## Bluetooth Architecture/Protocol Stack

Bluetooth is both a hardware-based radio system and a software stack that specifies the linkages between the architecture layers of the two.  The heart of this specification is the protocol stack, which is used to define how Bluetooth works.  The Bluetooth protocol stack is a set of layered programs.  Each layer in a protocol stack talks to the layer above it and to the layer below it.

Think of Bluetooth as having two well-defined layers of functionality in the stack.  These layers range from the lower level hardware-based radio system, to an upper level software stack that specifies the linkages between the layers (Figure 1).

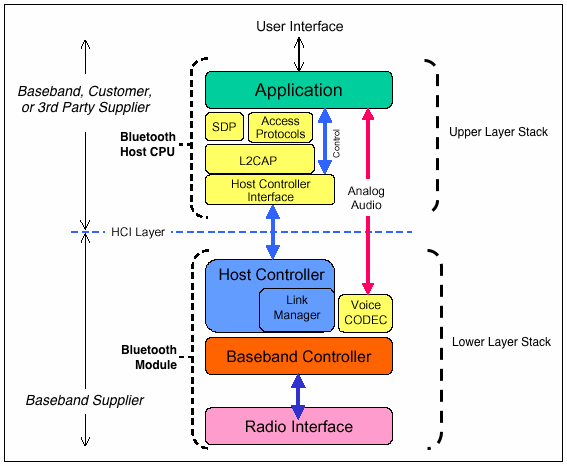
[](https://hearinghealthmatters.org/waynesworld/files/2014/01/BT-Stack.gif)

Figure 1. Bluetooth protocol stack consists of a three-layer hardware lower stack (radio, Baseband, LMP), and a three-layer software upper stack (HCI, L2CAP, and SDP). An application code then sits on top of this.

### Lower Stack Layers

The lower layers are the *basic core specifications* that describe how Bluetooth works.  The base of the Bluetooth protocol stack is the **radio layer**, or module.  The radio layer describes the physical characteristics of the transceiver.  It is responsible for modulation/demodulation of data for transmitting **or**receiving over radio frequencies in the 2.4 GHz band.  This is the physical wireless connection.  It splits the transmission band into 79 channels and performs fast [frequency hopping](https://hearinghealthmatters.org/waynesworld/2013/bluetooth-101-part-iv/) (1600 hops/sec) for security.

Above the radio layer is the **baseband** and **link controller/link manager protocol (LMP)**.  Perhaps the best way to think of these layers is that the *baseband* is responsible for properly formatting data for transmission to and from the radio.  It defines the timing, framing, packets, and flow control on the link.  The *link manager controller* translates the host controller interface (HCI) commands from the upper stack, and establishes and maintains the link.  It is responsible for managing the connection, enforcing fairness among slaves in the piconet, and provides for power management.

### Upper Stack Layers

The upper stack layers consist of *profile specifications* that focus on how to build devices that will communicate with each other, using the core technology.

The **host controller interface** (**HCI**) serves as the interface between the software part of the system and the hardware (i.e., the device driver).

The **L2CAP** (logical link control and adaptation protocol) layer is above the HCI in the upper stack.  Among other functions, it plays a central role in communication between the upper and lower layers of the Bluetooth stack.  It keeps track of where data packets come from and where they should go.  It is a required part of every Bluetooth system.

Above the L2CAP layer, the protocol stack is not as linearly ordered.  Still, the **service discovery** **protocol** (**SDP**) is important to mention because it exists independently of other higher-level protocol layers.  It provides the interface to the link controller and allows for interoperability between Bluetooth devices.

Of course, it is not required to partition the Bluetooth stack as shown in Figure 1.  Bluetooth headsets, for example, combine the module and host portions of the stack on one processor to meet self-containment and small size needs.  In such devices, the HCI may not be implemented at all unless device testing is required.

**Creating Bluetooth Android App to Control Arduino Board**

This tutorial is about creating your custom Android app to connect with an Arduino board using Bluetooth. Consequently, some basic prior knowledge of Android programming is required to follow this tutorial.

But don’t worry, if you don’t have basic knowledge on Android programming but still want to create your own Bluetooth app, then you can take the Basic Android Programming for Arduino Makers that is available in Udemy. You will learn how to create a Bluetooth app that can talk with your Arduino board from scratch and without prior knowledge in Android programming required.

