Zero to BLE on iOS - Part One

By: Evan K. Stone 11 June, 2015 in development

Everywhere you turn, there's a new wearable mobile device. Out of the box, consumers expect these devices to gather information and share that information with their primary mobile devices--their iPhones and iPads. How can we as developers start communication in this Internet of Things era? For now, Bluetooth Low Energy.

Introduction

This is the first of a three-part series on the topic of app development using Bluetooth Low Energy (BLE) on iOS with the Core Bluetooth Framework.

While specifically targeted at iOS developers with some knowledge of Objective-C or Swift and the iOS SDK and its basic concepts and patterns such as delegates, this series is intended as an introduction to help you get a look at what it takes to construct an app that can communicate with an off-the-shelf Bluetooth Low Energy device or interact with BLE components of your own design, whether you are a developer, designer, maker, or entrepreneur.

No previous experience with BLE is required, and all examples apply to both iPhone and iPad running iOS 8. After this series is over, you should have enough knowledge to begin developing your own BLE solutions for iOS, and resources to find more information if you would like to do more digging on the subject.

Zero to BLE

The product landscape is exploding with wearable mobile devices. More and more, we need these little secondary devices to communicate with our primary mobile devices, such as our iPhones and iPads.

Today, the primary means in which wearables, carryables, and pocketables communicate is through the **Bluetooth Low Energy** standard (also known as **Bluetooth LE** and **BLE**).

We are in the middle of an explosion of new bluetooth-enabled devices, and Cloud City Development is looking forward to engaging with and developing mobile solutions with enterprising companies who are developing new and exciting products, devices and services in the evermore-connected Internet of Things (IoT) space.

Background

Currently Cloud City has developers knowledgeable in the ways of Bluetooth on various platforms, however recently it has appeared that it would be helpful to put together a guide that discusses how to get started with the technology, as well as an iOS App project to go along with the guide.

The Guide: BLE – Making the jump to light speed!

Developing for Bluetooth Low Energy is relatively straightforward once one understands the basic concepts. As in most forms of computing these days, one device wants information from another device.

Typically we use terms like “client” and “server” when describing the “who wants data” and “who has the data” scenario. The group who put together the Bluetooth LE standard however decided to use slightly different terms.

In Bluetooth-speak, the device that has the data is referred to as the **Peripheral**, and the device that wants the data contained in the Peripheral is known as the **Central**.

I’m not entirely sure about the origins of these terms, but don’t let it confuse you — it’s really just another way of describing a client-server relationship. (TIP: I usually remember the Peripheral is the device that is “out there somewhere” collecting data. Therefore, by default, the one remaining is the Central – the player in the scenario that interacts with the Peripheral.)

In the iOS world, this means that typically the iPhone or iPad app we are developing will be the Central, interacting with one or more Peripherals to glean information that can then be processed, analyzed, or stored in some meaningful fashion. For example, one common device that would be used as a Peripheral is a heart rate monitor, which happens to be a pretty good device to start out with, because it generally provides a more limited set of functionality and data to sift through.

Getting Data

Naturally, if we are developing an app that will act as a Central, we will want to know how to get the data we're interested in out of the Peripheral. This is where **Services** and **Characteristics** come into play. You can think of Characteristics somewhat as being like properties of your device which and be read from and written to, and a Service is a collection of characteristics. A Peripheral can (and usually does) contain multiple services that we inspect to determine what characteristics are available with which to interact.

Discovery

Before you can read from or write to values on the device, you need to Discover it. As it turns out, if a Bluetooth device is turned on and is in range, it periodically sends out a little signal that lets interested parties know that it's alive and kicking. This process is known as Advertising, and the time between signals is known as the "advertising interval."

In the case where the Central is an iOS app, it listens for those advertisements, and the app can specify exactly which Services it is interested in. When going through its discovery phase, iOS will find devices in the area that support the desired Services. This is very important, because the number of Bluetooth LE devices in existence, and which could potentially be in our vicinity, is constantly increasing. Therefore, it is helpful to let the framework weed out the devices that are not broadcasting the types of services we are interested in.

Introducing Core Bluetooth

Apple has seen this wave coming, and it introduced a framework for iOS and OS X called Core Bluetooth, which, according to their documentation, is the "framework [that] provides the classes needed for your iOS and Mac apps to communicate with devices that are equipped with Bluetooth low energy wireless technology."

As you can see from the description in the docs – and contrary to what one might expect – Core Bluetooth is not designed for developers working with standard Bluetooth. Rather, it is specifically designed for working with Bluetooth LE, and this works out well for those of us working with BLE devices, because we have an entire framework developed, documented, and supported by Apple that enables us to interact with BLE devices. The downside is that even though Apple did an excellent job creating the Core Bluetooth framework, it can still be a challenging undertaking, and in general it is a task that should neither be tackled by the fledgling iOS developer, nor the impatient.

