Stream iPhone Sensor Data to Adafruit IO

Created by Trevor Beaton

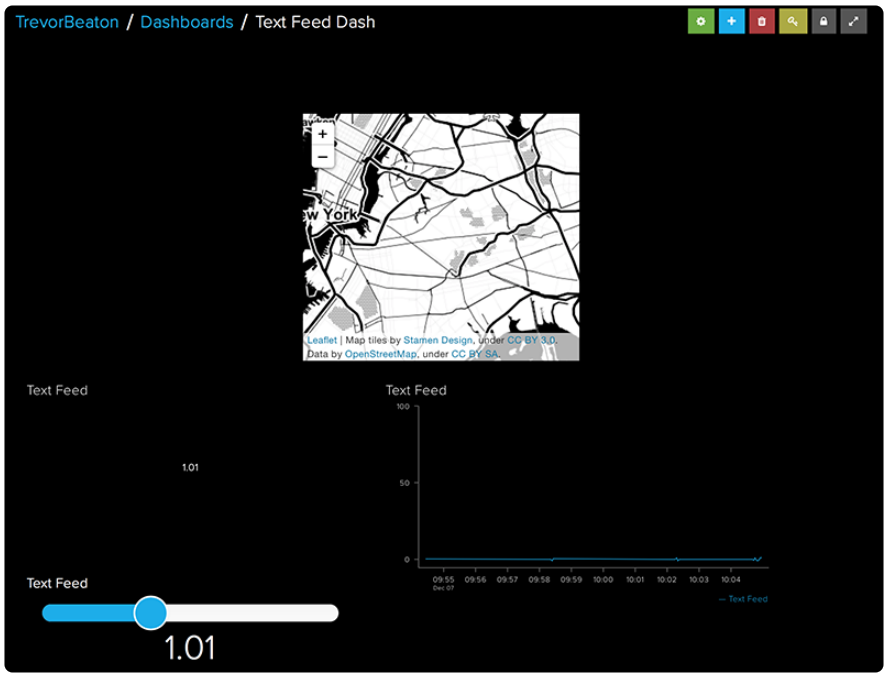
https://learn.adafruit.com/stream-iphone-ios-sensor-data-to-adafruit-io

Last updated on 2022-12-01 02:57:50 PM EST

Table of Contents

Overview

Sending your iPhone or iPad sensor data to Adafruit IO () is a great way to visualize your sensor data and also connect to an IO project from your mobile device. Adafruit IO also provides UI elements like sliders, charts, buttons, and switches to help you with your IoT (Internet of things) projects without extra programming.



This project requires a Mac computer.

In this guide, I'll take you through the steps necessary to stream your iOS iPhone or iPad sensor data to Adafruit IO using the Swift programming language. iOS mobile devices contains a handful of sensors such as Gyroscopic sensors (), a Barometer (), Magnetometer () and much more. In this guide we'll be sending our Accelerometer sensor ()data to our Adafruit IO account feed. This guide doesn't require an extensive knowledge of Swift.

For reference, this guide provides an example app called IO Connect Example. This example app sends your iPhone accelerometer sensor data to Adafruit IO.

In this learn guide we will:

• Setup an iOS app project in Xcode.

If you don't have Xcode installed on your Mac computer, check out the Installing Xcode () portion of the Introduction to iOS Development () learn guide.

• Create a function that helps display accelerometer data updates and a function to stop updates.

• Learn about REST to make a POST request with the REST API.

• We'll set up labels in the main.storyboard to display your accelerometer data. We will also add a switch to start and stop updates.

Your completed project will consist of an iOS app that sends our iPhone or iPad Accelerometer sensor data to Adafruit IO using a HTTP "POST" request and JSONSerialization.

Before we start...

• Make sure your Xcode IDE is up-to-date (version 8.3.3 or newer)

• While in Xcode, make sure the development target is set to 10.3 or higher.

• If you haven't setup your Adafruit IO account with a Feed and Dashboard, you should do so before beginning the guide.

