Core Bluetooth skill-up

Author	Сергей Погребняк
Last changes	04.01.2020
Swift	5.2
Xcode	11.3

Вместо введения

Данный мануал не описывает теоретическую часть iOS CoreBluetooth и GATT протокола и принципы работы Bluetooth девайсов, он предназначен для практической части, в нем описана настройка среды разработки Arduino для установки различных примеров на плату ESP32 для создания Bluetooth-сервера, протокол общения для каждого из примеров, нюансы работы с core Bluetooth. Данный мануал подкреплен тестовым примером для iPhone написанным на swift 5.2 и два примера на C++ для ESP32 под Arduino.

Протокол общения

В данных примерах протокол общения, следующий:
Peripheral name – ESP32
Периферал содержит 2 сервиса и по 1 характеристике в каждом из сервисов, подробнее в таблице:

Service UUID	Characteristic UUID	Capability	Description
6e400001-b5a3-f393-e0a9-e50e24dcca9e	6e400003-b5a3-f393-e0a9-e50e24dcca9e	PROPERTY_WRITE	Input: -on -off
6e400002-b5a3-f393-e0a9-e50e24dcca9e	6e400004-b5a3-f393-e0a9-e50e24dcca9e	PROPERTY_READ PROPERTY_NOTIFY	Output: -on -of -Wrong data

Итак, описание – у периферала есть сервис, у сервиса есть характеристика которая доступна только на запись и принимает на вход текст «on» или «off», второй сервис также имеет характеристику но она доступна только на чтение и она оповещает об изменении своего значения, ее выходные данные «on», «off» или «Wrong data».

Пример 1

Данный пример эмитирует работу iPhone с Bluetooth-выключателем (включение освещения, открытие машины), это действие не занимает длительного времени на выполнение отправленной команды. Так после подключения к девайсу и отправки соответствующей команды («on»/«off») или неверных данных, модуль моментально вернет результат ее выполнения (для того что бы это произошло, нужно подписаться на уведомления характеристики чтения), таким образом при включении или выключение освещения в доме модуль

будет возвращать текущий статус и девайс сможет реагировать на изменение статуса и менять иконки, показывать анимацию или выполнять следующее действие. Также при отключении от девайса и повторном подключении можно прочитать характеристику чтения и узнать текущий статус освещения и показать актуальное состояние юзеру. Также при мульти-подключении к девайсу (выключателю) при изменении одним пользователем состояния – все получат новое состояние автоматически. В примере после отправки команды на включение или выключение на плате будет включаться/выключаться светодиод.

Пример 2

Данный пример эмитирует работу iPhone с Bluetooth девайсом которое часто изменяет свои данные без отправки команд на него, например считывание показаний температуры и влажности в доме и при их изменении уведомление девайса о новых данных, данный пример при отправке команды «on» начинает на случайный период времени включать и выключать светодиод при изменении статуса светодиода он уведомляет слушателей о том что статус изменен на on или off. При отправке команды «off» генерация приостанавливается. Таким образом этот пример эмитирует, как и сказалось в самом начале датчик температуры в доме, пользователь открывает приложение, оно находит девайс и подключается к нему, подписывается на уведомления и ждет от девайса(датчика) обновленные данные, после изменения показаний температуры в доме датчик уведомит моментально о том, что данные изменились а приложение сможет всегда отображать юзеру актуальные данные.

Пример 3

Данный пример это обновленный примера 2, для которого потребуется датчик температуры DHT11, его нужно подключить к ESP32:
- Gnd (-) DHT11 к Gnd (-) ESP32
- Vcc (+) DHT11 к 3V3 (+) ESP32
- Out DHT11 к D14 ESP32
После чего загрузить скетч на ESP32, и подписаться на обновление характеристики, после чего при изменении температуры будут приходить данные типа Float, которые можно отображать юзеру.
Датчик DHT11 работает в диапазоне от 0 до 40 градусов Цельсия.

Установка и настройка IDE

Гайд по установке и настройке среды ардуино IDE для есп 32:
1) Скачать ардуино IDE - туть
2) Установить библиотеку чтоб увидеть ESP 32 среди плат - туториал
3) установить через ардуино IDE либу для Bluetooth ESP32_BLE_Arduino от Neil Kolban - либа

Перед заливкой скетча выбираем в пункте меню Инструменты выбираем:
Плата – FireBeetle-ESP32
Скорость загрузки – 115200
Частота – 80MHz
Порт – находим и выбираем свою плату

Можно приступать к разработке!) для того чтоб залить скетч нужно нажать в Arduino IDE кнопку загрузить и на ESP32 держать зажатой кнопку boot.

