TUTORIAL FOR INITIALIZING BLUETOOTH COMMUNICATION BETWEEN ANDROID AND ARDUINO

by :-Lohit Jain

some pre requirements

*First of all download arduino software from www.arduino.cc

*download software serial library from github.com (type on google).

*download four tier software development system for android app refer the following link for guidance - www.niktechs.wordpress.com.

1.

the Arduino code:

```
#include <SoftwareSerial.h>

int bluetoothTx = 2;
int bluetoothRx = 3;

SoftwareSerial bluetooth(bluetoothTx, bluetoothRx);

void setup()
{
  //Setup usb serial connection to computer
  Serial.begin(9600);

  //Setup Bluetooth serial connection to android
  bluetooth.begin(115200);
  bluetooth.print("$$$");
  delay(100);
  bluetooth.println("U,9600,N");
  bluetooth.begin(9600);
}

void loop()
{
  //Read from bluetooth and write to usb serial
  if(bluetooth.available())
  {
    char toSend = (char)bluetooth.read();
    Serial.print(toSend);
```

```
  }

  //Read from usb serial to bluetooth
  if(Serial.available())
  {
    char toSend = (char)Serial.read();
    bluetooth.print(toSend);
  }
}
```

For connection see tutorial on aubtm-20 and arduino interfacing with lcd.

That takes care of the Arduino side of things. The Android bits are little bit tougher.

2.

The Android project we are going to write is going to have to do a few things:

1. Open a bluetooth connection
2. Send data
3. Listen for incoming data
4. Close the connection

But before we can do any of that we need to take care of a few pesky little details. First, we need to pair the Arduino and Android devices. You can do this from the Android device in the standard way by opening your application drawer and going to **Settings**. From there open **Wireless and network.** Then **Bluetooth settings.** From here just scan for devices and pair it like normal. If it asks for a pin it's 8888. You shouldn't need to do anything special from the Arduino side other than to have it turned on. Next we need to tell Android that we will be working with bluetooth by adding this element to the **<manifest>** tag inside the **AndroidManifest.xml** file:
<uses-permission android:name="android.permission.BLUETOOTH" />
Alright, now that we have that stuff out of the way we can get on with opening the bluetooth connection. To get started we need a **BluetoothAdapter** reference from Android. We can get that by calling

```
BluetoothAdapter.getDefaultAdapter();
```

The return value of this will be null if the device does not have bluetooth capabilities. With the adapter you can check to see if bluetooth is enabled on the device and request that it be turned on if its not with this code:

```
if(!mBluetoothAdapter.isEnabled())
{
   Intent enableBluetooth = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
   startActivityForResult(enableBluetooth, 0);
}
```

3.

Now that we have the bluetooth adapter and know that its turned on we can get a reference to our Arduino's bluetooth device with this code:

```
Set pairedDevices = mBluetoothAdapter.getBondedDevices();
if(pairedDevices.size() > 0)
{
    for(BluetoothDevice device : pairedDevices)
    {
        if(device.getName().equals("AUBTM-20")) //Note, you will
need to change this to match the name of your device
        {
            mmDevice = device;
            break;
        }
    }
}
```

4.

Armed with the bluetooth device reference we can now connect to it using this code:

```
UUID uuid = UUID.fromString("00001101-0000-1000-8000-
00805f9b34fb"); //Standard SerialPortService ID
mmSocket = mmDevice.createRfcommSocketToServiceRecord(uuid);
mmSocket.connect();
mmOutputStream = mmSocket.getOutputStream();
mmInputStream = mmSocket.getInputStream();
```

This opens the connection and gets input and output streams for us to work with. You may have noticed that huge ugly UUID. Apparently there are standard UUID's for various devices and Serial Port's use that one.

If everything has gone as expected, at this point you should be able to connect to your Arduino from your Android device. The connection takes a few seconds to establish so be patient. Once the connection is established the red blinking light on the bluetooth chip should stop and remain off.

