**Transfer Wi-Fi credentials over BLE and connect to Wi-Fi**

**By Nirvan Tamhane**

/\*

  created 27 Aug 2019

  by ThatProject

  modified 11 July 2024

  by Nirvan Tamhane

  Ref.

  ThatProject, mironal.

  Parts required:

  - 1x ESP32 Devkit V1 Development Board

  Complete project detail @ https://github.com/Nirvan007/ESP32\_WIFI\_CREDS\_OVER\_BLE

  Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files.

  The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

\*/

#include "EEPROM.h"

#include <WiFi.h>

#include <BLEDevice.h>

#include <BLEServer.h>

#include <BLEUtils.h>

#include <BLE2902.h>

// Define UUIDs for BLE service and characteristic

#define SERVICE\_UUID "1d7ffe98-4251-4904-b4a1-9f63724c1346"

#define CREDS\_CHARACTERISTIC\_UUID "9a3f923f-d147-4372-8e56-0caaeb0d4834"

// Creating a BLE Server

BLEServer\* pServer = NULL;

BLECharacteristic\* pCharacteristic = NULL;

// Callback variables

bool deviceConnected = false;

bool oldDeviceConnected = false;

// EEPROM addresses for storing mode and WiFi credentials

const int modeAddr = 0;

const int wifiAddr = 10;

int modeIdx;

// Callbacks for BLE server events

class MyServerCallbacks : public BLEServerCallbacks {

  void onConnect(BLEServer\* pServer) {

    deviceConnected = true;

    BLEDevice::startAdvertising();  // Restart advertising

  }

  void onDisconnect(BLEServer\* pServer) {

    deviceConnected = false;

  }

};

// Callbacks for BLE characteristic events

class MyCallbacks : public BLECharacteristicCallbacks {

  void onWrite(BLECharacteristic\* pCharacteristic) {

    String value = pCharacteristic->getValue();

    if (value.length() > 0) {

      Serial.print("SSID,PASSWORD: ");

      Serial.println(value.c\_str());

      Serial.println("Reset Device to connect to WiFi");

      writeString(wifiAddr, value.c\_str());  // Store the received WiFi credentials in EEPROM

    }

  }

  // Write the string to EEPROM

  void writeString(int add, String data) {

    int \_size = data.length();

    for (int i = 0; i < \_size; i++) {

      EEPROM.write(add + i, data[i]);  // Write each character of the string to EEPROM

    }

    EEPROM.write(add + \_size, '\0');  // Write null terminator

    EEPROM.commit();                  // Commit the changes to EEPROM

  }

};

void setup() {

  Serial.begin(115200);

  if (!EEPROM.begin(128)) {

    delay(1000);

  }

  modeIdx = EEPROM.read(modeAddr);  // Read mode from EEPROM

  // Toggle mode and store the new mode in EEPROM

  EEPROM.write(modeAddr, modeIdx != 0 ? 0 : 1);

  EEPROM.commit();

  if (modeIdx != 0) {

    // BLE Mode

    Serial.println("BLE MODE");

    bleTask();

  } else {

    // WiFi Mode

    Serial.println("WIFI MODE");

    wifiTask();

  }

}

// Initializing and Advertising the BLE Device

void bleTask() {

  // Initialize the BLE device

  BLEDevice::init("ESP32\_NirvanT");

  // Create BLE server and set its callbacks

  pServer = BLEDevice::createServer();

  pServer->setCallbacks(new MyServerCallbacks());

  // Create BLE service

  BLEService\* pService = pServer->createService(SERVICE\_UUID);

  // Create BLE characteristic and set its properties and callbacks

  pCharacteristic = pService->createCharacteristic(

    CREDS\_CHARACTERISTIC\_UUID,

    BLECharacteristic::PROPERTY\_READ | BLECharacteristic::PROPERTY\_WRITE | BLECharacteristic::PROPERTY\_NOTIFY | BLECharacteristic::PROPERTY\_INDICATE);

  pCharacteristic->setCallbacks(new MyCallbacks());

  // Add descriptor to the characteristic

  pCharacteristic->addDescriptor(new BLE2902());

  // Start the service

  pService->start();

  // Start advertising

  BLEAdvertising\* pAdvertising = BLEDevice::getAdvertising();

  pAdvertising->addServiceUUID(SERVICE\_UUID);

  pAdvertising->setScanResponse(false);

  pAdvertising->setMinPreferred(0x0);  // Do not advertise this parameter

  BLEDevice::startAdvertising();

  Serial.println("Waiting for a client connection to notify...");

  Serial.println("Reset the device once the SSID and PASSWORD are received");

}

void wifiTask() {

  // Read WiFi credentials from EEPROM

  String receivedData = read\_String(wifiAddr);

  if (receivedData.length() > 0) {

    // Extract WiFi SSID and password from the received data

    String wifiName = getValue(receivedData, ',', 0);

    String wifiPassword = getValue(receivedData, ',', 1);

    if (wifiName.length() > 0 && wifiPassword.length() > 0) {

      Serial.print("WifiName : ");

      Serial.println(wifiName);

      Serial.print("wifiPassword : ");

      Serial.println(wifiPassword);

      // Connect to WiFi

      WiFi.begin(wifiName.c\_str(), wifiPassword.c\_str());

      Serial.print("Connecting to Wifi");

      while (WiFi.status() != WL\_CONNECTED) {

        Serial.print(".");

        delay(300);

      }

      Serial.println();

      Serial.print("Connected with IP: ");

      Serial.println(WiFi.localIP());

    }

  }

}

// Reading the stored value from EERPROM

String read\_String(int add) {

  char data[100];                      // Buffer to hold the read data

  int len = 0;                         // Length of the read data

  unsigned char k = EEPROM.read(add);  // Read the first byte from the specified address

  // Continue reading until a null terminator is encountered or the maximum length is reached

  while (k != '\0' && len < 500) {

    k = EEPROM.read(add + len);  // Read the next byte from EEPROM

    data[len] = k;               // Store the byte in the data buffer

    len++;                       // Increment the length counter

  }

  data[len] = '\0';  // Null-terminate the string

  return String(data);  // Return the read data as a String object

}

String getValue(String data, char separator, int index) {

  int found = 0;                     // Initialize the count of found separators

  int strIndex[] = { 0, -1 };        // Array to store start and end indices of the substring

  int maxIndex = data.length() - 1;  // The maximum valid index of the string

  // Iterate over the characters of the string

  for (int i = 0; i <= maxIndex && found <= index; i++) {

    // If the current character is the separator or the last character in the string

    if (data.charAt(i) == separator || i == maxIndex) {

      found++;                                    // Increment the count of found separators

      strIndex[0] = strIndex[1] + 1;              // Update the start index of the substring

      strIndex[1] = (i == maxIndex) ? i + 1 : i;  // Update the end index of the substring

    }

  }

  // If the number of found separators is greater than the desired index, return the substring

  return found > index ? data.substring(strIndex[0], strIndex[1]) : "";

}

void loop() {

  // Main loop code, runs repeatedly

}