ESP32 Formats and Communication

Updates

Software versions listed in Table 1-10.

```
The latest Arduino IDE software is version 2.3.2 (Feb 2024) www.arduino.cc/en/software The latest esp32 Boards Manager are versions v3.0.4 (July 2024) and v2.0.17 (May 2023).
```

The impact on the BLE, Hall sensor, I2S and LEDC APIs by migrating the Arduino ESP32 core from versions 2.0.N, based on ESP-IDF 4.4, to version 3.0, based on ESP-IDF 5.1, are outlined in the **Arduino ESP32 Core V3** folder on GitHub (github.com/Apress/ESP32-Formats-and-Communication/tree/main/ Arduino_ESP32_Core_V3).

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, 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:

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

Chapter 4 TTGO T-Watch V2

Pages 213-214 Espressif Flash Download Tool V3.9.7, dated 7 June 2024, is available.

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

BLE Receive and Transmit App

The *BLE Receive App* in Chapter 5 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;
                                                         // transmit characteristic
BLECharacteristic * TXChar;
                                                         // and descriptor
BLEDescriptor * TXDesc;
                                                         // receive characteristic
BLECharacteristic * RXChar;
BLE2902 * Notfy;
char ServUUID[] = "6e400001-b5a3-f393-e0a9-e50e24dcca9e";
                                                                      // receive UUID
                   "6e400002-b5a3-f393-e0a9-e50e24dcca9e";
                                                                      // transmit UUID
                  "6e400003-b5a3-f393-e0a9-e50e24dcca9e";
uint32 t RXval, TXval = 0, clientCon = 0;
String source, RXstr, TXstr;
                                                                // receive callback
class RXCharCallback: public BLECharacteristicCallbacks
  void onWrite(BLECharacteristic * RXChar)
                                                         // receive message from client
    std::string str = RXChar->getValue();
                                                         // convert to String
    RXstr = String(str.c str());
                                                         // first character of string
    source = RXstr.charAt(0);
                                                         // remove first character
    RXstr.remove(0,1);
                                                         // convert string to integer
    RXval = RXstr.toInt();
    if(source == "B") Serial.printf("Button %d \n", RXval);
    else if(source == "S") Serial.printf("Slider %d \n", RXval);
};
                                                         // same as Listing 5-6
class ServerCallback : public BLEServerCallbacks
void setup()
  Serial.begin(115200);
  BLEDevice::init("ESP32");
  Server = BLEDevice::createServer();
  Server->setCallbacks(new ServerCallback());
  Service = Server->createService(ServUUID);
                                                         // add transmit characteristic
  TXChar = Service->createCharacteristic(TXUUID,
                                                         // with notify status
           BLECharacteristic::PROPERTY NOTIFY);
                                                         // add transmit
  TXDesc = new BLE2902();
  TXChar->addDescriptor(TXDesc);
                                                         // descriptor to characteristic &
```

```
// transmit notifier to descriptor
  Notfy = (BLE2902*) TXDesc;
                                                           // add receive characteristic
  RXChar = Service->createCharacteristic(RXUUID,
                                                           // with write status
           BLECharacteristic::PROPERTY WRITE);
                                                           // and callback
  RXChar->setCallbacks(new RXCharCallback());
  Service->start();
  Server->getAdvertising()->start();
}
void loop()
                                                           // client connected to server
  if(clientCon == 1)
    TXval++;
    if(TXval % 2 == 0) TXstr ="even " + String(TXval);
                                                                  // create message
    else TXstr = "odd " + String(TXval);
                                                            // char array to store message
// char msq[24];
                                                           // convert to character array
// TXstr.toCharArray(msg, 24);
                                                           // set characteristic to message
// TXChar->setValue(msg);
                                                           // convert String to std::string
    TXChar->setValue(TXstr.c str());
                                                           // transmit message to client
    TXChar->notify();
    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.

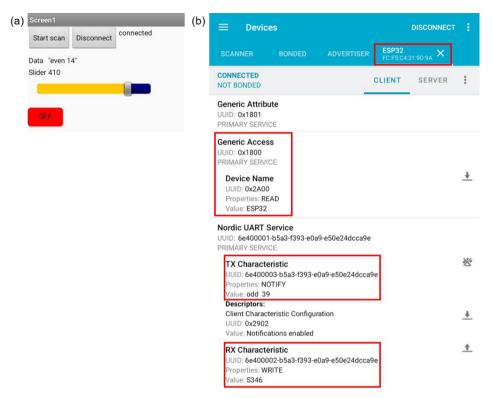


Figure 2 Screenshots of BLE Receive and Transmit App and nRF Connect App

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.

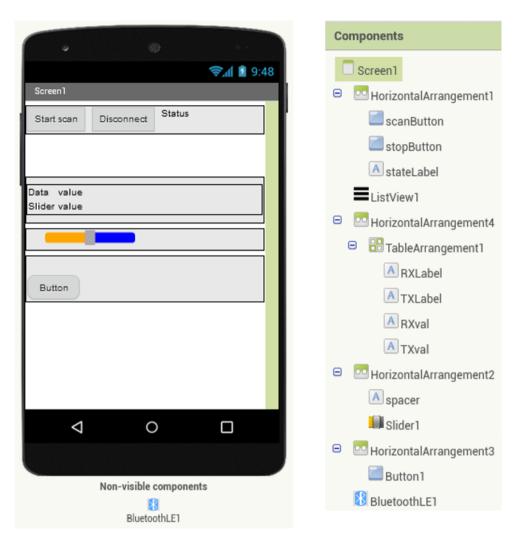


Figure 3 Components of BLE Receive and Transmit App

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.

```
initialize global serviceUUID to " 6e400001-b5a3-f393-e0a9-e50e24dcca9e"
initialize global TXUUID to " 6e400002-b5a3-f393-e0a9-e50e24dcca9e"
initialize global RXUUID to " 6e400003-b5a3-f393-e0a9-e50e24dcca9e"
when scanButton v .Click
do set ListView1 v . Visible v to true v
call BluetoothLE1 v .StartScanning

when BluetoothLE1 v .DeviceFound
do set ListView1 v . ElementsFromString v to BluetoothLE1 v . DeviceList v
```

