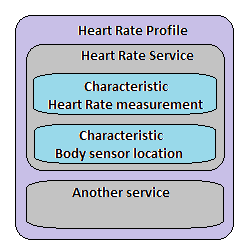
# **[Demo 27: How to use Arduino ESP32 BLE (Bluetooth Low Energy) as a GATT server](http://www.iotsharing.com/2017/07/how-to-use-ble-in-arduino-esp32.html)**

**1. Introduction**  
In this tutorial (2 parts: [part 1: GATT server](http://www.iotsharing.com/2017/07/how-to-use-ble-in-arduino-esp32.html) and [part 2: GATT client](http://www.iotsharing.com/2017/07/how-to-use-arduino-esp32-ble-as-gatt-client.html)), I will show you how to use BLE (Bluetooth Low Energy) in Arduino ESP32. Firstly, we need to know some basic concepts.   
-  GAP stands for Generic Access Profile. GAP makes your device visible to the other BLE devices (BLE devices can scan your BLE device), and determines how two devices can interact with each other.  
- There are 2 kinds of devices in BLE communication: Central devices and Peripheral devices.  
+ Peripheral devices are small, low power, resource constrained devices that can connect and give data to to a powerful central device. Example: a heart rate monitor.  
+ Central devices are usually the powerful  devices like smart phone or tablet. It can scan and connect to any peripheral device that is advertising information to get data from peripheral device.  
- When the connection is established between the peripheral and a central device, the advertising process will generally stop and you will use GATT **(Generic Attribute Profile)** services and characteristics to communicate in both directions.  
- GATT is based on a traditional client-server architecture including GATT Server and GATT Client.  
- The peripheral device keeps the role as the GATT Server, and the central device keeps the role as GATT Client, which sends requests to this server.    
- Beside that GATT also has some concepts called **Profiles**, **Services** and **Characteristics.**  
+ **Profiles**: is a collection of Services.  
+ **Services**: is a collection of characteristic. Service distinguishes from other services by a unique 16-bit numeric ID called a UUID.  
+ **Characteristics**: is data. Characteristic distinguishes from other Characteristics by a unique 16-bit or 128-bit numeric ID called a UUID.

[](https://3.bp.blogspot.com/-w4J184r_rEQ/WWxaoCH5QOI/AAAAAAAAEKs/zC8ekx7rKwMkcXgXYe-SfhvNz7XJrfG5wCLcBGAs/s1600/esp32-bluetooth.png)

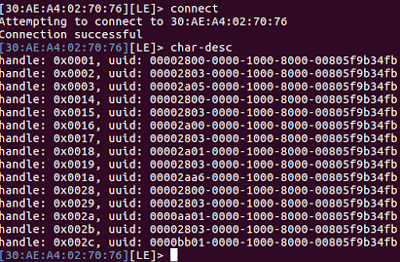
**Figure: Example of Hear Rate Profile**

- In this tutorial, I will make a demo using GATT to turn ON/OFF a LED. So ESP32 will act as a GATT server and a GATT client (I use Raspbbery Pi3 with BLE or if your laptop is equipped with BLE you can use it).  
  
**2. Software**  
- We will use BLE driver and Bluez Gatttool for our demo.  
**2.1 Bluez Gatttool**  
If the gatttool is noi installed on your RPi3 then you can follow these steps to install it:  
**+ wget http://www.kernel.org/pub/linux/bluetooth/bluez-5.46.tar.xz**  
**+ tar xvf bluez-5.46.tar.xz**  
**+ sudo apt-get install libglib2.0-dev libdbus-1-dev libusb-dev libudev-dev libical-dev systemd libreadline-dev**  
**+ ./configure --enable-library**  
**+ make -j8 && sudo make install  
+ sudo cp attrib/gatttool /usr/local/bin/**  
- Run a BLE scan: **sudo hcitool lescan** : It will return the MAC address of BLE device  
- Connect to the BLE device: **sudo gatttool -b BLE\_ADDR -I** : then type **connect**to connect. We use **sudo hcitool lescan**to get **BLE\_ADDR**  
**Note:**If you could not connect after typing connect, you should try typing connect some times.  
- List all uuids of services: **primary**  
- List all available handles (Handles are the «connection points» where you can read and write access data):**char-desc**  
- Read from a handle: **char-read-hnd**  
- Write to a handle: **char-write-req**