If you're not familiar or if you need help getting started with Adafruit IO, check out our learn guide:

Get started with Adafruit IO ()

Downloading IO Connect Example

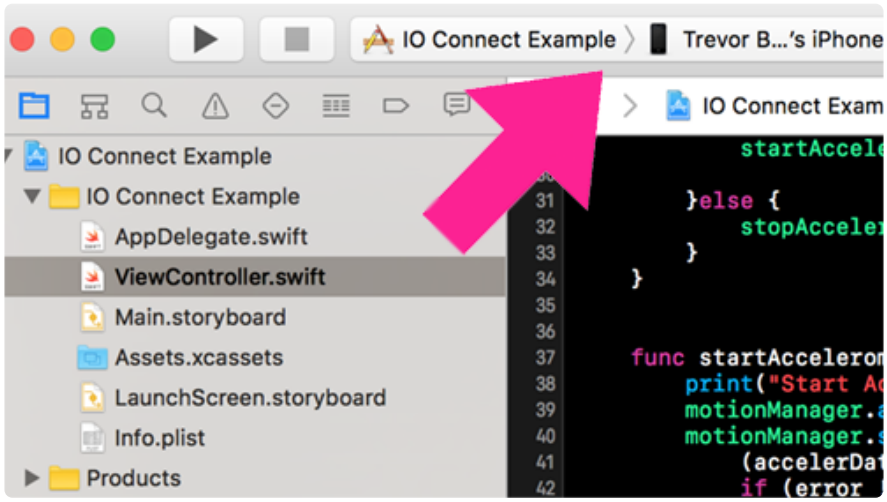
You can download the project here:

Download IO Connect Example

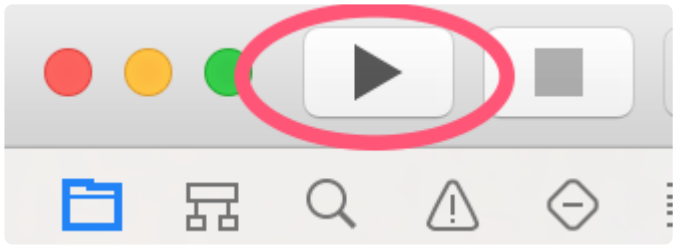
Once the file is downloaded, locate and click on the IO Connect Example.xcodeproj fil e and it will open up the project in Xcode.

Once we have the project open, select your iOS device's scheme in Xcode.

We will not be using the Xcode Simulator in this learn guide.



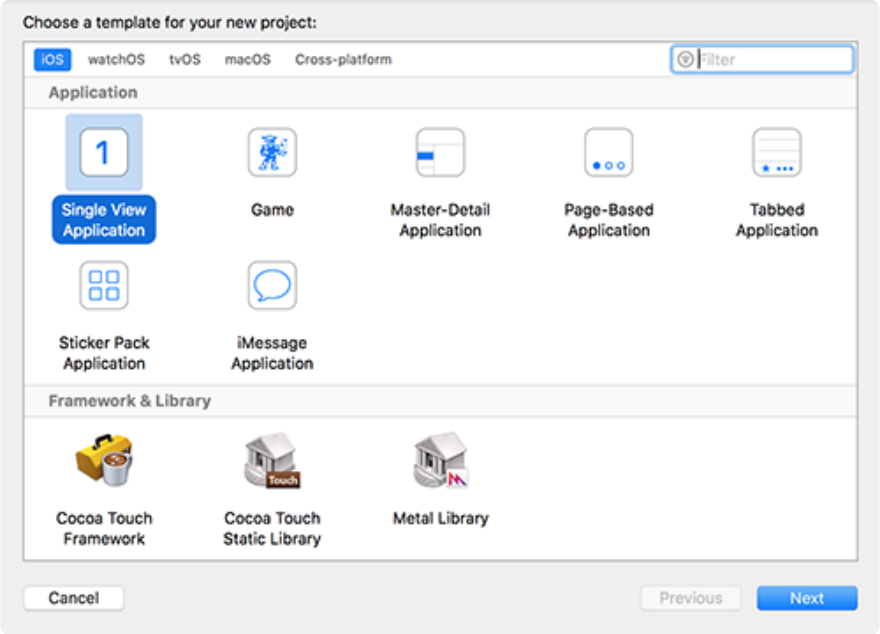
Now just press the Run button to test the app on your iOS device.



