Table of Contents

1. Foreword

This BLE example is intended to show how an Adroid device running an app made by MIT AI2 can exchange data (characters) with an ESP32 board by means of BLE (Bluetooth Low Energy) communication.

2. Hardware used

- An Android device: LENOVO pad TB-8505XS (8” pad) featuring Android 9
- An ESP32 board Dev Module featuring the BLE communication capability

3. Development environment

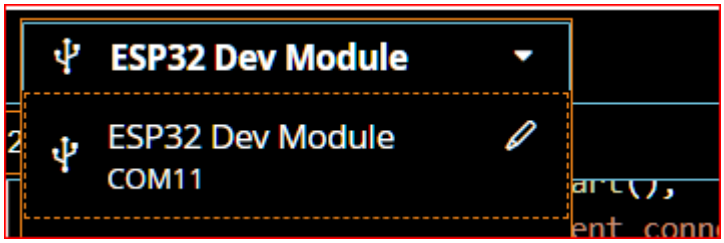
- DELL E7270 notebook c/w Windows 10 OS
- Brave browser
- Arduino 2.3.2 IDE

4. Operating instructions

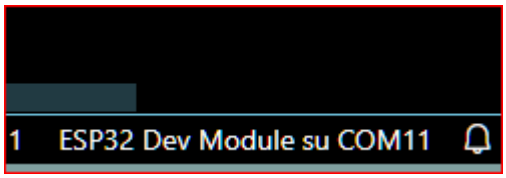
1) Download on your PC the following files:
   - The AI2 app: BLE_Test.aia
   - The ESP32 code: BLE_TestAI2.ino
2) Compile the .ino file with the Arduino IDE (be sure to have all the ESP32 relevant libraries installed).

Most probably in your IDE preferences menu tab you should have (green highlighted rows):

Be sure to have selected your relevant ESP32 board. This example is based on:

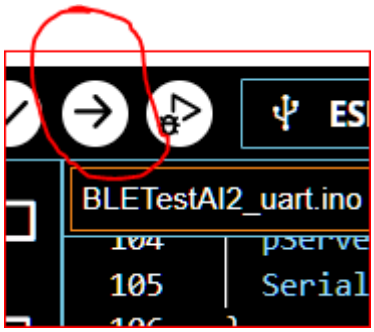


The example .ino file is intended to be connected to the Serial Monitor at a baudrate of 115200. Please be sure to have this setting in your IDE.



NOTE:(COM11 is the serial line to which my board was connected: please verify and set your true one !

Download the code on the ESP32 board:

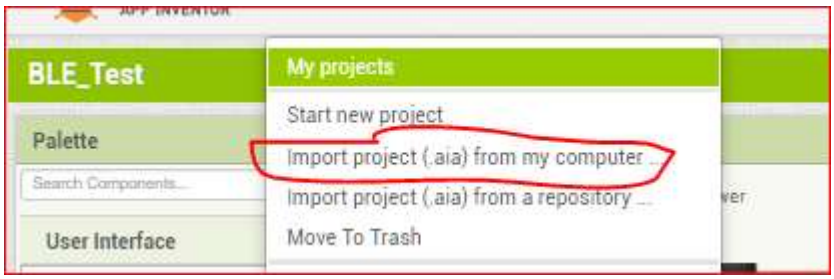


Once downloaded, press the reset button: on the serial Monitor it should appear:

```
11:43:43.715 -> ets Jul 29 2019 12:21:46
11:43:43.715 ->
11:43:43.715 -> rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
11:43:43.715 -> configsip: 0, SPIWP:0xee
11:43:43.715 -> clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
11:43:43.715 -> mode:DIO, clock div:1
11:43:43.715 -> load:0x3fff0030,len:1344
11:43:43.715 -> load:0x40078000,len:13964
11:43:43.715 -> load:0x40080400,len:3600
11:43:43.715 -> entry 0x400805f0
11:43:44.773 -> Waiting a client connection to notify...
```

The ESP32 is then waiting for a BLE connection toward a client (your AI2 app).

