



Università  
degli Studi  
della Campania  
*Luigi Vanvitelli*

# Università di degli Studi della Campania

## Luigi Vanvitelli - Dipartimento di Ingegneria

### *Laurea Triennale in Ingegneria Elettronica e Informatica*

**Laboratorio di Sviluppo di Applicazioni per IoT**  
**a.a. 2023-2024**

**Comunicazione Seriale e Wireless**

**Docente:** Carlo Mazzocca  
**e-mail:** carlo.mazzocca@unibo.it

# Motivazioni

---

- Scambio di dati
- Controllo da remoto
- Sincronizzazione
- Aggiornamenti firmware
- Condivisione di risorse

# Comunicazione Seriale

---

- Modalità di trasmissione dei dati sequenziale utilizzata per collegare dispositivi elettronici
- Fondamentale per il trasferimento affidabile di dati tra microcontrollori, sensori, attuatori e altri dispositivi
- Quando si parla di comunicazione seriale, solitamente ci si riferisce all'**Universal Asynchronous Receiver/Transmitter (UART)**

# Vantaggi

---

- **Affidabilità:** Utilizzando cavi fisici è meno soggetta a interferenze elettromagnetiche
- **Sicurezza:** Maggiore sicurezza, meno vulnerabile ad attacchi di intercettazione
- **Velocità e Latenza:** Velocità di trasferimento dati più elevata e una latenza inferiore
- **Controllo Diretto:** Controllo più diretto e preciso essendo i collegamenti fisici

# Svantaggi

---

- **Flessibilità:** Utilizzando cavi fisici per il collegamento, la flessibilità e mobilità è limitata
- **Complessità dell'Installazione:** Dovendo collegare fisicamente i dispositivi, l'installazione può essere più complessa
- **Limitazioni Distanza:** I cavi hanno dimensione limitata, influenzando la distanza massima
- **Esposizione Danni Fisici:** Vulnerabili a danni fisici come tagli o danni da torsioni che possono interrompere la connessione tra i dispositivi

# Applicazioni

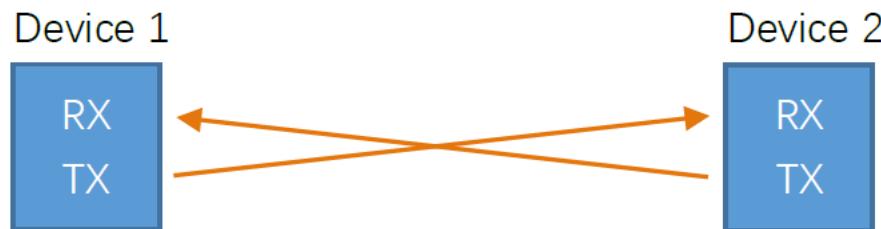
---

- Sistemi di controllo e monitoraggio
- Sistemi di trasporto e logistica
- Apparecchiature di comunicazione e reti
- Automobili

# UART

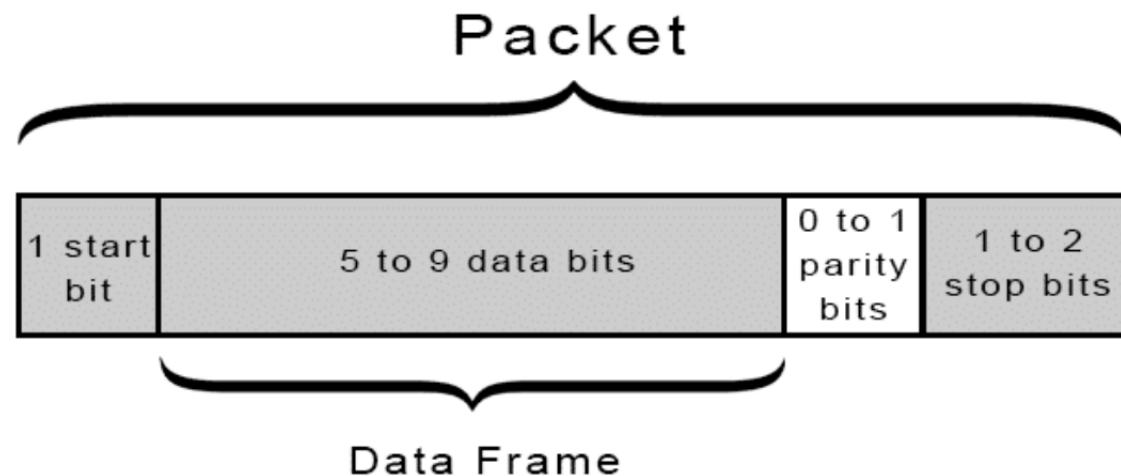
---

- UART prevede due linee di comunicazione: una per inviare i dati (TX) e una per riceverli
- Prima che la comunicazione possa avere inizio, i dispositivi devono utilizzare lo stesso **baud rate**
- Il baud rate definisce la velocità massima di trasmissione dei dati. Solitamente settato a 9600 o 115200



# UART

- UART trasmette i dati in maniera asincrona, non ci sono segnali di clock che temporizzano i bit di output
- L'UART trasmettente aggiunge un bit di inizio e uno di fine ai pacchetti trasmessi
- Il bit di parità serve a rilevare se ci sono state alterazioni nella trasmissione, può rilevare solo un solo errore



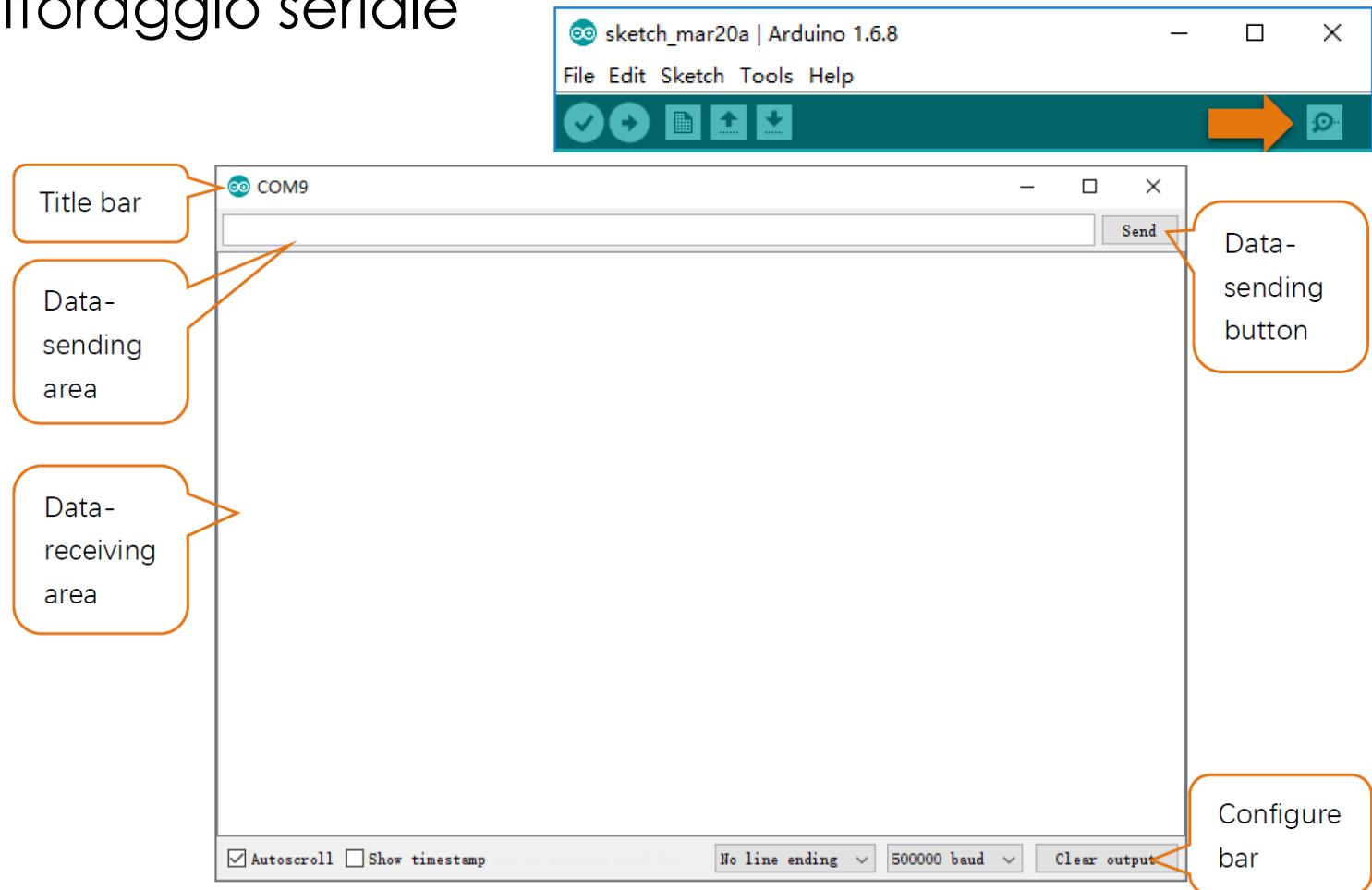
# Porta Seriale ESP32

- L'ESP32 utilizza la porta USB per trasferimenti seriali che può essere utilizzata per comunicare con il computer tramite un cavo USB
- Il vostro computer identifica i dispositivi seriali che si connettono come COMx



