Bluetooth Low Energy in iOS Swift

by Tony Gaitatzis

Services and Characteristics

Before data can be transmitted back and forth between a Central and Peripheral, the Peripheral must host a GATT Profile. That is, the Peripheral must have Services and Characteristics.

Identifying Services and Characteristics

Each Service and Characteristic is identified by a Universally Unique Identifier (UUID). The UUID follows the pattern 0000XXXX-0000-1000-8000-00805f9b34fb, so that a 32-bit UUID 00002a56-0000-1000-8000-00805f9b34fb can be represented as 0x2a56.

Some UUIDs are reserved for specific use. For instance any Characteristic with the 16-bit UUID 0x2a35 (or the 32-bit UUID 00002a35-0000-1000-8000-00805f9b34fb) is implied to be a blood pressure reading.

For a list of reserved Service UUIDs, see ***Appendix IV: Reserved GATT Services***.

For a list of reserved Characteristic UUIDs, see ***Appendix V: Reserved GATT Characteristics***.

Generic Attribute Profile

Services and Characteristics describe a tree of data access points on the peripheral. The tree of Services and Characteristics is known as the Generic Attribute (GATT) Profile. It may be useful to think of the GATT as being similar to a folder and file tree (Figure 6-1).

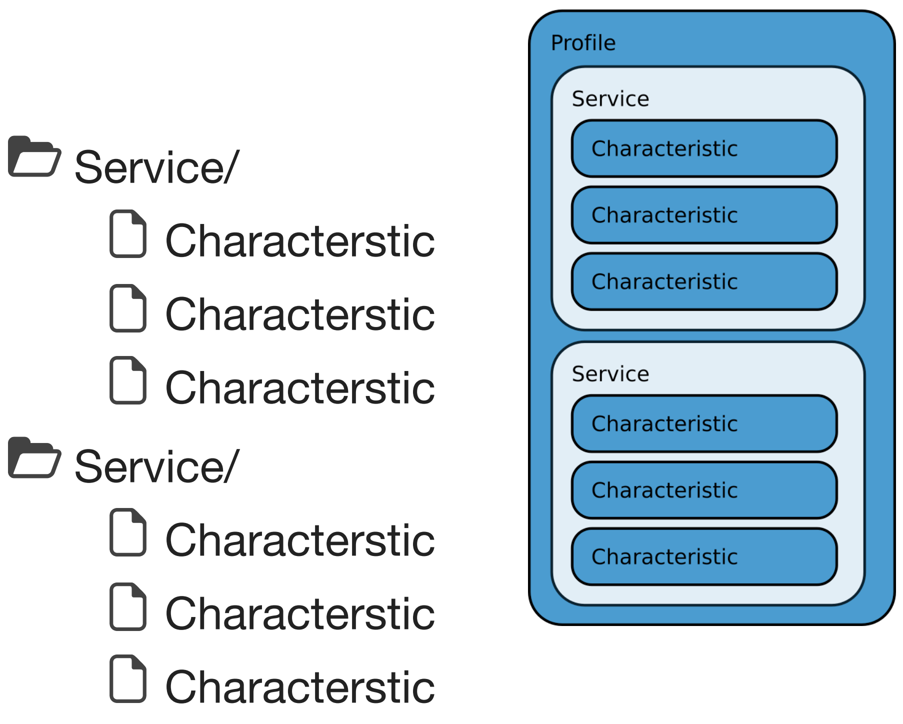


Figure 6-1. GATT Profile filesystem metaphor

Characteristics act as channels that can be communicated on, and Services act as containers for Characteristics. A top level Service is called a Primary service, and a Service that is within another Service is called a Secondary Service.

Permissions

Characteristics can be configured with the following attributes, which define what the Characteristic is capable of doing (Table 6-1):

Table 6-1. Characteristic Permissions

Descriptor	Description
Read	Central can read this Characteristic, Peripheral can set the value.
Write	Central can write to this Characteristic, Peripheral will be notified when the Characteristic value changes and Central will be notified when the write operation has occurred.
Write without Response	Central can write to this Characteristic. Peripheral will be notified when the Characteristic value changes but the Central will not be notified that the write operation has occurred.
Notify	Central will be notified when Peripheral changes the value.

Because the GATT Profile is hosted on the Peripheral, the terms used to describe a Characteristic's permissions are relative to how the Peripheral accesses that Characteristic. Therefore, when a Central uploads data to the Peripheral, the Peripheral can "read" from the Characteristic. The Peripheral "writes" new data to the Characteristic, and can "notify" the Central that the data is altered.