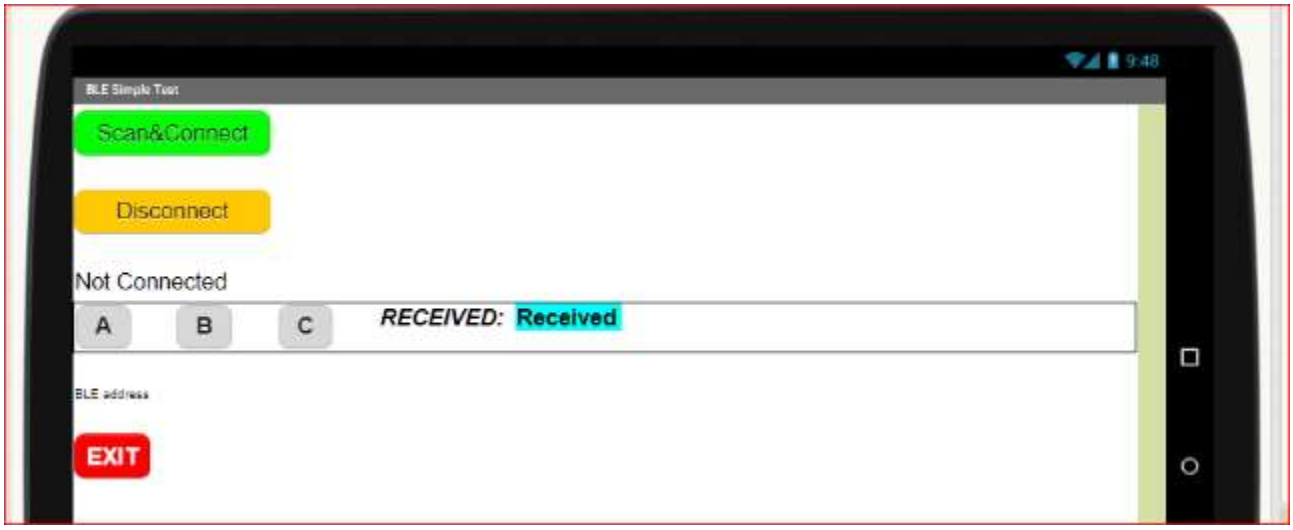
3) On your PC open AI2 and load the BLE_Test.aia.



4) Create the .apk and download it on your Adroid device (be sure that it is capable of BLE)

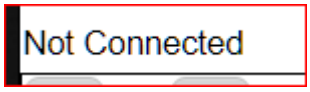
5) Install the .apk; the following icon should appear on the screen of your device (if not on home page, please search in app's general directory/screen page):

6) In your Android Settings tab: Bluetooth Devices, do a search for new devices and look for "UART Service". When found, do pair it.
7) When paired, you can tap on the icon, it should appear the following page:

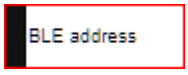


Where:

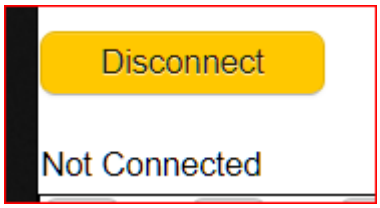
by tapping on it a search of BLE reachable devices starts. As soon as the UART Service is found, the search stops and the device is connected automatically.



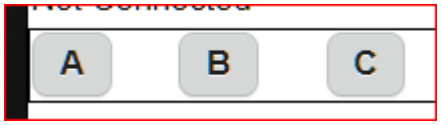
As soon as the UART Service device is connected, it shows "CONNECTED!"



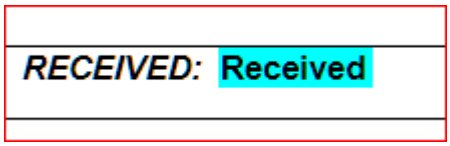
Shows the IP of the BLE device just connected



To Disconnect the ESP32 from the device, just hit on the orange button.

