ESP32 Formats and Communication Updates

Software versions listed in Table 1-10.

The latest Arduino IDE software is version 2.2.1.   www.arduino.cc/en/software

The latest *esp32* Boards Manager is version v2.0.14.   github.com/espressif/arduino-esp32/releases

Chapter 2 I2S Audio

When playing internet radio (Listing 2-6) or MP3 files (Listing 2-9) with the *M5Core2 0.1.3* and the *Audio* libraries, the sound quality is fine. However, with the *M5Core2 0.1.4* or *M5Core2 0.1.5* libraries, the audio plays at a "slow speed".

The "slow speed" problem, for the *M5Core2 0.1.7* library, is resolved by replacing the instructions:

`M5.begin(true, true, true, true)` in Listing 2-6 with

`M5.begin(true, true, true, true, kMBusModeOutput, false)` or replacing

`M5.begin()` in Listing 2-9 with

`M5.begin(true, true, true, false, kMBusModeOutput, false)`.

The changes set the *SpeakerEnable* variable to *false*, as the default `M5.begin` parameters are:

```
bool LCDEnable = true                        // LCD screen on
bool SDEnable = true                         // SD card on
bool SerialEnable = true                     // Serial display on
bool I2CEnable = false                       // I2C communication off
mbus_mode_t mode = kMBusModeOutput           // power option
bool SpeakerEnable = true                    // Speaker Enable on
```

with the power option equal to 0 or 1 when the M5Stack Core2 is powered by USB/battery or by outside input, respectively.

Chapter 5 BLE Beacons

The *MIT App Inventor BluetoothLE* extension has been updated to version 20230728, which is downloaded from mit-cml.github.io/extensions and saved as *edu.mit.appinventor.ble-20230728.aix*.

The *ESP32 BLE Arduino* library is not compatible with installation of the *ArduinoBLE* library as the compiler attempts to access both libraries. When using the *ESP32 BLE Arduino* library, the *ArduinoBLE* library must be deleted from the *User\Documents\Arduino\libraries* folder.

Formats for transmitting data with the *setValue* function are included in the *ESP32 BLE Arduino* library *src\BLECharacteristic.h* file, with the library located at *User \AppData\Local\Arduino15\packages\esp32\hardware\esp32\version\libraries\BLE.* Example

formats are `uint8_t* data, size_t size` for an integer array, `std::string value` for a string with `int& data` and `float& data` for an integer and a float, respectively.

The UUIDs of the *Characteristic User Description* and *Client Characteristic Configuration* are *0x2901* and *0x2902*, respectively (see Section 3.7 of the GATT Assigned Numbers document). A user-defined descriptor is defined with the new `BLEDescriptor((uint16_t)0x2901)` and `setValue("used-defined text")` instructions. Notifications are turned on automatically with the `setNotifications(true)` instruction. See updated Listing 5-6.

Table 1 User-defined description and automatic notification

Standard setup	Improved setup

`battService = Server`
` ->createService(battServUUID);`
`battService`
` ->addCharacteristic(&battChar);`
`battDesc = new BLE2902();`
 	
`battNotfy = (BLE2902*)battDesc;`
 	
`battChar.addDescriptor(battDesc);`
 	
`battService->start();`
	`battService = Server`
	` ->createService(battServUUID);`
	`battService`
	` ->addCharacteristic(&battChar);`
	`battDesc = new`
	` BLEDescriptor((uint16_t)0x2901);`
	`battDesc->setValue("Battery level text");`
	`battNotfy = new BLE2902();`
	`battNotfy->setNotifications(true);`
	`battChar.addDescriptor(battDesc);`
	`battChar.addDescriptor(battNotfy);`
	`battService->start();`

BLE Receive and Transmit App

The *BLE Receive App* in Chapter 9 used the Eddystone beacon framework for an ESP32 microcontroller to transmit data to an app built with MIT App Inventor. An alternative approach is to use the transmit and receive UUIDs outlined in Listings 5-6, 5-7 and 5-8. Comments specific to Listing 1 (see below) are included, rather than repeating comments from Listings 5-6, 5-7 and 5-8.