Типы поиска и подключения к девайсу:

1. manager.scanForPeripherals(withServices: nil, options: nil) – ищет все девайсы
2. manager.scanForPeripherals(withServices: [serviceOfUUIDForWrite, serviceOfUUIDForRead], options: nil) – в массиве нужно указать UUID сервисов. Этот способ рекомендуется к использованию в туториалах Apple, так как при использовании данного способа будут возвращаться только девайсы которые имеют данные сервисы. НО они могут не отображаться так как данные сервисы должны быть добавлены в advertisement, для того чтобы это проверить:
    a. Используем поиск девайсов - manager.scanForPeripherals(withServices: nil, options: nil)
    b. После нахождения девайсов в методе делегата - func centralManager(_ central: CBCentralManager, didDiscover peripheral:CBPeripheral, advertisementData: [String : Any], rssi RSSI: NSNumber), нужно расспечатать advertisementData, в нем должен быть массив сервисов UUID, если их нет то данный способ подключения работать не будет!
3. Возобновление подключения к девайсу
   Возобновление подключения возможно если соединение уже было установлено и после этого было прервано, для этого нужно:
    a. После успешного подключения сохранить идентификатор девайса - UserDefaults.standard.set(remotePeripheral?.identifier.uuidString, forKey: "peripheralIdentifier")
    b. Перед началом поиска выполнить следующую функцию:

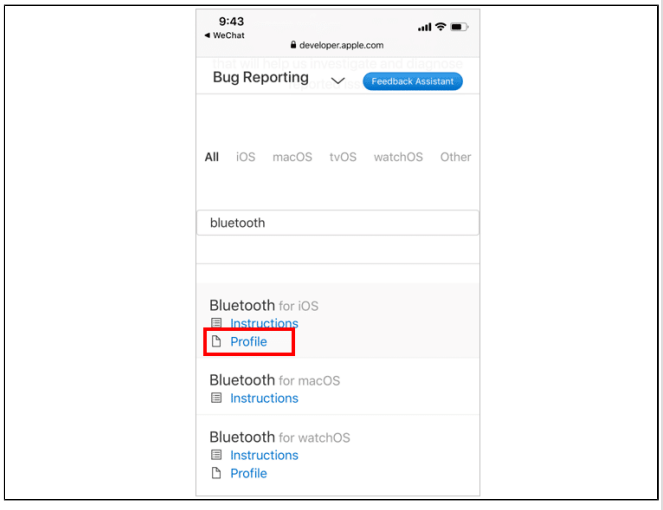
```
func connectToDevice() {
 guard let identifier = UserDefaults.standard.value(forKey:
"peripheralIdentifier") as? String,
 let uuid = UUID.init(uuidString: identifier),
 let savedPeripheral = manager.retrievePeripherals(withIdentifiers:
[uuid]).first
 else {
  self.startScan()
  return
 }

 remotePeripheral = savedPeripheral
 remotePeripheral!.delegate = self
 manager.connect(remotePeripheral!, options: nil)
}
```

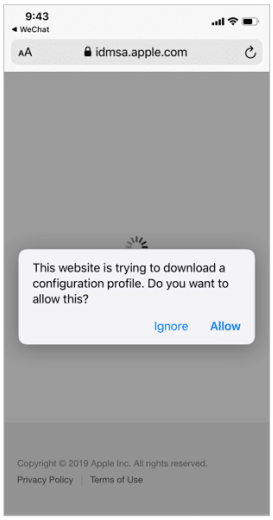
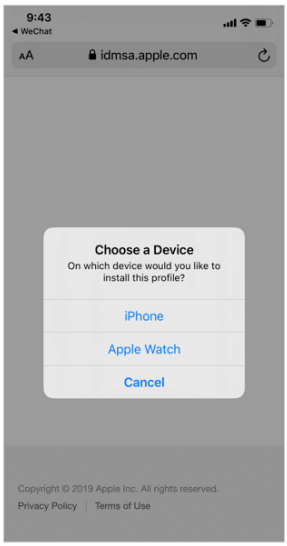
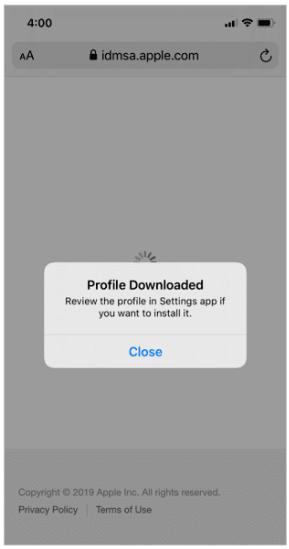
Если у нас сохранен идентификатор девайса то получаем его из песочницы, после этого создаем UUID девайса, теперь имея идентификатор вызываем функцию на восстановление подключения к девайсу – она вернет массив девайсов к которым можно подключиться, если нет идентификатора, не удалось создать UUID или массив перефиралов пуст – вызываем стандартные функции для поиска. Если все прошло успешно то берем первый девайс из массива и сохраняем его, подписываемся на делегата и дальше стандартной функцией устанавливаем соединение.