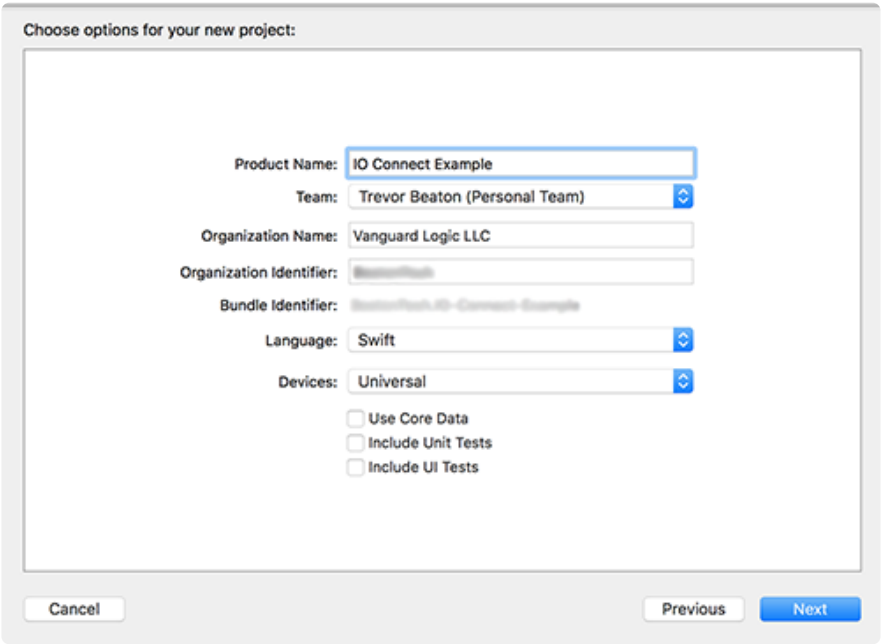
If all goes well, the IO Connect Example app will run on your iOS device and you'll be able to use the app as a reference while you explore the guide.

Getting Started

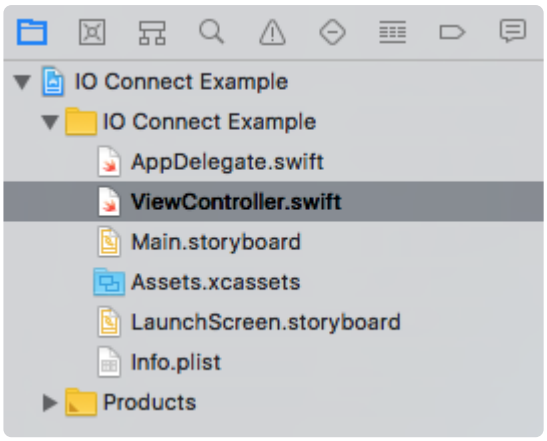
First, open Xcode and create a new project. The app we're making will be a single-view application. Select the Single-View Application icon then proceed by clicking Ne xt.



Now give your app a name in the Product Name text field. Once you've chosen a product name, make sure Use Core Data, Include Unit Tests and Include UI Tests are all unchecked before you click Next to save your project on your computer.



Before we start, check out the project's file hierarchy on the left side of the project window. The View Controller file is where you'll be entering your code and the Main.storyboard file is where you'll be adding UI to your app.



In the ViewController.swift file, we'll begin by importing the CoreMotion framework to our project. Below the `import UIKit` we will add:

```
import CoreMotion
```

The `CoreMotion` framework gives us the ability to receive and handle accelerometer data and other motion data such as gyroscope and barometer data for our app.

Next, we'll gain access to our acceleration data, so we'll create an instance of CMMoti onManager to access it in the ViewController class.

```
var motionManager = CMMotionManager()
```