The receive and transmit UUIDs obtained for Nordic Semiconductor (infocenter.nordicsemi.com and search for *UART/Serial BLE*) are relative to the client and are reversed when relative to the server, as in the sketch. The receive and transmit characteristics of the client have *NOTIFY* and *WRITE* properties, respectively, which correspond to the transmit and receive characteristics of the server. A message received from the client is formatted as `std::string` and is then converted to a `String` with the `c_str()` function. Messages are received from two client sources, a button and a slider, and to differentiate between sources, the message data is prefixed with "*B*" or "*S*", respectively. An alternative option is two receive UUIDs with the source identified from the UUID. Service and characteristic UUIDs are generated using an online generator, such as www.uuidgenerator.net.

A message transmitted to the client is initially formatted as a `String` and is then converted to a `std::string` with the `c_str()` function. Alternatively, a string is converted to a character array with the `toCharArray(string, N)` function, where *N* is the length of the character array.

Listing 1 *BLE Receive and Transmit App*

```
#include <BLEDevice.h>
#include <BLE2902.h>
BLEServer * Server;
BLEService * Service;
BLECharacteristic * TXChar;                        // transmit characteristic
BLEDescriptor * TXDesc;                            //  and descriptor
BLECharacteristic * RXChar;                        // receive characteristic
BLE2902 * Notfy;
char ServUUID[] = "6e400001-b5a3-f393-e0a9-e50e24dcca9e";
char RXUUID[] =   "6e400002-b5a3-f393-e0a9-e50e24dcca9e";   // receive UUID
char TXUUID[] =   "6e400003-b5a3-f393-e0a9-e50e24dcca9e";   // transmit UUID
uint32_t RXval, TXval = 0, clientCon = 0;
String source, RXstr, TXstr;

class RXCharCallback: public BLECharacteristicCallbacks     // receive callback
{
  void onWrite(BLECharacteristic * RXChar)         // receive message from client
  {
    std::string str = RXChar->getValue();
    RXstr = String(str.c_str());                   // convert to String
    source = RXstr.charAt(0);                       // first character of string
    RXstr.remove(0,1);                             // remove first character
    RXval = RXstr.toInt();                         // convert string to integer
    if(source == "B") Serial.printf("Button %d \n", RXval);
    else if(source == "S") Serial.printf("Slider %d \n", RXval);
  }
};

class ServerCallback : public BLEServerCallbacks          // same as Listing 5-6

void setup()
{
  Serial.begin(115200);
  BLEDevice::init("ESP32");
  Server = BLEDevice::createServer();
  Server->setCallbacks(new ServerCallback());
  Service = Server->createService(ServUUID);
  TXChar = Service->createCharacteristic(TXUUID,   // add transmit characteristic
          BLECharacteristic::PROPERTY_NOTIFY);     //   with notify status
  TXDesc = new BLE2902();                          // add transmit
  TXChar->addDescriptor(TXDesc);                   //   descriptor to characteristic &
  Notfy = (BLE2902*)TXDesc;                        // transmit notifier to descriptor
  RXChar = Service->createCharacteristic(RXUUID,   // add receive characteristic
          BLECharacteristic::PROPERTY_WRITE);      //   with write status
  RXChar->setCallbacks(new RXCharCallback());      //   and callback
  Service->start();
  Server->getAdvertising()->start();
}

void loop()
```

```
{
  if(clientCon == 1)                                      // client connected to server
  {
    TXval++;
    if(TXval % 2 == 0) TXstr ="even " + String(TXval);      // create message
    else TXstr = "odd  " + String(TXval);
//  char msg[24];                                         // char array to store message
//  TXstr.toCharArray(msg, 24);                           // convert to character array
//  TXChar->setValue(msg);                                // set characteristic to message
    TXChar->setValue(TXstr.c_str());                      // convert String to std::string
    TXChar->notify();                                     // transmit message to client
    Serial.printf("TXval %d \n", TXval);
    delay(2000);
  }
}
```
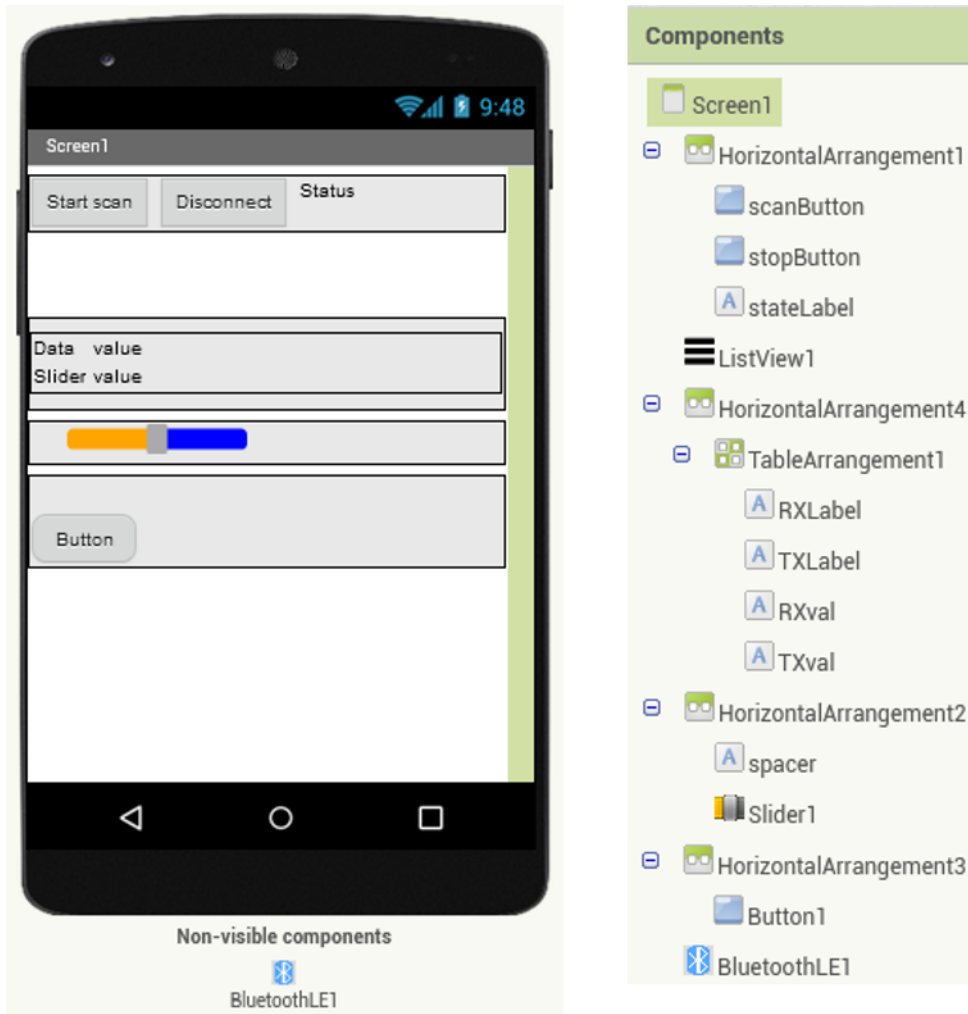
The *BLE Receive and Transmit App* (see Figure 2a) is built in *MIT App Inventor* with button to start BLE scanning and to disconnected from a BLE device, which is a ESP32 microcontroller. The *Data* text is received by the client from the server with the slider position and button state transmitted by the client to the server.

