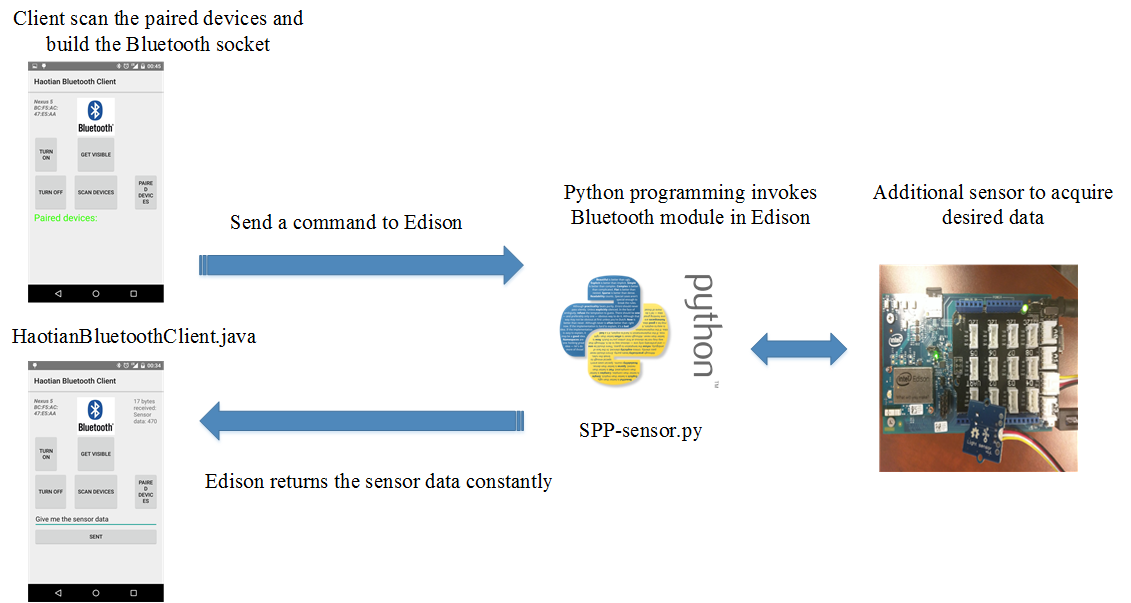
**Tutorial on Edison-Android Communications **

**EECS397/EECS600, Mobile Computing&&Sensor Network, Fall 2015**

**Advisor: Prof. Ming-Chun Huang**

**Introduction:**

In this project, we will learn how to set up Edison-Android communication for sending command from an Android device, and sending sensor data from Edison broad constantly to Android device. The Edison is a full computer without a video or key broad interface. This means you have to work with it via a remote console. There are lots of ways of talking to the Edison. Such as, Bluetooth (BT), WIFI, and USB. We will introduce Bluetooth connection in this tutorial. Now, since we have built-in BT support on Edison, you can use Edison module with Arduino breakout broad to do it.



**Figure 1** An example of an Edison-Android sensor data transmission system

In this project, Android device act as client and Edison is server. Figure 1 depicts a possible scheme for an Edison-Android sensor data transmission system. The user will first scan for the paired devices and send connection request to Edison. If Edison is the new device, which is never paired before, user should scan for the new device and pair with Edison first. After receiving the connection request, a python script on the Edison then invokes the server-side application to build the connection with Android. Now, user can send a command to Edison and tell Edison “Give me the sensor data”. Then, the Edison will send the captured sensor data to the Android device constantly for saving or processing further. Please note that the sensor can be any type of sensor supported by the base shield V2 in the figure 1. For example, light sensor, temperature sensor and UV sensor.

We will be using the above example throughout our tutorial. The tutorial is divided into two parts. The first part of the tutorial will focus on how to set up the Edison. We will implement a Python script to facilitate the built-in Bluetooth module on Edison. In the second part, we will explain how to handle Bluetooth communication on an Android client.

Disclaimer: This tutorial is for educational purpose only and it is not a 100% step by step tutorial, which means students still need to finish some parts by themselves. It is intended for students who would like to quickly overview the embedded system and Android development. Without considering security and reliability, according to this tutorial, students can build a fast-working prototype for their class project.