The codes presented in this tutorial is the minimum codes that enable an Android phone and Arduino board to send and receive messages (that can be translated into commands) with each other through Bluetooth.

**Development environment**

To manage some expectations in case the app doesn’t work like it’s supposed to be, this is the environment I use to develop this app:

1. Samsung Galaxy S8, with Android version 9.
2. Android Studio version 3.6.3 with compatible Gradle version.
3. Minimum SDK Version: 19 (You need to select this when creating a new project using Android Studio).
4. Mac OS 10.15.4 (Windows machines also works perfectly)

**How this app works**

This app will create a Bluetooth connection with a nearby Arduino board that has been connected with the HC05 Bluetooth module. It is created to be compatible with Arduino board from this tutorial. However, it is easy to modify the codes so that it can be used together with Arduino boards with different configurations.

**Creating Bluetooth Connection on Android**

Before we dive into the coding part, I would like to describe the step by step flow to create a Bluetooth connection on Android. This is a summary of the more detailed documentation from Google.

1. Initialize the default Bluetooth adapter (device) on your Android phone.
2. Get the MAC Address from the remote device that you are connecting to. In this case, the MAC Address of HC05 Bluetooth module connected to Arduino board.
3. Create a separate thread in your code to initiate a connection using the MAC Address that we previously obtained. This thread will manage what happens if a connection is successfully established or failed to be established. It also handles if we want to close the Bluetooth connection.
4. Once a connection is successfully established, the thread will do callback for the codes that manage data exchange (transmit and receive between 2 devices). For this, we need to create another thread.
5. This thread will read incoming data transmission and parse it if necessary (or you can parse it elsewhere on the code) and transmit the message or command that is generated by the Android app.

Now, the flow above needs to be translated into codes.

**Creating Activities and Java Class**

Create a new project with the empty activity template and select the appropriate name for your app. For this app we will create 2 activities and 2 Java classes :

1. MainActivity. This is automatically created when you create a new project. This is where most of the interactions take place.
2. SelectDeviceActivity. The UI where you select the Bluetooth device that you want to connect.
3. DeviceListAdapter. A class to display a list of paired Bluetooth devices for you to connect. The list will be displayed in SelectDeviceActivity.
4. DeviceInfoModel. A class that acts as a placeholder for the remote device information.

**AndroidManifest.xml**

Once you created all the activities and classes above, your AndroidManifest.xml file will look something like this :

Please note that you should add the Bluetooth permission so that you can access your phone’s Bluetooth device.

**MainActivity Layout**

MainActivity is the main UI where you can interact with the interfaces that will connect you to a remote Bluetooth device and retrieve data form it.

The XML code for the layout above is like this:

**SelectDeviceActivity Layout**

This activity will display a list of remote Bluetooth devices that are already paired with your phone. It is shown when the “Connect” button is clicked on the MainActivity. The layout XML code for this activity is as follows:

**MainActivity Code**

Now we continue with the code. Code in MainActivity is the one creating a Bluetooth connection to a remote device. You can just copy and paste the code below to your project. Some comments have been added to the code you can understand it better.

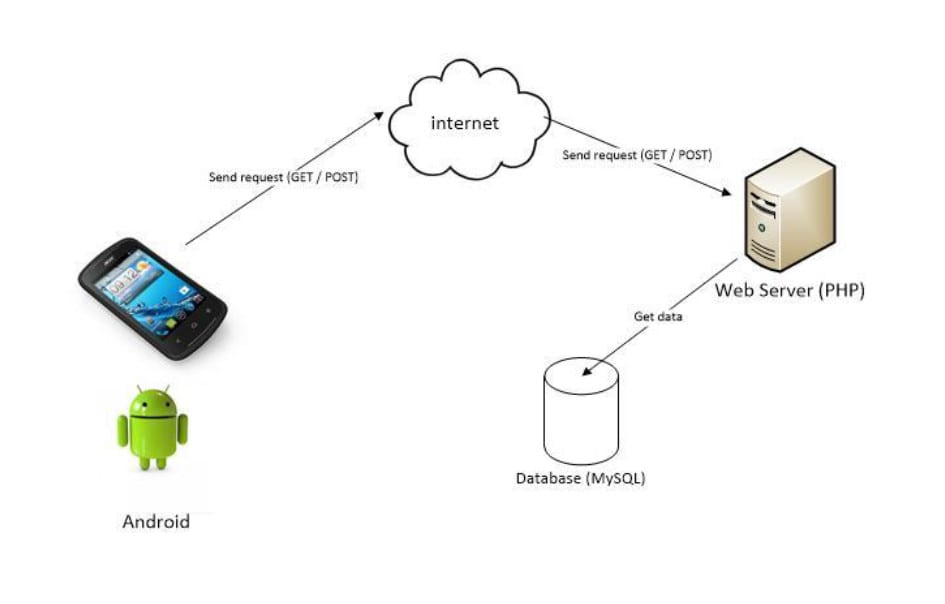
I make some minor style changes to the color resources.

