Google Maps and Intel Edison Sensor Kit

Bluetooth Integration

Junqi Ma and Shipei Tian

Part1.

Homework 5:
On Edison Arduino, we fixed the bugs on the demo code. As we don't have encoder at the first time, we modify our code to make the button on pin 0 (UART) works on pin 3 to achieve the function of encoder (one direction browser menu only).

Video demo: https://youtu.be/_6oPjqAIc_w

Edison Arduino sometimes have issue on timer interrupt, it cannot jump to the interrupt program code. Since the Wi-Fi connection is been setup and the web page data monitoring is done in the timer interrupt code, the connection cannot be established. This occurs especially after you have setup the Edison Linux system and have something running there. Our solution is to move the timer interrupt code into the routine while loop. Also, we add a print on serial port for the Wi-Fi connection status after connection to confirm the connection result (3 means success, 1 means failed). Note the time delay between each trial will affect the connection, a too fast connection retry will cause failed to establish the connection.

There is a documentation of the Grove demo, can be found at https://github.com/Seeed-Studio/Grove_Indoor_Environment_Demo

Homework 6:
For this homework we used an approach that is similar to the class tutorial, but better documented, which can be found at https://communities.intel.com/docs/DOC-100754

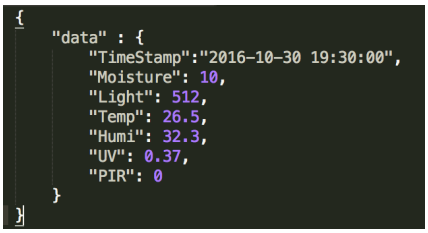
However, the there is an issue with Bluetooth service, if you follow the step above, you will end up getting a result that you Bluetooth drop connection once you connect to Edison Bluetooth in Bluetooth SPP app (I have already mentioned that in the E-mail). To solve this, you have to reinstall Bluetooth stack by:

```
opkg update
```

```
opkg install --force-reinstall bluez5
```

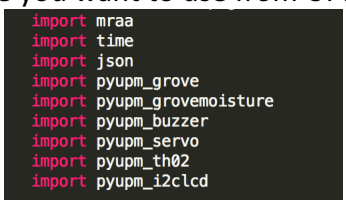
On Edison we are directly using python code to get sensor data and send to Android in JSON format. You can run a local sensor demo by running sensor.py in Edison Code. Sensor.py shows an approach of how to read devices and toggle output devices, including all devices you see in the Grove demo. To achieve, this we are using MRAA and UPM, they can be found at:
http://iotdk.intel.com/docs/master/mraa/python/
http://iotdk.intel.com/docs/master/upm/python/

The data from sensor will be convert to JSON format which is very common in APIs.

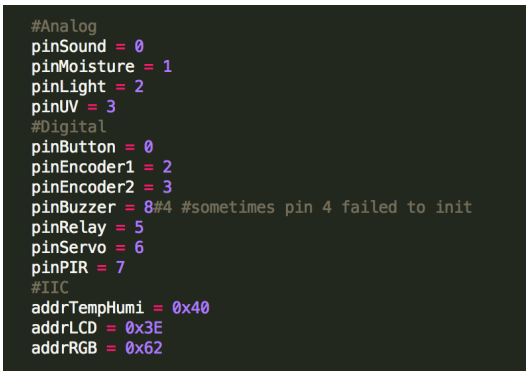


The reading from sensors and writing to devices are achieved in the flowing steps, note that Arduino part of Edison can affect initialing of GPIO pins, it is better to download a simple project to Edison Arduino before you start:

1. Import MRAA and the module you want to use from UPM.



2. Define device pins and address



Note the pin setup is all the same as Grove demo, which means you don't have to change the connection of devices when you run this code. The only different is we change the pin of buzzer from pin 4 to pin 8 since pin 4 sometimes may failed to initialize.