# Finestra per il Monitoraggio Seriale

- Arduino Software IDE offre una finestra per il monitoraggio seriale



# Inviare Dati - Sketch

```
1 void setup() {
2     Serial.begin(115200);
3     Serial.println("ESP32 initialization completed! ");
4 }
5
6 void loop() {
7     Serial.printf("Running time : %.1f s\n", millis() / 1000.0f);
8     delay(1000);
9 }
```

```
void begin(unsigned long baud, uint32_t config=SERIAL_8N1, int8_t rxPin=-1,
           int8_t txPin=-1, bool invert=false, unsigned long timeout_ms = 20000UL);
```

Initializes the serial port. Parameter baud is baud rate, other parameters generally use the default value.

```
size_t println( arg );
```

Print to the serial port and wrap. The parameter **arg** can be a number, a character, a string, an array of characters, etc.

```
size_t printf(const char * format, ...) __attribute__ ((format (printf, 2, 3)));
```

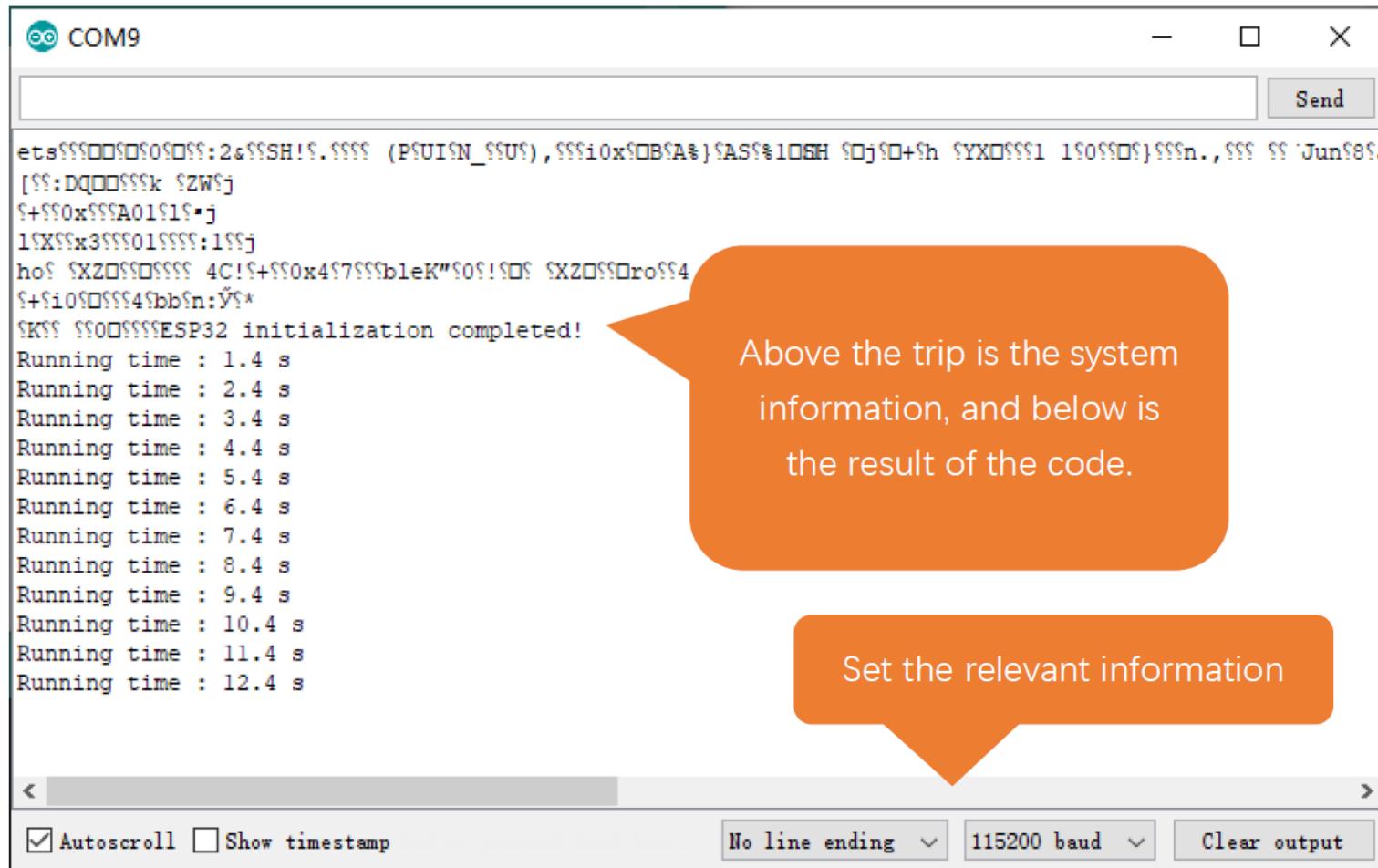
Print formatted content to the serial port in the same way as print in standard C.

```
unsigned long millis();
```

Returns the number of milliseconds since the current system was booted.

# Inviare Dati – Output

- All'avvio, è utilizzato un baud rate di 120000



# Ricevere Dati- Sketch

```
1 String inputString = "";      //a String to hold incoming data
2 bool stringComplete = false; // whether the string is complete
3
4 void setup() {
5     Serial.begin(115200);
6     Serial.println(String("\nESP32 initialization completed! \n")
7                     + String("Please input some characters, \n")
8                     + String("select \"Newline\" below and click send button. \n"));
9 }
10
11 void loop() {
12     if (Serial.available()) {          // judge whether data has been received
13         char inChar = Serial.read();    // read one character
14         inputString += inChar;
15         if (inChar == '\n') {
16             stringComplete = true;
17         }
18     }
19     if (stringComplete) {
20         Serial.printf("inputString: %s \n", inputString);
21         inputString = "";
22         stringComplete = false;
23     }
24 }
```

`String();`

Constructs an instance of the String class.

For more information, please visit

<https://www.arduino.cc/reference/en/language/variables/data-types/stringobject/>

`int available(void);`

Get the number of bytes (characters) available for reading from the serial port. This is data that's already arrived and stored in the serial receive buffer.

`Serial.read();`

Reads incoming serial data.

# Ricevere Dati – Output

The image shows two terminal windows titled "COM9" running on a Windows operating system. Both windows display the same serial output from an ESP32 microcontroller during its initialization process.

**Terminal Window 1 (Top):**

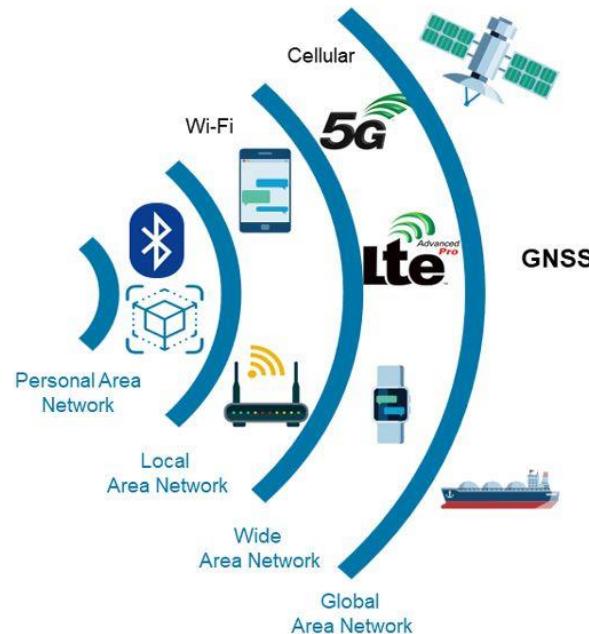
- Content: The terminal shows the ESP32's boot sequence, including the loading of the bootloader, the selection of the correct port ("COM9"), and the start of the main program. It also displays the message "ESP32 initialization completed!" followed by a prompt for user input.
- Buttons: A "Send" button is located in the top right corner of the window.

**Terminal Window 2 (Bottom):**

- Content: This window shows the same initialization log as the first one. At the bottom, it displays the command "inputString: ABCDEFG".
- Controls: The bottom right of this window features an orange callout bubble containing the text "Newline, 115200 baud". Below the bubble are three dropdown menus: "Newline", "115200 baud", and "Clear output".
- Bottom Buttons: At the very bottom of the window are two checkboxes: "Autoscroll" (checked) and "Show timestamp" (unchecked). To the right of these are three more buttons: "Newline", "115200 baud", and "Clear output".

