6.9 Practical Implementation with ESP32

This chapter provides a hands-on project to demonstrate the core concepts of a BLE peripheral device using an ESP32. We will move beyond a simple serial example and create a simulated **BLE Heart Rate Sensor**. This is a standard profile that teaches the essential concepts of services, characteristics, and notifications.

Project: Create a BLE Heart Rate Sensor

Goal: Configure the ESP32 to act as a BLE peripheral that advertises the standard Heart Rate service. When a central device (like a smartphone) connects and enables notifications, the ESP32 will periodically send a simulated heart rate measurement.

You Will Need:

- An ESP32 development board.
- The Arduino IDE with the ESP32 board package installed.
- A smartphone with a BLE scanner app (e.g., "nRF Connect for Mobile" or "LightBlue").

```
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>

// Standard Bluetooth Service and Characteristic UUIDs for Heart Rate
#define SERVICE_UUID        "0000180d-0000-1000-8000-00805f9b34fb" // Heart Rate Service
#define CHARACTERISTIC_UUID "00002a37-0000-1000-8000-00805f9b34fb" // Heart Rate Measurement

BLEServer* pServer = NULL;
BLECharacteristic* pCharacteristic = NULL;
bool deviceConnected = false;

// This class handles server events like client connect/disconnect
class MyServerCallbacks: public BLEServerCallbacks {
    void onConnect(BLEServer* pServer) {
      deviceConnected = true;
```

```
      Serial.println("Client Connected");
    }

    void onDisconnect(BLEServer* pServer) {
      deviceConnected = false;
      Serial.println("Client Disconnected");
    }
};

void setup() {
  Serial.begin(115200);
  Serial.println("Starting BLE Heart Rate Sensor...");

  // 1. Initialize the BLE device and set its name
  BLEDevice::init("ESP32 Heart Rate Sensor");

  // 2. Create the BLE Server
  pServer = BLEDevice::createServer();
  pServer->setCallbacks(new MyServerCallbacks()); // Set the event handler

  // 3. Create the BLE Service using the standard Heart Rate UUID
  BLEService *pService = pServer->createService(SERVICE_UUID);

  // 4. Create a BLE Characteristic for the Heart Rate Measurement
  pCharacteristic = pService->createCharacteristic(
                      CHARACTERISTIC_UUID,
                      BLECharacteristic::PROPERTY_READ |
                      BLECharacteristic::PROPERTY_NOTIFY
                    );

  // 5. Add a 2902 descriptor to the characteristic. This is ESSENTIAL
  // for the client to be able to enable notifications.
  pCharacteristic->addDescriptor(new BLE2902());

  // 6. Start the service
  pService->start();

  // 7. Start advertising, so other BLE devices can find this one
  BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
```

```
  pAdvertising->addServiceUUID(SERVICE_UUID); // Advertise our service
  pAdvertising->setScanResponse(true);
  pAdvertising->setMinPreferred(0x06);
  pAdvertising->setMinPreferred(0x12);
  BLEDevice::startAdvertising();

  Serial.println("Characteristic defined! Now you can scan for 'ESP32 Heart Rate Sensor' on
your phone.");
}

void loop() {
  // Check if a client is connected
  if (deviceConnected) {
    // Generate a simulated heart rate value
    // The first byte is a flag (0), the second is the 8-bit heart rate value
    static uint8_t heartRate = 60;
    heartRate++;
    if (heartRate > 100) {
      heartRate = 60; // Reset after 100
    }

    uint8_t heartRateData[2] = {0, heartRate};

    // Set the characteristic's new value
    pCharacteristic->setValue(heartRateData, 2);

    // Send a notification to the connected client
    pCharacteristic->notify();

    Serial.print("Heart Rate Notification Sent: ");
    Serial.println(heartRate);
  }
  delay(1000);
}
```

Code Walkthrough

1. **Initialization:** We initialize the BLE stack using `BLEDevice::init()` and give our device a public name.

2. **Server and Service:** We create a `BLEServer` to manage connections and a `BLEService` to hold our data. We use the official UUID for the "Heart Rate Service."
3. **Characteristic:** Inside the service, we create a `BLECharacteristic` for the "Heart Rate Measurement." We set its properties to allow a client to both READ the value and subscribe to NOTIFY (notifications).
4. **Descriptor (BLE2902):** This is a critical step. The BLE2902 descriptor is the Client Characteristic Configuration Descriptor (CCCD). A client (your phone) writes to this descriptor to tell the server (the ESP32) that it wants to receive notifications. Without this, notifications will not work.
5. **Advertising:** We start advertising and include the Service UUID. This tells scanning devices what services we offer before they even connect.
6. **The Loop:** In the main loop, we check if a client is connected. If so, we generate a new simulated heart rate value, update the characteristic with `setValue()`, and then send it to the client using `notify()`.

How to Test It

1. Upload the code to your ESP32.
2. Open the Arduino Serial Monitor to see the status messages.
3. On your smartphone, open a BLE scanner app (like nRF Connect for Mobile).
4. **Scan** for devices. You should see "ESP32 Heart Rate Sensor" in the list.
5. **Connect** to the device. In the Serial Monitor, you should see "Client Connected."
6. Find the **Heart Rate Service** and expand it to see the **Heart Rate Measurement** characteristic.
7. Tap the "subscribe" or "enable notifications" icon (often a single or triple downward arrow).
8. You should now see the value updating in your app every second, and the Serial Monitor will show the "Notification Sent" logs.

Revision #1
Created 2025-08-28 12:20:01 UTC by GI
Updated 2025-08-28 12:23:41 UTC by GI