```
#include <Arduino.h>
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEScan.h>
#include <BLEAdvertisedDevice.h>
// Client Code
#include "BLEDevice.h"

// TODO: change the service UUID to the one you are using on the server side.
// The remote service we wish to connect to.
static BLEUUID serviceUUID("b2f90fcf-4d6a-46e2-8a21-87011eb8be22");
// The characteristic of the remote service we are interested in.
static BLEUUID charUUID("c142758b-8783-46d0-8bbd-231bdd9934d4");

static boolean doConnect = false;
static boolean connected = false;
static boolean doScan = false;
static BLERemoteCharacteristic* pRemoteCharacteristic;
static BLEAdvertisedDevice* myDevice;

// TODO: define new global variables for data collection
static String serverName = "";  //
static unsigned long lastNamePrint = 0; // print timer
const unsigned long namePrintInterval = 5000;

// TODO: define a new function for data aggregation

static void notifyCallback(
  BLERemoteCharacteristic* pBLERemoteCharacteristic,
  uint8_t* pData,
  size_t length,
  bool isNotify) {
    // TODO: add codes to handle the data received from the server, and call the data
aggregation function to process the data

    // TODO: change the following code to customize your own data format for printing
    Serial.print("Notify callback for characteristic ");
    Serial.print(pBLERemoteCharacteristic->getUUID().toString().c_str());
    Serial.print(" of data length ");
    Serial.println(length);
    Serial.print("data: ");
    Serial.write(pData, length);
    Serial.println();
```

```
}

class MyClientCallback : public BLEClientCallbacks {
  void onConnect(BLEClient* pclient) {
  }

  void onDisconnect(BLEClient* pclient) {
    connected = false;
    Serial.println("onDisconnect");
  }
};

bool connectToServer() {
    Serial.print("Forming a connection to ");
    Serial.println(myDevice->getAddress().toString().c_str());

    BLEClient*  pClient  = BLEDevice::createClient();
    Serial.println(" - Created client");

    pClient->setClientCallbacks(new MyClientCallback());

    // Connect to the remove BLE Server.
    pClient->connect(myDevice);  // if you pass BLEAdvertisedDevice instead of address, it will
be recognized type of peer device address (public or private)
    Serial.println(" - Connected to server");
    pClient->setMTU(517); //set client to request maximum MTU from server (default is 23
otherwise)

    // Obtain a reference to the service we are after in the remote BLE server.
    BLERemoteService* pRemoteService = pClient->getService(serviceUUID);
    if (pRemoteService == nullptr) {
      Serial.print("Failed to find our service UUID: ");
      Serial.println(serviceUUID.toString().c_str());
      pClient->disconnect();
      return false;
    }
    Serial.println(" - Found our service");

    // Obtain a reference to the characteristic in the service of the remote BLE server.
    pRemoteCharacteristic = pRemoteService->getCharacteristic(charUUID);
    if (pRemoteCharacteristic == nullptr) {
      Serial.print("Failed to find our characteristic UUID: ");
      Serial.println(charUUID.toString().c_str());
      pClient->disconnect();
```

```
    return false;
  }
  Serial.println(" - Found our characteristic");

  // Read the value of the characteristic.
  if(pRemoteCharacteristic->canRead()) {
    String value = pRemoteCharacteristic->readValue().c_str();
    Serial.print("The characteristic value was: ");
    Serial.println(value);
  }

  if(pRemoteCharacteristic->canNotify())
    pRemoteCharacteristic->registerForNotify(notifyCallback);

  connected = true;
  lastNamePrint = 0; // reset print timer
  return true;
}

/**
 * Scan for BLE servers and find the first one that advertises the service we are looking for.
 */
class MyAdvertisedDeviceCallbacks: public BLEAdvertisedDeviceCallbacks {
  /**
   * Called for each advertising BLE server.
   */
  void onResult(BLEAdvertisedDevice advertisedDevice) {
    Serial.print("BLE Advertised Device found: ");
    Serial.println(advertisedDevice.toString().c_str());

    // We have found a device, let us now see if it contains the service we are looking for.
    if (advertisedDevice.haveServiceUUID() &&
advertisedDevice.isAdvertisingService(serviceUUID)) {
      serverName = String(advertisedDevice.getName().c_str());
      Serial.print("Connecting to server: ");
      Serial.println(serverName);

      BLEDevice::getScan()->stop();
      myDevice = new BLEAdvertisedDevice(advertisedDevice);
      doConnect = true;
      doScan = true;
    } // Found our server
  } // onResult
}; // MyAdvertisedDeviceCallbacks
```

```
void setup() {
  Serial.begin(115200);
  Serial.println("Starting Arduino BLE Client application...");
  BLEDevice::init("spencer_client");

  // Retrieve a Scanner and set the callback we want to use to be informed when we
  // have detected a new device.  Specify that we want active scanning and start the
  // scan to run for 5 seconds.
  BLEScan* pBLEScan = BLEDevice::getScan();
  pBLEScan->setAdvertisedDeviceCallbacks(new MyAdvertisedDeviceCallbacks());
  pBLEScan->setInterval(1349);
  pBLEScan->setWindow(449);
  pBLEScan->setActiveScan(true);
  pBLEScan->start(5, false);
} // End of setup.

// This is the Arduino main loop function.
void loop() {
  // If the flag "doConnect" is true then we have scanned for and found the desired
  // BLE Server with which we wish to connect.  Now we connect to it.  Once we are
  // connected we set the connected flag to be true.
  if (doConnect == true) {
    if (connectToServer()) {
      Serial.println("We are now connected to the BLE Server.");
    } else {
      Serial.println("We have failed to connect to the server; there is nothin more we will do.");
    }
    doConnect = false;
  }

  // If we are connected to a peer BLE Server, update the characteristic each time we are
reached
  // with the current time since boot.
  if (connected) {
    unsigned long now = millis();
    if (now - lastNamePrint >= namePrintInterval) {
      lastNamePrint = now;
      Serial.print("Connected to server name: ");
      if (serverName.length() > 0) Serial.println(serverName);
      else Serial.println("YewenZhouESP_server");
    }

    String newValue = "Time since boot: " + String(millis()/1000);
```

```
    Serial.println("Setting new characteristic value to \"" + newValue  + "\"");

    // Set the characteristic's value to be the array of bytes that is actually a string.
    pRemoteCharacteristic->writeValue(newValue.c_str(), newValue.length());
  }else if(doScan){
    BLEDevice::getScan()->start(0);  // this is just example to start scan after disconnect, most likely there is better way to do it in arduino
  }

  delay(1000); // Delay a second between loops.
} // End of loop
```