Data Length and Speed

It is worth noting that Bluetooth Low Energy has a maximum data packet size of 20 bytes, with a 1 Mbit/s speed.

Programming the Central

The Central can be programmed to read the GATT Profile of the Peripheral after connection, like this:

```
peripheral.discoverServices(nil)
```

If only a subset of the Services hosted by the Peripheral are needed, those Service UUIDs can be passed into the discoverServices function like this:

```
let serviceUuids = [ "1800", "1815" ]
peripheral.discoverServices(serviceUuids)
```

When the Services are discovered, a callback will be executed by the CBPeripheralManagerDelegate, containing an updated CBPeripheral object. This updated object contains an array of Services:

```
func peripheral(
    _ peripheral: CBPeripheral,
    didDiscoverServices error: Error?)
{
    if error != nil {
        print("Discover service Error: \(error)")
    }
}
```

In order for the class to access these methods, it must implement CBPeripheralManagerDelegate.

There are Primary Services and Secondary services. Secondary Services are contained within other Services; Primary Services are not. The type of Service can be discovered by inspecting the CBService.isPrimary flag.

```
boolean isPrimary = service.isPrimary
```

To discover the Characteristics hosted by these services, simply loop through the discovered Services and handle the resulting peripheral didDiscoverCharacteristicsFor callback:

```
func peripheral(
    _ peripheral: CBPeripheral,
    didDiscoverServices error: Error?)
{
    if error != nil {
        print("Discover service Error: \(error)")
    } else {
        for service in peripheral.services!{
            self.peripheral.discoverCharacteristics(nil, for: service)
        }
    }
}

func peripheral(
    _ peripheral: CBPeripheral,
    didDiscoverCharacteristicsFor service: CBService,
    error: Error?)
{
    let serviceIdentifier = service.uuid.uuidString
    if let characteristics = service.characteristics {
        for characteristic in characteristics {
         // do something with Characteristic
        }
    }
}
```

Each Characteristic has certain permission properties that allow the Central to read, write, or receive notifications from it (Table 6-2).

Table 6-2. CBCharacteristicProperties

Value	Permission	Description
read	**Read**	Central can read data altered by the Peripheral
write	**Write**	Central can send data, Peripheral reads
writeWithoutResponse	**Write**	Central can send data. No response from Peripheral
notify	**Notify**	Central is notified as a result of a change

In iOS, these properties are expressed as a binary integer which can be extracted like this:

```
let properties = characteristic.properties.rawValue
let isWritable = (properties & \
    CBCharacteristicProperties.write.rawValue) != 0;
let isWritableNoResponse = (properties & \
    CBCharacteristicProperties.writeWithoutResponse.rawValue) != 0;
let isReadable = (properties & \
    CBCharacteristicProperties.read.rawValue) != 0;
let isNotifiable = (properties & \
    CBCharacteristicProperties.notify.rawValue) != 0;
```

A Note on Caching

Because Bluetooth was designed to be a low-power protocol, measures are taken to limit redundancy and power consumption through radio and CPU usage. As a result, a Peripheral's GATT Profile is cached on iOS. This is not a problem for normal use, but when developing, it can be confusing to change Characteristic permissions and not see the updates reflected on iOS.

To get around this, the iOS device must be restarted each time a Peripheral with the same Identifier has has changed its GATT Profile

Putting It All Together

This app will work like the one from the previous chapter, except that once the it connects to the Peripheral, it will also list the GATT Profile for that Peripheral. The GATT Profile will be displayed in an UITableView (Figure 6-2).

