ESP32 BLE + Android + Arduino IDE = AWESOME

by arduinofanboy

Introduction

As you might know, the ESP32 is an incredibly feature-packed module that has not only WiFi but also Bluetooth Low Energy (BLE), touch sensors, tons of ADC pins, DAC pins, audio support, SD card support... did I mention enough to impress you?

In this tutorial we will be working with the Bluetooth Low Energy feature of this in Arduino IDE and create a custom Android app using Thunkable, a free and visual app building tool. What actually sparked me to do this tutorial was this YouTube video by Andreas Spiess in which he experiments with the BLE feature a little. What's really sweet is that some awesome dude has already done all the hard coding behind the BLE libraries for Arduino IDE (hats off to Neil Kolban !) and his contributions were recently added as part of the official ESP32 Arduino release.

Note: For using the ESP32's traditional Bluetooth as a serial device, please see the example Arduino sketch that is now included in the ESP32 Arduino package.

Goals for this Tutorial

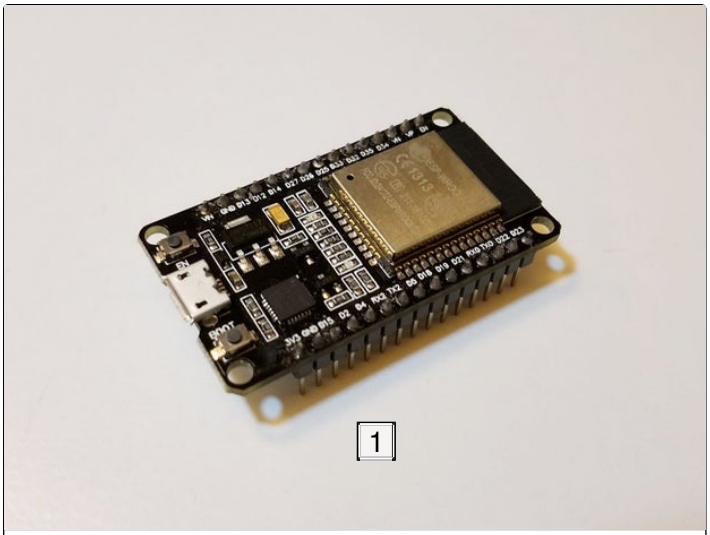
First of all what are we making here? In this tutorial we'll be building an Android app that connects to the ESP32 via Bluetooth to establish two-way communication. We'll be able to control an LED on/off remotely and we'll also be able to see some arbitrary values that are sent from the ESP32 to the Android app. These values could be things like sensor readings, door states for a home security system, etc. The cool part about all this is that you don't need to have any crazy skills to do this! So with that, let's get started!

1. Photo credits: phone picture from Samsung.com, Bluetooth logo from bluetooth.com, Arduino logo from arduino.cc

Step 1: Gather Parts

Fortunately this list is pretty simple!

- Android device with Bluetooth 4.0 or higher (most smartphones)
- ESP32 development board (note that there are many versions that would also work just fine)
- Micro USB to program the ESP32 dev board
- Optional: sensors, LED's, etc. to spice up the project!
- Depending on your setup and project you may want a breadboard and some jumper wires



1. Almost any ESP32 development board should work, but I used this one.

1. I've specifically designed the Reflowduino32 add-on to plug into the "DOIT" ESP32 dev board but I suppose other boards with the same pinouts and pin spacing would work too!

Step 2: Arduino IDE Setup

This is pretty obvious, but the first thing you need to do is install Arduino IDE. Enough said.

ESP32 Package Installation

The next thing you need to do is install the ESP32 package for Arduino IDE by following the Windows instructions or the Mac instructions. I will say that for Windows when the instructions tell you to open "Git GUI" you have to download and set up "Git" from the link provided and if you have a hard time finding an application called "Git GUI" then all you need to do is search "Git GUI" in the start menu and you will see a little command prompt-ish looking icon (see attached screenshot above). It's also located in "C:\Program Files\Git\cmd\git-gui.exe" by default. From there, follow the instructions and you should be good to go!

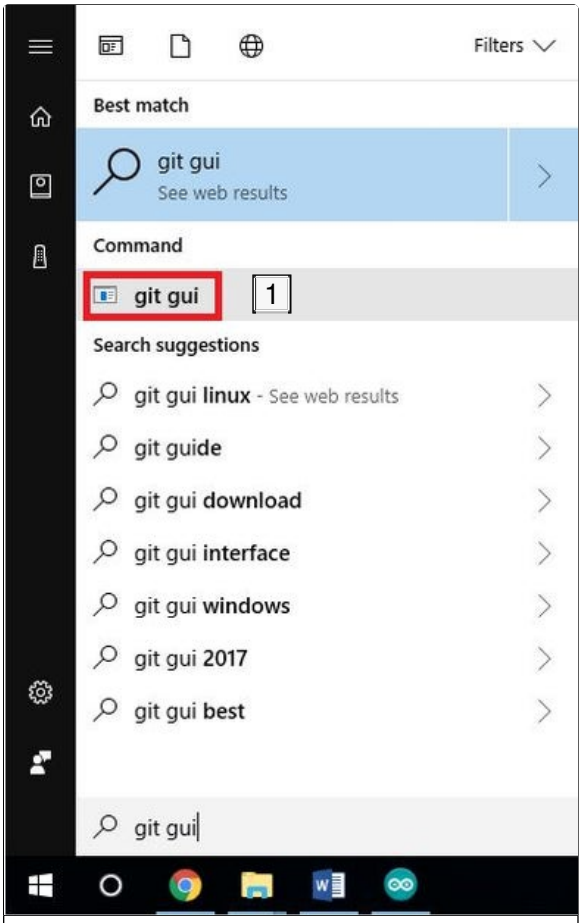
Note: If you already have the ESP32 package installed in Arduino IDE but you didn't get it after BLE support was added to the package, I'd recommend going to "Documents/hardware/espressif" and deleting the "esp32" folder and re-doing the setup instructions above. I'm saying this because I ran into an issue where even after following the update procedure at the bottom of the instructions the BLE examples weren't appearing in the "Examples" under "Examples for ESP32 Dev Module" in Arduino IDE.

In order to check if the ESP32 installation went well, go to File / Examples / ESP32 BLE Arduino and you should see several example sketches, like "BLE_scan", "BLE_notify", etc. This means everything is set up properly in Arduino IDE!