3. Create device instances

```
#Devices
moisture = pyupm_grovemoisture.GroveMoisture(pinMoisture)
light = pyupm_grove.GroveLight(pinLight)
UV = mraa.Aio(pinUV)

button = pyupm_grove.GroveButton(pinButton)
Encoder1 = pyupm_grove.GroveButton(pinEncoder1) #treat encoder as two buttons
Encoder2 = pyupm_grove.GroveButton(pinEncoder2)
buzzer = pyupm_buzzer.Buzzer(pinEncoder2)
relay = pyupm_grove.GroveRelay(pinRelay)
servo = pyupm_servo.Servo(pinServo)
PIR = mraa.Gpio(pinPIR)     #no PIR lib in upm, use mraa

tempHumi = pyupm_th02.TH02()
LCD = pyupm_i2clcd.Jhd1313m1(0, addrLCD, addrRGB)
```

moisture, button, buzzer, relay, servo, TH02(the temperature and humidity sensor), LCD can be found at UPM library, while UV censor and PIR sensor is not available in UPM library so we use GPIO function from MRAA instead. There is no encoder implementation in UPM, however an encoder can be treat as two buttons since they are using two pins to indicate increase/decrease in angle.

4. Main loop

```
def main():
  PIR.dir(mraa.DIR_IN)

  LCD.setCursor(0,0)
  LCD.setColor(53, 39, 249)
  LCD.backlightOn()
  LCD.write(str("Hello World"))
  i = 0
  j = 0
  while(1):
    localtime = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
    print("Time: "+localtime)
    print("light: "+(str)(light.value())+" lux")
    print("light_raw: "+(str)(light.raw_value()))
    print("moisture: "+(str)(moisture.value()))
    print("UV: "+(str)(UV.readFloat()))
    print("temp: "+(str)(tempHumi.getTemperature())+" C")
    print("humi: "+(str)(tempHumi.getHumidity())+" %")
    print("PIR: "+(str)(PIR.read()))
    print("button: "+(str)(button.value()))
    print(" ")
    data = {
    "TimeStamp": time.strftime("%Y-%m-%d %H:%M:%S", time.localtime()),
    "Moistrue": moisture.value(),
    "Light": light.value(),
    "Temp": tempHumi.getTemperature(),
    "Humi": tempHumi.getHumidity(),
    "UV": UV.readFloat(),
    "PIR": PIR.read()}
    json_str = json.dumps(data)
    print(json_str)
    print(" ")
    # if relay.isOn() == True:
    #   relay.off()
    # else:
    #   relay.on()
    # buzzer.playSound(i, 100)
    # i = i+1
    # if i>9:
    #   i=0;
    # servo.setAngle(j)
    # j = j + 10
    # if j >= 160:
    #   j = 0
    time.sleep(0.5)
```

The program will first initialize PIR digital IO pin and the LCD then goes into the while loop. It will read and print the reading value from sensors together with timestamp every half second. Addition to that, you can see the JSON formatted sensor data that will be send to Android. Note that for light intensity, we are using converted value in lux as instead of the raw sensor read (which is used in the Grove demo).

By default, time synchronize is enabled on Edison, however, the time zone on have been set to Universal, we have to change that to New York time so that we can get the right time value. Changing time zone can refer to http://qiita.com/CLCL/items/e991e23f4bdbca5ff28b

