**IoT con Bluetooth Low Energy (BLE)**

*Lo scopo di questa attività è utilizzare ESP32 per realizzare dispositivi che utilizzano il protocollo Bluetooth Low Energy (BLE).*

*Esp32 è programmabile con l’IDE di Arduino; fare riferimento al seguente link per le informazioni relative alla configurazione:* [*https://www.vincenzov.net/tutorial/ESP/ESP32/Arduino-IDE.htm*](https://www.vincenzov.net/tutorial/ESP/ESP32/Arduino-IDE.htm)

*Per informazioni di carattere generale su BLE, fare riferimento a*

[*https://www.dta.mil.nz/assets/Publications/A-Summary-of-Bluetooth-Low-Energy.pdf*](https://www.dta.mil.nz/assets/Publications/A-Summary-of-Bluetooth-Low-Energy.pdf)

**Parte 1: applicazione BLE server su ESP32**

Con l’installazione delle librerie per ESP32 vengono forniti esempi di applicazione ESP32 BLE; una di queste è un’applicazione server.

Fare riferimento a <https://randomnerdtutorials.com/esp32-bluetooth-low-energy-ble-arduino-ide/>

oppure a <https://www.electronicshub.org/esp32-ble-tutorial/>

per l’uso della libreria e la spiegazione dei programmi di esempio.

Un ulteriore esempio di programma che espone un servizio con due caratteristiche è riportato di seguito. È stato ottenuto modificando l’esempio fornito con la libreria.

#include <BLEDevice.h>

#include <BLEUtils.h>

#include <BLEServer.h>

// definizione degli UID del servizio e delle due caratteristiche

#define SERVICE\_UUID "00000000-1fb5-459e-8fcc-c5c9c331914b"

#define CHARACTERISTIC\_UUID1 "11111111-36e1-4688-b7f5-ea07361b26a8"

#define CHARACTERISTIC\_UUID2 "22222222-36e1-4688-b7f5-ea07361b26a8"

BLEServer \*pServer;

BLEService \*pService;

BLECharacteristic \*pCharacteristic1;

BLECharacteristic \*pCharacteristic2;

void setup()

{

Serial.begin(9600);

Serial.println("Starting BLE Server!");

// attivazione del device BLE e creazione delle caratteristiche

// la prima in R/W la seconda solo R

BLEDevice::init("Emilio");

pServer = BLEDevice::createServer();

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

pCharacteristic1 = pService->createCharacteristic(

CHARACTERISTIC\_UUID1,

BLECharacteristic::PROPERTY\_READ |

BLECharacteristic::PROPERTY\_WRITE

);

pCharacteristic2 = pService->createCharacteristic(

CHARACTERISTIC\_UUID2,

BLECharacteristic::PROPERTY\_READ

);

// impostazione dei valori iniziali delle caratteristiche

pCharacteristic1->setValue("init");

pCharacteristic2->setValue("1");

// attivazione del servizio e del'advertising

pService->start();

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

pAdvertising->addServiceUUID(SERVICE\_UUID);

pAdvertising->setScanResponse(true);

pAdvertising->setMinPreferred(0x12);

BLEDevice::startAdvertising();

}

int i = 0;

void loop()

{

// acquisizione del valore della prima caratteristica

std::string value = pCharacteristic1->getValue();

// impostazione del valore della seconda caratteristica

// (valore progressivo)

char str[100];

sprintf(str, "%d", i++);

pCharacteristic2->setValue(str);

// stampa

Serial.print("Caratteristica 1: ");

Serial.print(value.c\_str());

Serial.print(" - Caratteristica 2: ");

Serial.println(i);

// ripresa dell'advertising

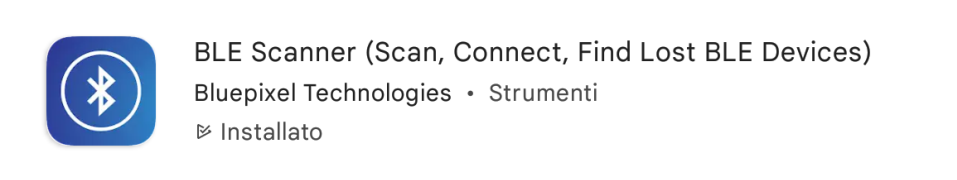
BLEDevice::startAdvertising();

delay(2000);