***Pre-requisite:***

1. EECS 132 and EECS 233

2. Prior experience with the Unix command line.

3. Android Studio (recommended) and SSH client such as PuTTY (for Windows).

**Part I: Edison Board**

***Edison Setup***

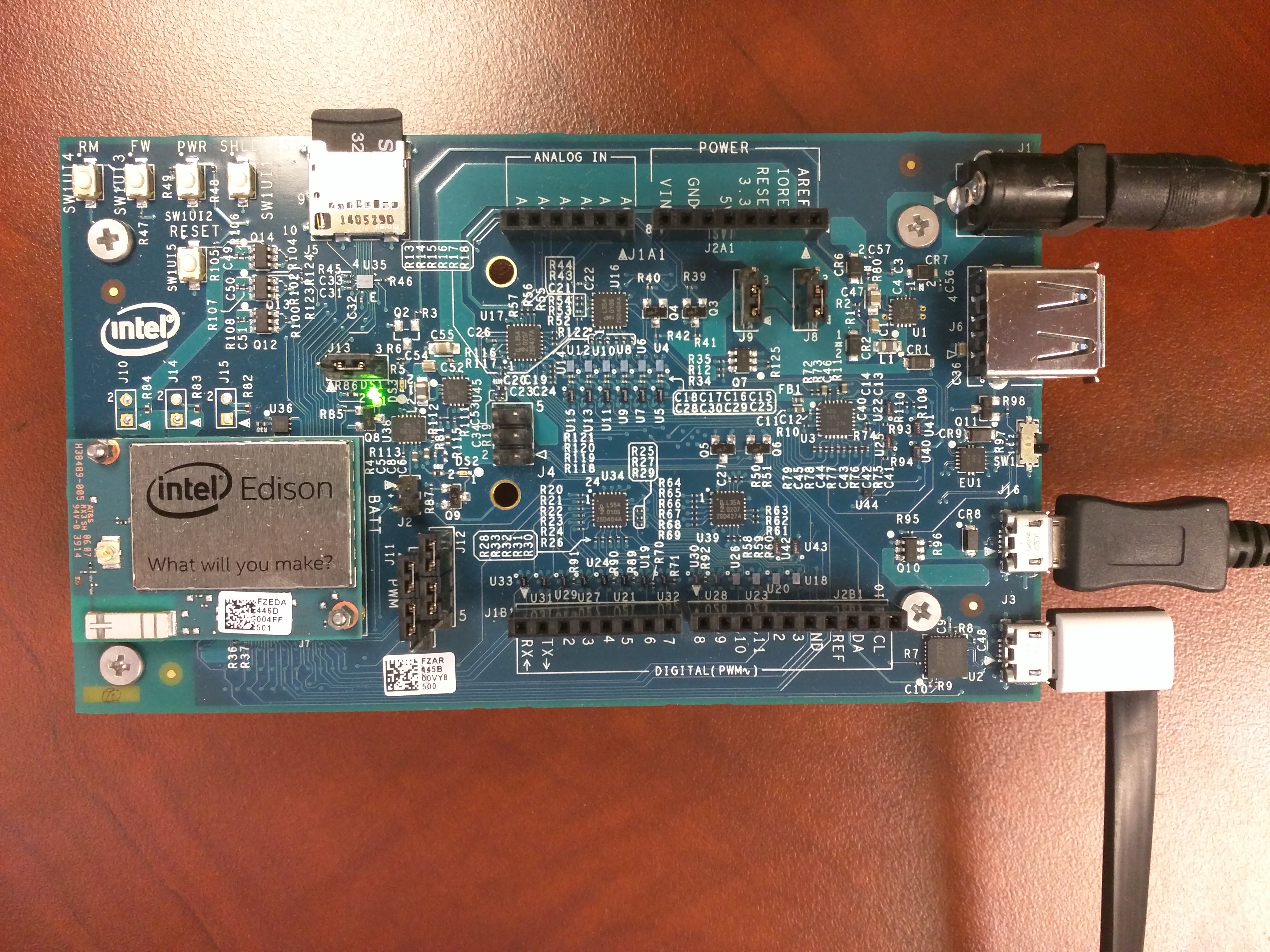
This tutorial assumes you have a PuTTY terminal emulator installed in Windows. Otherwise, please refer to the following links on how to setup an Edison. You can try several built-in demo by yourself and we recommend you choose Arduino IDE. Please go through the following link before you read the left part of the tutorial. After you get your Edison Board, please update the firmware according to the third link.