In our Bluetooth approach, we encapsulate the initialization, reading sensor data and get JSON formatted data into functions. In the Bluetooth connection program, we combined Bluetooth function from SPP-loopback with our sensor part. When receiving a string “start”, Edison will reply a JSON string with the current sensor data, also print out the message it received and sensor value in the terminal.

```
def init():
    PIR.dir(mraa.DIR_IN)
    LCD.setCursor(0,0)
    LCD.setColor(53, 39, 249)
    LCD.backlightOn()
    LCD.autoscrollOn()
    LCD.write(str("Running BT"))
    LCD.setCursor(1,0)
    LCD.write(str("Socket Server"))

def printSensor():
    localtime = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
    print("Time: "+localtime)
    print("light: "+(str)(light.value())+" lux")
    print("light_raw: "+(str)(light.raw_value()))
    print("moisture: "+(str)(moisture.value()))
    print("UV: "+(str)(UV.readFloat()))
    print("temp: "+(str)(tempHumi.getTemperature())+" C")
    print("humi: "+(str)(tempHumi.getHumidity())+" %")
    print("PIR: "+(str)(PIR.read()))
    print("button: "+(str)(button.value()))
    print(" ")

def getJSON():
    data = {
        "data": {
            "TimeStamp": time.strftime("%Y-%m-%d %H:%M:%S", time.localtime()),
            "Moisture": moisture.value(),
            "Light": light.value(),
            "Temp": tempHumi.getTemperature(),
            "Humi": tempHumi.getHumidity(),
            "UV": UV.readFloat(),
            "PIR": PIR.read()
        }
    }
    json_str = json.dumps(data)
    return json_str
```

```
def NewConnection(self, path, fd, properties):
    self.fd = fd.take()
    print("NewConnection(%s, %d)" % (path, self.fd))


    server_sock = socket.fromfd(self.fd, socket.AF_UNIX, socket.SOCK_STREAM)
    server_sock.setblocking(1)
    server_sock.send("Send \"start\" to recive current sensor data in JSON format")

    try:
        while True:
            data = server_sock.recv(1024)
            print("Got: %s" % data)
            if data == "start":
                print("Sending sensor data")
                printSensor()
                server_sock.send("%s" % getJSON())
    except IOError:
        pass

    server_sock.close()
    print("all done")
```

The Bluetooth socket server on Edison can be run by:

```
./connect.sh
./BTsensor.py –C 22
```

Sometimes, commands in .sh cannot been execute correctly, you can run it by:

```
rfkill list
rfkill unblock bluetooth
bluetoothctl
agent DisplayYesNo
default-agent
scan on
discoverable on
connect 1C:56:FE:B9:B3:8D
quit
./BTsensor.py –C 22
```

Running this in a screen terminal is recommended, in this way you can keep it running in the background. Since it supports multi time connect and disconnect.

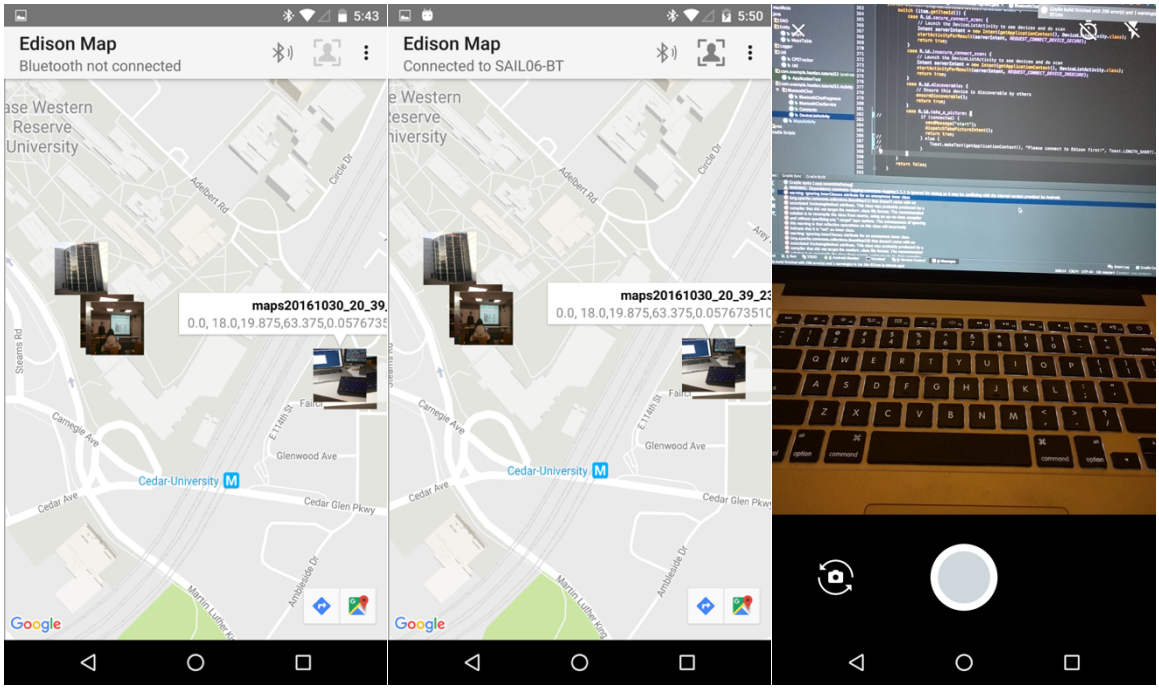
The data transmission is shown in the following demo, the write to CSV part is implemented in part 2, using the same CSV write and read function as we written for lab01.