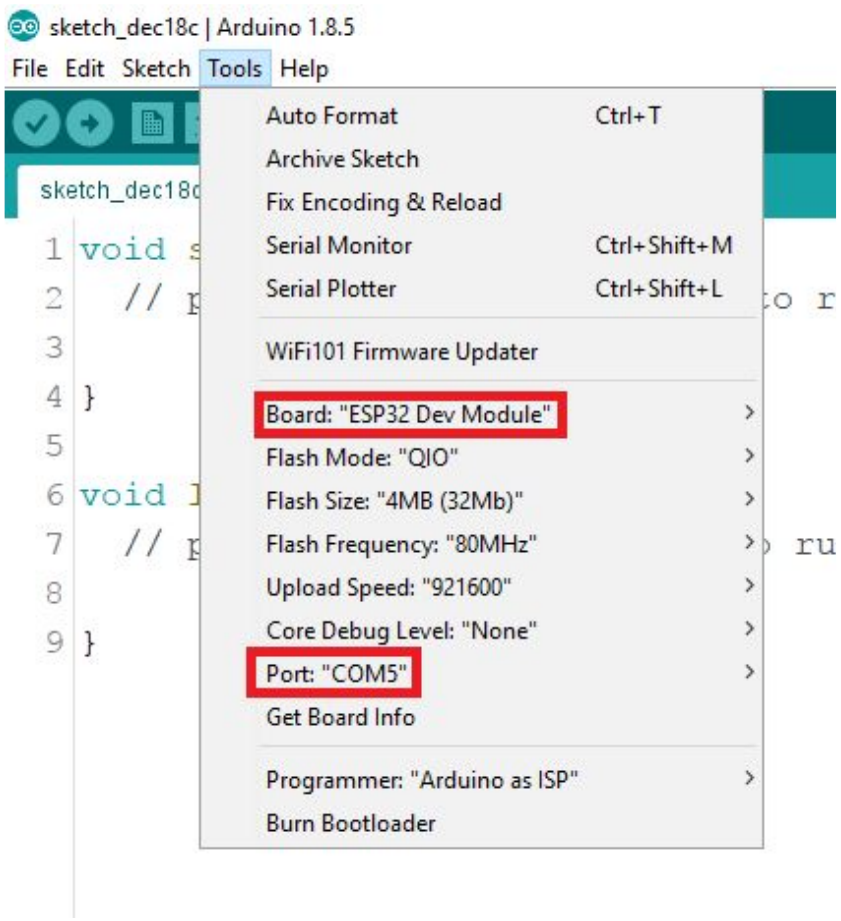
ESP32 BLE Example Sketch

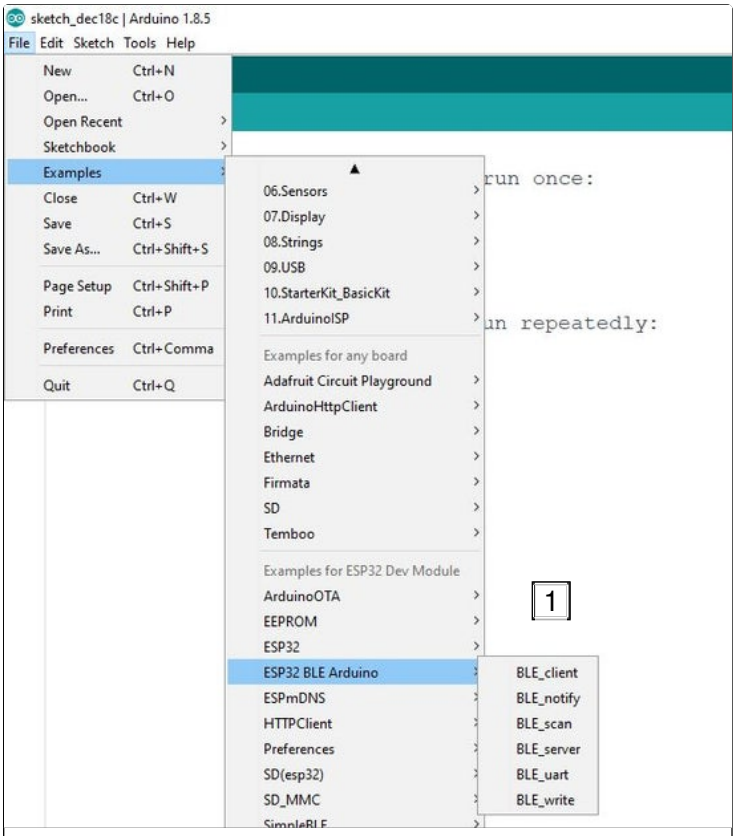
In Arduino IDE the first thing you should do is go to Tools / Board and select the appropriate board. It doesn't really matter which one you choose, but some things might be board-specific. I chose "ESP32 Dev Module" for my board. Also go ahead and choose the correct COM port after connecting the board to your computer via the USB cable.

Now that Arduino IDE is all set up, open the code I've provided for this tutorial (attached below), which is a slightly edited version of the "BLE_uart" example sketch. Since I've kept the file extension as ".ino" Arduino IDE will ask you if you want to create a folder around it with the same name, so click "yes" to open it.



1. In Windows just search "Git GUI" in the Start menu. It's easy to miss!





1. You should see these examples if the ESP32 package was set up right!

https://www.instructabl…

Download

Step 3: App Setup

UPDATE: Thunkable recently transitioned from Thunkable Classic to a completely new platform called ThunkableX which allows users to create apps for both Android and iOS from the same platform but requires a paid membership for making private apps. This tutorial was written using Thunkable Classic and unfortunately isn't accessible to ThunkableX users and there's no way to import from Classic into ThunkableX. I have uploaded images of the screen and blocks from Thunkable Classic if anyone wants to try recreating the app in ThunkableX, and I've already started on recreating it (you can find it here**).**

Setting up Thunkable

For the Android app we'll be using Thunkable, a fantastic visual app-building tool for Android and iOS. Here we'll just be making an Android app since their iOS support is still in the early stage and doesn't have Bluetooth stuff yet. (Not to mention Apple holds a tight grip on app distribution, etc.)

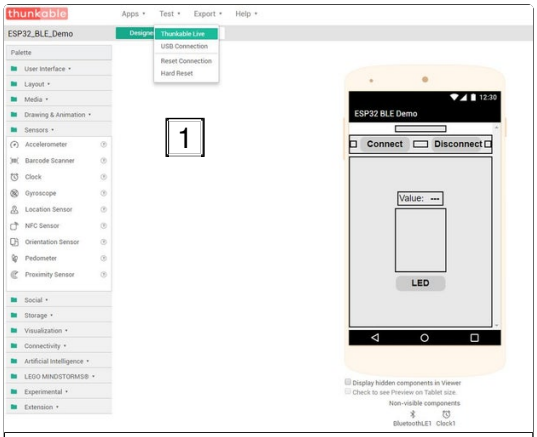
Go to the Thunkable site and set up an account or log in with a Google account. If you're new to Thunkable you won't see any existing projects, but that's about to change! Click "Apps" at the top left and click "Upload app project (.aia) from my computer". The "native" file type for Thunkable is ".aia" files and these files will allow you to view and edit code blocks within Thunkable. First download the attached file called "ESP32_BLE_Demo.aia" and then load this file in Thunkable. This should now bring you to the app's home screen where you can edit the user interface. To view and edit the code blocks, click "Blocks" sort of at the top left, next to "Designer". This tutorial isn't meant to teach you all the ins and outs of Thunkable, but I definitely recommend you explore it yourself and have fun with it!