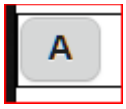


By hitting any of them the relevant character is sent to the ESP32

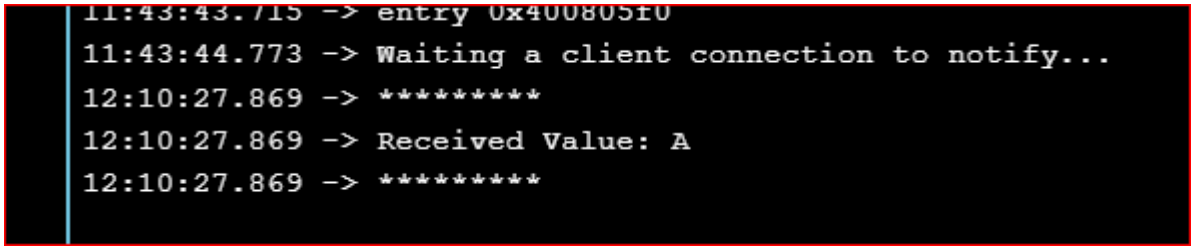


If the BLE loop = transmission from APP to the ESP and loopback of the same character from ESP to APP has worked fine, it shows the same character just sent ('A', 'B', 'C').

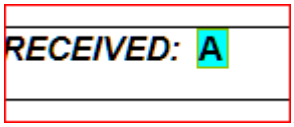
For example by hitting the



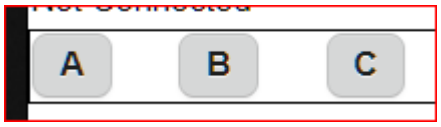
button, on the ESP32 Serial Monitor you should read:



And on the app screen you should see:



.

NOTE: the 

buttons are enabled only when the ESP32 is connected, otherwise they are "greyed" and disabled.

5. INO code explanation

Many comments are available, row-by-row, in the code, nevertheless some details are written here.

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

Required libraries

```
#define SERVICE_UUID           "6E400001-B5A3-F393-E0A9-E50E24DCCA9E"     // UART service UUID
#define CHARACTERISTIC_UUID_RX "6E400002-B5A3-F393-E0A9-E50E24DCCA9E"     // its characteristics
#define CHARACTERISTIC_UUID_TX "6E400003-B5A3-F393-E0A9-E50E24DCCA9E"
```

Service and Characteristics creation (one for reading and one for writing data)

```
std::string myStringForUnit8((char*)&rxValue[0], 1);        // trick to allow sending back one single character to the client
```

Mandatory to allow a single character to be sent back to the app.

```
for (int i = 0; i < rxValue.length()-1; i++)              // AI2 sends the linefeed to close the string: it shall be discarded
```

Since the app sends a string terminated by a linefeed character (0x0A), it shall be discarded, this explains why .length()-1

```
pTxCharacteristic->addDescriptor(new BLE2902());
```

Don't ask me why, but it seems better to have this descriptor !

Last, but not, least, don't use the Arduino delay(milliseconds) function because it stops any CPU's activity. You'd rather use your own one like:

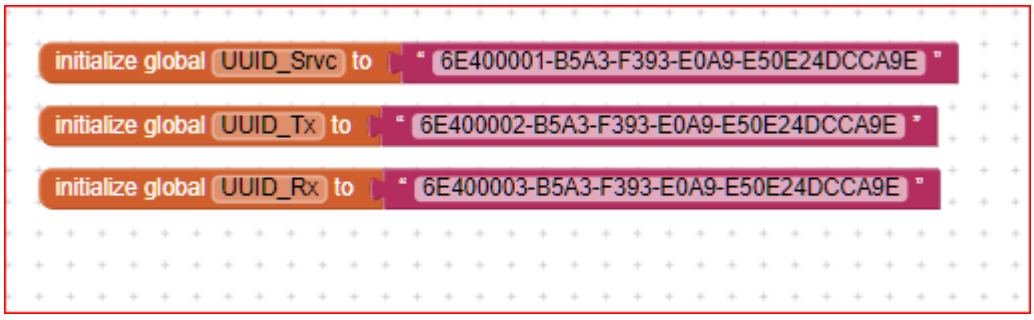
```
//---------------------------------------------------------------------
// Private delay function that does not block the CPU
// Input parameter number of millisecond to wait (unsigned long)
void Delay(unsigned long timeToWait)
{
  unsigned long now = millis();                     // reads the clock
  while ((millis()-now) < TimeToWait);              // waits until the delte milliseconds is smaller than the wanted timeout
                                                    // but it does not block the CPU, thus leaving the interrupts to work
}
```

that does not stop the CPU.

6. AI2 code explanation

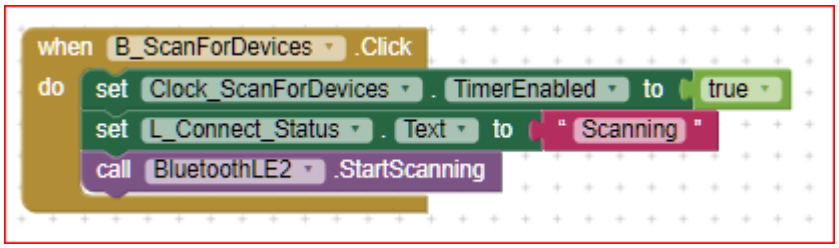
Condensed image:

The following UUID's do replicate the same that are set in the ESP32 code



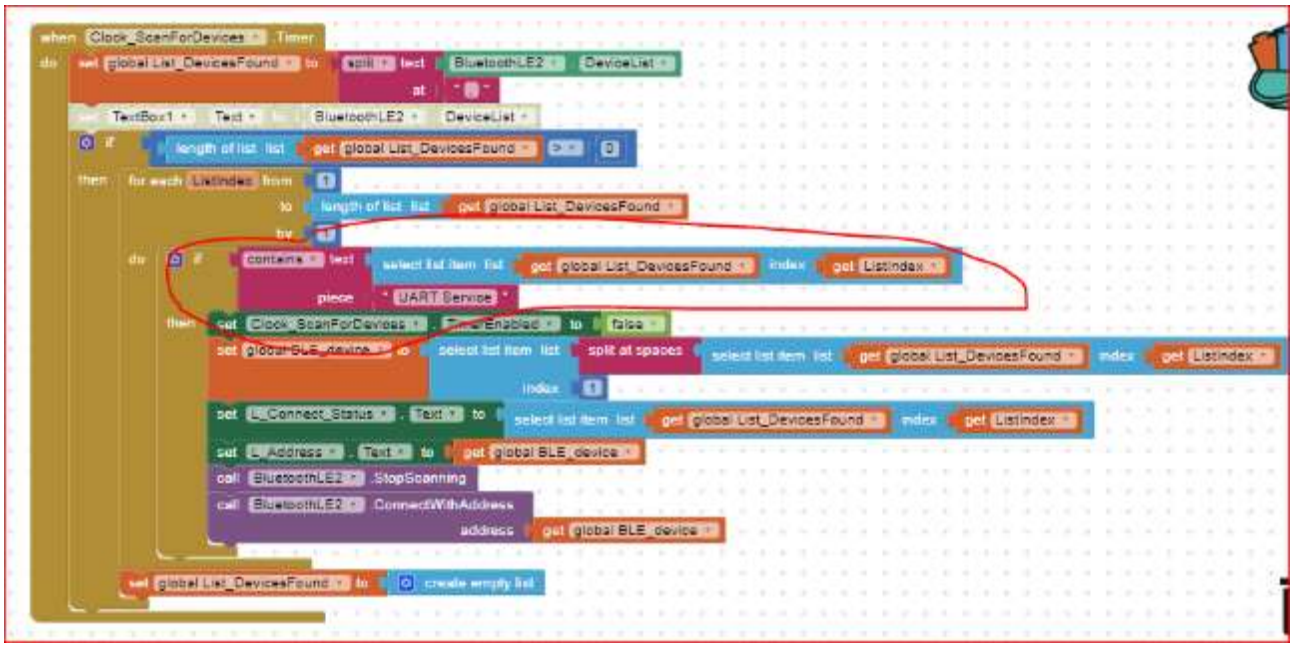
When you hit the

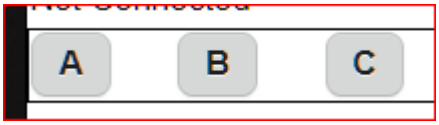
The scanning clock is enabled:



This clock has a period of 500 milliseconds: i.e. each 500 ms it performs a new scan, looking for new devices presenting themselves on the BLE radio.

The search (scan) continues until the UART Service device is found. See below:



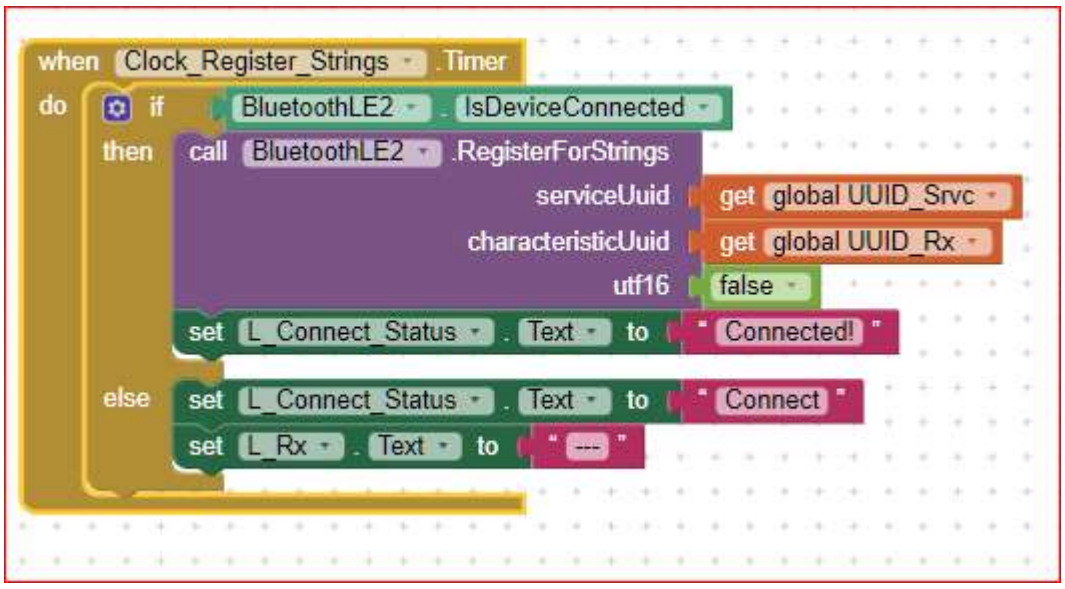
Once connected the server, the three buttons 

become active and the scan clock is deactivated.