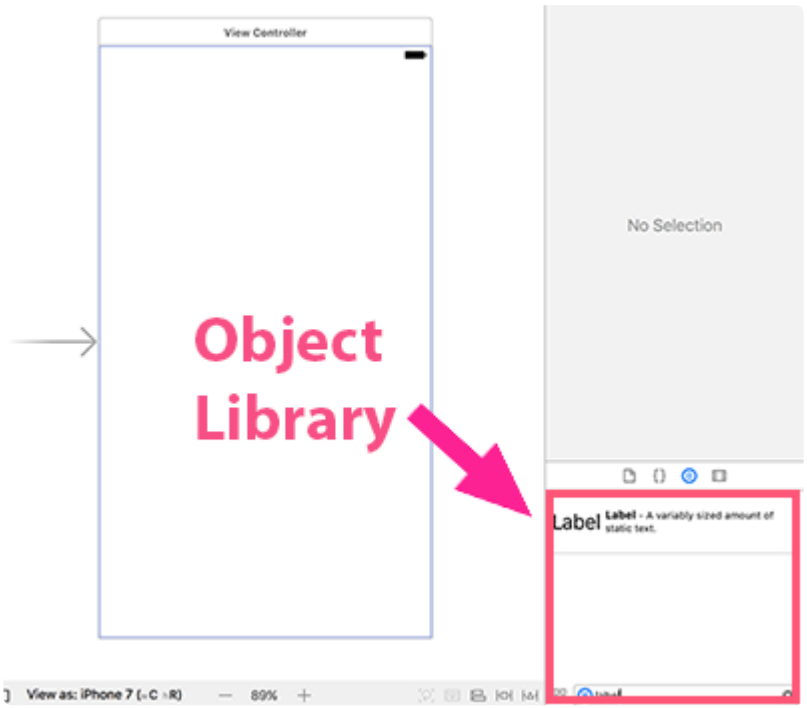
CMMotionManager provides services like accelerometer data, rotation-rate data, magnetometer data, and other device-motion data.

Before we go further, let's set up a user interface so that we can see our accelerometer sensor data.

Setting up the UI

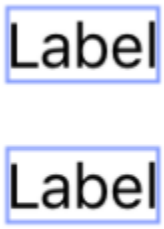
The user interface will be pretty simple since we're only displaying one value. In the M ain.storyboard we're going to give our app two labels and a switch. One of the labels will display our accelerometer data as it updates and our switch will stop and start the updates.

Let's create our labels.



Select the Main.storyboard file. You can find the label and switch in the Object Library on the right lower corner of our project window.

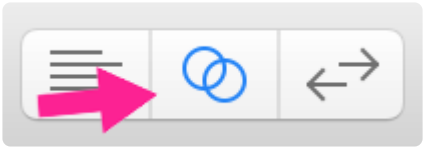
Click and drag two Labels to the center of the storyboard. Place one Label over the other Label. It should look like this.



Then search for a Switch, and do the same.

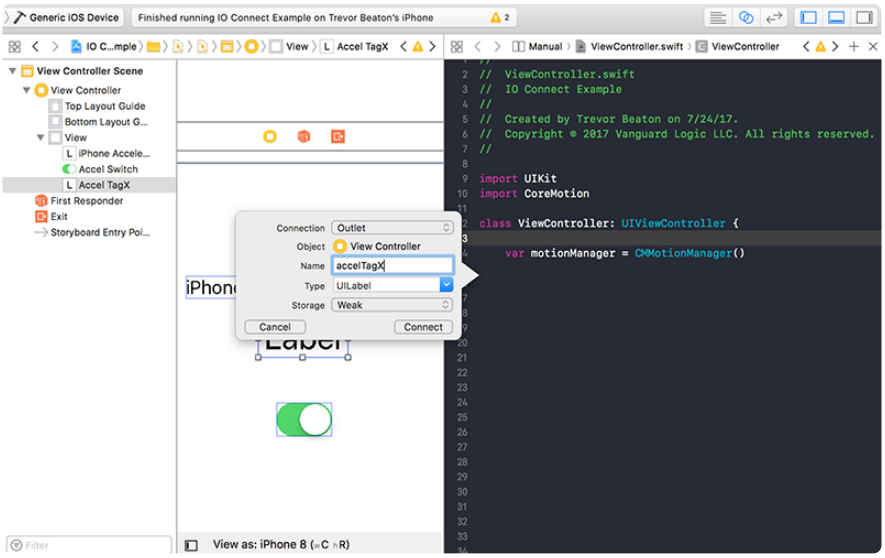
Change the top label's text to "iPhone Accelerometer-X Data:" by double clicking the label.

Then, open the assistant editor menu. Your main.storyboard and view controller class should appear side-by-side.