Thunkable Companion App

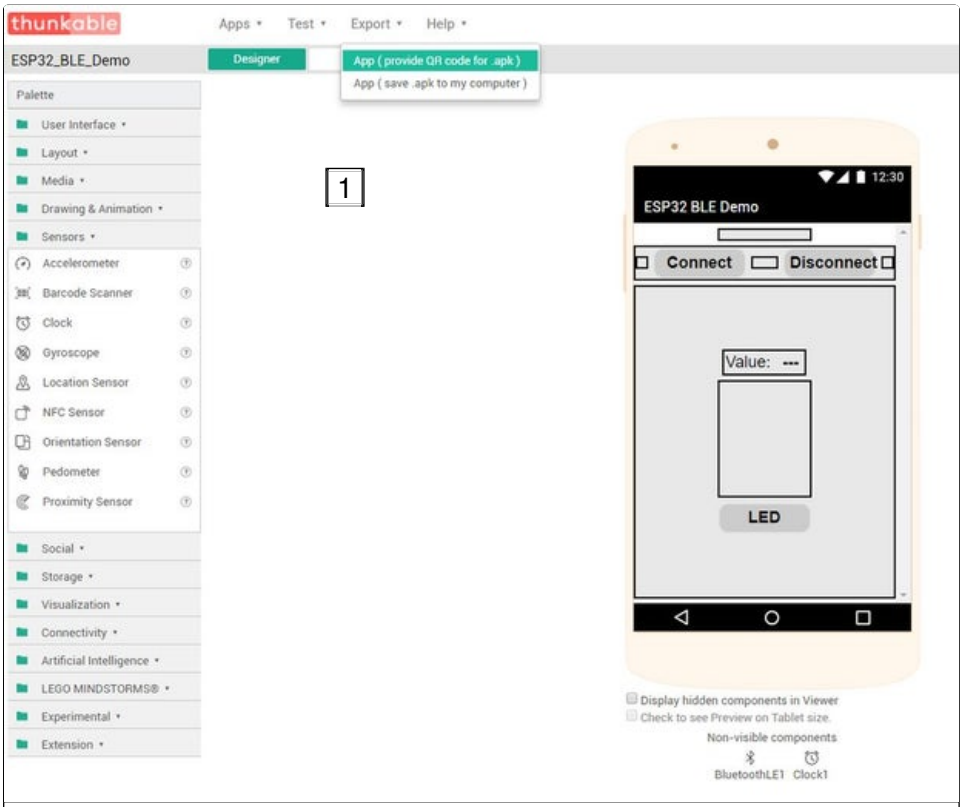
You can also download the Thunkable companion app on your mobile device and do live testing with it, which is really darn cool because you can test the app without having to first compile and download it every time! Simply install it on your mobile device and under the "Test" tab at the top click "Thunkable Live" and it will bring up a QR code on the screen. Open the Thunkable app on your mobile device and scan the QR code to live test!

Now to actually get the app on your phone all you have to do is click "Export" and "App (provide QR code for .apk)" and scan the QR code with your phone using the Thunkable app. You can then install the app and open it! Alternatively, you can download the .apk file I've attached above and email it to yourself to get it on your phone.

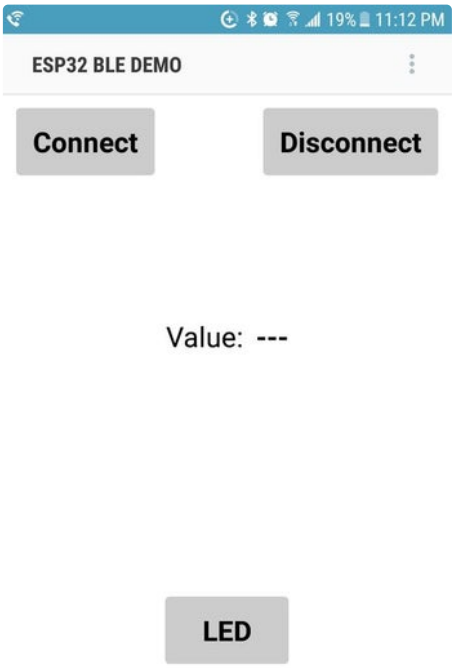
When you first open the app it will ask you to turn on Bluetooth if you haven't already, and click "Yes". When the app is connected to your ESP32 it will print out arbitrary values that are sent to it from the ESP32 and the "LED" button allows you to toggle the LED on or off by sending "A" or "B" to the ESP32. But for now let's not jump the gun just yet!

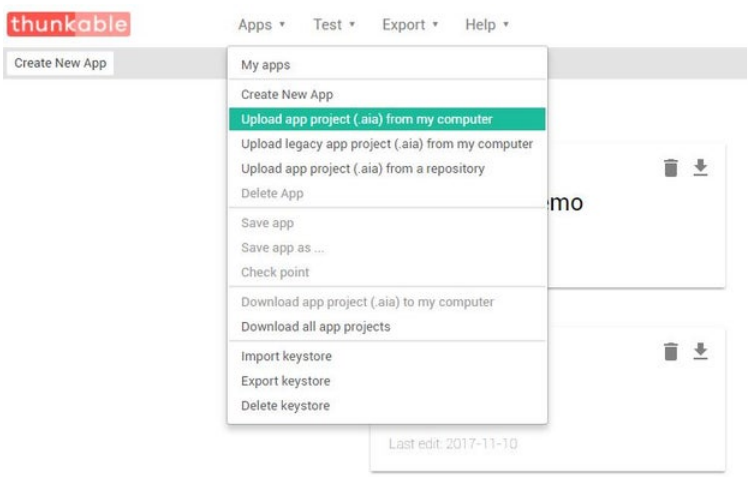


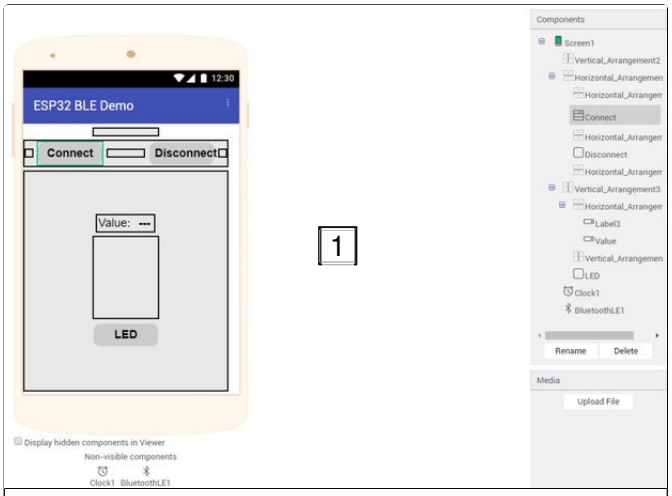
1. Download the Thunkable app on your mobile device and use the "Thunkable Live" feature to live test your app! Cool stuff huh?