# Comunicazione Wireless

- Modalità di trasmissione di informazioni tra dispositivi che non prevede l'utilizzo di fili o cavi fisici
- Ogni sistema di comunicazione wireless è costituito da un trasmittitore, un ricevitore e delle antenne



# Vantaggi

---

- **Flessibilità:** Abolizione del cablaggio, maggiore flessibilità nell'installazione dei dispositivi
- **Accessibilità:** Rende l'accesso alle comunicazioni e alle informazioni più ampio e conveniente senza restrizioni di posizione
- **Mobilità:** Consente connessione e comunicazione in movimento, senza vincoli di cavi o fili
- **Scalabilità:** Un singolo dispositivo di ricetrasmissione può coprire un'ampia zona ad un prezzo di impianto notevolmente più basso

# Svantaggi

---

- **Interferenze:** Influenzate da altre apparecchiature e da fattori ambientali, causando interferenze e riducendo le prestazioni
- **Sicurezza:** Soggette a **intercettazioni e intrusioni** non autorizzate, richiedendo meccanismi di sicurezza aggiuntivi
- **Prestazioni:** Variano a seconda della **distanza** e dell'**ambiente**
- **Consumo Energetico:** Richiedono una **maggior** quantità di **energia** rispetto alle loro controparti cablate

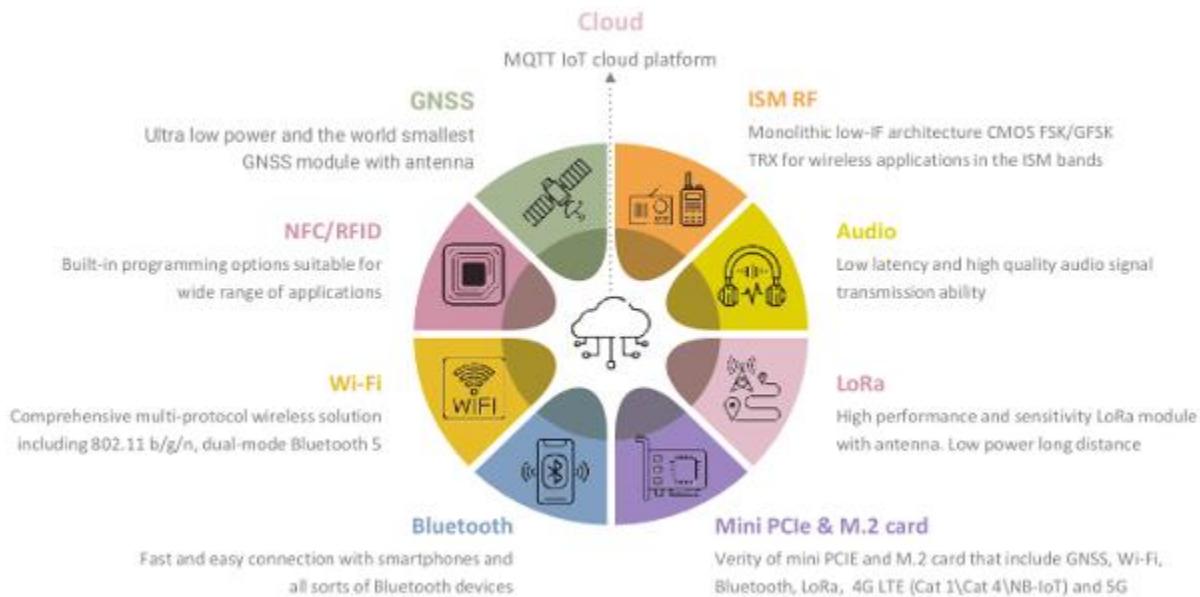
# Applicazioni

---

- Reti Wi-Fi
- Telefonia mobile
- GPS
- Televisione e radio
- Radar e sistemi di navigazione aerea
- Comunicazioni via satellite
- Sistemi di sicurezza e sorveglianza
- Domotica

# Tecnologie Chiave

- **Bluetooth:** comunicazioni a corto raggio tra dispositivi
- **Wi-Fi:** connessione a Internet senza fili
- **Cellulari:** Tecnologie come GSM, 3G, 4G, e 5G per le comunicazioni mobili
- **RFID:** Identificazione e tracciamento degli oggetti



# Bluetooth

---

- Standard per comunicazioni wireless a **corto raggio tra dispositivi**, utilizzato per trasmettere dati e segnali audio
- Sviluppato da **Ericsson** negli anni '90, è stato successivamente standardizzato e gestito da Bluetooth Special Interest Group (SIG)
- Il **raggio operativo** è di circa 10 metri, con la possibilità di estendersi fino a 100 in condizioni ottimali
- La **velocità di trasferimento** dipende dalla versione **integrata nel dispositivo**. Ad esempio, la versione 1.1 gestisce trasferimenti fino a 723,1 kbit/s, mentre la versione 2.0 arriva fino a 3 Mbit/s



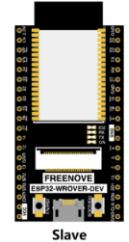
# Bluetooth

---

- Il protocollo Bluetooth lavora nelle frequenze libere di 2.4 e 2.5 GHz
- La banda è divisa in 79 canali per ridurre le interferenze e provvede a commutare tra i vari canali 1600 volte al secondo
- Progettato per un consumo energetico molto basso rendendolo ideale per dispositivi alimentati a batteria
- Utilizza protocolli di crittografica per garantire la sicurezza delle comunicazioni tra dispositivi
- I dispositivi devono essere accoppiati prima di poter comunicare tra loro riducendo il rischio di accesso non autorizzato

# Bluetooth

- Ogni dispositivo Bluetooth è in grado di gestire simultaneamente la comunicazione con altri 7
- Il collegamento è di tipo **master-slave**, quindi solo un dispositivo per volta può comunicare con lo slave
- **Master Mode:** un dispositivo master può collegarsi con uno slave. Il master cerca e seleziona uno slave nelle vicinanze, richiede informazioni sull'altro device
- **Slave Mode:** lo slave può solo accettare richieste di connessione, ma non può avviinarne

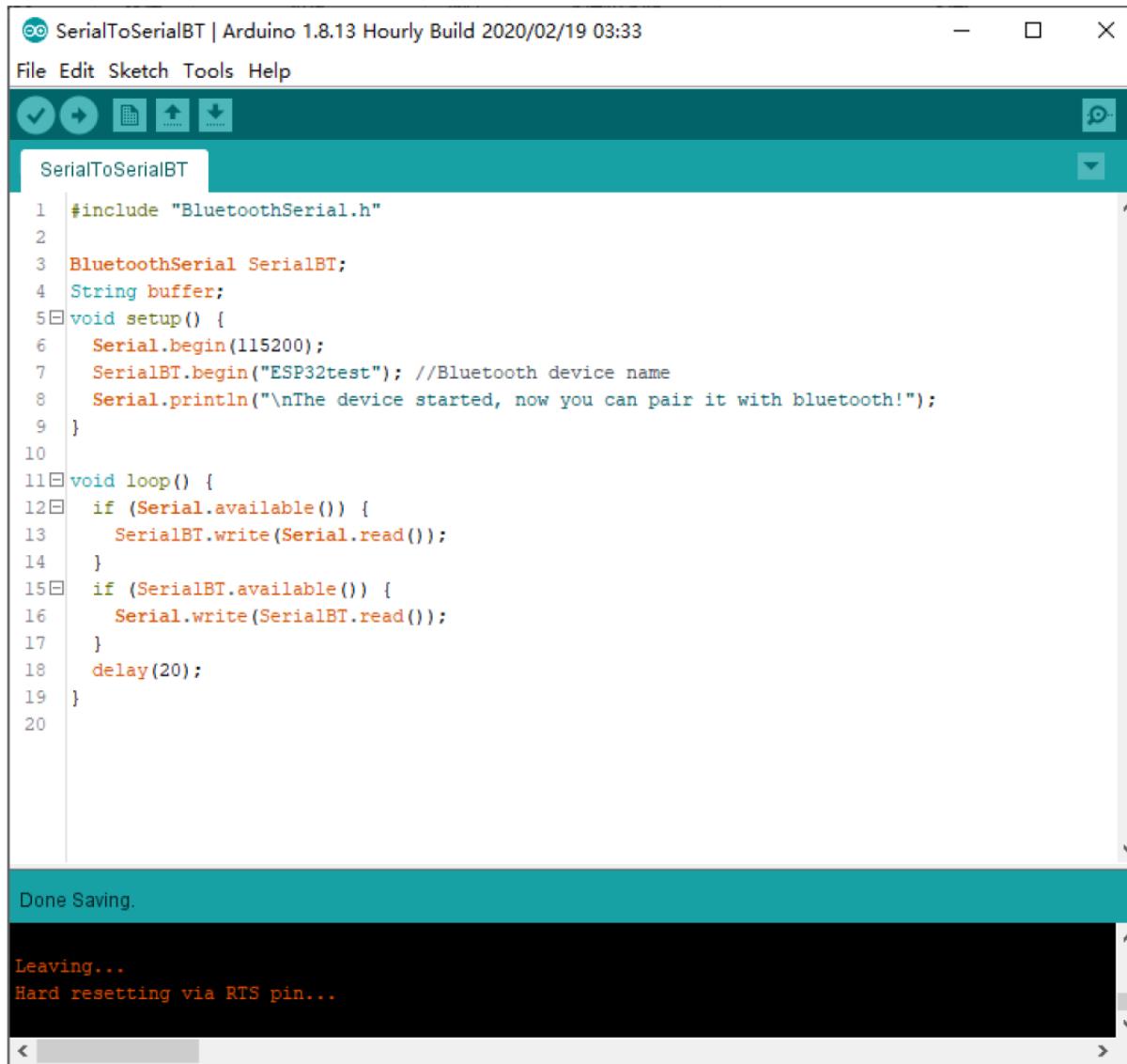


# ESP32 Bluetooth

---

- L'ESP32 integra un modulo Bluetooth sia nella versione classica che in quella a basso consumo
- **Bluetooth Low Energy (BLE)** è progettato per applicazioni che richiedono una connettività a basso consumo energetico
- Le modalità di trasmissione dati supportate sono master e slave