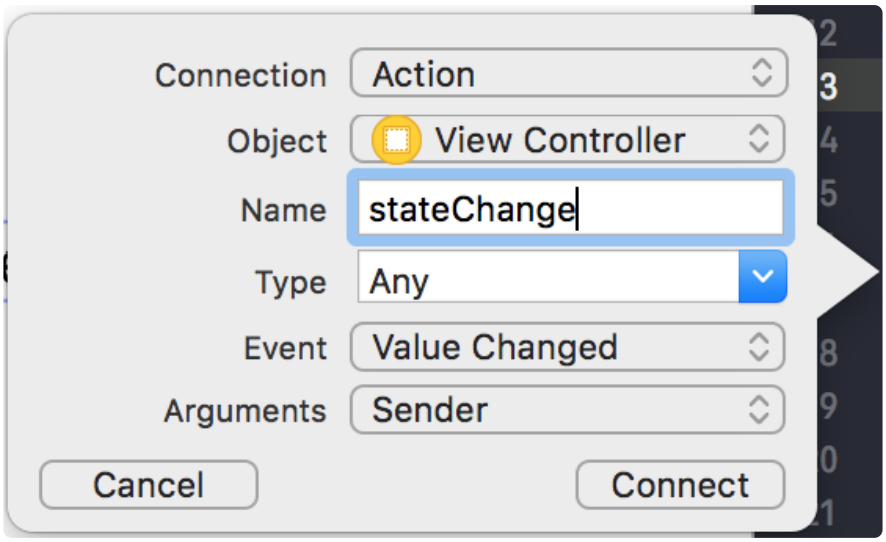
We'll need to create a reference to one of our labels. Select the bottom label then con trol+drag and drop to the view controller window.

A smaller window will pop up, here you'll give the label its reference name. For the sake of the project, we'll name this reference accelTagX.



We'll do the same with the switch UI and we will name the switch accelSwitch.

Select the switch again, we'll control+drag and drop into the view controller, but this time in the small pop-up window, we'll select the Connection drop down menu and select Action instead of Outlet, then we'll name the action "stateChange".



After you've done that, in the Type drop down option, change Any to UISwitch. When you're all done, hit Connect.

Let's head back to our view controller to create a function that will get us our accelerometer data.

Getting Accelerometer data

Now, in our ViewController.swift file we're going to create a function that will give us our accelerometer data. Here's the function that I've created:

```
func startAccelerometerX () {
        motionManager.accelerometerUpdateInterval = 2.5
        motionManager.startAccelerometerUpdates(to: OperationQueue.current!,
withHandler: {
            (accelerData:CMAccelerometerData?, error: Error?) in
            if (error != nil ) {
                print("Error")
            } else {

                let accelX = accelerData?.acceleration.x
                self.accelTagX.text = String(format: "%.02f",
accelX!)
                print("Accelerometer X: \(accelX!)")
            }
        })
    }
```

First, we'll create a function called startAccelerometerX. Within our function we'll call the `accelerometerUpdateInterval` method to receive updates of the accelerometer's current x-axis position in intervals of the time we choose.

Here, we'll set the interval time to 2.5 seconds:

```
motionManager.accelerometerUpdateInterval = 2.5
```

Next, we'll start acceleration updates with the `startAccelerometerUpdate` method. This method starts accelerometer updates on an operation queue with a specified handler.

```
motionManager.startAccelerometerUpdates(to: OperationQueue.current!, withHandler: {}
```

Within the handler, if we run into an error, the string "error" will be printed in the console. If we don't encounter an error, we'll run the else statement where we'll create a constant called accelX and give it the value of the acceleration data on the x-axis:

```
 let accelX = accelerData?.acceleration.x
```

Now that we have our accelerometer data, we'll give our accelTagX label the accelX value so that our label will display our accelerometer data.

```
self.accelTagX.text = String(format: "%.02f", accelX!)
```

Now we are going to create a new function that stops the startAccelerometerX function from updating.