Video demo: https://youtu.be/Mx2bMPr7CfI

Part2.

Homework1

We relocate the “take a picture” button to the menu bar. The button will be grey when the device is not connected to Edison board, user can’t click it. Otherwise, if mobile phone is connected to Edison board, the button will turn black allowing user to take a picture.



In order to achieve that, we will judge if mobile phone is connected to Edison by using a Boolean variable called **connected.** It will control the color of taking a picture icon and if jumping to camera activity. The screenshot shows the code.

```
case BluetoothChatService.STATE_CONNECTED:
    setStatus("Connected to ${device_name}");
    connected = true;
    if (shootItem != null)
        shootItem.setIcon(R.drawable.selfie_bk);
      mConversationArrayAdapter.clear();
    break;
```

```
case BluetoothChatService.STATE_NONE:
    setStatus("Bluetooth not connected");
    connected = false;
    if (shootItem != null)
        shootItem.setIcon(R.drawable.selfie_bk_ban);
    break;
```

```
case R.id.take_a_picture: {
    if (connected) {
        sendMessage("start");
        dispatchTakePictureIntent();
        return true;
    } else {
        Toast.makeText(getApplicationContext(), "Please connect to Edison first!",
    }
}
```

Taking picture is done the method called **dispatchTakePictureIntent()**. It will first check the directory that stores photos exists or not, if not it will create one. **imageUri** is the storage path for selfies, we use **"takenTime"** as timestamp to distinguish each photo. **startActivityForResult()** is called to jump to taking photo activity. When photo is taken, it will jump back to ImageActivity and store the photo in tis directory: **/sdcard/DCIM/Maps** and named with "mapsyyyyMMdd_HH_mm_ss".

```
Uri imageUri;
String takenTime;

private void dispatchTakePictureIntent() {

    Intent takePictureIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    // Ensure that there's a camera activity to handle the intent
    if (!checkAndCreateDirectory(photos_dir)) {
        Toast.makeText(getApplicationContext(), "Fail to create photo folder", Toast.LENGTH_SHORT).show();
        return;
    }
    takenTime = photo_timeformat.format(new Date());
    imageUri = Uri.fromFile(new File(photos_dir, photos_prefix + takenTime + ".jpg"));
    takePictureIntent.putExtra(android.provider.MediaStore.EXTRA_OUTPUT, imageUri);
    Log.i("photo_dir:", imageUri.toString());
    startActivityForResult(takePictureIntent, REQUEST_TAKE_PHOTO);
}

private boolean checkAndCreateDirectory(String Dir) {
    File imageFolder = new File(Dir);
    return imageFolder.exists() || imageFolder.mkdirs();
}
```

In **onActivityResult()**, we will get location information through gps class which we will discuss later.

```
case REQUEST_TAKE_PHOTO:
    if (resultCode == Activity.RESULT_OK) {
        // Check if GPS enabled
        sendMessage("start");
        double latitude = 41.502825;
        double longitude = -81.606651;

        if (gps.canGetLocation()) {

            Location location = gps.getLocation();
            latitude = gps.getLatitude();
            longitude = gps.getLongitude();
        } else {
            // Can't get location.
            // GPS or network is not enabled.
            // Ask user to enable GPS/network in settings.
            gps.showSettingsAlert();
        }
```