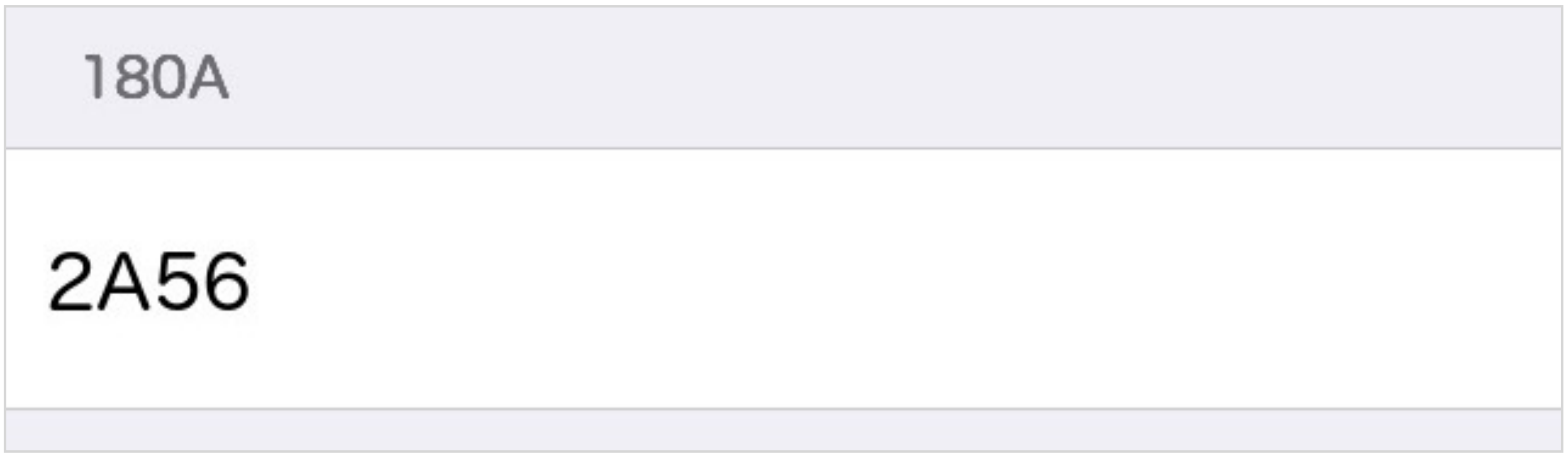
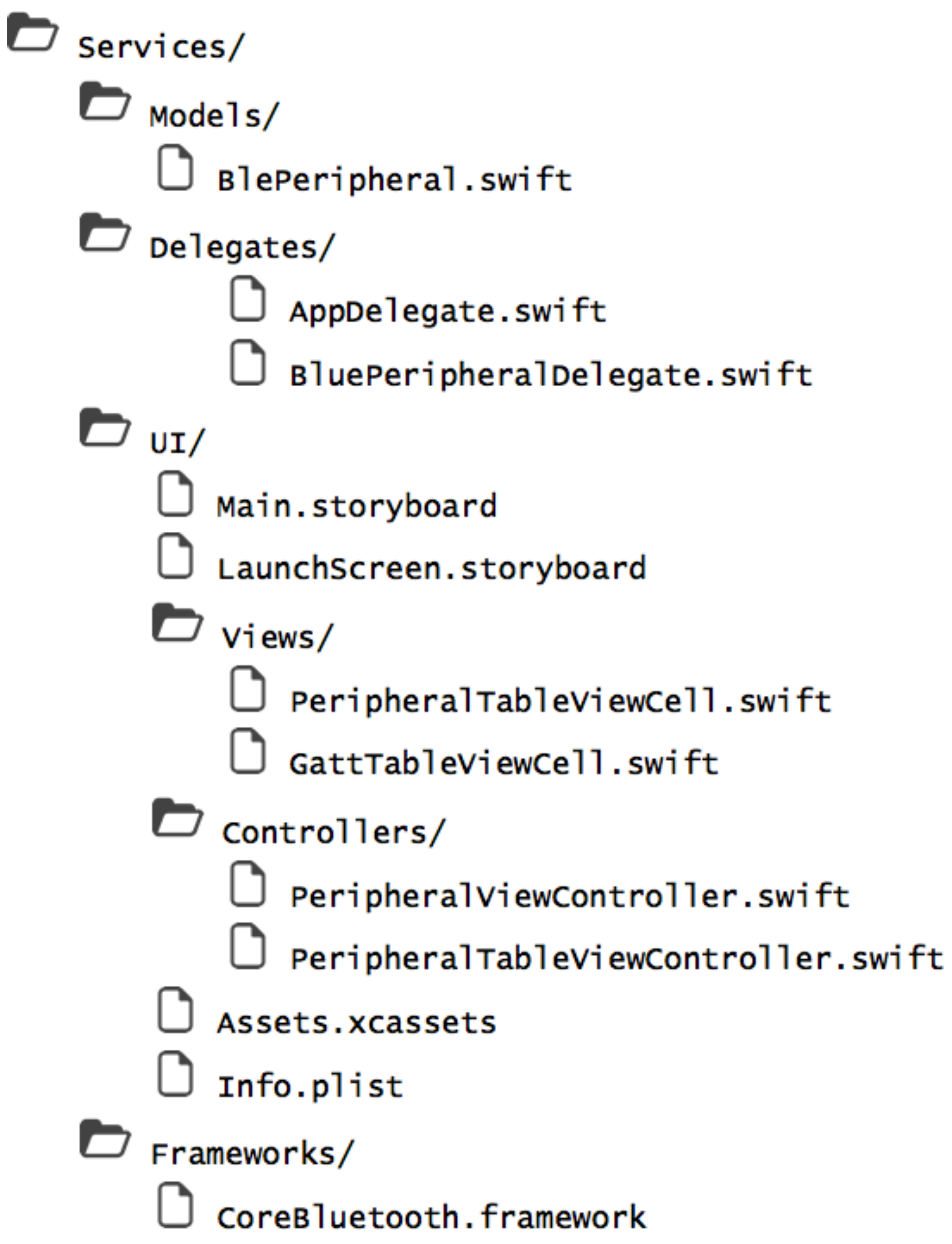