# Serial Bluetooth - Sketch



The screenshot shows the Arduino IDE interface with a sketch titled "SerialToSerialBT". The code implements a serial-to-Bluetooth converter using the `BluetoothSerial.h` library. It initializes the serial port at 115200 baud, names the Bluetooth device "ESP32test", and prints a startup message. The `loop()` function reads from the serial port and writes to the Bluetooth port, and vice versa, with a 20ms delay between reads.

```
#include "BluetoothSerial.h"

BluetoothSerial SerialBT;
String buffer;

void setup() {
    Serial.begin(115200);
    SerialBT.begin("ESP32test"); //Bluetooth device name
    Serial.println("\nThe device started, now you can pair it with bluetooth!");
}

void loop() {
    if (Serial.available()) {
        SerialBT.write(Serial.read());
    }
    if (SerialBT.available()) {
        Serial.write(SerialBT.read());
    }
    delay(20);
}
```

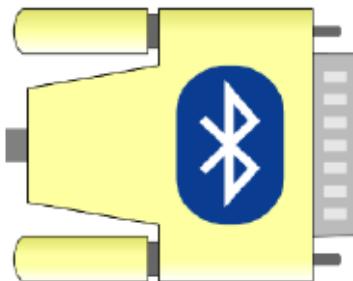
Done Saving.

Leaving...  
Hard resetting via RTS pin...

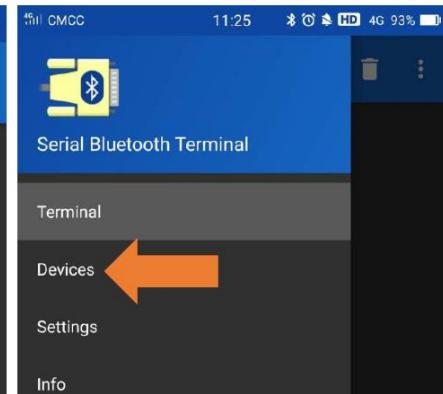
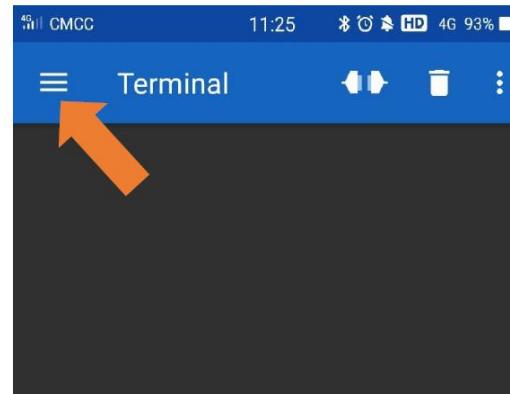
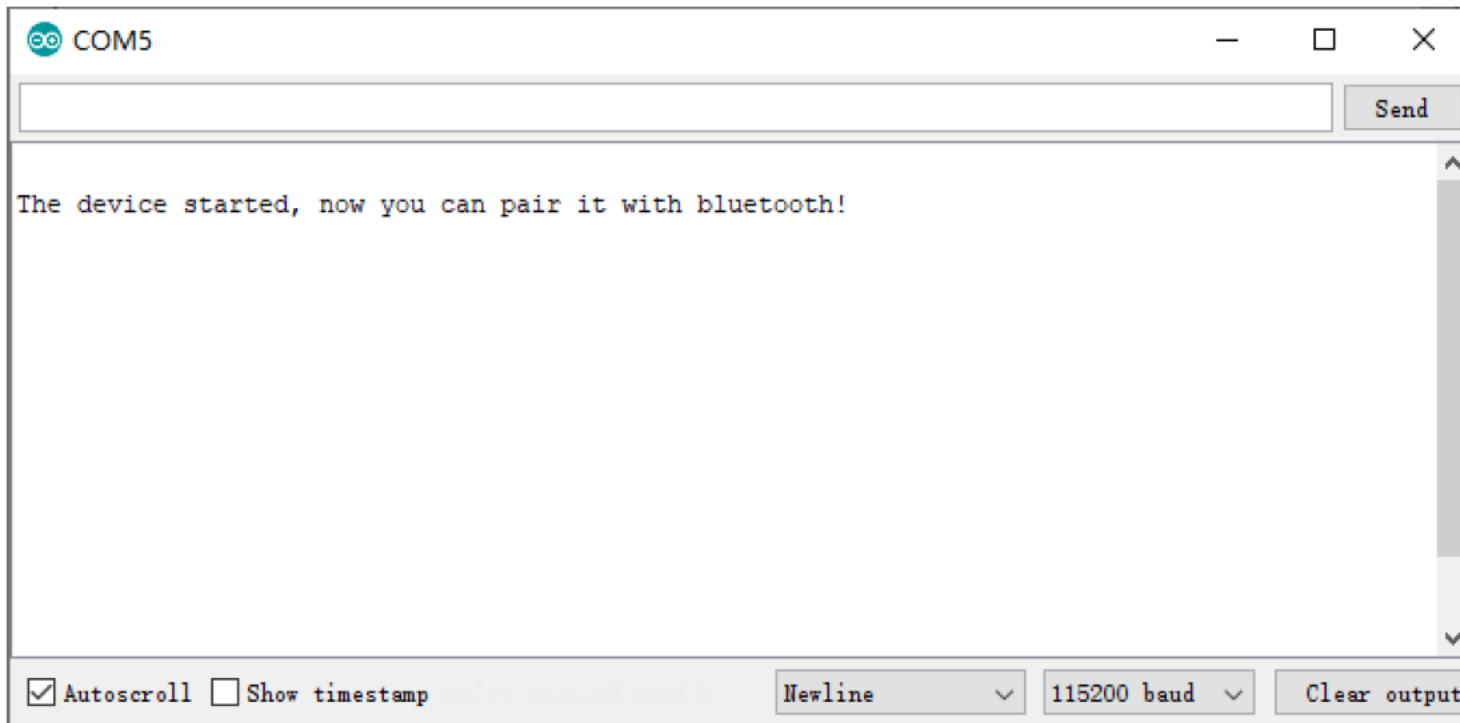
# Terminale Bluetooth

---

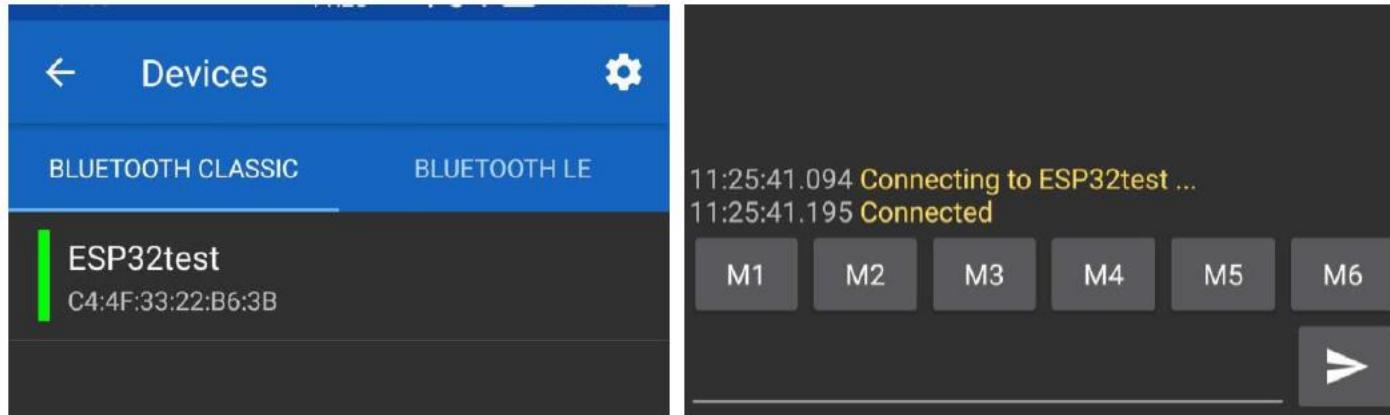
- Scaricate sul vostro telefono **Serial Bluetooth Terminal**



# Serial Bluetooth - Output



# Serial Bluetooth - Output



Hi!