Create a new function, and call it stopAccelerometerX. This one is pretty straightforward, we'll call the stopAccelerometerUpdates method to stop accelerometer updates. Then we're going to change the accelTagX label to "--".

```
    func stopAccelerometerX () {
        self.motionManager.stopAccelerometerUpdates()
        self.accelTagX.text = "--"
        print("Accelerometer X Stopped")
    }
```

Ok, let's test it out. In the `viewDidLoad` function lets call our startAccelerometerX function to make sure everything is running as it should.

```
    override func viewDidLoad() {
        super.viewDidLoad()
        startAccelerometerX()
    }
```

Hit the Build button to run your app. If everything runs smoothly, the app should launch on your iOS device. As you turn your device, you should see the accelX label display your iOS accelerometer data as your device tilts.

If your label is not updating, try going through the guide again or look at the IO Connect Example app.

Next, we are going to send our accelX data to Adafruit IO using the REST "POST" method.

Sending Data with REST "POST" Method

What is REST?

The REST API is an application program interface that uses HTTP request method to transfer data between a client and a server. REST is based on REpresentational State Transfer, the architectural style that underpins the internet.

We use HTTP resource methods for REST. There are a handful of different methods:

• POST - Creates new resources or creates new data. When we creating and sending newer data.

• GET - Gives you the ability to read and receive new data.

• PUT - This method is usually used to updating or replace data that's being sent.

• DELETE - This method is pretty straightforward. This method deletes data.

For this guide we'll be using the "POST" method.

Post Data to Adafruit IO

Since we've previously collected accelerometer data, we'll continue by sending that data to your Adafruit IO feed. In the ViewController.swift file, we'll create a function named `postAccelerometerDataX()`.

Then we'll create a constant dictionary and attach our accelerometer data to the dictionary in a string format:

```
let parameters = ["value": "\(String(format: "%.02f",
(motionManager.accelerometerData?.acceleration.x)!))"]
```

Now we're going to setup the URL request. We'll use a guard let statement so that we can check if the URL we've given is valid:

```
guard let url = URL(string: "https://io.adafruit.com/api/feeds/your-Feed-Key-Here/
data.json?X-AIO-Key=Your-A-IO-Key-Here) else { return }
```

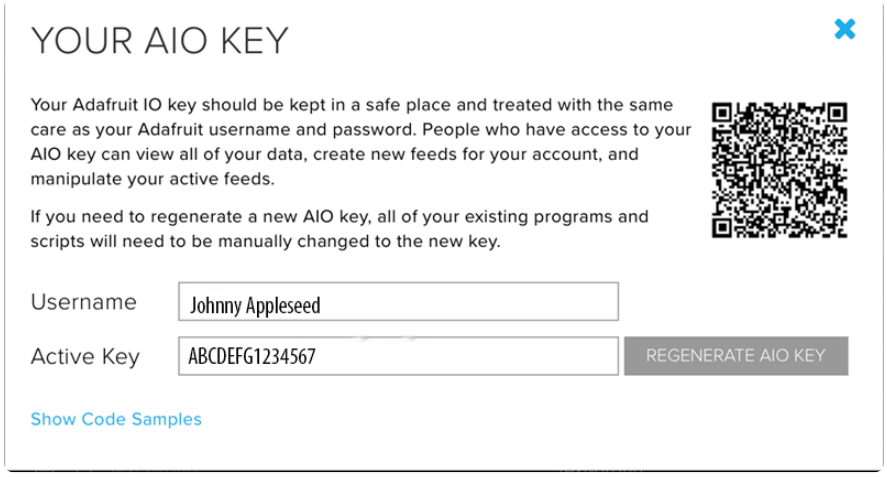
As you can see, there are placeholders in the URL Endpoint "Your-Feed-Key-Here" and "Your-A-IO-Key-Here".

You'll need to add your Feed Key and Adafruit IO Account Key here. To find your Feed Key, click on the Feeds tab on the left side of the Adafruit IO interface to access your Feeds page:

Once here, look to your right and your feed key should be below the Key column. Copy the key specific to the feed that you'll be using and replace "Your-Feed-Key-Here" with your Feed Key.

Next, to get your Adafruit IO account Key, on your left you should see "View AIO Key". Click on that, and you should see a pop-over menu that displays your Account username and Active Key. The Active Key is your Adafruit IO Key.

Copy the key and replace "Your-A-IO-Key-Here" with your Adafruit IO Key.