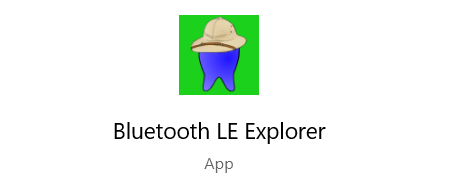
}

**Parte 2: connessione tramite un client standard**

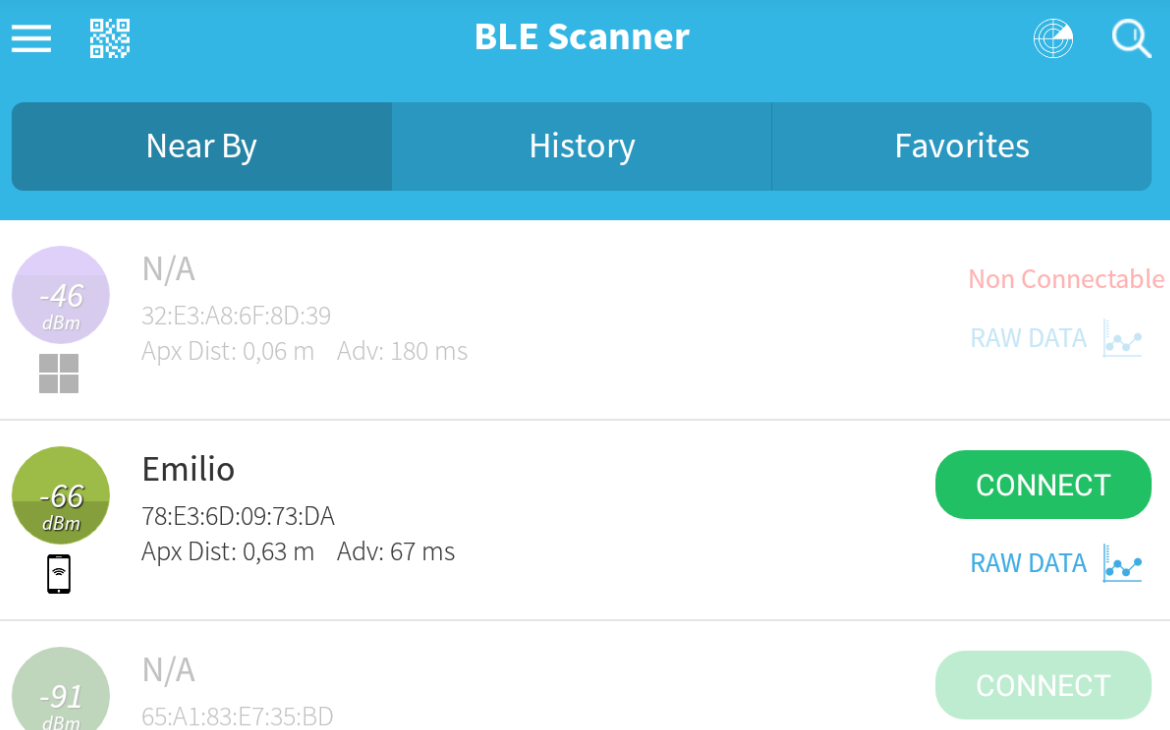
La verifica del funzionamento della parte 1 può essere effettuato utilizzando un BLE Scanner su Android, ad esempio