Figure 6-2. GATT Profile downloaded from Peripheral

Create a new project called Services and copy everything from the previous example. (Figure 6-3).



```
Services/
    Models/
        BlePeripheral.swift
    Delegates/
            AppDelegate.swift
            BluePeripheralDelegate.swift
    UI/
        Main.storyboard
        LaunchScreen.storyboard
        Views/
            PeripheralTableViewCell.swift
            GattTableViewCell.swift
        Controllers/
            PeripheralViewController.swift
            PeripheralTableViewController.swift
        Assets.xcassets
        Info.plist
    Frameworks/
        CoreBluetooth.framework
```

Figure 6-3. Project Structure

Objects

Modify BlePeripheral.swift to discover Services and Characteristics

Example 6-1. Models/BlePeripheral.swift

```
...
    /**
     Servicess were discovered on the connected Peripheral
     */
    func peripheral(
        _ peripheral: CBPeripheral,
        didDiscoverServices error: Error?)
    {
        print("services discovered")
        // clear GATT profile - start with fresh services listing
        gattProfile.removeAll()
        if error != nil {
            print("Discover service Error: \(error)")
        } else {
            print("Discovered Service")
            for service in peripheral.services!{
                self.peripheral.discoverCharacteristics(nil, for: service)
            }
            print(peripheral.services!)
        }
    }

    /**
     Characteristics were discovered
     for a Service on the connected Peripheral
     */
    func peripheral(
        _ peripheral: CBPeripheral,
        didDiscoverCharacteristicsFor service: CBService,
        error: Error?)
    {
        print("characteristics discovered")
        // grab the service
        let serviceIdentifier = service.uuid.uuidString
        print("service: \(serviceIdentifier)")

        gattProfile.append(service)
        if let characteristics = service.characteristics {
            print("characteristics found: \(characteristics.count)")
            for characteristic in characteristics {
                print("-> \(characteristic.uuid.uuidString)")
            }
            delegate?.blePerihperal?(
                discoveredCharacteristics: characteristics,
                forService: service,
                blePeripheral: self)
        }
    }
...
```

Delegates

Add a function to the BlePeripheralDelegate to alert when Characteristics have been discovered:

Example 6-2. Delegates/BlePeripheralDelegate.swift

```
...
    /**
     Characteristics were discovered for a Service

     - Parameters:
     - characteristics: the Characteristic list
     - forService: the Service these Characteristics are under
     - blePeripheral: the BlePeripheral
     */
    @objc optional func blePerihperal(
        discoveredCharacteristics characteristics: [CBCharacteristic],
        forService: CBService,
        blePeripheral: BlePeripheral)
...
```

Storyboard

Add a UITableView and UITableViewCell to the PeripheralViewController. Make the UITableView a "grouped" TableVew and make the UITableViewCell of class "GattTableViewCell" Give it the Reuse Identifier "GattTableViewCell." In the new GattTableViewCell, create and link a UILabel to be used to hold the Characteristic UUID (Figure 6-4):