When the *nRF Connect* app is connected to the server, the server device name is displayed under *Generic Access* and with the microcontroller MAC address at the top of the screen (see Figure 2b). The client receives and transmits on the *nRF Connect* app *TX* and *RX* characteristics, which are the UUIDs defined for the server in the sketch.
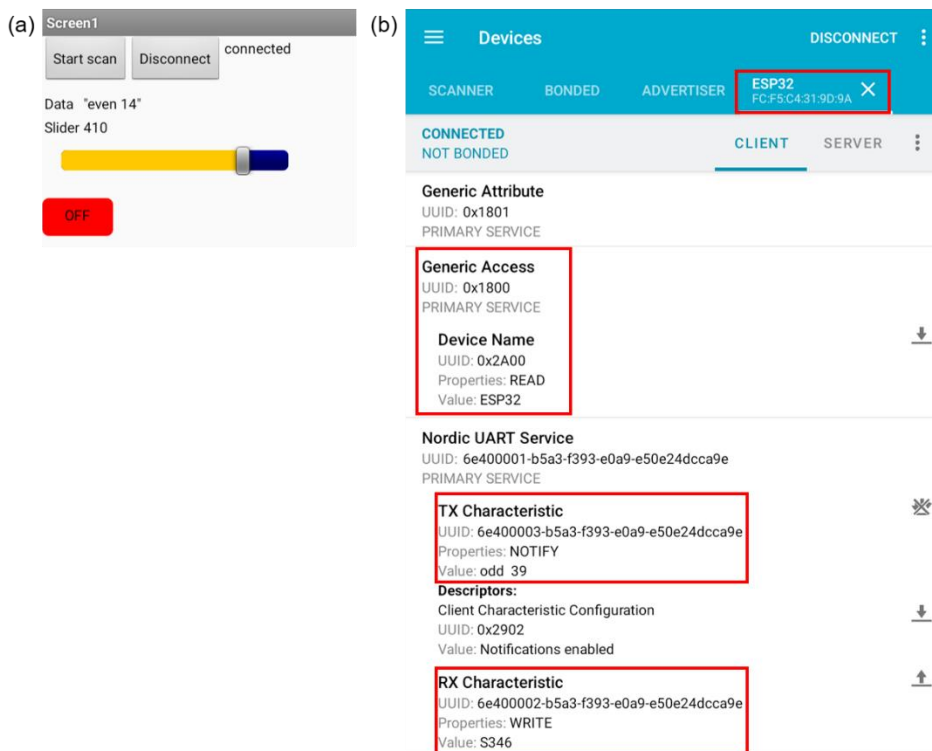