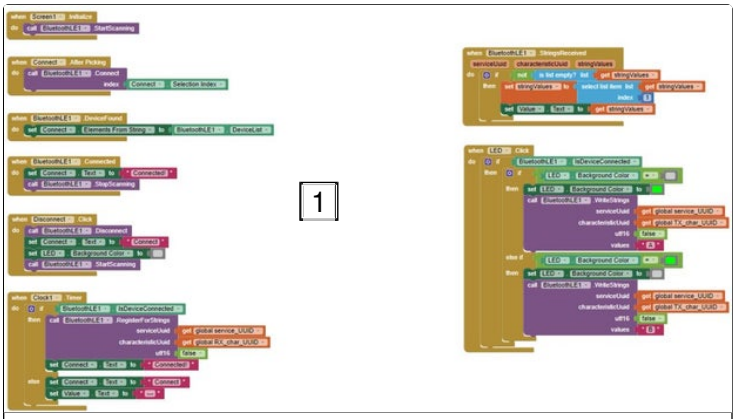
1. Export the app and it will show a QR code for you to scan with your Thunkable app on the mobile device.







1. Thunkable Classic is going to be obsolete and has moved to ThunkableX, a completely different platform (but with similar GUI)



1. These are the blocks from Thunkable Classic

https://www.instructabl... Download

https://www.instructabl... Download

Step 4: Code Explanation

First load the example sketch I attached a few steps ago and I'll try to give a brief explanation of what's happening. If you're using a different ESP32 dev board you should make sure that the LED pin is initialized correctly. Note that in Arduino IDE you should write the GPIO number, not necessarily the pin number shown on the board's pinout diagram.

BLE Intro

Bluetooth Low Energy (BLE) is a slightly different protocol than the traditional Bluetooth we might find in things like Bluetooth audio, for example. Instead of constantly streaming data, BLE "servers" (like the ESP32 reading sensor

data) can "notify" clients (like your smartphone) periodically to send them bits of data. Therefore, BLE is more suitable for low-power IoT applications where large amounts of data aren't required.

Now in order to know which server and client to connect to, both the server and clients use a "service UUID" which describes the overarching service (kind of like a grocery store, Walmart for example). Inside this service there can be several "characteristics" which are defined by characteristic UUID's. This can be thought of kind of like the snack section in the Walmart, or the canned food section. Then we have "descriptors", which are attributes of the characteristics describing what it's being used for, and can be thought of like the brand of potato chips in the snack aisle of Walmart. This allows interoperability and standardization between various BLE devices so that you can, for example, connect your ESP32 with a heart rate monitor like what Andreas Spiess does in this YouTube video. You can view some example descriptors here.

So to summarize, when you (the client) check out Walmart (the service) you might be looking for potato chips (the characteristic) and pick up some Pringles (the descriptor). Because the product is labeled "Pringles" and not "Great Value" you know which product to choose from and what to expect. This is sort of how BLE devices operate. In our example, we use two different characteristics, TX and RX under the overarching "service" to send data to and receive data from a client (Android device) via these two channels. The ESP32 (acting as the server) "notifies" the client via the TX characteristic UUID and data is sent to the ESP32 and received via the RX characteristic UUID. However, since there is sending and receiving, TX on the ESP32 is actually RX on the Android app, so inside Thunkable you will notice that the UUID's are swapped from those in the Arduino sketch.

You could also think about this like AT&T customer service:

- Server --> Waiting for client to connect
- Client --> Connects to service
- Server --> Via Customer Support characteristic: "Hi, how may I help you? Would you like to consider our special family bundle?"
- Client --> Via Raised Voice characteristic: "No thanks, I would just like to know why my bill went up this time"
- Server --> Via Customer Support characteristic: "OK, no problem. Would you also like to upgrade your Internet speed for only $5 more per month?"
- Client --> Disconnects

Arduino Code Explained

In this section I'll point out a few important things. At the top of the sketch we include the necessary libraries for the code to run:

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

Thankfully these libraries were bestowed upon us by Neil Kolban (thanks again!) and are now included in the ESP32 package distribution by default.

Also near the top of the sketch we define the analog read pin as well as the LED pin. Note that it's the GPIO pin number and not necessarily other pin numbers you see on Google.

```
const int readPin = 32; // Use GPIO number. See ESP32 board pinouts<br>const int LED = 2; // Could be different depending on the dev board. I used the DOIT ESP32 d
ev board.
```

Next, we define the service and characteristic UUID's for both TX and RX (two-way communication):

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

Because what's transmitted on one end is received on the other and vice versa, the RX UUID in the *app* is the TX UUID for the *ESP32* and vice versa. Next let's look at the callback function that handles the Bluetooth connection status:

```
class MyServerCallbacks: public BLEServerCallbacks {<br>    void onConnect(BLEServer* pServer) {
      deviceConnected = true;
    };
    void onDisconnect(BLEServer* pServer) {
      deviceConnected = false;
    }
};
```

All this does is set the "deviceConnected" flag true or false when you connect or disconnect from the ESP32. Similarly there's another callback function that handles receiving data being sent from the client (phone):

```
<p>class MyCallbacks: public BLECharacteristicCallbacks {<br>    void onWrite(BLECharacteristic *pCharacteristic) {
      std::string rxValue = pCharacteristic->getValue();</p><p>      if (rxValue.length() > 0) {
        Serial.println("**********");
        Serial.print("Received Value: ");</p><p>        for (int i = 0; i < rxValue.length(); i++) {
          Serial.print(rxValue[i]);
        }</p><p>        Serial.println();</p><p>        // Do stuff based on the command received from the app
        if (rxValue[0] == '1') {
          Serial.print("Turning ON!");
          digitalWrite(LED, HIGH);
        }
        else if (rxValue.find("B") != -1) {
          Serial.print("Turning OFF!");
          digitalWrite(LED, LOW);
        }</p><p>        Serial.println();
        Serial.println("**********");
      }
    }
};</p>
```