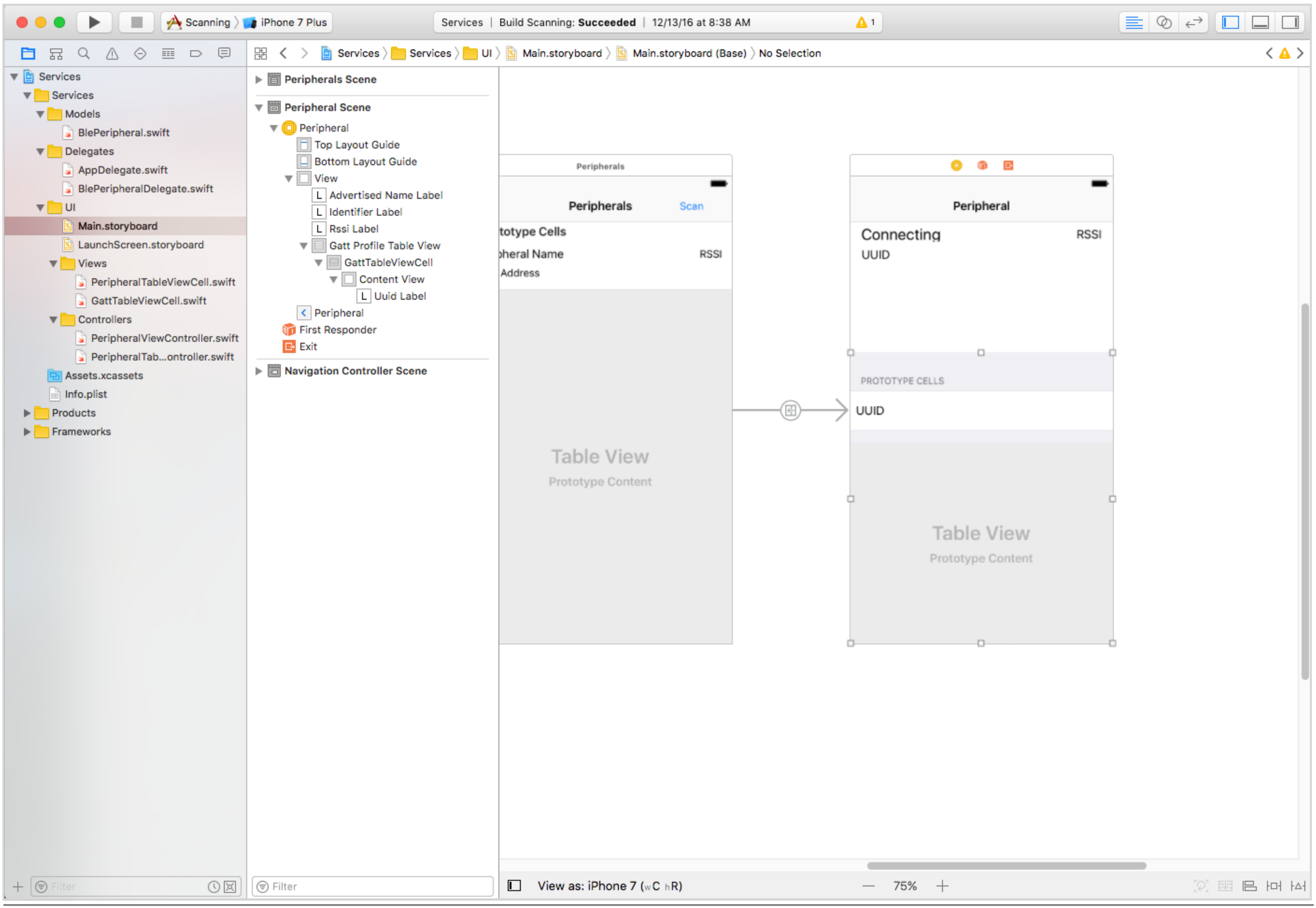


Figure 6-4. Project Storyboard

Views

The GATT Profile will be represented as a Grouped UITableView, with Services as the table header and Characteristics as the GattTableViewCell table cell.

Each GattTableViewCell will display the UUID of a Characteristic.

Example 6-3. UI/Views/GattTableViewCell.swift

```
import UIKit
import CoreBluetooth

class GattTableViewCell: UITableViewCell {
    @IBOutlet weak var uuidLabel: UILabel!

    func renderCharacteristic(characteristic: CBCharacteristic) {
        uuidLabel.text = characteristic.uuid.uuidString
        print(characteristic.uuid.uuidString)
    }
}
```

Controllers

Add functionality in the PeripheralViewController to render the GATT Profile table and to handle the blePeripheral discoveredCharacteristics callback from the BlePeripheral-Delegate class:

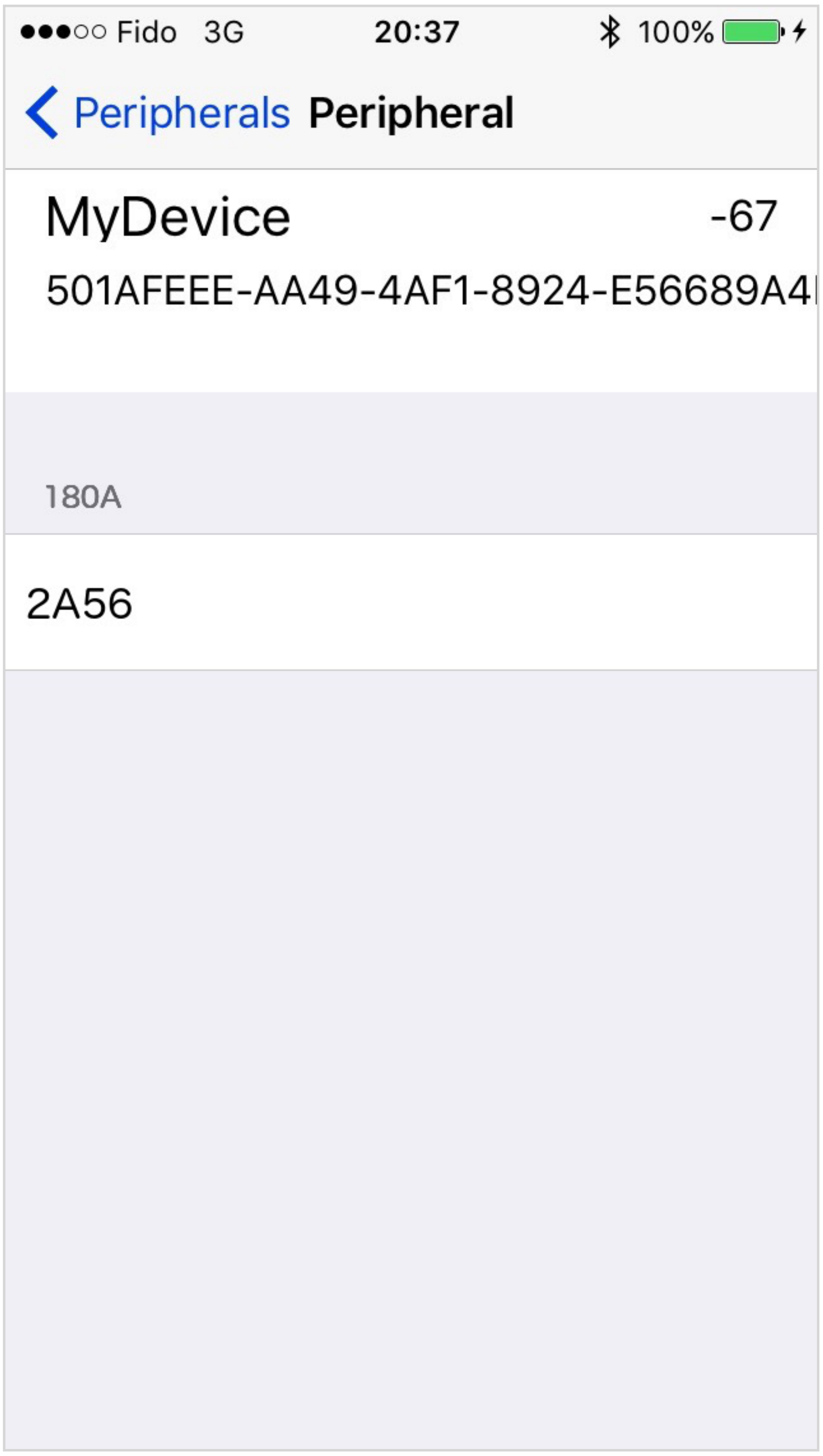
Example 6-4. UI/Controllers/PeripheralViewController.swift

```
class PeripheralViewController: UIViewController, UITableViewDataSource, \
    UITableViewDelegate, CBCentralManagerDelegate, BlePeripheralDelegate {

    // MARK: UI Elements
    @IBOutlet weak var advertisedNameLabel: UILabel!
    @IBOutlet weak var identifierLabel: UILabel!
    @IBOutlet weak var rssiLabel: UILabel!
    @IBOutlet weak var gattProfileTableView: UITableView!
    @IBOutlet weak var gattTableView: UITableView!

    // Gatt Table Cell Reuse Identifier
    let gattCellReuseIdentifier = "GattTableViewCell"
...
```

```
// MARK: BlePeripheralDelegate

/**
 Characteristics were discovered.  Update the UI
 */
func blePerihperal(
    discoveredCharacteristics characteristics: [CBCharacteristic],
    forService: CBService,
    blePeripheral: BlePeripheral)
{
    gattTableView.reloadData()
}

/**
 RSSI discovered.  Update UI
 */
func blePeripheral(
    readRssi rssi: NSNumber,
    blePeripheral: BlePeripheral)
{
    rssiLabel.text = rssi.stringValue
}

// MARK: UITableViewDataSource

/**
 Return number of rows in Service section
 */
func tableView(
    _ tableView: UITableView,
    numberOfRowsInSection section: Int) -> Int
{
    print("returning num rows in section")
    if section < blePeripheral.gattProfile.count {
        if let characteristics = \
            blePeripheral.gattProfile[section].characteristics
        {
            return characteristics.count
        }
    }
    return 0
}

/**
 Return a rendered cell for a Characteristic
 */
func tableView(
    _ tableView: UITableView,
    cellForRowAt indexPath: IndexPath) -> UITableViewCell
{
    print("returning table cell")
    let cell = tableView.dequeueReusableCell(
        withIdentifier: gattCellReuseIdentifier,
        for: indexPath) as! GattTableViewCell
    let section = indexPath.section
    let row = indexPath.row

    if section < blePeripheral.gattProfile.count {
        if let characteristics = \
            blePeripheral.gattProfile[section].characteristics
        {
            if row < characteristics.count {
                cell.renderCharacteristic(
                    characteristic: characteristics[row])
            }
        }
    }
    return cell
}

/**
 Return the number of Service sections
```

```
     */
    func numberOfSections(in tableView: UITableView) -> Int {
        print("returning number of sections")
        print(blePeripheral)
        print(blePeripheral.gattProfile)
        return blePeripheral.gattProfile.count
    }

    /**
     Return the title for a Service section
     */
    func tableView(
        _ tableView: UITableView,
        titleForHeaderInSection section: Int) -> String?
    {
        print("returning title at section \(section)")
        if section < blePeripheral.gattProfile.count {
            return blePeripheral.gattProfile[section].uuid.uuidString
        }
        return nil
    }

    /**
     User selected a Characteristic table cell.
     Update UI and open the next UIView
     */
    func tableView(
        _ tableView: UITableView,
        didSelectRowAt indexPath: IndexPath)
    {
        let selectedRow = indexPath.row
        print("Selected Row: \(selectedRow)")
        tableView.deselectRow(at: indexPath, animated: true)
    }
    // MARK: Navigation
    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        print("leaving view - disconnecting from peripheral")
        if let peripheral = blePeripheral.peripheral {
            centralManager.cancelPeripheralConnection(peripheral)
        }
    }
}
```