Figure 2 *Screenshots of BLE Receive and Transmit App and nRF Connect App*

4

Components of the app are shown in Figure 3. The displayed *Data* and *Slider* titles and values are contained in a table, *TableArrangement1*, with two rows and columns. The *BluetoothLE* extension is located in the *Extension Palette* of *MIT App Inventor* and is dragged onto the *Viewer* screen.


Figure 3 *Components of BLE Receive and Transmit App*

When the *scanButton* is clicked, Bluetooth BLE scanning is initiated and each detected BLE device is added to the *ListView* (see Figure 4). Note that naming of the *TX* and *RX* UUIDs are relative to the client.
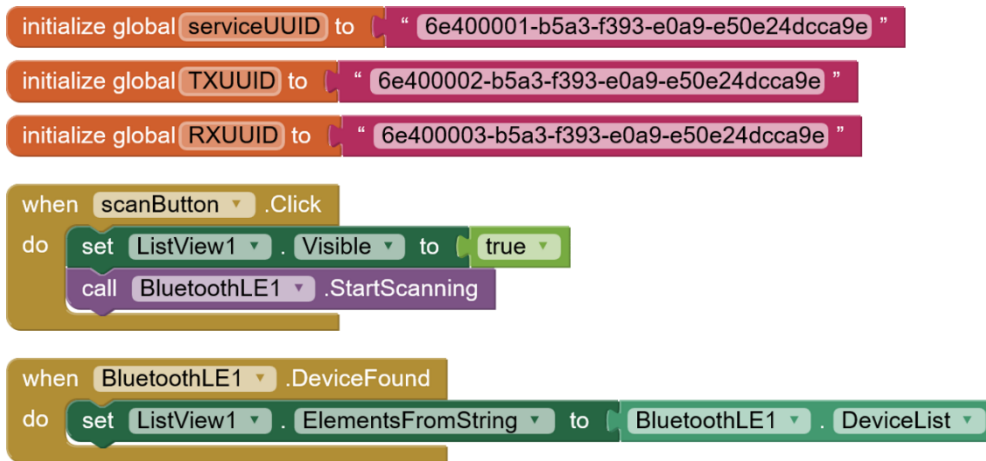
Figure 4 *App UUIDs and BLE scanning*