**

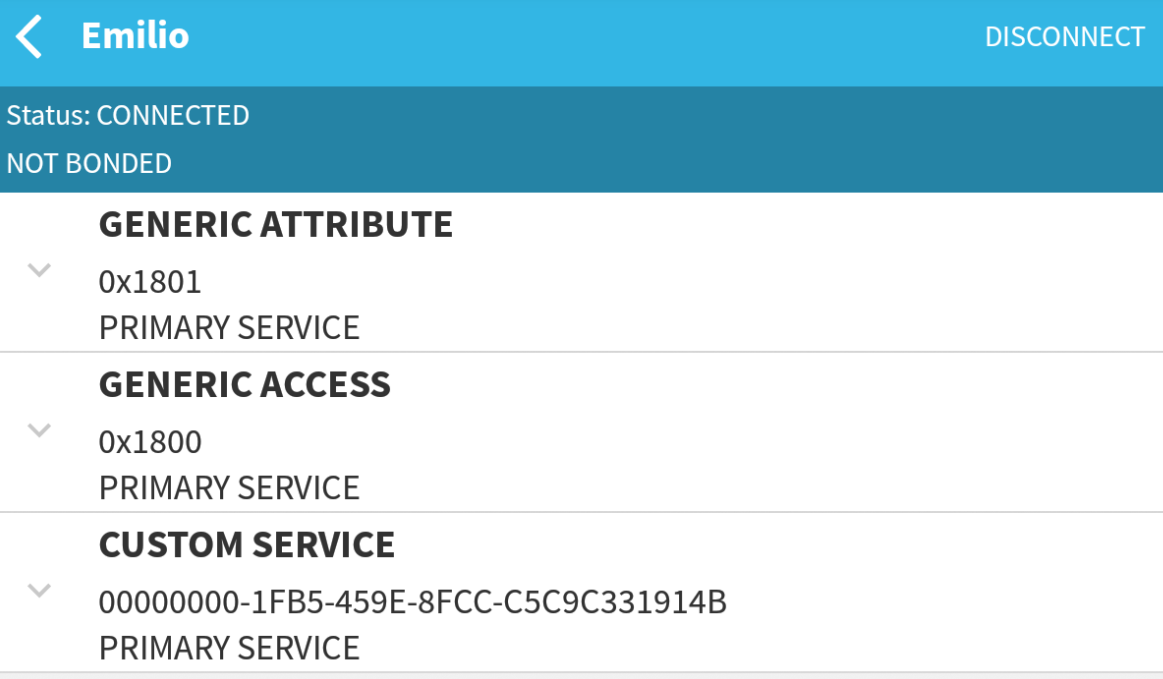
oppure, sotto Windows, con



Utilizzando BLE Scanner su Android con il precedente programma di esempio in esecuzione su ESP32 si ottiene:



e, scegliendo poi **CONNECT:**



dove, insieme ai servizi standard GENERICA ATTRIBUTE e GENERIC ACCESS, è presente il servizio con l’UID assegnato nel programma:

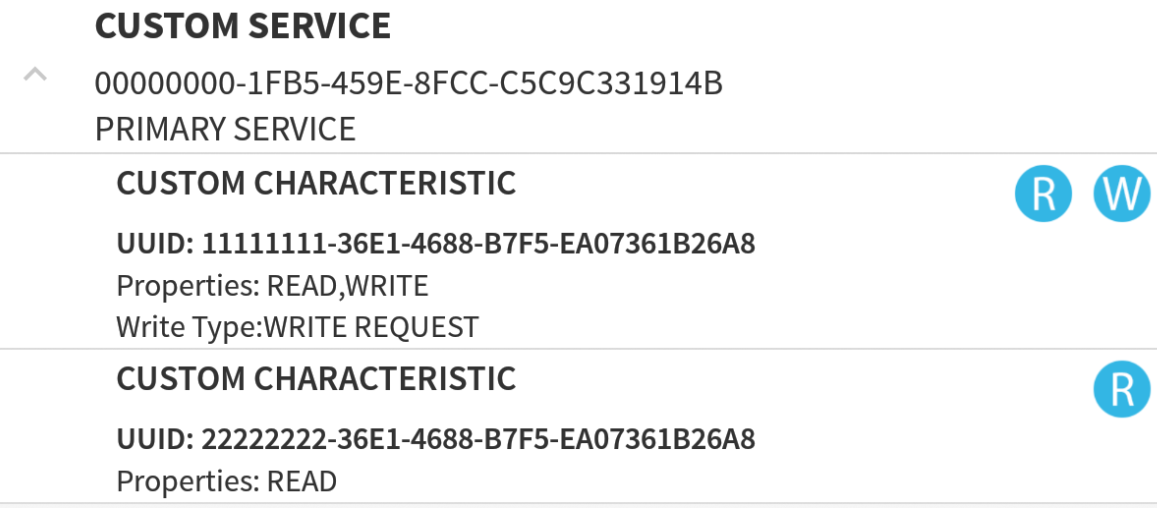
// definizione degli UID del servizio e delle due caratteristiche

#define SERVICE\_UUID "00000000-1fb5-459e-8fcc-c5c9c331914b"

#define CHARACTERISTIC\_UUID1 "11111111-36e1-4688-b7f5-ea07361b26a8"

#define CHARACTERISTIC\_UUID2 "22222222-36e1-4688-b7f5-ea07361b26a8

Aprendo il CUSTOM SERVICE si ottiene:



con le due caratteristiche definite, la prima R/W e la seconda solo R.

E’ possibile leggere il valore delle caratteristiche (la seconda in continuo aggiornamento nel programma su ESP32) e la prima anche scrivibile. Il monitor seriale nell’IDE di Arduino riporterà traccia della scrittura.

**Parte 4: applicazione client con Python**

E’ possibile utilizzare la libreria Bleak per accedere alle caratteristiche fornite da ESP32.

(<https://bleak.readthedocs.io/en/latest/index.html>)

Un esempio di programma (desunto dal programma di esempio) per la lettura della seconda caratteristica è sotto riportato. Per Accedere al device è necessario utilizzare il MAC Address, nell’esempio:



import asyncio

from bleak import BleakClient

import time

ESP32\_ADDRESS = "78:E3:6D:09:73:DA"

CHARACTERISTIC\_UUID2 = "22222222-36e1-4688-b7f5-ea07361b26a8"

async def main(address,uuid):

async with BleakClient(address) as client:

while True:

value = await client.read\_gatt\_char(uuid)

print(value)

time.sleep(5)

asyncio.run(main(ESP32\_ADDRESS,CHARACTERISTIC\_UUID2))