Core Bluetooth has several moving parts, and Apple has done their best to abstractify the functionality of Bluetooth Low Energy specification, and by and large has used the same nomenclature that the specification uses for those components involved in the exchange.

As we mentioned earlier, the two (or more) players in a BLE conversation are the Peripheral, which collects and dispenses the data we're interested in, and the Central, which, for the purposes of this blog post, will probably be the iPhone or iPad app we are developing. For better or for worse, Apple uses these the same terms that the specification uses in the Core Bluetooth framework – even when the names "Central" and "Peripheral" may not initially invoke their proper meaning when you first hear them.

The classes in the framework we commonly use when developing apps that are Centrals interacting with Peripherals are `CBCentralManager`, which manages and interacts with Peripherals, and `CBPeripheral`, which is as it sounds – an abstraction of the Peripheral that wraps the functionality surrounding the retrieval and updating of data in the remote device. Services and Characteristics are represented by the `CBService` and `CBCharacteristic` classes, respectively. Notice that the classes and delegates in the framework begin with “CB”, which indicates that they are all part of the Core Bluetooth framework.

In addition to the classes already mentioned, we also use those objects’ corresponding delegate protocols, `CBCentralManagerDelegate` and `CBPeripheralDelegate`.

Developers using the Core Bluetooth Framework will need to be very comfortable with working with delegates and understanding how they function. As of today, the framework relies heavily on delegates rather than on blocks. I wouldn’t be surprised if that changes in the future, but for now, you need to be comfortable with delegates.

The reason why the Framework uses delegates is that just about all of the interactions between the Central and Peripheral are, thankfully, asynchronous and non-blocking. Therefore, the use of delegates in this scenario allows for the calling of a method on one of the Core Bluetooth Framework objects, and it will get back to you at some undetermined time in the future by way of a delegate method.

Initially, the class we are most interested in is the `CBCentralManager`, which acts as the coordinator of the conversation with the Peripheral devices, and it’s this class that one uses as the starting point when building an app that will act as a Central.

UUIDs Galore

One other thing to be aware of and get familiar with is that all Services and Characteristics in Core Bluetooth are defined by either a 16-bit UUID or a 128-bit UUID. The distinction between the two is that the 16-bit UUIDs are defined by the Bluetooth LE specification, and are listed in the Bluetooth Developer Portal, in the Services or Characteristics sections, and the 128-bit UUID format is for proprietary Services and Characteristics.

The fact that this distinction exists shows that It is possible to create your own Services and Characteristics for your own devices. For example, if you are developing a cool new BLE device, and you are building Services and Characteristics not listed in the standard, then you would create your own 128-bit UUIDs for that purpose.

A UUID (Universally Unique Identifier) is basically just a number, and if you are developing your own proprietary Services and Characteristics, it is highly recommended by Apple that you use a tool like uuidgen to create the 128-bit random UUIDs, which “generates a Universally Unique IDentifier (UUID), a 128-bit value guaranteed to be unique over both space and time.” Therefore, such a tool enables your identifiers to have a very high probability of being unique, so as not to collide with identifiers being used by other devices nearby.

Core Bluetooth Workflow

The basic workflow when developing with Core Bluetooth goes something like this:

1. Create an instance of a `CBCentralManager`.
2. If Bluetooth services are powered on, start scanning for `CBPeripherals` with desired Services.
3. When you find a Peripheral you want to connect to, stop scanning, and then connect to the `CBPeripheral` you have found.
4. Inspect the `CBPeripheral` for available `CBServices`. You can either ask the Peripheral if it supports specific services or ask it to return all available Services that it supports.
5. When we find the Peripheral’s Services, we iterate through them, looking for `CBCharacteristics` we are interested in.
6. On finding the Characteristics we are seeking, we can then read from or write to the values of those Characteristics.

This was a super high-level overview of developing for BLE with iOS, but it gives you a taste of how the process works.

In our next installment of the Zero to BLE series, we will discuss some more in-depth topics and situations one can run into when developing for Bluetooth LE devices, such as connecting to a device and reading and writing data. You can find the Objective-C version of Part Two here and the Swift version here.

Conclusion

For the foreseeable future, Bluetooth LE will be the standard way for devices to communicate with each other and with the apps that we develop to run on our mobile devices, such as iPhone and iPad. Conveniently, Apple has supplied the tools to make developing for the growing wave of interconnected devices, and Cloud City Development can help you to take advantage of those technologies to provide engaging solutions for your customers to interact with.

References

Apple Developer - Core Bluetooth Programming Guide
https://developer.apple.com/library/ios/documentation/NetworkingInternetWeb/Conceptual/CoreBluetooth_concepts/AboutCoreBluetooth/Introduction.html

WWDC 2012 Videos - Core Bluetooth 101
https://developer.apple.com/videos/wwdc/2012/#703

WWDC 2012 Videos - Advanced Core Bluetooth
https://developer.apple.com/videos/wwdc/2012/#705

WWDC 2013 Videos - Core Bluetooth
https://developer.apple.com/videos/wwdc/2013/#703