I've added an "if" statement at the end that toggles the LED on or off depending on what letter is sent by the app. Now let's have a look at the setup() function. As usual, we set up Serial and set the LED pin to OUTPUT but then we also initialize the ESP32 as a BLE device and set its name:

```
<p>// Create the BLE Device<br>BLEDevice::init("ESP32 UART Test"); // Give it a name</p>
```

Next, we create the BLE server,

```
// Create the BLE Server
BLEServer *pServer = BLEDevice::createServer();
pServer->setCallbacks(new MyServerCallbacks());
```

create a BLE service using the service UUID,

```
<p>// Create the BLE Service<br>BLEService *pService = pServer->createService(SERVICE_UUID);</p>
```

and add the characteristics

```
// Create a BLE CharacteristicpCharacteristic = pService->createCharacteristic(
                    CHARACTERISTIC_UUID_TX,
                    BLECharacteristic::PROPERTY_NOTIFY
                  );
pCharacteristic->addDescriptor(new BLE2902());
BLECharacteristic *pCharacteristic = pService->createCharacteristic(
                                       CHARACTERISTIC_UUID_RX,
                                       BLECharacteristic::PROPERTY_WRITE
                                     );
pCharacteristic->setCallbacks(new MyCallbacks());
```

According to Andreas Spiess' video, here the BLE2902 descriptor makes it so that the ESP32 won't notify the client unless the client wants to open its ears up to read the values to eliminate "talking to the air" and we also set the callback that handles receiving values via the RX channel. However, if you try uncommenting the BLE2902 line and even the BLE2902 #include line, the code still seems to run just as it did before! Maybe someone more knowledgeable can tell us what's going on here! Next, we start the BLE service and start advertising, but the ESP32 ain't gonna send nothin' until a client connects!

```
// Start the service<br>pService->start();
```

```
// Start advertising
pServer->getAdvertising()->start();
Serial.println("Waiting a client connection to notify...");
```

Now let's take a look at the loop() function. Here we check if the device is connected or not (handled by the callback function), and if so, we continue to read a sensor value (for now it's just a random analog reading), convert it to a char array using "dtostrf" so that the app can process it, set the value to send, and notify the client!

```
void loop() {
  if (deviceConnected) {
    // Fabricate some arbitrary junk for now...
    txValue = analogRead(readPin) / 3.456; // This could be an actual sensor reading!
    // Let's convert the value to a char array:
    char txString[8]; // make sure this is big enuffz
    dtostrf(txValue, 1, 2, txString); // float_val, min_width, digits_after_decimal, char_buffer

//    pCharacteristic->setValue(&txValue, 1); // To send the integer value
//    pCharacteristic->setValue("Hello!"); // Sending a test message
    pCharacteristic->setValue(txString);

    pCharacteristic->notify(); // Send the value to the app!
    Serial.print("*** Sent Value: ");
    Serial.print(txString);
    Serial.println(" ***");
  }
  delay(1000);
}
```

Upload the Sketch!

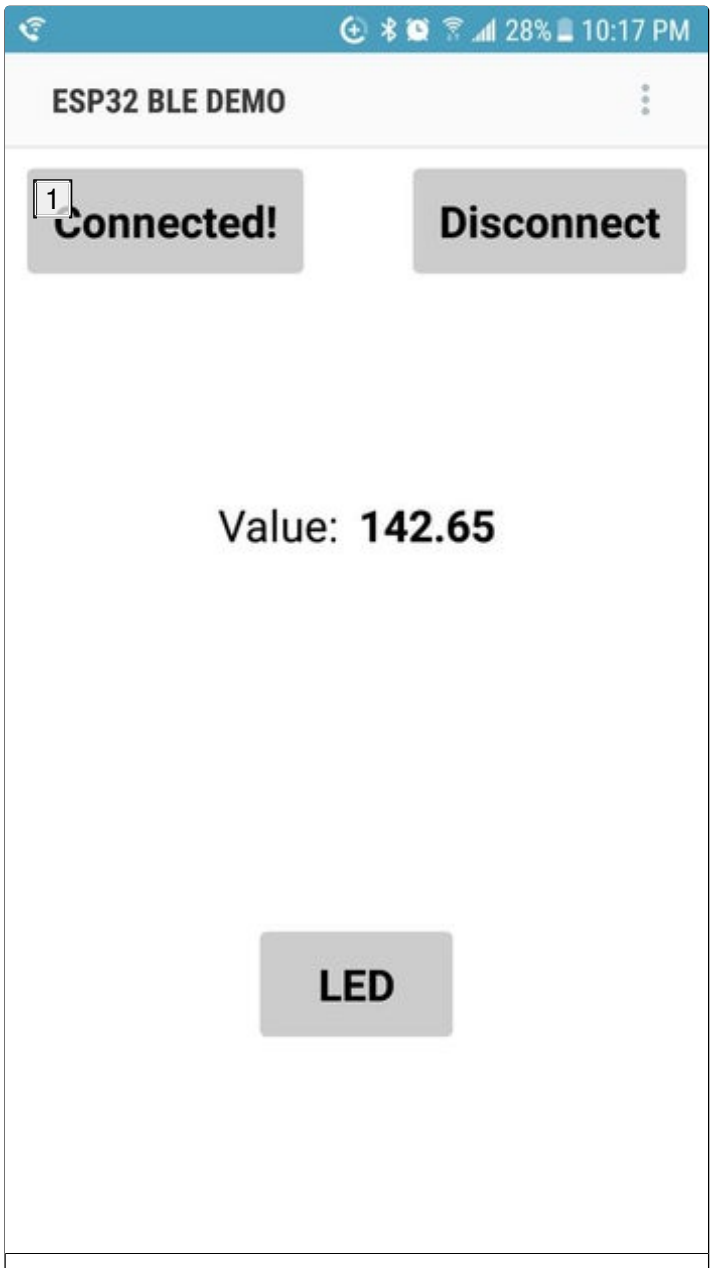
Now upload the sketch to your ESP32 board, making sure that you have the right board and COM port selected. When it's done, open the serial monitor under Tools / Serial Monitor and you should see "Waiting a client connection to notify..." Now open the Android app, click the "Connect" button at the top left, and you should see a list of available nearby devices. Select the ESP32 and you should see the button text change to "Connected!" and start seeing values on the screen. To toggle the LED on or off press the "LED" button and check the serial monitor to see how it sends "A" or "B" to the ESP32. Pretty neat stuff huh?