<https://software.intel.com/en-us/iot/library/edison-getting-started> (getting start)

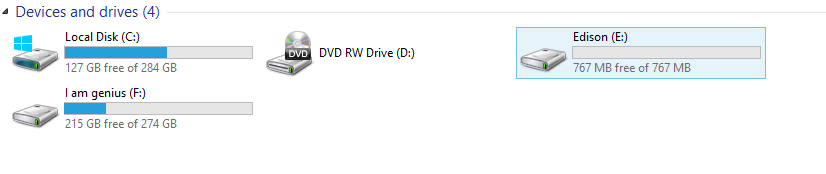
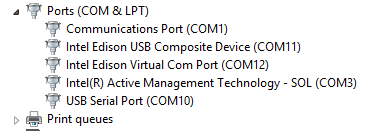
<https://software.intel.com/en-us/setting-up-serial-terminal-on-system-with-windows#terminal>

<https://software.intel.com/en-us/flashing-firmware-with-flash-tool-lite> (update the firmware)

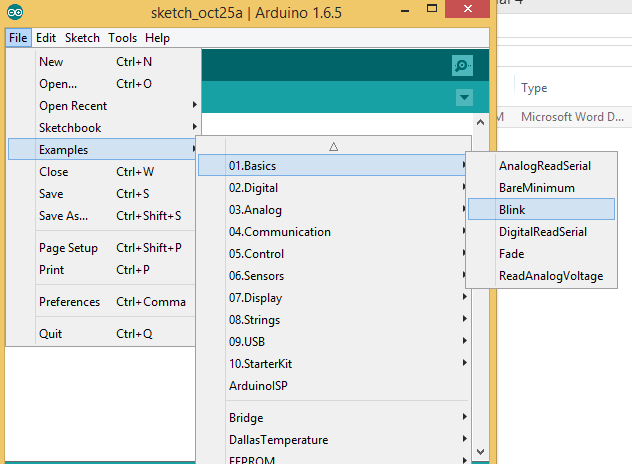
Note that, edge USB connector is designed for the USB Serial Port, you can access Unix terminal of the Edison when you plug the cable in edge USB connector. Middle USB connector is designed for communicating to the Arduino IDE, you can upload and run your code using Arduino IDE when you plug the cable in middle USB connector.



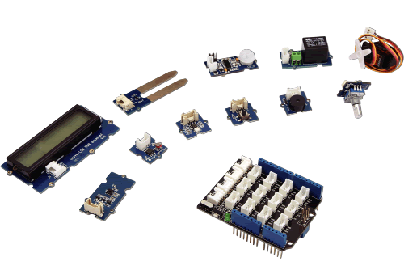
Once your Edison is fully initialized, your PC should mount a new drive and your device manager should have the following new Ports,



Now, you can import an existing example to test your Edison. A led will blink in the following example.



Now, you can play with Grove Indoor Environment Kit for Intel® Edison.



You can download the official getting start tutorial from the following link.

<http://www.seeedstudio.com/wiki/images/d/d6/110060064.pdf>

You also can refer the following link to learn how to play with sensor using Edison.

<http://www.seeedstudio.com/depot/Grove-Indoor-Environment-Kit-for-Intel-Edison-p-2427.html>

The following link give you an example which shows how to read and write data to and from an SD card.

<https://www.arduino.cc/en/Tutorial/ReadWrite>

Homework 1: Submit a video demo and show us how you play with the Grove Indoor Environment Kit. At least, you should repeat the example in the official tutorial. In addition, once you repeat the example successfully, you should be able to collect multiple sensor data and you need to store these sensor data in the SD card.

The data format is not the matter. The purpose of the homework 1 is to make sure all of you know how to set up Edison, how to set up sensor board, how to develop the Edison board and how to read/write SD card.

***Getting Started with Bluetooth***

The following link demonstrates how to enable Edison Bluetooth and pair to a device.

<https://software.intel.com/en-us/articles/intel-edison-board-getting-started-with-bluetooth>

rfkill unblock bluetooth

bluetoothctl

scan on

pair 78:24:AF:13:58:B9

discoverable on

trust 78:24:AF:13:58:B9

***Python Programming***

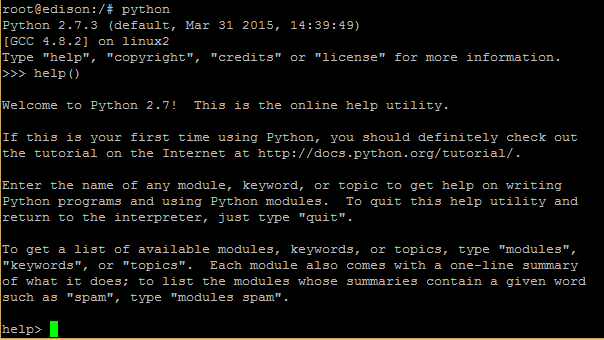
There's so much motivation to learn Python nowadays. As a python programmer, you might be wondering that what role Python would play in the rapidly emerging field of internet of Things as well. This tutorial will lead you to go through it.