□) \$^ \$@J!i\$6!B\$) \$\*§  
C9\$R\$OV!§  
The device started, now you can pair it with bluetooth!  
Hello!

11:25:41.094 Connecting to ESP32test ...  
11:25:41.195 Connected  
11:26:11.913 Hello!  
11:26:24.759 Hi!

Autoscroll  Show timestamp

Newline 9600 baud Clear output

# Class Bluetooth Serial

---

## Class BluetoothSerial

This is a class library used to operate **BluetoothSerial**, which can directly read and set **BluetoothSerial**.

Here are some member functions:

**begin(localName, isMaster)**: Initialization function of the Bluetooth

name: name of Bluetooth module; Data type: String

**isMaster**: bool type, whether to set Bluetooth as Master. By default, it is false.

**available()**: acquire digits sent from the buffer, if not, return 0.

**read()**: read data from Bluetooth, data type of return value is int.

**readString()**: read data from Bluetooth, data type of return value is String.

**write(val)**: send an int data val to Bluetooth.

**write(str)**: send an Srting data str to Bluetooth.

**write(buf, len)**: Sends the first len data in the buf Array to Bluetooth.

**setPin(const char \*pin)**: set a four-digit Bluetooth pairing code. By default, it is 1234

**connet(remoteName)**: connect a Bluetooth named remoteName, data type: String

**connect(remoteAddress[])**: connect the physical address of Bluetooth, data type: uint8-t.

**disconnect()**: disconnect all Bluetooth devices.

**end()**: disconnect all Bluetooth devices and turn off the Bluetooth, release all occupied space

# Bluetooth Low Energy - Sketch

```
1 #include <BLEDevice.h>
2 #include <BLEServer.h>
3 #include <BLEUtils.h>
4 #include <BLE2902.h>
5 #include <String.h>
6
7 BLECharacteristic *pCharacteristic;
8 bool deviceConnected = false;
9 uint8_t txValue = 0;
10 long lastMsg = 0;
11 String rxload="Test\n";
12
13 #define SERVICE_UUID      "6E400001-B5A3-F393-E0A9-E50E24DCCA9E"
14 #define CHARACTERISTIC_UUID_RX "6E400002-B5A3-F393-E0A9-E50E24DCCA9E"
15 #define CHARACTERISTIC_UUID_TX "6E400003-B5A3-F393-E0A9-E50E24DCCA9E"
16
17 class MyServerCallbacks: public BLEServerCallbacks {
18     void onConnect(BLEServer* pServer) {
19         deviceConnected = true;
20     };
21     void onDisconnect(BLEServer* pServer) {
22         deviceConnected = false;
23     }
24 };
25
26 class MyCallbacks: public BLECharacteristicCallbacks {
27     void onWrite(BLECharacteristic *pCharacteristic) {
28         std::string rxValue = pCharacteristic->getValue();
29         if (rxValue.length() > 0) {
30             rxload="";
31             for (int i = 0; i < rxValue.length(); i++) {
32                 rxload +=(char)rxValue[i];
33             }
34         }
35     }
36 };
37
38 void setupBLE(String BLEName) {
39     const char *ble_name=BLEName.c_str();
40     BLEDevice::init(ble_name);
41     BLEServer *pServer = BLEDevice::createServer();
42     pServer->setCallbacks(new MyServerCallbacks());
43     BLEService *pService = pServer->createService(SERVICE_UUID);
44     pCharacteristic=
45     pService->createCharacteristic(CHARACTERISTIC_UUID_RX,BLECharacteristic::PROPERTY_WRITE);
46     pCharacteristic->addDescriptor(new BLE2902());
47     BLECharacteristic *pCharacteristic =
48     pService->createCharacteristic(CHARACTERISTIC_UUID_TX,BLECharacteristic::PROPERTY_NOTIFY);
49     pCharacteristic->setCallbacks(new MyCallbacks());
50     pService->start();
51     pServer->getAdvertising()->start();
52     Serial.println("Waiting a client connection to notify... ");
53 }
54
55 void setup() {
56     Serial.begin(9600);
57     setupBLE("ESP32_Bluetooth");
58 }
59
60 void loop() {
61     long now = millis();
62     if (now - lastMsg > 1000) {
63         if (deviceConnected&&rxload.length()>0) {
64             Serial.println(rxload);
65             rxload="";
66         }
67         if(Serial.available()>0){
68             String str=Serial.readString();
69             const char *newValue=str.c_str();
70             pCharacteristic->setValue(newValue);
71             pCharacteristic->notify();
72         }
73         lastMsg = now;
74     }
75 }
```

# UUID

---

- Universally Unique Identifiers (UUID) sono utilizzati per identificare in maniera unica i servizi e le caratteristiche offerte dal dispositivo
- Una caratteristica in BLE è un attributo o una funzionalità

```
13 #define SERVICE_UUID          "6E400001-B5A3-F393-E0A9-E50E24DCCA9E"  
14 #define CHARACTERISTIC_UUID_RX "6E400002-B5A3-F393-E0A9-E50E24DCCA9E"  
15 #define CHARACTERISTIC_UUID_TX "6E400003-B5A3-F393-E0A9-E50E24DCCA9E"
```

# Callback

- **MyServerCallbacks:** invocato quando un device si connette o disconnette al BLE server

```
17 class MyServerCallbacks: public BLEServerCallbacks {  
18     void onConnect(BLEServer* pServer) {  
19         deviceConnected = true;  
20     };  
21     void onDisconnect(BLEServer* pServer) {  
22         deviceConnected = false;  
23     }  
24 };
```

- **MyCallbacks:** invocato quando si scrive alla caratteristica BLE corrispondente

```
26 class MyCallbacks: public BLECharacteristicCallbacks {  
27     void onWrite(BLECharacteristic *pCharacteristic) {  
28         std::string rxValue = pCharacteristic->getValue();  
29         if (rxValue.length() > 0) {  
30             rxload="";  
31             for (int i = 0; i < rxValue.length(); i++) {  
32                 rxload +=(char)rxValue[i];  
33             }  
34         }  
35     }  
36 };
```

# Terminale BLE per iPhone



**LightBlue®** 4+

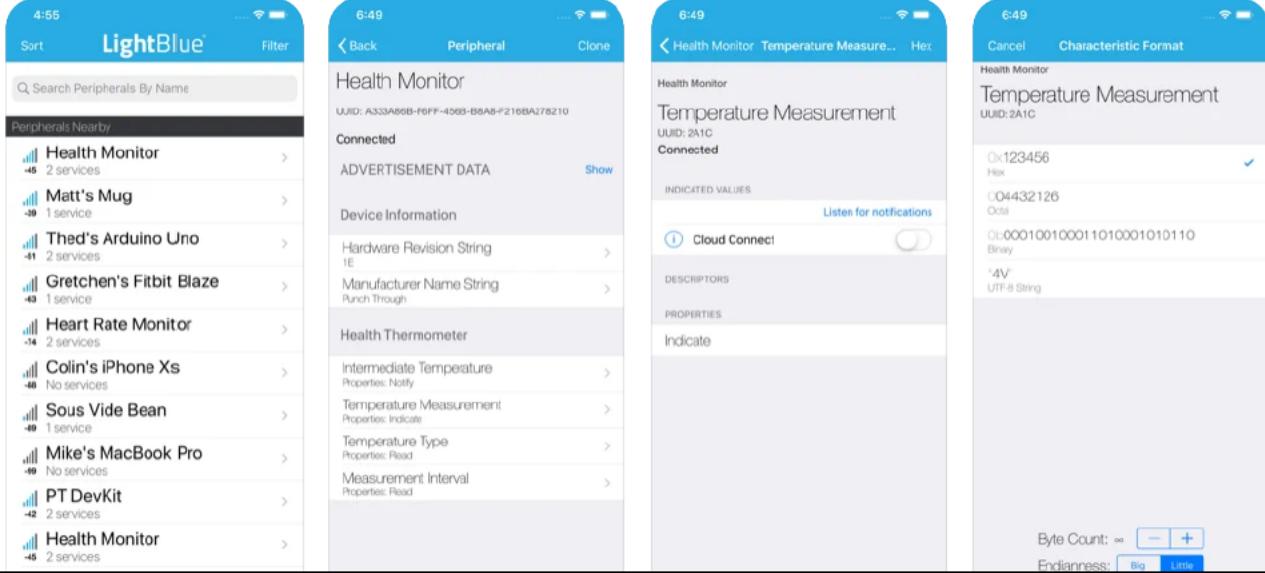
The go-to BLE development tool

Punch Through

★★★★★ 4.0 • 4 Ratings

Free

**Screenshots** Mac iPhone iPad



# Connessione e Ricezione

LightBlue®

17:17 4G 53%

ESP32\_Bluetooth  
-54dBm 24:62:48:FD:D1:8E

M31\_BT\_1D30C8  
-97dBm 57:4C:54:10:30:C8

Unnamed  
-72dBm 17:AB:4E:D5:F7:1A

Unnamed  
-76dBm 6E:8B:A3:4F:D6:C4

Scan Bonded

Receive

ESP32\_Bluetooth

17:30 4G 48%

← ESP32\_Bluetooth ⋮

GENERIC ATTRIBUTE

Service Changed  
Indicate

GENERIC ACCESS

Device Name  
Readable →

Appearance  
Readable →

Central Address Resolution  
Readable →

6e400001-b5a3-f393-e0a9-e50e24dcca9e

6e400003-b5a3-f393-e0a9-e50e24dcca9e  
Notify →

6e400002-b5a3-f393-e0a9-e50e24dcca9e  
Writable →

17:33 4G 47%

← 6e400003-b5a3-f393-e0a9-...  
0e400003-b5a3-f393-e0a9-e50e24dcca9e

✗ Readable Able to be read from

✗ Writable Able to be written to

✓ Supports notifications/indications Able to be subscribed to for notifications/indications on changes to the characteristic