Sending Multiple Values

A lot of people have asked this question: "how do I send multiple values to and from the ESP32 and app?" That's a good question, and luckily it's not hard at all! The easiest way I've found is to simply send the values in comma-separated variable (CSV) format. For example, if you're measuring temperature and humidity and you measured 21 *C and 55% humidity and want to send it to the app, simply program the ESP32 to send "21,55" and the app can parse it easily.

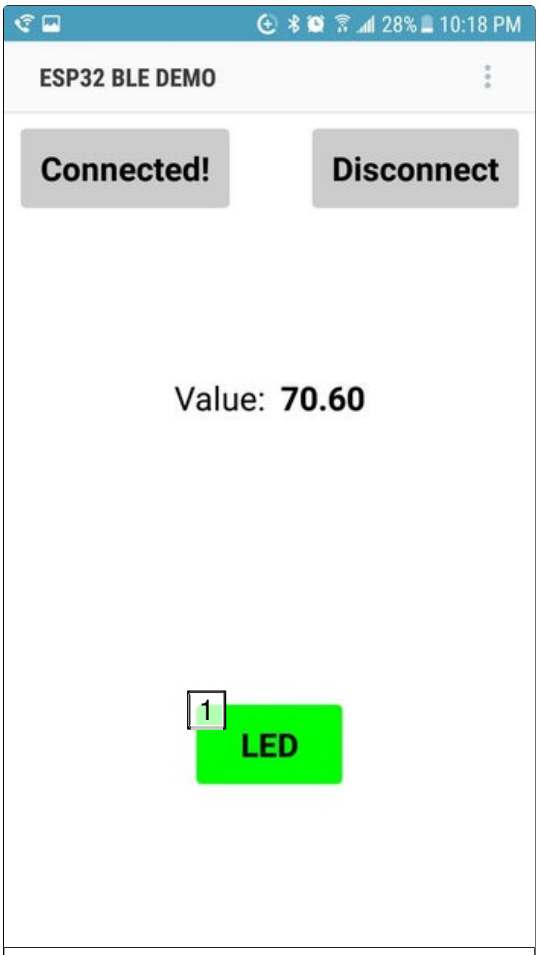
Sending Lots of Data

Unfortunately BLE isn't really meant for large streams of data (that's more for traditional Bluetooth, like those used in audio-streaming devices). The max allowable data size per packet is 20 bytes for BLE specification, so if you want to send anything more you'll have to split it up into multiple packets. Fortunately this is not that hard to do either. Simply use a delimiter like "*" or "!" or something unique at the end of your entire message to let the app know the message is complete and to start listening for a new message. For example, if you want to send and and cumulatively + > 20 bytes, then what you can do is send then proceed with the next message if needed.

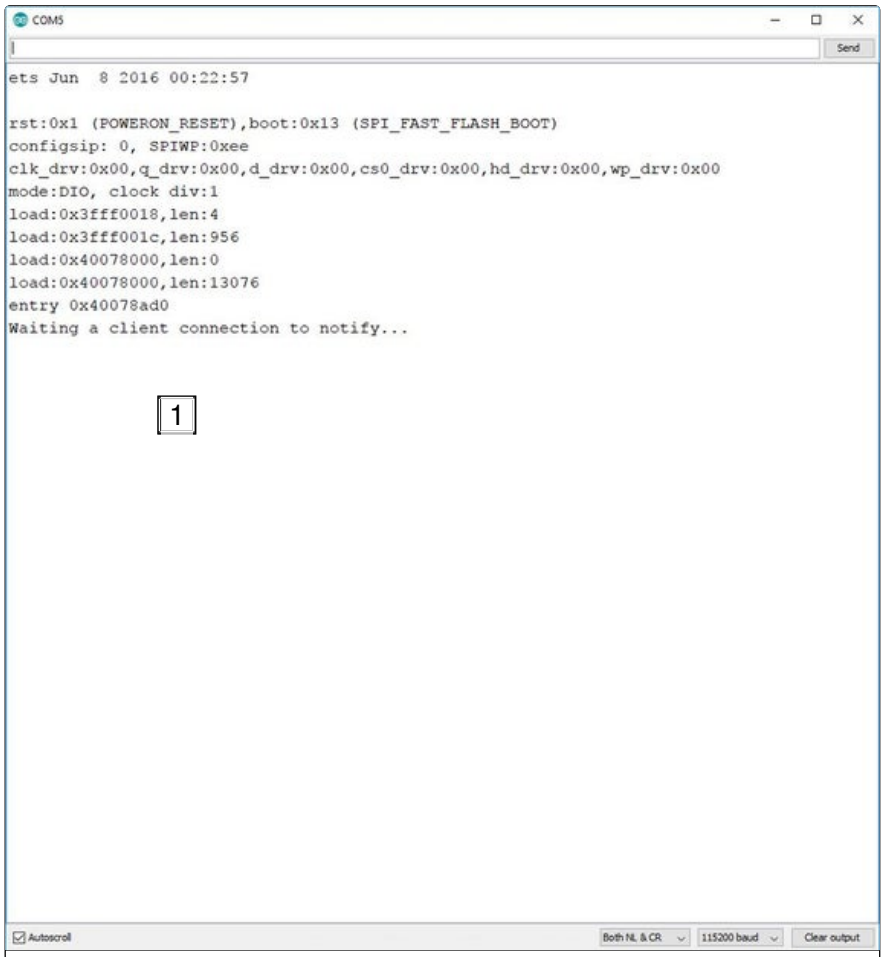


1. Once it says "Connected!" we're in good shape and you should see values start to roll in below!

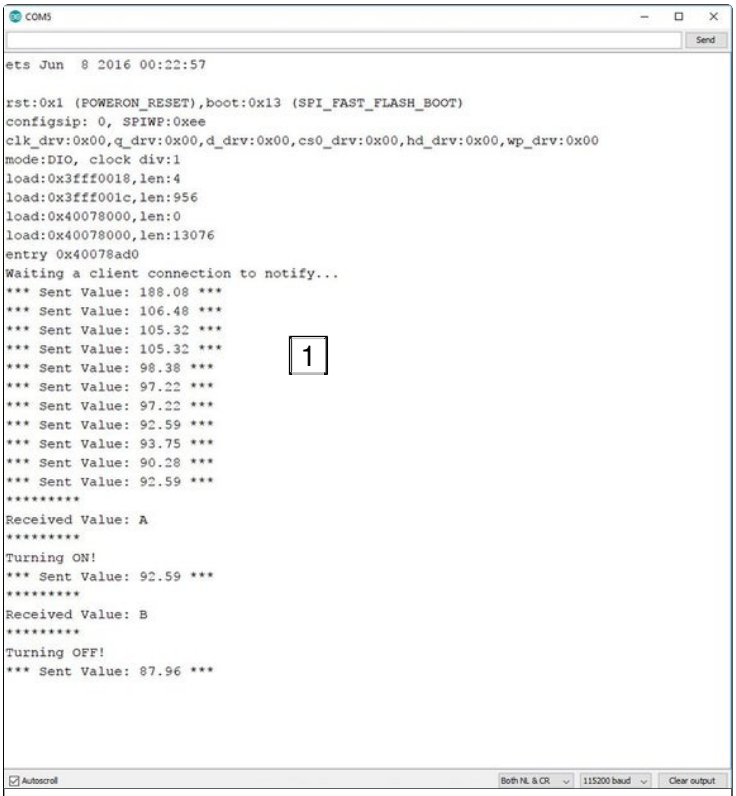
values start to roll in below!