The resulting app will be able to connect to a Peripheral and list the Services and Characteristics (Figure 6-5).



MyDevice -67
501AFEEE-AA49-4AF1-8924-E56689A4

180A

2A56

Figure 6-5. App screen showing GATT profile from a connected Peripheral

Programming the Peripheral

The Peripheral can be programmed to host a GATT Profile - the tree structure of Services and Characteristics that a connected Central will use to communicate with the Peripheral.

Services are created and added to the CBPeripheralManager, like this:

```
// Service UUID
let serviceUuid = CBUUID(string: "0000180c-0000-1000-8000-00805f9b34fb")
let service = CBMutableService(type: serviceUuid, primary: true)
peripheralManager.add(service)
```

When a Service is added to the Peripheral, the peripheralManager didAdd callback will be triggered by the CBPeripheralManagerDelegate.

```
func peripheralManager(
    _ peripheral: CBPeripheralManager,
    didAdd service: CBService, error: Error?)
{
}
```

Characteristics must have defined properties. These properties allow a connected Central to read data from, write data to, and/or subscribe to notifications from a Characteristic. Some common properties are enumerated in the CBCharacteristicProperties class:

Table 6-3. Common CBCharacteristicProperties

Value	Permission	Description
read	**Read**	The characteristic's value can be read.
write	**Write**	The characteristic's value can be written, with a response from the peripheral to indicate that the write was successful.
writeWithoutResponse	**Write**	The characteristic's value can be written, without a response from the peripheral.
notify	**Notify**	Notifications of the characteristic's value are permitted.

Create the Characteristics properties by instanciating and merging CBCharacteristicProperties:

```
// create read Characteristic properties
var characteristicProperties = CBCharacteristicProperties.read
// append write propertie
characteristicProperties.formUnion(CBCharacteristicProperties.write)
// append notify support
characteristicProperties.formUnion(CBCharacteristicProperties.notify)
```

A Characteristic must also define it's attribute permissions. An example of an attribute is a flag that is set when a connected Characteristic wants to subscribe to a Charactersitic.

Examples of common Attribute Permissions are enumerated in the CBAttributePermissions class.

Table 6-3. Common CBAttributePermissions

Value	Description
readable	The Characteristic’s Attributes can be read by a connected Central.
writeable	The Characteristic’s Attributes can be altered by a connected Central.