Data format Hex

READ/INDICATE

READ AGAIN

No value read recently Tap on one of the buttons above — if available — to begin

Unsigned Little-Endian → to

Signed Little-Endian

DESCRIPTIONS

Client Characteristic 00002902-0000-10

Unsigned Big-Endian

Signed Big-Endian

17:34 4G 47%

← 6e400003-b5a3-f393-e0a9-...  
0e400003-b5a3-f393-e0a9-e50e24dcca9e

✗ Readable Able to be read from

✗ Writable Able to be written to

✓ Supports notifications/indications Able to be subscribed to for notifications/indications on changes to the characteristic

Data format UTF-8 String

READ/INDICATED VALUES

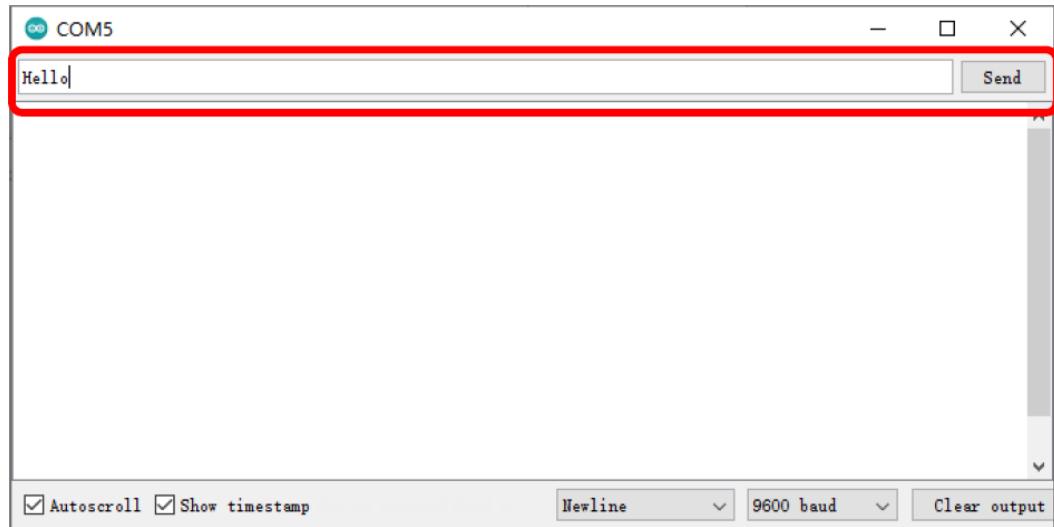
READ AGAIN SUBSCRIBE

No value read recently Tap on one of the buttons above — if available — to begin

DESCRIPTIONS

Successfully opted out of characteristic changes

# Ricezione



A screenshot of a mobile application interface for a BLE characteristic. The characteristic's UUID is shown as "6e400003-b5a3-f393-e0a9-...". The data format is set to "UTF-8 String", which is highlighted with a red rectangle. The value of the characteristic is "Hello", also highlighted with a red rectangle. The application also displays other information such as "Readable", "Writable", and "Supports notifications/indications".

6e400003-b5a3-f393-e0a9-...

6e400003-b5a3-f393-e0a9-e50e24dcca9e

**Readable**  
Able to be read from

**Writable**  
Able to be written to

**Supports notifications/indications**  
Able to be subscribed to for notifications/indications on changes to the characteristic

Data format: UTF-8 String

READ/INDICATED VALUES

READ AGAIN UNSUBSCRIBE

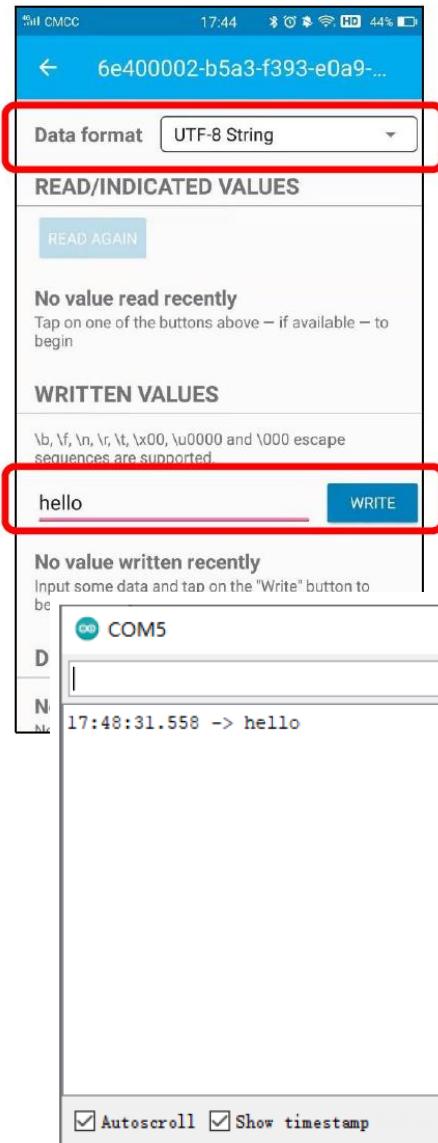
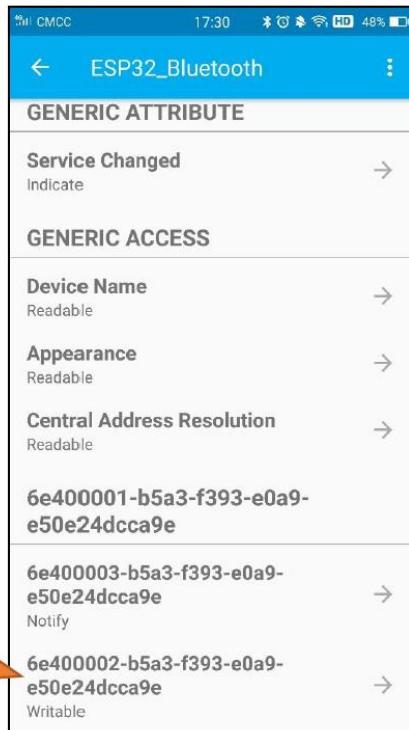
Hello