SelectDeviceActivity, DeviceListAdapter and DeviceInfoModel

This activity works with DeviceListAdapter class and DeviceInfoModel class to display the list of paired devices.

Once you build the whole project, you need to install the app to your actual device to be able to use the Bluetooth function.

**Communication Between Cloud And Android Device using PHP**



Cloud

Pushing data from android application to server is very important feature for every android application. That makes our application dynamic. This tutorial is helpful for beginners who wish to understand dynamic content management in android development. In this tutorial we are going to do some basic programming and learn to insert data from application to MySQL database Android PHP .

## Contents in this project Android PHP Insert data from Application to MySQL Database.

1. Watch the live demo video of Android PHP Insert data to server.
2. Starting a new project in Android Studio.
3. Creating a database on your online server.
4. Make a table in your database.
5. Upload the PHP script on your server.
6. Testing the PHP script.
7. Adding internet permission in your project’s AndroidManifest.xml file.
8. Add Org.Apache.Http.Legacy in your project.
9. Adding two EditText and one button in your layout file.
10. Declare EditText and button objects in your Activity & Assign their ID’s to them.
11. Define method to get data from EditText into String variables.
12. Defining AsyncTask method class in your Activity.

## Upload the PHP script on your server :

After finishing table creation process just upload the below two PHP scripts on your server using file manager. There are two different type of files present here first one is **DatabaseConfig.php** file and second is **get\_data.php** file. Please change the details of your server in DatabaseConfig.php file.

**Code for DatabaseConfig.php file.**

<?php

//Define your host here.

$HostName = "mysql.hostinger.in";

//Define your database username here.

$HostUser = "u288012116\_json";

//Define your database password here.

$HostPass = "N1c45hlf";

//Define your database name here.

$DatabaseName = "u288012116\_json";

?>

**Code for get\_data.php file.**

<?php

include 'DatabaseConfig.php' ;

$con = mysqli\_connect($HostName,$HostUser,$HostPass,$DatabaseName);

$name = $\_POST['name'];

$email = $\_POST['email'];

$Sql\_Query = "insert into GetDataTable (name,email) values ('$name','$email')";

if(mysqli\_query($con,$Sql\_Query)){

echo 'Data Submit Successfully';

}

else{

echo 'Try Again';

}

mysqli\_close($con);

?>

## Testing the PHP script :

After done uploading procedure just open the get\_data.php file URL in your web browser and you can see that its showing us the **Data Submit Successfully** message and when you open the PhpMyAdmin control panel there is a blank value inserted in your table. This value is blank because there is no such data present for insertion.

## Adding internet permission in your project’s AndroidManifest.xml file :

Next step is to add internet permission in your AndroidManifest.xml file .

<uses-permission android:name="android.permission.INTERNET"/>

Code for Arduino Hardware

#include <OneWire.h>

#include <LiquidCrystal.h>

#include <Wire.h>

#include "MAX30100\_PulseOximeter.h"

#define REPORTING\_PERIOD\_MS 1000

// PulseOximeter is the higher level interface to the sensor

// it offers:

// \* beat detection reporting

// \* heart rate calculation

// \* SpO2 (oxidation level) calculation

PulseOximeter pox;

OneWire ds(A0); // on pin 10 (a 4.7K resistor is necessary)

LiquidCrystal lcd(13, 12, 11, 10, 9, 8);

uint32\_t tsLastReport = 0;