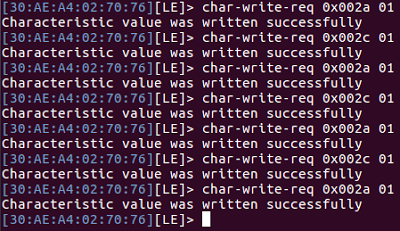
**2.2 ESP32**  
I also wrote many comments in the code, so you can read and map them with theory above.

|  |
| --- |
| #pragma GCC diagnostic push  #pragma GCC diagnostic warning "-fpermissive"  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include "freertos/FreeRTOS.h"  #include "freertos/task.h"  #include "freertos/event\_groups.h"  #include "esp\_system.h"  #include "esp\_log.h"  #include "nvs\_flash.h"  #include "bt.h"  #include "bta\_api.h"  #include "esp\_gap\_ble\_api.h"  #include "esp\_gatts\_api.h"  #include "esp\_bt\_defs.h"  #include "esp\_bt\_main.h"  #include "esp\_bt\_main.h"  #include "sdkconfig.h"  #pragma GCC diagnostic pop  #define GATTS\_TAG "GATTS\_DEMO"  /\* this function will be invoked to handle incomming events \*/  static void gatts\_profile\_event\_handler(esp\_gatts\_cb\_event\_t event, esp\_gatt\_if\_t gatts\_if, esp\_ble\_gatts\_cb\_param\_t \*param);  #define LED 4  #define GATTS\_SERVICE\_UUID\_TEST\_ON 0x00FF  #define GATTS\_CHAR\_UUID\_TEST\_ON 0xAA01  #define GATTS\_CHAR\_UUID\_TEST\_OFF 0xBB01  #define GATTS\_NUM\_HANDLE\_TEST\_ON 8  #define TEST\_DEVICE\_NAME "ESP\_GATTS\_ON\_OFF"  /\* maximum value of a characteristic \*/  #define GATTS\_DEMO\_CHAR\_VAL\_LEN\_MAX 0xFF  /\* value range of a attribute (characteristic) \*/  uint8\_t attr\_str[] = {0x00};  esp\_attr\_value\_t gatts\_attr\_val =  {  .attr\_max\_len = GATTS\_DEMO\_CHAR\_VAL\_LEN\_MAX,  .attr\_len = sizeof(attr\_str),  .attr\_value = attr\_str,  };  /\* service uuid \*/  static uint8\_t service\_uuid128[32] = {  /\* LSB <--------------------------------------------------------------------------------> MSB \*/  //first uuid, 16bit, [12],[13] is the value  0xfb, 0x34, 0x9b, 0x5f, 0x80, 0x00, 0x00, 0x80, 0x00, 0x10, 0x00, 0x00, 0xAB, 0xCD, 0x00, 0x00,  };  static esp\_ble\_adv\_data\_t test\_adv\_data = {  .set\_scan\_rsp = false,  .include\_name = true,  .include\_txpower = true,  .min\_interval = 0x20,  .max\_interval = 0x40,  .appearance = 0x00,  .manufacturer\_len = 0,  .p\_manufacturer\_data = NULL,  .service\_data\_len = 0,  .p\_service\_data = NULL,  .service\_uuid\_len = 16,  .p\_service\_uuid = service\_uuid128,  .flag = (ESP\_BLE\_ADV\_FLAG\_GEN\_DISC | ESP\_BLE\_ADV\_FLAG\_BREDR\_NOT\_SPT),  };  esp\_ble\_adv\_params\_t test\_adv\_params;  #define PROFILE\_ON\_APP\_ID 0  /\* characteristic ids 0 and 1 \*/  #define CHAR\_NUM 2  #define CHARACTERISTIC\_ON\_ID 0  #define CHARACTERISTIC\_OFF\_ID 1  struct gatts\_characteristic\_inst{  esp\_bt\_uuid\_t char\_uuid;  esp\_bt\_uuid\_t descr\_uuid;  uint16\_t char\_handle;  esp\_gatt\_perm\_t perm;  esp\_gatt\_char\_prop\_t property;  uint16\_t descr\_handle;  };  struct gatts\_profile\_inst {  esp\_gatts\_cb\_t gatts\_cb;  uint16\_t gatts\_if;  uint16\_t app\_id;  uint16\_t conn\_id;  uint16\_t service\_handle;  esp\_gatt\_srvc\_id\_t service\_id;  struct gatts\_characteristic\_inst