When a BLE device is selected on the *ListView*, Bluetooth BLE scanning is stopped, a connection with the selected BLE device is established and the client receive UUID is defined for the client to receive strings from the server (see Figure 5). When the *stopButton* is clicked, the Bluetooth BLE connection is disconnected.
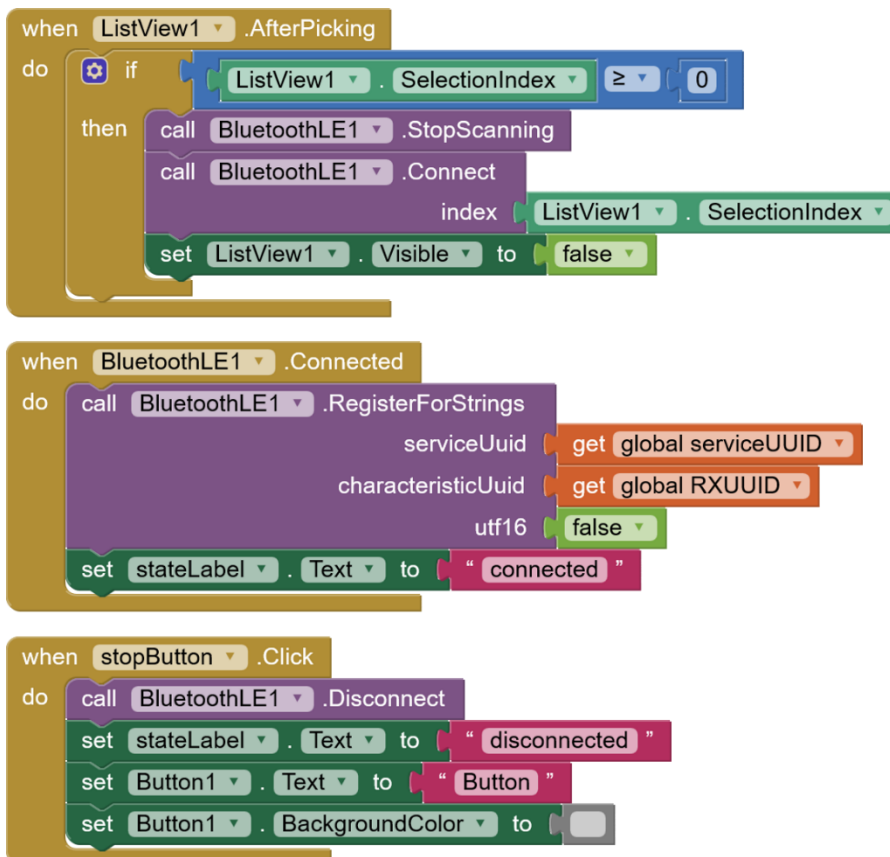

Figure 5 *App Bluetooth BLE connection and disconnect*

When the client receives a string from the server, the "*[*" and "*]*" characters are removed and the string is displayed on the app (see Figure 6).
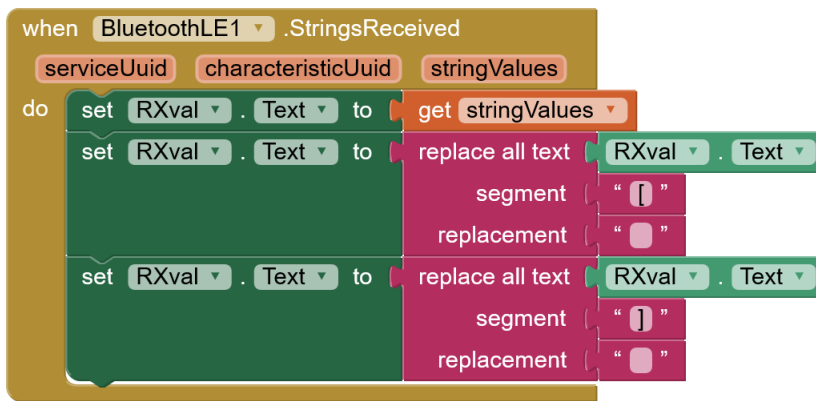
Figure 6 *Client receiving from server*