// Callback (registered below) fired when a pulse is detected

void onBeatDetected()

{

//Serial.println("B:1");

}

byte i;

byte present = 0;

byte type\_s;

byte data[12];

byte addr[8];

float celsius, fahrenheit;

int sensorValue;

int count=0,fl;

unsigned long time1=0; // store the initial time

unsigned long time2; // store the current time

int count1;

int hb1,hb2;

void setup()

{

Serial.begin(9600);

lcd.begin(16,2);

pinMode(A0, INPUT\_PULLUP);

pinMode(2,OUTPUT);

digitalWrite(2,0);

lcd.clear();

lcd.setCursor(0,0);

lcd.print("Temp:");

if (!pox.begin()) {

Serial.println("ERROR: Failed to initialize pulse oximeter");

for(;;);

}

pox.setOnBeatDetectedCallback(onBeatDetected);

if ( !ds.search(addr)) {

ds.reset\_search();

}

}

int pox2,hb;

int cel;

void loop()

{

// Make sure to call update as fast as possible

pox.update();

// Asynchronously dump heart rate and oxidation levels to the serial

// For both, a value of 0 means "invalid"

if (millis() - tsLastReport > REPORTING\_PERIOD\_MS) {

//pox2=pox.getHeartRate();

pox2=pox.getSpO2();

hb=count1;

lcd.clear();

lcd.setCursor(8,1);

lcd.print("HB:");

lcd.print(hb);

lcd.setCursor(0,1);

lcd.print("PX:");

lcd.print(pox2);

tsLastReport = millis();

Serial.print("\*");

Serial.print("T=");cel=3;

Serial.write((cel/100)+0x30);

Serial.write(((cel%100)/10)+0x30);

Serial.write((cel%10)+0x30);

Serial.print("\_HB=");hb=1;

Serial.write((hb/100)+0x30);

Serial.write(((hb%100)/10)+0x30);

Serial.write((hb%10)+0x30);

Serial.print("\_PX=");pox2=2;

Serial.write((pox2/100)+0x30);

Serial.write(((pox2%100)/10)+0x30);

Serial.write((pox2%10)+0x30);

Serial.print("#");

}

if (OneWire::crc8(addr, 7) != addr[7]) {

// Serial.println("CRC is not valid!");

return;

}

// Serial.println();

ds.reset();

ds.select(addr);

ds.write(0x44, 1); // start conversion, with parasite power on at the end

// we might do a ds.depower() here, but the reset will take care of it.

present = ds.reset();

ds.select(addr);

ds.write(0xBE); // Read Scratchpad

for ( i = 0; i < 9; i++) { // we need 9 bytes

data[i] = ds.read();

}

int16\_t raw = (data[1] << 8) | data[0];

if (type\_s) {

raw = raw << 3; // 9 bit resolution default

if (data[7] == 0x10) {

raw = (raw & 0xFFF0) + 12 - data[6];

}

} else {

byte cfg = (data[4] & 0x60);

if (cfg == 0x00) raw = raw & ~7; // 9 bit resolution, 93.75 ms

else if (cfg == 0x20) raw = raw & ~3; // 10 bit res, 187.5 ms

else if (cfg == 0x40) raw = raw & ~1; // 11 bit res, 375 ms

}

celsius = (float)raw / 16.0;

fahrenheit = celsius \* 1.8 + 32.0;

cel=(int)celsius;

lcd.setCursor(0, 0);

lcd.print("Temp=");

lcd.print(cel);

lcd.print(" ");

hb1++;

// read the value from the sensor:

if(count==0)

{

time1=millis();

}

time2=millis();

sensorValue = analogRead(A1);

if(sensorValue<356)

{

if(fl==0)

{

increment();

fl=1;

}

}

else

{

fl=0;

}

if(hb1>60)

{

if(count>8)

count1=66+(count%10);

else

count1=0;

hb1=0;

count=0;

}

lcd.setCursor(12,0);

lcd.print(count);

}

void increment()

{

count++;

}

void counter()

{

if(count>2)

{

count1=count\*2;

}

else

{

count1=0;

}

time1=0;

time2=0;

count=0;

delay(200);

}

IOT Link (github)

http://iotclouddata.com/project20/037/iot20view.php