1. You can press the LED button to toggle the LED on or off.



1. You need to connect to the ESP32 from the app before it will do anything!



1. When connected via Bluetooth the ESP32 will send random values to the app and you can also control the LED by pressing the button on the app. You will see it send "A" or "B" to toggle the LED state.

Step 5: Easy, Peasy, BL-Easy!

The ESP32 is literally exploding with features! In this tutorial we've just learned the basics of how to create a simple Android app for two-way communication between your mobile device and the ESP32 using Bluetooth Low Energy. With this knowledge combined with WiFi and sensors we can now make some really cool projects with this! Also feel free to experiment with the app and throw in extra features for things like voice recognition, color pickers for LED control, slide bars for motor speed, or use your phone's accelerometer for controlling a robot via Bluetooth!

- If you liked this Instructable, please give it a heart, vote for it, and share!
- Feel free to check out my website here and my humble YouTube channel here for more cool projects like this!
- You might also be interested in this follow-up tutorial I made about building your own circuit board reflow oven with a toaster oven and ESP32!
- If you have any questions, comments, suggestions, or replicated this project, let me know in the comments section below!

Hello, I tried the .apk and when i click connect, there is a black screen, no BLE device. I'm on android 8.0

Hi, Your app is working well. I have a issue that when I use your .aia file and try to compile it using thunkable the app does not work. I am unable to pick bluetooth in the app after creating the apk file from you .aia file. Is the .aia file not correct? Please confirm. Thanks

I have the exact same problem! Did you by any chance find a solution??
Thanks!

Hi, Your app is working well. But if I need to control 4 LEDs , What we need to change ? in your .aia file and in your ESP32_BLE_UART_Demo file ? Please confirm. Thanks

Hi.
Thanks for the tutorial.

Can anyone confirm if in the current version of the web THUNKABLE (2019) projects * .aia can be uploaded?

Thank you.

As far as I know, it's not possible.
You can use the classic thunkable via app.thunkable.com, there it's definitely possible to upload .aia files.

Hi, great tutorial!...How can I connect to multiple devices, so I could to to read many BLE sensors?

Someone asked the same question before. You just need to add a second instance of the BLE client extension in Thunkable.

Can't thank you enough. Exceptional tutorial. I often think our own individual understanding is the ability to find the “key” to unlock what we are trying to understand. Different things work for different people. Your analogy about Walmart and the ATT Customer Service did it for me. So far the app and code worked perfectly and now I can reverse engineer it to see how it actually works.

Very, very well done. Good luck with your work and I subscribed to your YouTube channel. Time is hard to find but if you ever decide to do an Instructable for Bluetooth Classic like the BLE, it'd be great.

Hi, did you have any problem with :

C:\Users\dso\Documents\Arduino\libraries\ESP32_BLE_Arduino-master\src/BLEDevice.h:16:20: fatal error: esp_bt.h: No such file or directory

during build with arduino IDE ?

Regards

No, it runs fine. Looks like you didn't install the ESP32 package for Arduino properly. Please check again.

Hi, thanks for the example.

i want to change the bluetooth device (ESP32 UART Test) server name to recognize different board.

i had tried to change *BLEDevice::init("ESP32 UART Test"); to BLEDevice::init("ESP32 UART Test1");*

but it doesn't work when i check on my phone . what can i do?

It might be that both devices have the same UUID's and are therefore are treated as the same device. Try using the UUID generator to create different UUID's for the second device.

I have a question, can you explain to me how to use this option

"pCharacteristic->setValue(&txValue, 1); // To send the integer value"

because i can't send integers to my app, i only receive "()" .

Thanks

If you declare "txValue" as type uint8_t (integer) then, for example, you can do something like

txValue = analogRead(readPin);

pCharacteristic->setValue(&txValue, 1); // Format: setValue(data, length)

pCharacteristic->notify();

To send the data. Please note that this does not work if you declare it simply as "int". Also, in the app you need to use the BluetoothLE.IntegersReceived block rather than .StringsReceived because it will try to convert the numbers into ASCII character equivalent (if you send 65 the app will show "A"). You can see the built-in example of this in Arduino IDE under Examples -> ESP32 BLE Arduino -> BLE_uart. This sketch sends a uint8_t value and increments it.

All that being said, I think it's easiest just to convert the number to a string and send that instead and use the .StringsReceived block in the app.

Well ... i've done exactly what u mention (uint8_t txValue and IntegersReceived) but it only receives "()" i don't know why, so i decide to use strings, i'll check the example anyways.

Thanks

Hello, during compiling the code i get this error: ESP32_BLE_UART_Demo:91: error: 'createServer' is not a member of 'BLEDevice' BLEServer *pServer = BLEDevice::createServer()

how may ia to solve it?

thanks

Please check if you installed the ESP32 package for Arduino correctly. You should see some BLE examples like mentioned in the tutorial if the installation was done properly.

Great tutorial. Very clean and works fine from the start. Helped me discover "thunkable" which is an easier alternative than Android Studio. Thanks again.

Glad you liked it! :)

Do you have any example to send post request to a server?

Hi there, this Instructable doesn't focus on the WiFi aspects of the ESP32 but almost everything about the WiFi is exactly the same as the ESP8266 except that you have to change the #include <ESP8266WiFi.h> to simply #include <WiFi.h>. There are many examples but here's a good one that you can easily change to do a POST request: https://github.com/espressif/arduino-esp32/tree/master/libraries/HTTPClient/examples/BasicHttpClient

I am having trouble to get connected. There is no .click event handling for the Connect button. Or does it suppose to scan automatically and then reacts on a pick of one of the listed addresses? Probably some setting of Thunkable not OK? Any suggestions to get it going will bring me from the ground.