chars[CHAR\_NUM];  };  /\* One gatt-based profile one app\_id and one gatts\_if, this array will store the gatts\_if returned by ESP\_GATTS\_REG\_EVT \*/  static struct gatts\_profile\_inst test\_profile;  typedef struct {  uint8\_t \*prepare\_buf;  int prepare\_len;  } prepare\_type\_env\_t;  static prepare\_type\_env\_t on\_prepare\_write\_env;  static prepare\_type\_env\_t off\_prepare\_write\_env;  void example\_write\_event\_env(esp\_gatt\_if\_t gatts\_if, prepare\_type\_env\_t \*prepare\_write\_env, esp\_ble\_gatts\_cb\_param\_t \*param);  /\* this callback will handle process of advertising BLE info \*/  static void gap\_event\_handler(esp\_gap\_ble\_cb\_event\_t event, esp\_ble\_gap\_cb\_param\_t \*param)  {  switch (event) {  case ESP\_GAP\_BLE\_ADV\_DATA\_SET\_COMPLETE\_EVT:  esp\_ble\_gap\_start\_advertising(&test\_adv\_params);  break;  case ESP\_GAP\_BLE\_ADV\_DATA\_RAW\_SET\_COMPLETE\_EVT:  esp\_ble\_gap\_start\_advertising(&test\_adv\_params);  break;  case ESP\_GAP\_BLE\_SCAN\_RSP\_DATA\_RAW\_SET\_COMPLETE\_EVT:  esp\_ble\_gap\_start\_advertising(&test\_adv\_params);  break;  case ESP\_GAP\_BLE\_ADV\_START\_COMPLETE\_EVT:  //advertising start complete event to indicate advertising start successfully or failed  if (param->adv\_start\_cmpl.status != ESP\_BT\_STATUS\_SUCCESS) {  Serial.println("Advertising start failed\n");  }  break;  case ESP\_GAP\_BLE\_ADV\_STOP\_COMPLETE\_EVT:  if (param->adv\_stop\_cmpl.status != ESP\_BT\_STATUS\_SUCCESS) {  Serial.println("Advertising stop failed\n");  }  else {  Serial.println("Stop adv successfully\n");  }  break;  default:  break;  }  }  void example\_write\_event\_env(esp\_gatt\_if\_t gatts\_if, prepare\_type\_env\_t \*prepare\_write\_env, esp\_ble\_gatts\_cb\_param\_t \*param){  /\* check char handle and set LED \*/  if(test\_profile.chars[CHARACTERISTIC\_ON\_ID].char\_handle == param->write.handle){  digitalWrite(LED, HIGH);  }else if(test\_profile.chars[CHARACTERISTIC\_OFF\_ID].char\_handle == param->write.handle){  digitalWrite(LED, LOW);  }  /\* send response if any \*/  if (param->write.need\_rsp){  esp\_ble\_gatts\_send\_response(gatts\_if, param->write.conn\_id, param->write.trans\_id, ESP\_GATT\_OK, NULL);  }  }  /\* this callback handle BLE profile such as registering services and characteristics, send response to central device \*/  static void gatts\_profile\_event\_handler(esp\_gatts\_cb\_event\_t event, esp\_gatt\_if\_t gatts\_if, esp\_ble\_gatts\_cb\_param\_t \*param) {  switch (event) {  /\* create service event \*/  case ESP\_GATTS\_REG\_EVT:  printf("REGISTER\_APP\_EVT, status %d, app\_id %d\n", param->reg.status, param->reg.app\_id);  test\_profile.service\_id.is\_primary = true;  test\_profile.service\_id.id.inst\_id = 0x00;  test\_profile.service\_id.id.uuid.len = ESP\_UUID\_LEN\_16;  test\_profile.service\_id.id.uuid.uuid.uuid16 = GATTS\_SERVICE\_UUID\_TEST\_ON;  