Next, we'll create a request variable and give it the value of the URLRequest:

```
var request = URLRequest(url: url)
```

Then, we'll give variable request a HTTP `POST` method:

```
request.httpMethod = "POST"
```

Now, we use the JSONSerialization class to convert JSON into Swift data types like our Dictionary "parameters".

```
guard let httpBody = try? JSONSerialization.data(withJSONObject: parameters,
options: []) else { return }
```

This JSONSerialization method, returns a value of type Any and throws an error if the data couldn't be parsed.

Now we'll pass our request to session.dataTask.

```
session.dataTask(with: request) { (data, response, error)
```

This creates a task that sends our contents to a specified URL, then calls a handler upon completion. The completion handler is called when the request is completed. This handler is executed on the delegate queue.

This completion handler takes the following parameters:

• Data - The data returned by the server.

• Response - An object that provides response metadata, such as HTTP headers and status code.

• Error - An error object that indicates why the request failed, or `nil` if the request was successful.

The remainder of the function will print in our data in the console or print an error message. Here's the whole function in it's completed form:

```
func postAccelerometerDataX() {

     let parameters = ["value": "\(String(format: "%.02f",
(motionManager.accelerometerData?.acceleration.x)!))"]
     guard let url = URL(string: "https://io.adafruit.com/api/feeds/your-Feed-Key-
Here/data.json?X-AIO-Key=Your-A-IO-Key-Here)

     var request = URLRequest(url: url)
     request.httpMethod = "POST"
     request.addValue("application/json", forHTTPHeaderField: "Content-Type")
     guard let httpBody = try? JSONSerialization.data(withJSONObject: parameters,
options: []) else { return }
     request.httpBody = httpBody
     let session = URLSession.shared
     session.dataTask(with: request) { (data, response, error) in
         if let response = response {
             print(response)
         }
         if let data = data {
             do {
                 let json = try JSONSerialization.jsonObject(with: data, options:
[])
                 print(json)
             } catch {
                 print(error)
             }
         }
      }.resume()
 }
```

Ok, now we're going to finish up our app.

Finishing Up

Previously, we created a function that would send your accelerometer data to Adafruit IO. Now we need to call this function whenever there's an update. We can call our `postAccelerometerDataX()` function inside of the `startAccelerometerX()` fu

nction so whenever an update happens, our sensor data will be sent to our Adafruit IO feed.

Let's add `self.postAccelerometerDataX()` inside of the else `startAccelerometerX ()` statement.

```
    func startAccelerometerX () {
        print("Start Acceleromter Updates")
        motionManager.accelerometerUpdateInterval = 2.5
        motionManager.startAccelerometerUpdates(to: OperationQueue.current!,
withHandler: {
            (accelerData:CMAccelerometerData?, error: Error?) in
            if (error != nil ) {
                print("Error")
            } else {

                let accelX = accelerData?.acceleration.x
                self.accelTagX.text = String(format: "%.02f", accelX!)
                self.postAccelerometerDataX()
                print("Accelerometer X: \(accelX!)")
            }
        })
    }
```

Now, whenever there are any update events that happens, the postAccelerometerDataX function sends current accelerometer x position data to Adafruit IO.