Hi, please make sure Bluetooth is enabled on your Android device. Also, make sure your device is BLE-capable (recent phones and tablets should all work). Once you open the app it will ask you to turn on Bluetooth. Click yes to turn it on. Then click the Connect button to open the list of available BLE devices. Select the ESP 32 and it should say "Connected!"

The app scans for BLE right after you open the app. When you click on the top left button it will show you the nearby devices.

Let me know if you have any other issues.

Regards,
Tim

Thx for the explanation. I did the BLE example of Adreas Spiess in the mean time and that works as expected so BLE should be OK.

I understand a bit more after your explanation, but there is no device listed to pick when I click the Connect list. I am well in range as I found out with Andreas' example. On my Server I get "flash read err, 1000" after reset button.
==========
(COM4) rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)

(COM4) flash read err, 1000
(COM4) ets_main.c 371
==========

But the program starts running and comes to the end of Setup and loops too.

I tried flashing in QIO and DIO mode, no difference, both this error. Working with Arduino IDE and ESP32 Dev module

It seems there is something wrong on the client side in the Thunkable app.

Hmmm that's strange, it works just fine for me for "ESP32 Dev Board" in Arduino IDE and I tried it again yesterday. It's hard to believe that the app would mess up the ESP32. Can anyone please comment if they got it to work (or not)?

Did you use the .APK file or the .aia file? Just curious. Maybe try Thinkable Live. It works on both here.

Sorry for the slow answer. This is just a thing I do aside. I used the .aia and generate with QR code my app. Thunkable Live also works, but I have to find out all features of this beside seeing changes on my Android after I edited my PC Thunkable. For this I am going to study the ino's of Andreas Spiess to understand what is really going on, because this works and I can add debug commands. Is there something like debugging the app?

So the app works? I don't think there's a way to really debug because it's more of a hobbyist type of app-building rather than using Android Studio or something more standard. However, sometimes what I do is add a label on the screen and throw debug text in it. Just use your imagination and you can actually do quite a bit with Thunkable!

Thx for your answer! OK I now know not to look for any fancy debugging but will try this Thunkable to see happening things like a serial monitoring on my Arduino IDE.At the moment lots of other things to do, so this have to wait a little while.

Ok, have fun! Glad you can at least find it useful :)

To answer your question about the app programming, here's what it does:
- When the screen initializes it immediately starts scanning for BLE devices (assuming the user turned BT on)
- Once a device is found from that scan the ".DeviceFound" event handler populates the "Connect" list. Note that "Connect" isn't really a button; it's actually a list.
- Once the user opens the "Connect" list of available devices and picks one, the app will then try to

connect with that device
- Once connected to that device the ".Connected" event handler will make the app stop scanning and change the "Connect" button/list name to "Connected!"

Hope that helps!

Hello,

I wanted to try out your example, but I'm really stuk at the following point.

On thunkable if i try to create an APK it says:

"Error generating Yail for screen 5457454623096832_Screen1: : Cannot read property 'map' of undefined. Please fix and try packaging again."

If i look at the blocks there are a few blocks with "Call bluetooth LE" which are freely floating and other things like "get global service UUID" which are floating arroung with an explaination mark and cannot be docket to the "Call"s.

As I have no experience with Thunkable - what can i do to make it work?

Hmmmmm that is super weird! I just tried loading the app and now it's bringing up the floating blocks as well. It seems like it's deleting the "BluetoothLE" extension entirely when I load it. I'll contact Thunkable about this because it's the first time I've had an issue with it. This is quite bizarre actually!

Hi,

I retried it. Not really working and again have floating blocks.

Now I tried the same with the original AppInventor a few times (upload, delete, look at the blocks, upload, delete and so on). The same AIA sometimes work after upload - but most of the time it doees not work (Floating blocks or it cannot load the blocks at all).

Even if it works it seems to be a bit unstable.

From one working copy i tried to save a copy directly. After the save both the original and the copy do not work any longer.

I tried to do the example from scratch and imported the latest Bluetooth LE extension. First it worked with one button, also with USB connection. After copying just a block for a second button it still worked with USB Connection - but only showed errors when trying to save as APK. Deleting the block - the APK showed the same Errors, that the Bluetooth extension is unknown.

Tried the same with multiple Checkpoints after each button i added. At different points it seems to fail. The same operation of copy, change button name and Information to be send to Bluetooth server worked sometimes, sometimes not.

Also exporting the new project to AIA and importing it with a second account it sometime worked, sometimes not.

So the original AppInventor engine which also drives Thunkable as fork seems to be a somehow unstable with the extension.

I also noticed if it works the thunkable part seem to be more unstable then the more complicated way with the Android Builder.

Your example is good - but sorry the Thunkable part seems to have some issues (have you tried to connect, disconnect and reonnect a few times? after a while I get index errors)

One point of your example in the C++ Code is a bit weird.

pCharacteristic->addDescriptor(new BLE2902());

BLECharacteristic *pCharacteristic = pService->createCharacteristic(

You do first use pCharacteristic and then define it again?

Were you using the latest files (not just extension) I provided in this Instructable? I just tried logging in again and it loads all the blocks just fine. I also loaded some other apps using the same extension and they seem OK. The main reason it was flaky was because I was using an older version of the extension but now that I'm using the latest one it seems to be just fine.

As far as the code, I didn't write that part. It was from Neil Kolban's example UART sketch: https://github.com/nkolban/ESP32_BLE_Arduino/blob/master/examples/BLE_uart/BLE_uart.ino

I completely re-did the app by deleting the extension, adding it back, and using the same blocks as before. However, now it can control the LED but it's not receiving data... I'm trying to ask Thunkable if it's a glitch in the extension or Thunkable because I had it working at the time I wrote this Instructable. Very strange.
Oh, and I should mention that I tested it on an older version of the app and it was working, then when I switched to the new version that I re-made, it didn't work! Super weird. Anyway, I'll keep you updated but at least you shouldn't have the problem where the blocks end up floating.

I just uploaded a new .aia file in the Instructable. Could you please try that one? After re-installing the BLE extension in Thunkable it seems to be OK (and after re-adding the blocks). At least now it doesn't go away when I close Thunkable and open it back up.

Hmmm it's doing the same thing, I logged back in later and it created the floating blocks. Let me see what I can do to fix it and I'll get back to you on this.

Figured it out guys! The reason it wasn't reading data from the ESP32 correctly after I re-made the app was because I downloaded an older version of the BLE extension. Just for reference, if you do make a new app and need to add the BLE extension, go here to get the latest one: https://puravidaapps.com/extensions.php and click "Bluetooth Low Energy Extension (New)". I've re-uploaded the .aia and .apk files in the Instructable and the app works great again!

Let me know if you have any further issues!