Create Attribute permissions.

```
// set the Characteristic's Attribute permissions
var characterisitcPermissions = CBAttributePermissions.writeable
// append permissions
characterisitcPermissions.formUnion(CBAttributePermissions.readable)
```

Create a new CBMutableCharacteristic with the defined properties. Optionally an initial value can be set.

```
let characteristicUuid = \
    CBUUID(string: "00002a56-0000-1000-8000-00805f9b34fb")
var value:Data!

// instantiate a Characteristic
let characteristic = CBMutableCharacteristic(
    type: characteristicUuid,
    properties: characteristicProperties,
    value: value,
    permissions: characterisitcPermissions)
```

Add one or more Characteristics to a Service by creating a [CBMutableCharacteristic] array.

```
// set the service Characterisic array
service.characteristics = [ characteristic ]
```

A Note on GATT Profile Best Practices

All Peripherals should contain Device information and a Battery Service, resulting in a minimal GATT profile for any Peripheral that resembles this (Figure 6-6).

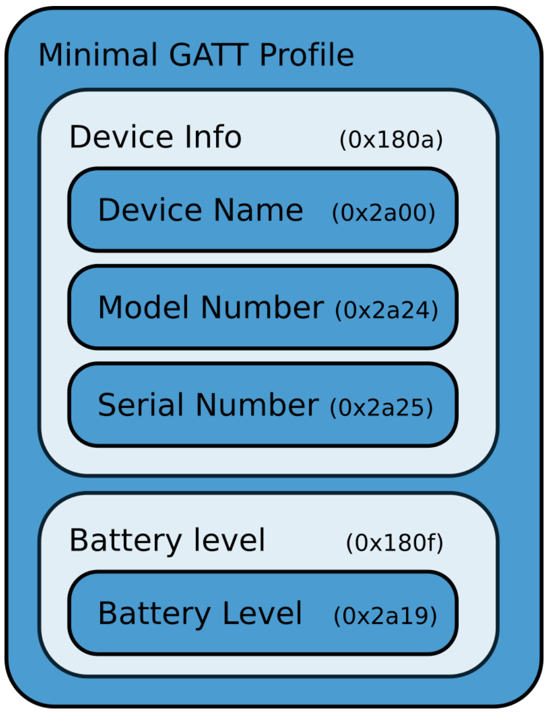


Figure 6-6. Minimal GATT Profile for Peripherals

This provides Central software, surveying tools, and future developers to better understand what each Peripheral is, how to interact with it, and what the battery capabilities are.

For pedagogical reasons, many of the examples will not include this portion of the GATT Profile.

Putting It All Together

Create a new project called GattProfile and copy everything from the previous example.

Models

Modify BlePeripheral.swift to build a minimal Gatt Services profile. Build the GATT Profile structure, and handle the callback when Services are added.

Example 6-5. Models/BlePeripheral.swift

```
...
```

```
    // MARK: GATT Profile

    // Service UUID
    let serviceUuid = CBUUID(string: "0000180c-0000-1000-8000-00805f9b34fb")
    // Characteristic UUIDs
    let characteristicUuid = CBUUID(
        string: "00002a56-0000-1000-8000-00805f9b34fb")
    // Read Characteristic
    var characteristic:CBMutableCharacteristic!
...
    /**
     Build Gatt Profile.
     This must be done after Bluetooth Radio has turned on
     */
    func buildGattProfile() {
        let service = CBMutableService(type: serviceUuid, primary: true)
        var characteristicProperties = CBCharacteristicProperties.read
        characteristicProperties.formUnion(
            CBCharacteristicProperties.notify)
        var characterisitcPermissions = CBAttributePermissions.writeable
        characterisitcPermissions.formUnion(CBAttributePermissions.readable)

        characteristic = CBMutableCharacteristic(
            type: characteristicUuid,
            properties: characteristicProperties,
            value: nil,
            permissions: characterisitcPermissions)
        service.characteristics = [ characteristic ]
        peripheralManager.add(service)
    }
...
    /**
     Peripheral added a new Service
     */
    func peripheralManager(
        _ peripheral: CBPeripheralManager,
        didAdd service: CBService,
        error: Error?)
    {
        print("added service to peripheral")
        if error != nil {
            print(error.debugDescription)
        }
    }

    /**
     Bluetooth Radio state changed
     */
    func peripheralManagerDidUpdateState(
        _ peripheral: CBPeripheralManager)
    {
        peripheralManager = peripheral
        switch peripheral.state {
        case CBManagerState.poweredOn:
            buildGattProfile()
            startAdvertising()
        default: break
        }
        delegate?.blePeripheral?(stateChanged: peripheral.state)
    }
...
```

The resulting app will be able to host a minimal GATT Profile.

Example code

The code for this chapter is available online at: https://github.com/BluetoothLowEnergyIniOSSwift/Chapter06

Other Books in this Series