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.

```
when ListView1
                 .AfterPicking
    🗯 if
                ListView1 ▼
                             SelectionIndex •
              BluetoothLE1 .StopScanning
    then
           call BluetoothLE1 .Connect
                                         ListView1 •
                                                     SelectionIndex *
                                 index
          set ListView1 .
                           Visible to false
when BluetoothLE1 .Connected
    call BluetoothLE1 .RegisterForStrings
                                           get global serviceUUID
                              serviceUuid
                         characteristicUuid
                                           get global RXUUID 🔻
                                    utf16
                                           false 🔻
    set stateLabel •
                                    connected
                     Text ▼
when stopButton .Click
    call BluetoothLE1 .Disconnect
    set stateLabel . Text to
                                    disconnected
                   Text ▼ to
                                " Button
        Button1 ▼
    set Button1
                   BackgroundColor v to
```

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).

```
when BluetoothLE1 .StringsReceived
serviceUuid characteristicUuid
                                stringValues
    set RXval . Text to
                               get stringValues •
    set RXval . Text to
                               replace all text
                                              RXval •
                                                        Text ▼
                                               0 "
                                    segment
                                replacement
    set RXval v . Text v to
                               replace all text
                                              RXval Text
                                    segment
                                                N "
                                replacement
```

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).

```
when Slider1 .PositionChanged
 thumbPosition
    🗯 if
              BluetoothLE1 . IsDeviceConnected .
              TXval . Text to round
                                                get thumbPosition
           call BluetoothLE1 .WriteStrings
                               serviceUuid
                                             get global serviceUUID *
                          characteristicUuid
                                             get global TXUUID 🔻
                                            false 🔻
                                    values
                                             ioin
                                                        S "
                                                      TXval •
                                                                Text ▼
```

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.

```
initialize global buttonVal to 0
when Button1 .Click
     set global buttonVal ▼ to
                                          get global buttonVal
     call BluetoothLE1 .WriteStrings
                          serviceUuid
                                        get global serviceUUID *
                    characteristicUuid
                                        get global TXUUID
                                utf16
                                        false *
                                        ioin 🔯
                                                   " B "
                               values
                                                   get global buttonVal
     🔯 if
                                              1
                 get global buttonVal *
                                        set Button1 .
                           Text ▼ to
                                         " ON "
               Button1 *
                           BackgroundColor *
           set Button1 v . Text v to
                                         " OFF
           set Button1 *
                           BackgroundColor *
```

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).

Chapter 10 Managing Images

The plugin to load files into SPIFFS for Arduino IDE version 1 is available from:

github.com/me-no-dev/arduino-esp32fs-plugin/releases/tag/1.1 esp32fs last updated 6 Mar 2023

The *mklittlefs* folder containing the *mklittlefs* application (Sept 2023) is downloaded from github.com/earlephilhower/mklittlefs/releases as referenced by github.com/lorol/LITTLEFS.

The plugin to load files into SPIFFS for Arduino IDE version 2 is downloaded from:
github.com/earlephilhower/arduino-littlefs-upload

Since publishing *ESP32 Formats and Communication*, a plugin for Arduino IDE version 2 is now available. Instructions for installing and running the plugin are available at github.com/earlephilhower/arduino-littlefs-upload. The *arduino-littlefs-upload-1.1.8.vsix* file is downloaded and the *.vsiz* file is moved to the folder *User\.arduinoIDE\plugins*, which may have to be created. Note that the *plugin-storage* folder already exists. If the Arduino IDE is open, then the Arduino IDE must be closed and restarted after the *arduino-littlefs-upload-1.1.8.vsix* file is moved to the *plugin* folder. In the Arduino IDE, the command palette is opened by pressing *CRTL Shift P* to list the available commands, which will now include the *Upload LittleFS to Pico/ESP8266/ESP32* command.

A text file or image file is stored in SPIFFS by locating the file in the *data* folder within the sketch folder. In the Arduino IDE, select the *Sketch* menu, click *Show Sketch Folder*, create a folder named *data*, and then move the file into the *data* folder. A file *filename.txt* or *image.jpg* is referenced in the sketch as */filename.txt* or */image.jpg*. Prior to compiling and uploading the sketch in the Arduino IDE, ensure that both a *Board* and a *Port* are selected and that the Serial Monitor is closed.

Open the command palette by pressing CRTL Shift P, scroll through the list of commands to the Upload LittleFS to Pico/ESP8266/ESP32 command and click on the command. After the Completed upload message is displayed in the LittleFS upload panel, compile and upload the sketch.

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 116 Table 2-6 Connect *OLED VCC* to 3.3V not to *VIN*

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"

Page 409 Lines 9-10 Change to $User \land AppData \land Local \land Arduino 15 \land packages \land esp32 \land tools$

Neil Cameron

August 2024