Then we will create a Maps() instance to put the information into this class.

```
Maps maps = new Maps();
maps.setPhoto(imageUri.toString());
maps.setTimestamp(takenTime);
maps.setLatitude(latitude);
maps.setLongitude(longitude);
```

Then we will have a dead loop to wait for the data transforming from Edison, once Android receives the data, the received will be set true, so it will jump out of the loop. We use JSON as data format, after parsing, we put the remaining data into Maps and write to CSV file. In the end, we will call **setUpMapIfNeeded()** to update the markers on the map.

```
//waiting to receive data
while (received == false) {
    try {
        Thread.sleep(500);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
//once received message, reset it to false for next message.
received = false;

try {
    JSONObject jsonRootObject = new JSONObject(dataFromEdison);
    JSONObject jsonObject = jsonRootObject.optJSONObject("data");
    Double bb = jsonObject.optDouble("Moisture");
    maps.setMoisture(bb);
    maps.setLight(jsonObject.optDouble("Light"));
    maps.setTemp(jsonObject.optDouble("Temp"));
    maps.setHumi(jsonObject.optDouble("Humi"));
    maps.setUV(jsonObject.optDouble("UV"));
    maps.setPIR(jsonObject.optInt("PIR"));
} catch (JSONException e) {
    e.printStackTrace();
}

mapsTable.add(maps);
mapsDao.writeToCSV(mapsTable, csv_dir);


List<String[]> image_list = readImages();
setUpMapIfNeeded(image_list);
```

In **setUpMap(),** the **image_list()** stores all the photo information, I use **FileInputStream()** to read the photo and convert it into bitmap with 200 height and 200 width. Finally call **addMarker()** to put all the photo related information on the map.

```
private void setUpMap(List<String[]> image_list) {
    if (image_list != null && image_list.size() > 0)
        for (String[] image : image_list) {
            try {
                FileInputStream in = new FileInputStream(image[0]);
                BufferedInputStream buf = new BufferedInputStream(in);
                byte[] bMapArray = new byte[buf.available()];
                buf.read(bMapArray);
                Bitmap bitmap = BitmapFactory.decodeByteArray(bMapArray, 0, bMapArray.length);
                bitmap = bitmap.createScaledBitmap(bitmap, 200, 200, false);
             Bitmap bitmap = Bitmap.createBitmap(image[0])
                BitmapDescriptor icon = BitmapDescriptorFactory.fromBitmap(bitmap);
                mMap.addMarker(new MarkerOptions().position(new LatLng(Double.parseDouble(imag
            } catch (FileNotFoundException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

}
```

We create a new class called **GPSTracker** to get the location information, first we need to get locationManager and check if GPS and network is enabled.

```
locationManager = (LocationManager) activity.getSystemService(LOCATION_SERVICE);

// Getting GPS status
isGPSEnabled = locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER);

// Getting network status
isNetworkEnabled = locationManager.isProviderEnabled(LocationManager.NETWORK_PROVIDER);
```

Then we can get the latitude and longitude according to the following code.

```
if (isGPSEnabled) {
    if (location == null) {
        locationManager.requestLocationUpdates(
                LocationManager.GPS_PROVIDER,
                MIN_TIME_BW_UPDATES,
                MIN_DISTANCE_CHANGE_FOR_UPDATES, this);
        Log.d("GPS Enabled", "GPS Enabled");
        if (locationManager != null) {
            location = locationManager
                    .getLastKnownLocation(LocationManager.GPS_PROVIDER);
            if (location != null) {
                latitude = location.getLatitude();
                longitude = location.getLongitude();
            }
        }
    }
}
```

Homework2:

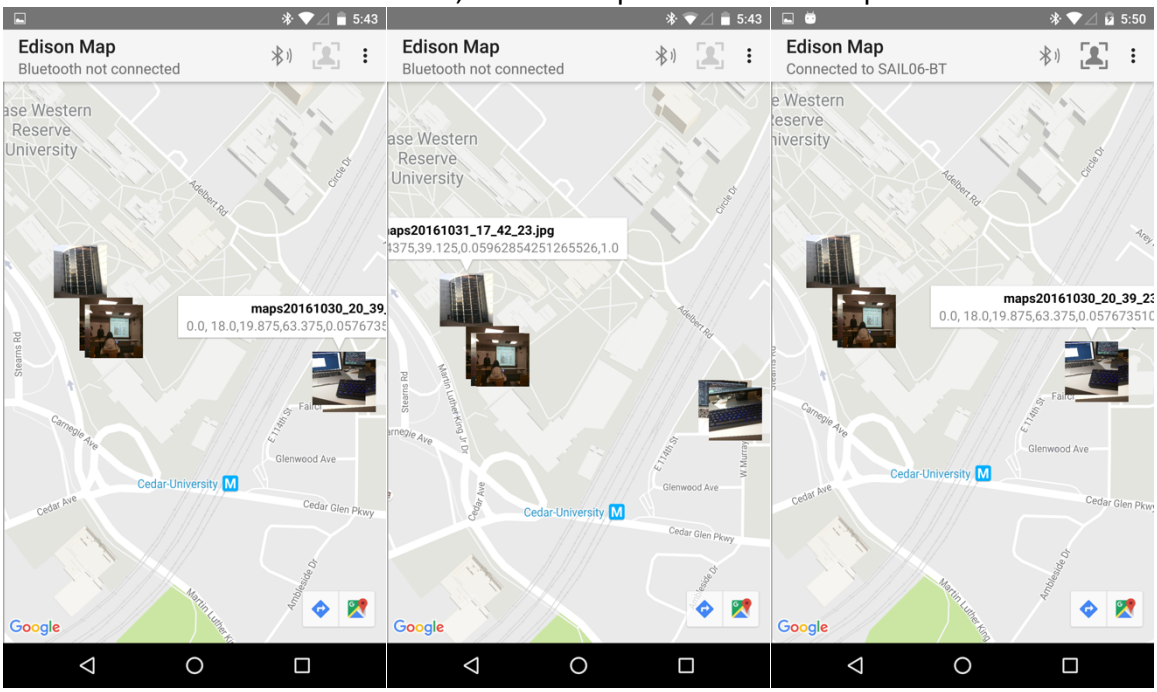
We save the CSV in the following location, **/sdcard/DCIM/Maps/maps.csv**, CSV has the following attributes: **photo, timestamp, latitude, longitude, moisture, light, temp, humi, UV, PIR.**

The file looks like this.

	A	B	C	D	E	F	G	H	I	J
1	photo	timestamp	latitude	longitude	moistrue	light	temp	humi	UV	PIR
2	maps20161030_20_36_45.jpg	20161030_20_36_45	41.5008342	-81.603292	0	18	19.8225	64.525	0.05865341	0
3	maps20161030_20_39_23.jpg	20161030_20_39_23	41.5009243	-81.603463	0	18	19.875	63.375	0.05767351	0
4	maps20161030_20_45_29.jpg	20161030_20_45_29	41.5009243	-81.603463	0	18	19.8125	64.625	0.05962854	0
5	maps20161030_20_56_20.jpg	20161030_20_56_20	41.5009243	-81.603463	0	18	20.0625	65.875	0.05865103	0
6	maps20161031_13_55_54.jpg	20161031_13_55_54	41.5015945	-81.607317	0	6	21.46875	35.75	0.05767351	1
7	maps20161031_14_02_45.jpg	20161031_14_02_45	41.5015043	-81.607146	0	6	21.40625	35.875	0.05767351	0
8	maps20161031_14_04_01.jpg	20161031_14_04_01	41.5015106	-81.607232	0	12	21.5	36.6875	0.05865103	0
9	maps20161031_17_41_01.jpg	20161031_17_41_01	41.5022465	-81.607746	0	17	21.4375	39.0625	0.05962854	1
10	maps20161031_17_41_49.jpg	20161031_17_41_49	41.5022465	-81.607746	0	17	21.40625	39.0625	0.06060606	1
11	maps20161031_17_42_23.jpg	20161031_17_42_23	41.5022465	-81.607746	0	18	21.4375	39.125	0.05962854	1
12										

Homework3:

This is the screenshot of homework3, we have explained how we implemented this.



