simply check if a new acceleration event has changed beyond a certain threshold, then alert the user using NSLog.

Step 8) Follow the steps from lab 1 to install the app on a device. You should use the device to test the accelerometer, as the simulator will not send any acceleration events (though you may send your app a Shake Gesture from the Hardware file menu). You may show the TA your code to detect strumming now or later as part of the Guitar App Project (Part 4).

Part 2: Introduction to Sound

Step 1) Create a new project called **SoundReceiver** using the Single View Application template.
Step 2) Download the sounds.zip from the BB Learn. Optionally, you may find different sounds on the internet and use them.
Step 3) Add the sound files to your project. You can do this by right-clicking in the Xcode Project Navigator and selecting *Add Files to "Project Name"*.
Step 4) Add the `AVFoundation` framework to your project. You can do this by selecting your project settings, selecting the *Build Phases* tab for your target, and adding the framework in the *Link Binary with Libraries* group.


Accel
1 target, iOS SDK 5.1
AVFoundation.framework
Accel
AppDelegate.h
AppDelegate.m
MainStoryboard.storyboard
ViewController.h
ViewController.m
Supporting Files
Frameworks
Products
PROJECT
Accel
TARGETS
Accel
Summary
Info
Build Settings
Build Phases
Build Rules
Target Dependencies (0 items)
Compile Sources (3 items)
Link Binary With Libraries (4 items)
AVFoundation.framework Required
UIKit.framework Required
Foundation.framework Required
CoreGraphics.framework Required
Drag to reorder frameworks
Add items
Copy Bundle Resources (2 items)

Step 5) Import AVFoundation.h from the AVFoundation framework. An easy was to import the header file in all of your files is to add the following line to you Prefix.pch file (locating in the *Supporting Files* folder):

```
#import <AVFoundation/AVFoundation.h> //Imports all header files in the framework
```

Or you could simple add the following line to the header file of the view controller that uses audio:

```
#import <AVFoundation/AVFoundation.h>
```

Step 6) Apple recommends using the AVAudioPlayer:

```
NSString *path = [[NSBundle mainBundle] pathForResource:@"mysound" ofType:@"wav"];
NSURL *fileURL = [NSURL fileURLWithPath:path];

AVAudioPlayer *newPlayer =
                    [[AVAudioPlayer alloc] initWithContentsOfURL:fileURL error:nil];
[newPlayer play];
```

Using the code above as an example, create five buttons that each play one of the sounds you downloaded.

Step 7) Test your app in the simulator or on the device. You can show the TA that you can successfully play sounds now or later as part of the Guitar App Project (Part 4).

Part 3: Using Bluetooth

Step 1) Reopen your **AccelSender** project.
Step 2) Add the **GameKit framework** to your project.
Step 3) Import any necessary header files.

```
#import <GameKit/GameKit.h>
```

Step 4) Add the protocols for GKPeerPickerControllerDelegate and GKSessionDelegate to your view controller:

@interface ViewController : UIViewController ***<GKPeerPickerControllerDelegate, GKSessionDelegate>***

Tip: You can review a list of methods the protocol specifies by ctrl-clicking (right-clicking) on the delegate protocol in Xcode and selecting Jump to Definition.

Step 5) Add the following instance variables to your view controller's header file:

```
GKPeerPickerController *myPicker;
GKSession *mySession;
NSString *myPeerID;
```

Step 6) First, we'll want the Peer Picker to show up on the screen. Edit the viewDidLoad method and add the following:

```
- (void)viewDidLoad {
    [super viewDidLoad];

    myPicker = [[GKPeerPickerController alloc] init];
    myPicker.delegate = self;
    myPicker.connectionTypesMask = GKPeerPickerConnectionTypeNearby;
    [myPicker show];
}
```

Step 7) Next, you'll want to start implementing the GKPeerPickerControllerDelegate protocol. Add the following method in the implementation block of your view controller:

```
- (void)peerPickerController:(GKPeerPickerController *)picker
              didConnectPeer:(NSString *)peerID
                   toSession:(GKSession *)session
{
    myPeerId = peerID;
    mySession = session;
    mySession.delegate = self;
    [mySession setDataReceiveHandler:self withContext:nil];

    // Remove the picker
    [picker dismiss];
}
```

Step 8) To send data over your Bluetooth connection, use the **sendDataToAllPeers:withDataMode:error:** method on your instance of GKSession, mySession.

```
NSData *data = [@"Hello World" dataUsingEncoding:NSUTF8StringEncoding];
[mySession sendDataToAllPeers:data withDataMode:GKSendDataUnreliable error:nil];
```

Stpe 9) Edit AccelSender so that once a connection has been established, it starts sending messages to its peers whenever the user shakes the device.

Step 10) Reopen your **SoundReceiver** project. Repeat steps 2, 3, 4, 5, and 7 for this project.

Step 11) Look at the protocol definition for GKSessionDelegate; ctrl-click (right click) on GKSessionDelegate in your header file and select *Jump to Definition*. Decide which method seems mandatory for accepting an incoming connection and implement it.

One interesting function is the *setDataReceiveHandler:withContext:* method. If we set the view controller as the data receive handler, we expect the view controller to handle the following method to accept data:

```
- (void) receiveData:(NSData *)data
            fromPeer:(NSString *)peer
           inSession:(GKSession *)session
             context:(void *)context
{
    // TODO - implement
}
```

Therefore, to receive data, you'll need to implement this method in your view controller and figure out how to register your view controller using the setDataReceiveHandler.

Step 12) Finish implementing Bluetooth connectivity and make it so one device can send a message to another.

An easy implementation would be to press a button on the AccelSender device, making it call sendDataToAllPeers:withDataMode:error, and having the other device print a message using NSLog in the receiveData:fromPeer:inSession method.

Step 16) When testing, make sure you enable Bluetooth on both of your devices (Goto Settings->General->Bluetooth). The Bluetooth logo by the battery should turn Blue when Bluetooth is being used.

Step 17) Show the TA you successfully linked two iPods together now or later as part of the Guitar App Project (Part 4).

Part 4: Guitar App Project

Step 1) Use what you have learned from parts 1, 2, and 3 to create a Guitar App. This app requires two iPods. The first iPod (the strummer) uses the accelerometer to detect strumming events and sends a message over Bluetooth to the second. The second iPod (the frets) uses UIButtons to select a sound file, and plays the selected sound file when it receives a message over Bluetooth.
Step 2) After you have tested your application, show the TA your accomplishments and turn in your completed lab evaluation form and a lab feedback form.