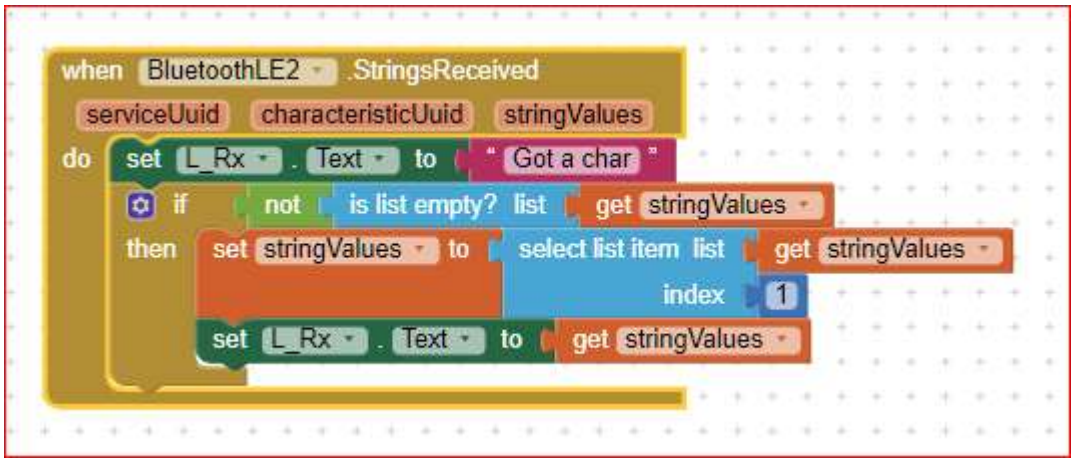
NOTE: 

it is not used, for the time being, the BLE device to be connected is fixed (UART_Service).

A second clock is used to periodically register the client to be allowed to read strings, used also to show the connection status updated in “real time” (i.e. 500 ms).



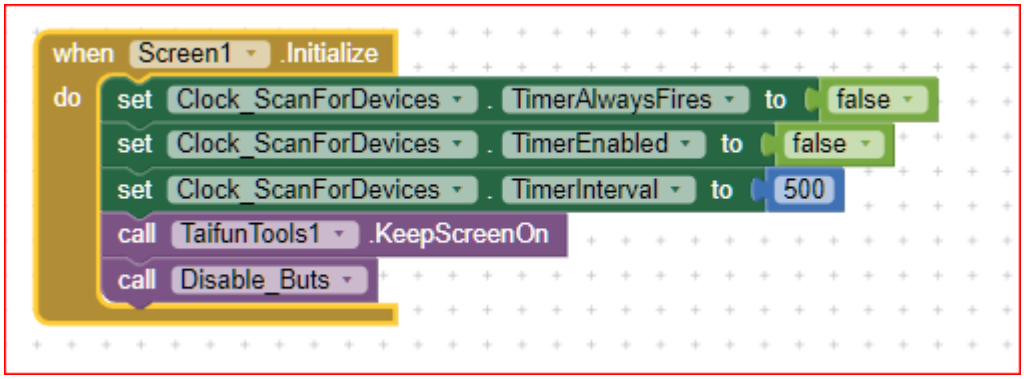
The following event is raised whenever a new characteristic is available at ESP32 side. In this case the ESP echoes the received character, therefore this event is raised “immediately” after a button (‘A’ ‘B’ ‘C’) is pressed, and its character is sent to the ESP.

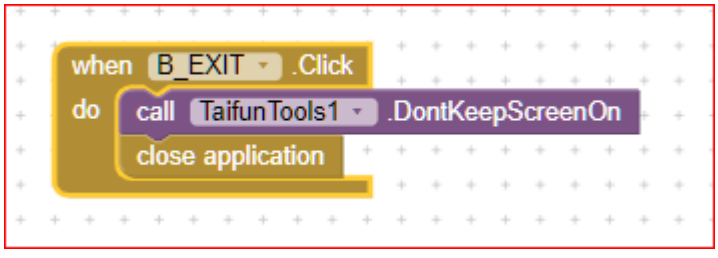


The received data is in a list form, therefore to show the received character, the first element of such list shall be extracted. In case the received data is not recognized, a simple warning “Got a char” is shown instead.

7. EXTENSIONS used

1) The app uses the “TaifunTools” extension to keep the screen on until the EXIT pushbutton is hit.





Mamy thanks to Taifun. You can find many Extensions, Snippets and ready made code on their web site:

https://puravidaapps.com/

2) BluetoothLE, from ewPatton release 20230728