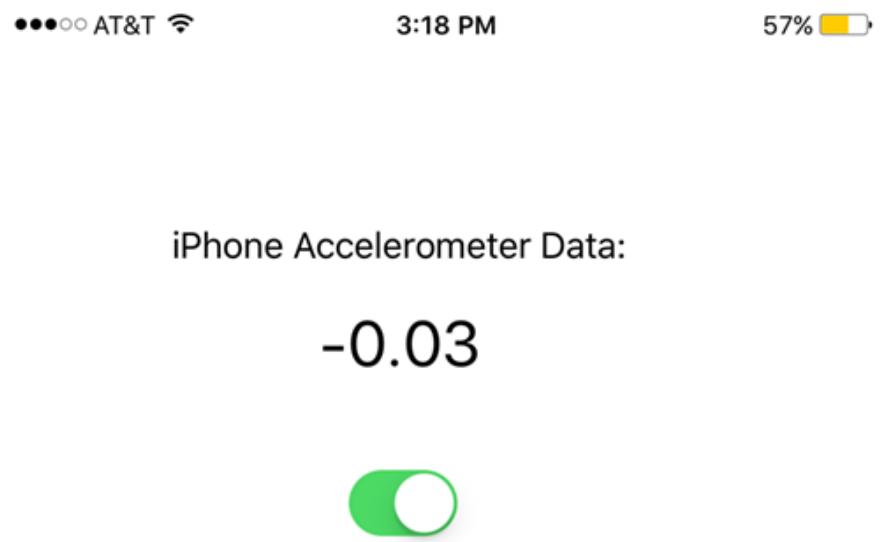
Setting up the Switch

With this switch, we'll be able to start and stop updates. In the stateChange function, if the switch is set to On, then we'll run the `startAccelerometerX` function. If not, it'll run the `stopAccelerometerX` function.

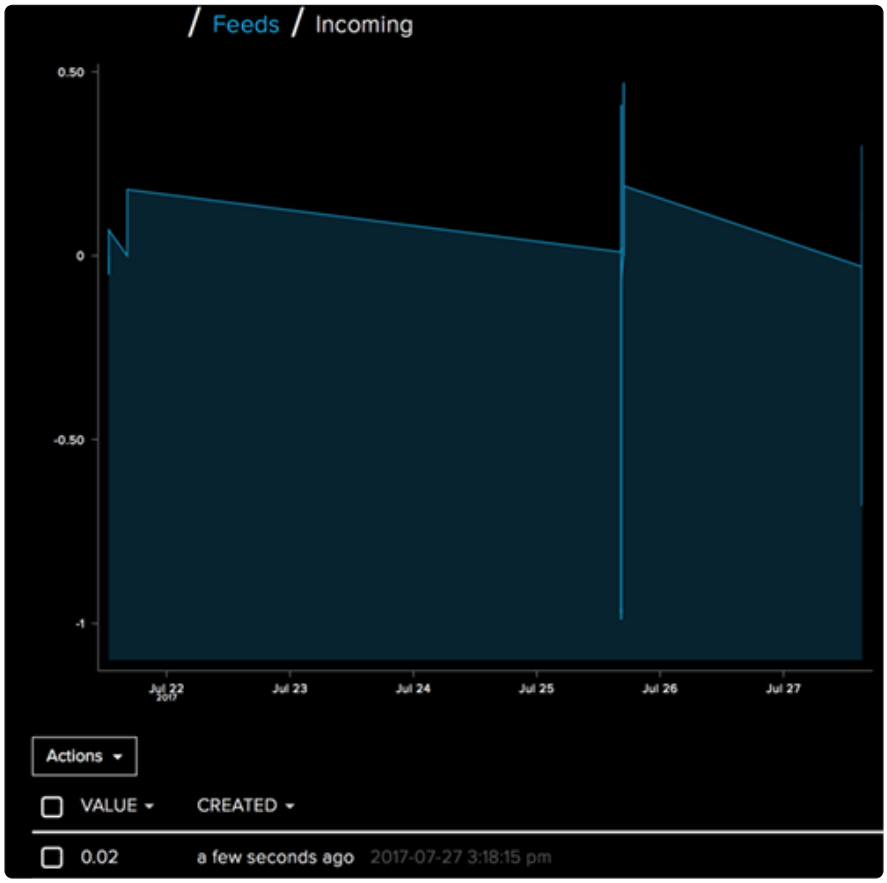
```
@IBAction func stateChange(_ sender: UISwitch) {
        if (sender.isOn == true){
            startAccelerometerX()
           }else {
            stopAccelerometerX()
        }
    }
```

Ok, lets run our app.

As soon as we open our app, the app will start its updates.



Which means that it'll send our sensor data straight to Adafruit IO and can be seen on a graph. Below shows a graph of my acceleration data from my iPhone.



Congrats!

You've successfully created your own app that not only displays your iOS device's sensor data, but also sends data using REST to Adafruit IO.

Now that you've completed this learn guide, download Adafruit IO Connect to use as a reference to send different sensor data. This version makes it possible to send accelerometer, magnetometer, or gyroscopic data to Adafruit IO.

Download Adafruit IO Connect

Happy Coding :]