When the slider position is updated, the client transmits a string, prefixed with the character "*S*", to the server with the transmit UUID (see Figure 7).
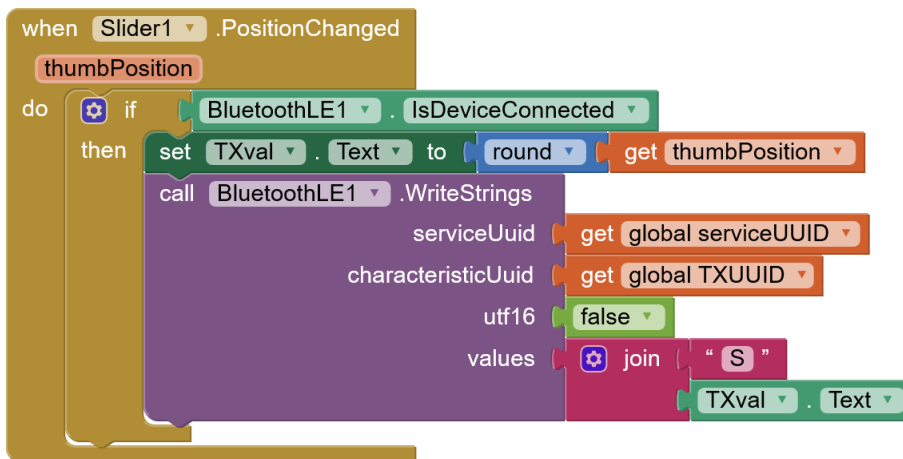
Figure 7 *Client transmitting to server (1)*

Similarly, when the button is pressed, the client transmits a string, prefixed with the character "*B*" (see Figure 8) with the button colour and label also updated.
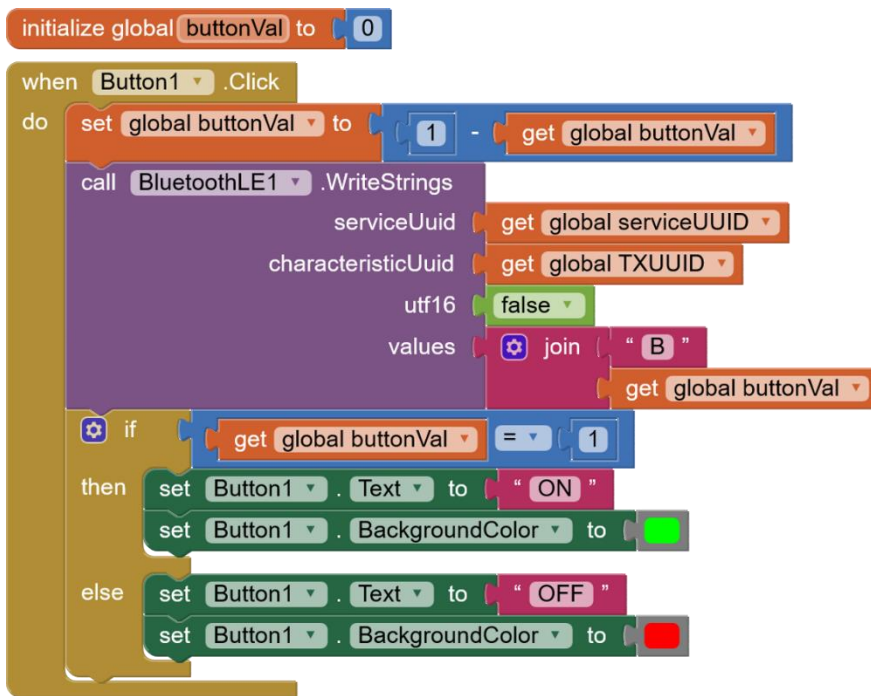
Figure 8 *Client transmitting to server (2)*

Chapter 9 MQTT

Cayenne (developers.mydevices.com) announced that Cayenne MQTT broker will be discontinued in September 2023 (community.mydevices.com/t/end-of-life-announcement-for-cayenne/18926). Chapter 9 *MQTT* has been updated with the Blynk, Arduino IoT Cloud and Adafruit IO MQTT broker services. The updated Chapter and accompanying sketches are included in the **Chapter 9_NEW_V2** folder on GitHub (github.com/Apress/ESP32-Formats-and-Communication/tree/main/Chapter9_NEW_V2).

Corrections

Despite repeatedly checking the page proofs, some typos may have persisted.

If you find typos, then please either email *esp32ndc@gmail.com* or use *GitHub Issues* to log the typo or other issues.

Page 88 last – 5 line	Change expression to `(A > B) ? C : D` by inserting a `?`
Page 243 last line	Change `LEDpin = 27` to `LEDpin = 5`
Page 367 Line 10	Change `SCTpin = 37` to `SCTpin = 36` (as in Figure 9-7)
Page 395 Line 9	Change comment to "Clear LCD screen"

Neil Cameron

December 2023