Closing the connection is simply a matter of calling close() on the input and output streams as well as the socket

Now we are going to have a little fun and actually send the Arduino some data from the Android device. Make yourself a text box (also known as an EditText in weird Android speak) as well as a send button. Wire up the send button's click event and add this code:

```
String msg = myTextbox.getText().toString();
msg += "\n";
mmOutputStream.write(msg.getBytes());
```

With this we have one way communication established! Make sure the Arduino is plugged in to the computer via the USB cable and open the Serial Monitor from within the Arduino IDE. Open the application on your Android device and open the connection. Now whatever your type in the textbox will be sent to the Arduino over air and magically show up in the serial monitor window.

Sending data is trivial. Sadly, listening for data is not. Since data could be written to the input stream at any time we need a separate thread to watch it so that we don't block the main ui thread and cause Android to think that we have crashed. Also the data written on the input stream is relatively arbitrary so there isn't a simple way to just be notified of when a line of text shows up. Instead lots of short little packets of data will show up individually one at a time until the entire message is received. Because of this we are going to need to buffer the data in an array until enough of the data shows up that we can tell what to do. The code for doing this and the threading is a little be long winded but it is nothing too complicated.

First we will tackle the problem of getting a worker thread going. Here is the basic code for that:

```
            final Handler handler = new Handler();
            workerThread = new Thread(new Runnable()
            {
                public void run()
                {
                   while(!Thread.currentThread().isInterrupted() && !
    stopWorker)
                   {
                        //Do work
                   }
                }
            });
            workerThread.start();
```

The first line there declares a Handler which we can use to update the UI, but more on that later. This code will start a new thread. As long as the run() method is running the thread will stay alive. Once the run() method completes and returns the thread will die. In our case, it is stuck in a while loop that will keep it alive until our boolean **stopWorker** flag is set to true, or the thread is interrupted. Next lets talk about actually reading data.

The input stream provides an **.available()** method which returns us how many (if any) bytes of data are waiting to be read. Given that number we can make a temporary byte array and read the bytes into it like so:

```
                            int bytesAvailable = mmInputStream.available();
                            if(bytesAvailable > 0)
                            {
                                byte[] packetBytes = new
    byte[bytesAvailable];
                                mmInputStream.read(packetBytes);
                            }
```

This gives us some bytes to work with, but unfortunately this is rather unlikely to be all of the bytes we need (or who know its might be all of them plus some more from the next command). So now we have do that pesky buffering thing I was telling you about earlier. The read buffer is just byte array that we can store incoming bytes in until we have them all. Since the size of message is going to vary we need

to allocate enough space in the buffer to account for the longest message we might receive. For our purposes we are allocating 1024 spaces, but the number will need to be tailored to your specific needs. Alright, back to the code. Once we have packet bytes we need to add them to the read buffer one at a time until we run in to the end of line delimiter character, in our case we are using a newline character (Ascii code 10)

```
                            for(int i=0;i<bytesAvailable;i++)
                            {
                                byte b = packetBytes[i];
                                if(b == delimiter)
                                {
 byte[] encodedBytes = new byte[readBufferPosition];
System.arraycopy(readBuffer, 0, encodedBytes, 0, encodedBytes.length);
        final String data = new   String(encodedBytes, "US-ASCII");
                                    readBufferPosition = 0;

                    //The variable data now contains our full command
                                }
                                else
                                {
                                    readBuffer[readBufferPosition++] = b;
                                }
                            }
```

At this point we now have the full command stored in the string variable **data** and we can act on it however we want. For this simple example we just want to take the string display it in on a on screen label. Sticking the string into the label would be pretty simple except that this code is operating under a worker thread and only the main UI thread can access the UI elements. This is where that Handler variable is going to come in. The handler will allow us to schedule a bit of code to be executed by main UI thread. Think of the handler as delivery boy who will take the code you wanted executed and deliver it to main UI thread so that it can execute the code for you. Here is how you can do that:

```
handler.post(new Runnable()
{
    public void run()
    {
        myLabel.setText(data);
    }
});
```

And that is it! We now have two way communication between the Arduino and an Android device! Plugin the Arduino and open the serial monitor. Run your Android application and open the bluetooth connection. Now what type in the textbox on the Android device will show up in the serial monitor window for the Arduino, and what you type in the serial monitor window will show up on the Android device.