Fri Feb 05 17:40:09 GMT+08:00 2021

DESCRIPTORS

Client Characteristic Configuration  
00002902-0000-1000-8000-00805f9b34fb

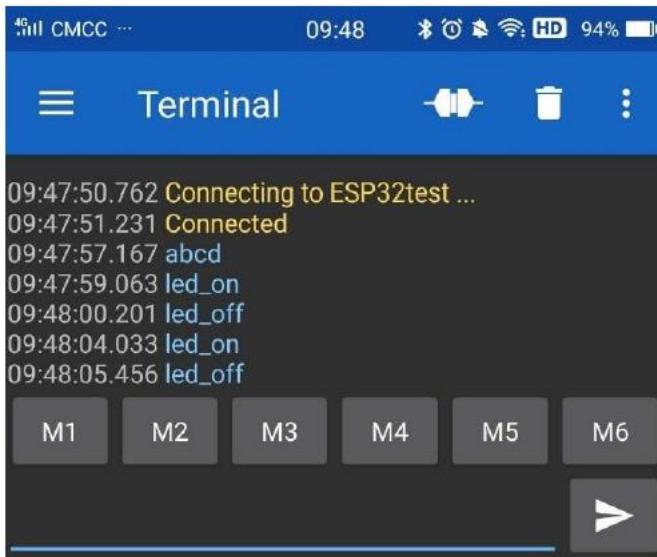
# Invio



# Controllo LED tramite BLE - Sketch

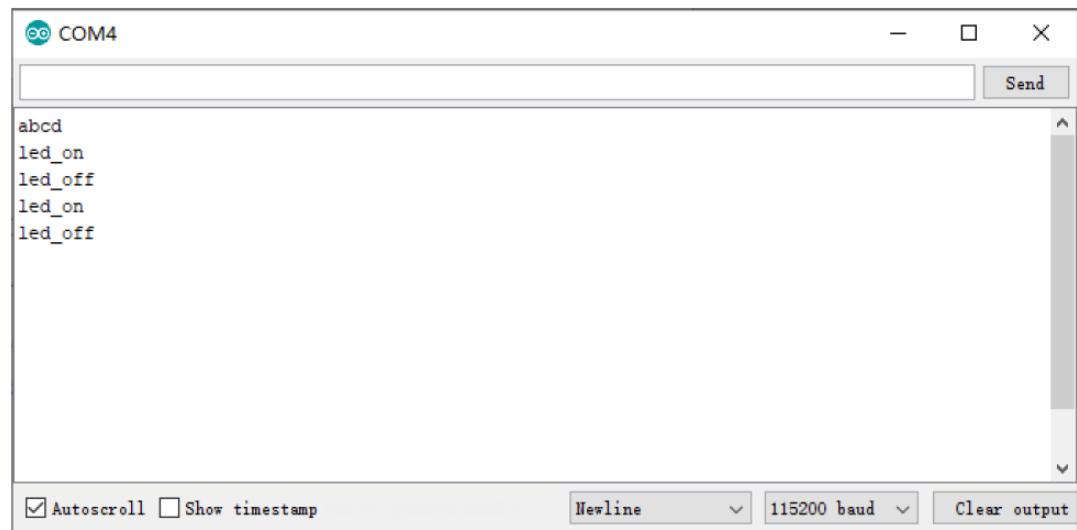
```
1 #include "BluetoothSerial.h"
2 #include "string.h"
3 #define LED 2
4 BluetoothSerial SerialBT;
5 char buffer[20];
6 static int count = 0;
7 void setup() {
8     pinMode(LED, OUTPUT);
9     SerialBT.begin("ESP32test"); //Bluetooth device name
10    Serial.begin(115200);
11    Serial.println("\nThe device started, now you can pair it with Bluetooth! ");
12 }
13
14 void loop() {
15     while(SerialBT.available())
16     {
17         buffer[count] = SerialBT.read();
18         count++;
19     }
20     if(count>0) {
21         Serial.print(buffer);
22         if(strncmp(buffer, "led_on", 6)==0) {
23             digitalWrite(LED, HIGH);
24         }
25         if(strncmp(buffer, "led_off", 7)==0) {
26             digitalWrite(LED, LOW);
27         }
28         count=0;
29         memset(buffer, 0, 20);
30     }
31 }
```

# Controllo LED tramite BLE - Output



```
09:47:50.762 Connecting to ESP32test ...
09:47:51.231 Connected
09:47:57.167 abcd
09:47:59.063 led_on
09:48:00.201 led_off
09:48:04.033 led_on
09:48:05.456 led_off
```

M1 M2 M3 M4 M5 M6 >



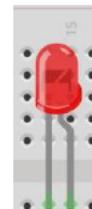
```
abcd
led_on
led_off
led_on
led_off
```

Autoscroll  Show timestamp Newline 115200 baud Clear output

Send: "led\_on"



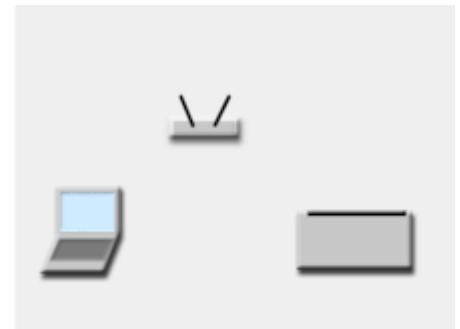
Send: "led\_off"



# Wi-Fi

---

- Wi-Fi è un insieme di tecnologie che consente a dispositivi elettronici di scambiare dati su una rete, incluso connessioni Internet ad alta velocità
- Solo i prodotti che completano i test di certificazione di interoperabilità possono utilizzare il termine Wi-Fi Certified
- I dispositivi compatibili Wi-Fi possono connettersi a Internet tramite un WLAN e un accesso wireless (access point)



# Wi-Fi

---

- La scheda wireless all'interno dei dispositivi trasforma i dati in segnali radio, che vengono poi trasmessi attraverso l'antenna
- I router Wi-Fi ricevono questi segnali e li decodificano per instradare le informazioni verso Internet tramite una rete locale o una connessione Ethernet cablata
- Wi-Fi utilizza diversi standard come 802.11ac o Wi-Fi 6 che si differenziano per velocità, frequenze, a distanza per soddisfare
- Lavora nelle frequenze libere di 2.4 GHz

# Wi-Fi vs Bluetooth

---

- **Utilizzo:** Bluetooth è stato progettato per la connessioni di dispositivi personali, Wi-Fi per fornire accesso ad Internet
- **Distanza:** Bluetooth è una tecnologia a corto raggio, Wi-Fi riesce ad offrire una copertura più ampia a seconda della potenza del router
- **Prestazioni:** Bluetooth richiede un minore consumo energetico. Wi-Fi offre una maggiore velocità di trasferimento

# Wi-Fi – Modalità Station

- Se configurato in modalità station, l'ESP32 si comporta come un client Wi-Fi: può connettersi al router e comunicare con gli altri dispositivi nella rete



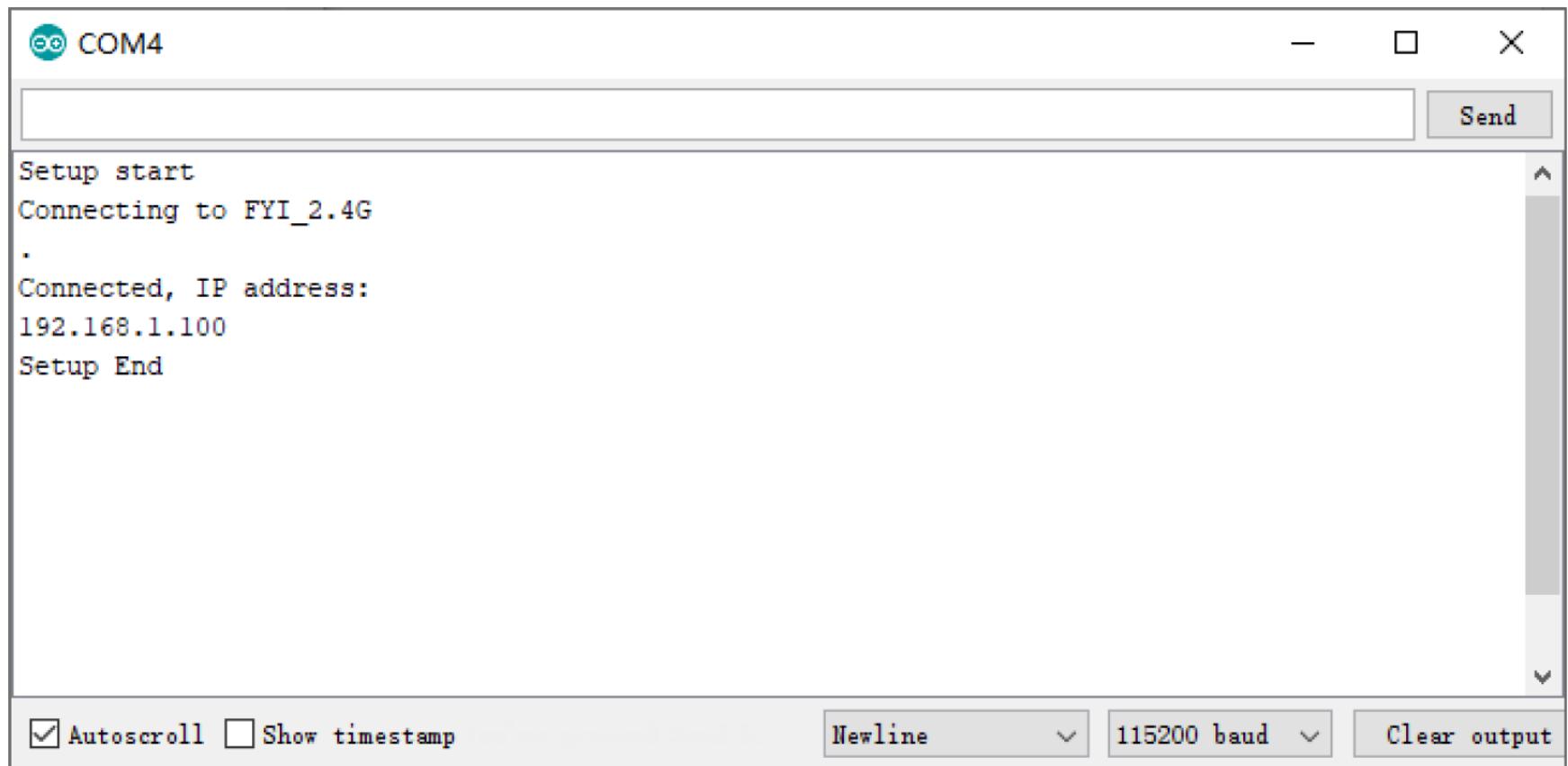
# Modalità Station - Sketch

```
1 #include <WiFi.h>
2
3 const char *ssid_Router      = "*****"; //Enter the router name
4 const char *password_Router = "*****"; //Enter the router password
5
6 void setup() {
7     Serial.begin(115200);
8     delay(2000);
9     Serial.println("Setup start");
10    WiFi.begin(ssid_Router, password_Router);
11    Serial.println(String("Connecting to ")+ssid_Router);
12    while (WiFi.status() != WL_CONNECTED) {
13        delay(500);
14        Serial.print(".");
15    }
16    Serial.println("\nConnected, IP address: ");
17    Serial.println(WiFi.localIP());
18    Serial.println("Setup End");
19 }
20
21 void loop() {
22 }
```