Part3.

We first need to get a **bluetoothAdapter**

```
// Get local Bluetooth adapter
mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

// If the adapter is null, then Bluetooth is not supported
if (mBluetoothAdapter == null) {
    Toast.makeText(getApplicationContext(), "Bluetooth is not available", Toast.LENGTH_LONG).show();
}
```

We also need to override **onStart()** and **onDestroy()**, to start and stop bluebooth service.

```
@Override
public void onStart() {
    super.onStart();
    // If BT is not on, request that it be enabled.
    // setupChat() will then be called during onActivityResult
    if (!mBluetoothAdapter.isEnabled()) {
        Intent enableIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
        startActivityForResult(enableIntent, REQUEST_ENABLE_BT);
        // Otherwise, setup the chat session
    } else if (mChatService == null) {
        setupChat();
          mChatService = new BluetoothChatService(getApplicationContext(), mHandle
    }
}

@Override
public void onDestroy() {
    super.onDestroy();
    if (mChatService != null) {
        mChatService.stop();
    }
}
```

In **setupChar()**, we register the BlurtoothChatService(). Then we can use Bluetooth successfully.

```
private void setupChat() {

    // Initialize the BluetoothChatService to perform bluetooth connections
    mChatService = new BluetoothChatService(getApplicationContext(), mHandler);

    // Initialize the buffer for outgoing messages
    mOutStringBuffer = new StringBuffer("");

}
```

We use a new activity called **DeviceListActivity()** to list all the connectable devices, if user click the connect button on menu bar, it will call **DeviceListActivity().**

```java
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.secure_connect_scan: {
            // Launch the DeviceListActivity to see devices and do scan
            Intent serverIntent = new Intent(getApplicationContext(), DeviceListActivity.class);
            startActivityForResult(serverIntent, REQUEST_CONNECT_DEVICE_SECURE);
            return true;
        }
```

Once the **MapsActivity()** receive the data from **DeviceListActivity()**, **connectDevice()** is called to complete the connection.

```java
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    switch (requestCode) {
        case REQUEST_CONNECT_DEVICE_SECURE:
            // When DeviceListActivity returns with a device to connect
            if (resultCode == Activity.RESULT_OK) {
                connectDevice(data, true);
            }
            break;
```

We use **connectDevice()** to connect the device user selected.

```java
private void connectDevice(Intent data, boolean secure) {
    // Get the device MAC address
    String address = data.getExtras().getString(DeviceListActivity.EXTRA_DEVICE_ADDRESS);
    // Get the BluetoothDevice object
    BluetoothDevice device = mBluetoothAdapter.getRemoteDevice(address);
    // Attempt to connect to the device
    mChatService.connect(device, secure);
}
```

sendMessage() is used to send message to other devices.

```java
private void sendMessage(String message) {
    // Check that we're actually connected before trying anything
    if (mChatService.getState() != BluetoothChatService.STATE_CONNECTED) {
        Toast.makeText(getApplicationContext(), "You are not connected to a device", Toast.LENGTH_SHORT).show();
        return;
    }

    // Check that there's actually something to send
    if (message.length() > 0) {
        // Get the message bytes and tell the BluetoothChatService to write
        byte[] send = message.getBytes();
        mChatService.write(send);

        // Reset out string buffer to zero and clear the edit text field
        mOutStringBuffer.setLength(0);
          Toast.makeText(getApplicationContext(), mOutStringBuffer.toString(), Toast.LENGTH_SHORT).show();
          mOutEditText.setText(mOutStringBuffer);
    }
```

In Bluetooth handler, we can receive the messages from other devices and store it in **dataFromEdison**. Later we can retrieve the data from it.

```java
case Constants.MESSAGE_READ:
    byte[] readBuf = (byte[]) msg.obj;
    // construct a string from the valid bytes in the buffer
    dataFromEdison = new String(readBuf, 0, msg.arg1);
    received = true;
    break;
```