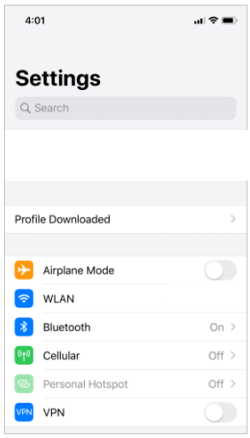
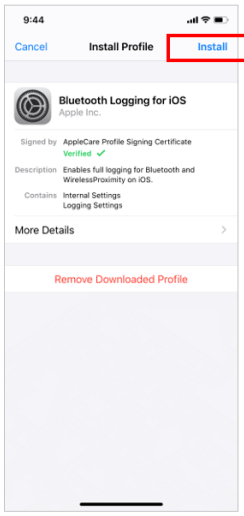
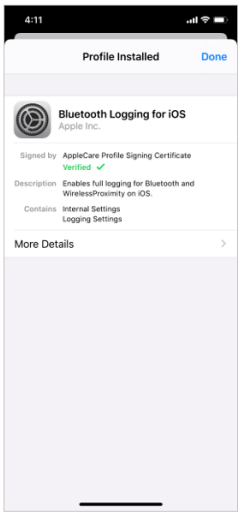
PacketLogger:

PacketLogger - это bluetooth sniffer (анализатор трафика), он отображает отсылаемый и принимаемый трафик на девайс. Для того что бы отследить движение трафика нужно иметь:
- mac os Catalina (можно и Mojave, но рекомендуется именно Catalina)
- девайс с iOS 13 на борту
- Xcode 11
- кабель для подключение iPhone к макбуку

Для того чтобы настроить анализатор трафика нужно скачать профайл с сайта на айфон и установить его.
Для этого:

Переходим на сайт Apple с профайлами и нажимаем на скачивание интересующего нас профайла	

Соглашаемся на загрузку профайла	
Выбираем девайс на который профайл будет установлен	
Профиль успешно скачан, теперь нужно перейти в настройки	

В настройках переходим в пункт меню profile downloaded	
И устанавливаем только что скачанный профайл, нажатием на кнопку Install, дожидаемся окончания установки и жмем на кнопку Done.	 

Итак, iPhone настроен теперь нужно настроить mac, скачиваем Additional Tools for Xcode 11, открываем Hardware и запускаем PacketLogger.

Теперь подключаем iPhone к mac, и открываем приложение которое может установить с любым Bluetooth девайсом и отправлять/принимать команды, после этого в packet logger выбираем пункт меню File->New iOS trace, в результате откроется новое окно и мы будем видеть передаваемые и получаемы девайсом пакеты.

Анализируем полученные данные:

1. Это тип пакета, отправлен, получен или уведомление
2. Это данные которые были получены или отправлены в 16тиричном формате

В ссылке на тестовый продж имеется конвертор для перевода в тип Float и String.
Таким образом с помощью данного sniffer-а можно отслеживать трафик как своего так и стороннего приложения для анализа и устранения ошибок или проверки корректности работы.

Несколько полезных ништячков:

Плата ESP32 пропала из плата или mac ее не видит:

Если возникнет проблема что плата ESP32 после очередного подключения/перезагрузки пк/обновления mac os перестанет видеть ESP32 в портах то поможет вот эта дока.
Спас следующий гайд:
1) sudo rm -rf /System/Library/Extensions/usb.kext
2) sudo rm -rf /Library/Extensions/usbserial.kext
3) Перезагрузить пк
4) Устанавливаем OEM driver, скачать - туть
5) Перезагрузить пк

Скетч не заливается и/или IDE выдает не понятные ошибки:
Если возникает проблема с заливкой скетча лучше его пересоздать и просто перебросить код в новый.

В XCode нету девайса с нужным именем после заливки прошивки на ESP:
Core Bluetooth кеширует имена девайсов, после заливки новой прошивки или просто смены имени девайса (ESP32), iPhone может по прежнему продолжать отображать девайс со старым именем, для этого можно подключиться к нему по старому имени, iPhone обновит его имя и в дальнейшем будет отображать корректное имя. Прошивка на плате ESP не поддается никаким изменениям и перезаливке!

Я отправил команду на девайс, а она не сработала и соединение разорвалось:
Некоторые девайсы не могут принимать данные больше 20 байт, также есть версия что iPhone не может отправлять данные больше 20 байт (проверить не удалось), таким образом если есть необходимость отправить на девайс данные больше 20 байт то их нужно отправлять чанками, попросить увеличить размер принимаемых данных, сократить команды до нужной длинны. Если данные очень большие то нужно использовать потоковую запись.

Также для самой ESP32 можно выставить ограничение данных (MTU), НО оно выставляется исходя из формулы: нужное кол-во байт + 3, так как 1 байт это заголовок команды и 2 байта дескриптор.

Команда не срабатывает:
Все команды должны выполнятся только после того как сработал метод делегата centralManagerDidUpdateState. Например, метод - manager.scanForPeripherals(…) не сработает и будет проигнорирован пока не сработает метод centralManagerDidUpdateState.

Ссылка на тестовый проект

https://github.com/Serg-Pogrebnyak/CoreBluetooth_Skillup_with_Arduino
https://gitlab.nixdev.co/nix_iphone_skillup/corebluetooth_skillup_with_arduino

---