## Class Station

Every time when using WiFi, you need to include header file "WiFi.h".  
**begin(ssid, password,channel, bssid, connect):** ESP32 is used as Station to connect hotspot.  
**ssid:** WiFi hotspot name  
**password:** WiFi hotspot password  
**channel:** WiFi hotspot channel number; communicating through specified channel; optional parameter  
**bssid:** mac address of WiFi hotspot, optional parameter  
**connect:** boolean optional parameter, defaulting to true. If set as false, then ESP32 won't connect WiFi.  
**config(local\_ip, gateway, subnet, dns1, dns2):** set static local IP address.  
    **local\_ip:** station fixed IP address.  
    **subnet:** subnet mask  
    **dns1,dns2:** optional parameter. define IP address of domain name server  
**status:** obtain the connection status of WiFi  
**local IP():** obtain IP address in Station mode  
**disconnect():** disconnect wifi  
**setAutoConnect(boolean):** set automatic connection Every time ESP32 is power on, it will connect WiFi automatically.  
**setAutoReconnect(boolean):** set automatic reconnection Every time ESP32 disconnects WiFi, it will reconnect to WiFi automatically.

# Modalità Station - Output



The screenshot shows a terminal window titled "COM4". The window contains the following text output:

```
Setup start
Connecting to FYI_2.4G
.
Connected, IP address:
192.168.1.100
Setup End
```

At the bottom of the window, there are several control buttons and settings:

- Autoscroll  Show timestamp
- Newline
- 115200 baud
- Clear output

# Wi-Fi – Modalità Access Point

- Se configurato in modalità access point, l'ESP32 consente di creare una rete distinta da Internet



# Modalità Access Point - Sketch

```
1 #include <WiFi.h>
2
3 const char *ssid_AP      = "WiFi_Name"; //Enter the router name
4 const char *password_AP = "12345678"; //Enter the router password
5
6 IPAddress local_IP(192, 168, 1, 100); //Set the IP address of ESP32 itself
7 IPAddress gateway(192, 168, 1, 10); //Set the gateway of ESP32 itself
8 IPAddress subnet(255, 255, 255, 0); //Set the subnet mask for ESP32 itself
9
10 void setup() {
11     Serial.begin(115200);
12     delay(2000);
13     Serial.println("Setting soft-AP configuration ... ");
14     WiFi.disconnect();
15     WiFi.mode(WIFI_AP);
16     Serial.println(WiFi.softAPConfig(local_IP, gateway, subnet) ? "Ready" : "Failed!");
17     Serial.println("Setting soft-AP ... ");
18     boolean result = WiFi.softAP(ssid_AP, password_AP);
19     if(result) {
20         Serial.println("Ready");
21         Serial.println(String("Soft-AP IP address = ") + WiFi.softAPIP().toString());
22         Serial.println(String("MAC address = ") + WiFi.softAPmacAddress().c_str());
23     } else{
24         Serial.println("Failed!");
25     }
26     Serial.println("Setup End");
27 }
28
29 void loop() {
30 }
```

## Class AP

Every time when using WiFi, you need to include header file "WiFi.h".

**softAP(ssid, password, channel, ssid\_hidden, max\_connection):**

**ssid:** WiFi hotspot name

**password:** WiFi hotspot password

**channel:** Number of WiFi connection channels, range 1-13. The default is 1.

**ssid\_hidden:** Whether to hide WiFi name from scanning by other devices. The default is not hide.

**max\_connection:** Maximum number of WiFi connected devices. The range is 1-4. The default is 4.

**softAPConfig(local\_ip, gateway, subnet):** set static local IP address.

**local\_ip:** station fixed IP address.

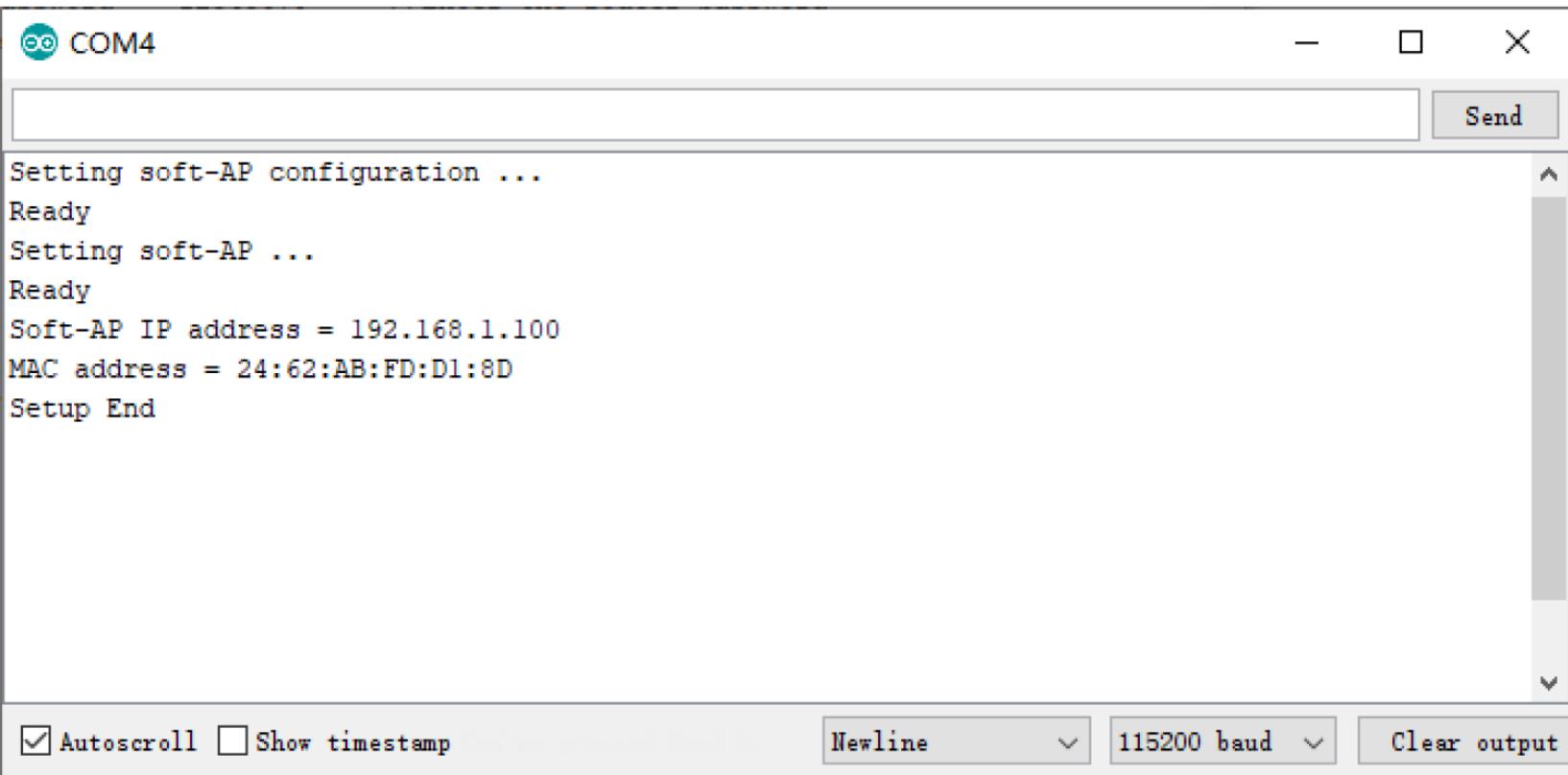
**Gateway:** gateway IP address

**subnet:** subnet mask

**softAP():** obtain IP address in AP mode

**softAPdisconnect ():** disconnect AP mode.

# Modalità Access Point - Output



```
Setting soft-AP configuration ...
Ready
Setting soft-AP ...
Ready
Soft-AP IP address = 192.168.1.100
MAC address = 24:62:AB:FD:D1:8D
Setup End
```

Autoscroll  Show timestamp

Newline  115200 baud  Clear output