esp\_ble\_gatts\_create\_service(gatts\_if, &test\_profile.service\_id, GATTS\_NUM\_HANDLE\_TEST\_ON);  break;  /\* when central device request info from this device, this event will be invoked and respond \*/  case ESP\_GATTS\_READ\_EVT: {  printf("ESP\_GATTS\_READ\_EVT, conn\_id %d, trans\_id %d, handle %d\n", param->read.conn\_id, param->read.trans\_id, param->read.handle);  esp\_gatt\_rsp\_t rsp;  memset(&rsp, 0, sizeof(esp\_gatt\_rsp\_t));  rsp.attr\_value.handle = param->read.handle;  rsp.attr\_value.len = 14;  rsp.attr\_value.value[0] = 105;  rsp.attr\_value.value[1] = 111;  rsp.attr\_value.value[2] = 116;  rsp.attr\_value.value[3] = 115;  rsp.attr\_value.value[4] = 104;  rsp.attr\_value.value[5] = 97;  rsp.attr\_value.value[6] = 114;  rsp.attr\_value.value[7] = 105;  rsp.attr\_value.value[8] = 110;  rsp.attr\_value.value[9] = 103;  rsp.attr\_value.value[10] = 46;  rsp.attr\_value.value[11] = 99;  rsp.attr\_value.value[12] = 111;  rsp.attr\_value.value[13] = 109;  esp\_ble\_gatts\_send\_response(gatts\_if, param->read.conn\_id, param->read.trans\_id,  ESP\_GATT\_OK, &rsp);  break;  }  /\* when central device send data to this device, this event will be invoked \*/  case ESP\_GATTS\_WRITE\_EVT: {  printf("ESP\_GATTS\_WRITE\_EVT, conn\_id %d, trans\_id %d, handle %d\n", param->write.conn\_id, param->write.trans\_id, param->write.handle);  printf("value len %d, value %08x\n", param->write.len, \*(uint8\_t \*)param->write.value);  example\_write\_event\_env(gatts\_if, &on\_prepare\_write\_env, param);  break;  }  /\* start service and add characterstic event \*/  case ESP\_GATTS\_CREATE\_EVT:  printf("status %d, service\_handle %d\n", param->create.status, param->create.service\_handle);  /\* 1 service LED and 2 characteristics ON and OFF \*/  test\_profile.service\_handle = param->create.service\_handle;  /\* add char ON \*/  esp\_ble\_gatts\_add\_char(test\_profile.service\_handle, &test\_profile.chars[CHARACTERISTIC\_ON\_ID].char\_uuid,  ESP\_GATT\_PERM\_READ | ESP\_GATT\_PERM\_WRITE,  ESP\_GATT\_CHAR\_PROP\_BIT\_READ | ESP\_GATT\_CHAR\_PROP\_BIT\_WRITE | ESP\_GATT\_CHAR\_PROP\_BIT\_NOTIFY,  &gatts\_attr\_val, NULL);  test\_profile.service\_handle = param->create.service\_handle;  /\* add char OFF \*/  esp\_ble\_gatts\_add\_char(test\_profile.service\_handle, &test\_profile.chars[CHARACTERISTIC\_OFF\_ID].char\_uuid,  ESP\_GATT\_PERM\_READ | ESP\_GATT\_PERM\_WRITE,  ESP\_GATT\_CHAR\_PROP\_BIT\_READ | ESP\_GATT\_CHAR\_PROP\_BIT\_WRITE | ESP\_GATT\_CHAR\_PROP\_BIT\_NOTIFY,  &gatts\_attr\_val, NULL);  esp\_ble\_gatts\_start\_service(test\_profile.service\_handle);  break;  /\* add characteristic descriptor for 2 char ON and OFF.  when it's done, a callback event BTA\_GATTS\_ADD\_DESCR\_EVT is called \*/  case ESP\_GATTS\_ADD\_CHAR\_EVT: {  printf("ADD\_CHAR\_EVT, status %d, attr\_handle %d, service\_handle %d\n",  param->add\_char.status, param->add\_char.attr\_handle, param->add\_char.service\_handle);  /\* store char handle \*/  if(param->add\_char.char\_uuid.uuid.uuid16 == GATTS\_CHAR\_UUID\_TEST\_ON){  test\_profile.chars[CHARACTERISTIC\_ON\_ID].char\_handle = param->add\_char.attr\_handle;  }else if(param->add\_char.char\_uuid.uuid.uuid16 == GATTS\_CHAR\_UUID\_TEST\_OFF){  test\_profile.chars[CHARACTERISTIC\_OFF\_ID].char\_handle = param->add\_char.attr\_handle;  }    break;  }  case ESP\_GATTS\_ADD\_CHAR\_DESCR\_EVT:  printf("ESP\_GATTS\_ADD\_CHAR\_DESCR\_EVT, status %d, attr\_handle %d, service\_handle %d\n",  param->add\_char.status, param->add\_char.attr\_handle, param->add\_char.service\_handle);  break;  /\* when disconneting, send advertising information again \*/  case ESP\_GATTS\_DISCONNECT\_EVT:  esp\_ble\_gap\_start\_advertising(&test\_adv\_params);  break;  /\* When gatt client connect, the event comes \*/  case ESP\_GATTS\_CONNECT\_EVT: {  printf("ESP\_GATTS\_CONNECT\_EVT\n");  esp\_ble\_conn\_update\_params\_t conn\_params = {0};  memcpy(conn\_params.bda, param->connect.remote\_bda, sizeof(esp\_bd\_addr\_t));  /\* For the IOS system, please reference the apple official documents about the ble connection parameters restrictions. \*/  conn\_params.latency = 0;  conn\_params.max\_int = 0x50; // max\_int = 0x50\*1.25ms = 100ms  conn\_params.min\_int = 0x30; // min\_int = 0x30\*1.25ms = 60ms  conn\_params.timeout = 1000; // timeout = 1000\*10ms = 10000ms  printf("ESP\_GATTS\_CONNECT\_EVT, conn\_id %d, remote %02x:%02x:%02x:%02x:%02x:%02x:, is\_conn %d\n",  param->connect.conn\_id,  param->connect.remote\_bda[0], param->connect.remote\_bda[1], param->connect.remote\_bda[2],  param->connect.remote\_bda[3], param->connect.remote\_bda[4], param->connect.remote\_bda[5],  param->connect.is\_connected);  test\_profile.conn\_id = param->connect.conn\_id;  //start sent the update connection parameters to the peer device.  esp\_ble\_gap\_update\_conn\_params(&conn\_params);  break;  }  default:  break;  }  }  static void gatts\_event\_handler(esp\_gatts\_cb\_event\_t event, esp\_gatt\_if\_t gatts\_if, esp\_ble\_gatts\_cb\_param\_t \*param)  {  /\* If event is register event, store the gatts\_if for each profile \*/  if (event == ESP\_GATTS\_REG\_EVT) {  if (param->reg.status == ESP\_GATT\_OK) {  test\_profile.gatts\_if = gatts\_if;  } else {  printf("Reg app failed, app\_id %04x, status %d\n",  param->reg.app\_id,  param->reg.status);  return;  }  }  /\* here call each profile's callback \*/  if (gatts\_if == ESP\_GATT\_IF\_NONE || /\* ESP\_GATT\_IF\_NONE, not specify a certain gatt\_if, need to call every profile cb function \*/  gatts\_if == test\_profile.gatts\_if) {  if (test\_profile.gatts\_cb) {  test\_profile.gatts\_cb(event, gatts\_if, param);  }  }  }  void setup(){  Serial.begin(115200);  pinMode(LED, OUTPUT);  digitalWrite(LED, LOW);  /\* initialize advertising info \*/  test\_adv\_params.adv\_int\_min = 0x20;  test\_adv\_params.adv\_int\_max = 0x40;  test\_adv\_params.adv\_type = ADV\_TYPE\_IND;  test\_adv\_params.own\_addr\_type = BLE\_ADDR\_TYPE\_PUBLIC;  test\_adv\_params.channel\_map = ADV\_CHNL\_ALL;  test\_adv\_params.adv\_filter\_policy = ADV\_FILTER\_ALLOW\_SCAN\_ANY\_CON\_ANY;  /\* initialize profile and characteristic \*/  test\_profile.gatts\_cb = gatts\_profile\_event\_handler;  test\_profile.gatts\_if = ESP\_GATT\_IF\_NONE; /\* Not get the gatt\_if, so initial is ESP\_GATT\_IF\_NONE \*/  test\_profile.chars[CHARACTERISTIC\_ON\_ID].char\_uuid.len = ESP\_UUID\_LEN\_16;  test\_profile.chars[CHARACTERISTIC\_ON\_ID].char\_uuid.uuid.uuid16 = GATTS\_CHAR\_UUID\_TEST\_ON;  test\_profile.chars[CHARACTERISTIC\_ON\_ID].perm = ESP\_GATT\_PERM\_READ | ESP\_GATT\_PERM\_WRITE;  test\_profile.chars[CHARACTERISTIC\_ON\_ID].property = ESP\_GATT\_CHAR\_PROP\_BIT\_READ | ESP\_GATT\_CHAR\_PROP\_BIT\_WRITE | ESP\_GATT\_CHAR\_PROP\_BIT\_NOTIFY;  test\_profile.chars[CHARACTERISTIC\_OFF\_ID].char\_uuid.len = ESP\_UUID\_LEN\_16;  test\_profile.chars[CHARACTERISTIC\_OFF\_ID].char\_uuid.uuid.uuid16 = GATTS\_CHAR\_UUID\_TEST\_OFF;  test\_profile.chars[CHARACTERISTIC\_OFF\_ID].perm = ESP\_GATT\_PERM\_READ | ESP\_GATT\_PERM\_WRITE;  test\_profile.chars[CHARACTERISTIC\_OFF\_ID].property = ESP\_GATT\_CHAR\_PROP\_BIT\_READ | ESP\_GATT\_CHAR\_PROP\_BIT\_WRITE | ESP\_GATT\_CHAR\_PROP\_BIT\_NOTIFY;    esp\_err\_t ret;  /\* initialize BLE and bluedroid \*/  btStart();  ret = esp\_bluedroid\_init();  if (ret) {  printf("%s init bluetooth failed\n", \_\_func\_\_);  return;  }  ret = esp\_bluedroid\_enable();  if (ret) {  printf("%s enable bluetooth failed\n", \_\_func\_\_);  return;  }  /\* set BLE name and broadcast advertising info  so that the world can see you\*/  esp\_ble\_gap\_set\_device\_name(TEST\_DEVICE\_NAME);  esp\_ble\_gap\_config\_adv\_data(&test\_adv\_data);  /\* register callbacks to handle events like register device,  sending and receiving data \*/  esp\_ble\_gatts\_register\_callback(gatts\_event\_handler);  esp\_ble\_gap\_register\_callback(gap\_event\_handler);  /\* register profiles \*/  esp\_ble\_gatts\_app\_register(CHARACTERISTIC\_ON\_ID);  }  void loop(){  } |

**3. Result**  
- I used **sudo hcitool lescan** to get the address of my BLE device. It is **30:AE:A4:02:70:76.**Then I used **sudo gatttool -b 30:AE:A4:02:70:76 -I** to establish connection to it.  
- Using **char-desc** command we will see our characteristic uuids that we set in the code, are **aa01** and **bb01** with handles are **0x002a** and **0x002c**. So we just operate on these handles with commands below. 

[](https://2.bp.blogspot.com/-FMVIt1Dn_Bw/WW-EwrGmjNI/AAAAAAAAELM/JqwanRcjGIU0wiXm5LDUMinxKoVKYdRiQCLcBGAs/s1600/esp32_ble2.png)

**Figure: Available characteristics**

[](https://4.bp.blogspot.com/-3iy6kWSLiSg/WW-DS6j3hLI/AAAAAAAAELI/LQ-ixw6LVT0S2T9Lapj5D4qo_vTSiCQZACLcBGAs/s1600/esp32_ble.png)

**Figure: write data to ESP32 BLE**