Logging into serial terminal, run the following commands to check the python environment:

**python**

**help()**

If the python environment is ok, you will see the following process:



***Serial Port Profile***

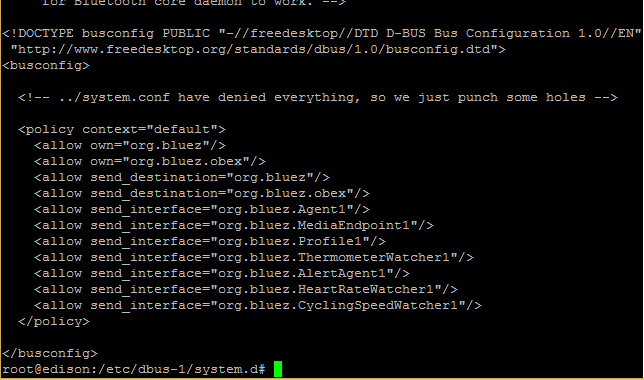
SPP (serial port profile), which is based on ETSI 07.10 and RFCOMM protocol, defines how two Bluetooth-enabled devices create a virtual/emulated serial port connection and communicate with each other.

We can test this by creating a virtual serial port between two devices via Bluetooth and using SPP to send info from one Bluetooth device to another. We can verify using the following methods:

• SPP verification using DBUS APIs (recommended)  
• SPP verification using the RFCOMM tool

In our tutorial, we choose DBUS APIs to verify the SPP process.

Important:  
Go to the [root@edison:/etc/dbus-1/system.d/bluetooth.conf](mailto:root@edison:/etc/dbus-1/system.d/bluetooth.conf). If the bluetooth.conf file doesn’t have the line <allow send\_interface="org.bluez.Profile1"/>, add this line and save the file.



You can get at the application layer of the RFCOMM socket file directly using the test-profile python script in the BlueZ test folder. But in this tutorial, you can download the modified version from the following link and loopback received data on the other side to verify SPP.

<http://downloadmirror.intel.com/24909/eng/SPP-loopback.py>

Navigating to the location of SPP-loopback.py, run the script in the background using the following command:

**python SPP-loopback.py &**

SPP-loopback.py is just an example of verifying the SPP. When the android device sends a message to the Edison board, the Edison will send the message in a loop back to the android device. You can refer the following YouTube link, <https://www.youtube.com/watch?v=yuTXnxI4UWc>

However, our purpose is to send the sensor data to the Android device constantly. This means you need to modify SPP-loopback.py by yourself. Do not worry too much about python. The key point is to figure out how to use python to acquire sensor data and how to implement SPP-loopback.py. You can use nano to modify SPP-loopback.py in the terminal.

Homework 2: Try to write sensor data to the terminal window first and modify SPP-loopback.py. You cannot verify your modification now, after you finish the Android Bluetooth part, you can verify your homework 2.

Hint: import mraa. Keyword: Python, Edison, Sensor data

**Part II: Android Client**

If you haven’t installed Android Studio, please go to the following link:

<https://developer.android.com/sdk/index.html>

If you haven’t worked with Android before, please go to the following link in order to start quickly:

<https://developer.android.com/training/basics/firstapp/index.html>

***Bluetooth in Android***

In our tutorial, we will only introduce the section of connecting as a client and some basic knowledge about Bluetooth. For students who have never touched Android before and want to learn Bluetooth thoroughly, we highly recommend you go to the following link,

<http://developer.android.com/guide/topics/connectivity/bluetooth.html>

All of the Bluetooth APIs are available in the android.bluetooth package. Here's a summary of the classes and interfaces you will need to create Bluetooth connections in our project:

* [*BluetoothAdapter*](http://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html)

Represents the local Bluetooth adapter (Bluetooth radio). The [BluetoothAdapter](http://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html) is the entry-point for all Bluetooth interaction. Using this, you can discover other Bluetooth devices, query a list of bonded (paired) devices, instantiate a [BluetoothDevice](http://developer.android.com/reference/android/bluetooth/BluetoothDevice.html) using a known MAC address, and create a [BluetoothServerSocket](http://developer.android.com/reference/android/bluetooth/BluetoothServerSocket.html) to listen for communications from other devices.

* [*BluetoothDevice*](http://developer.android.com/reference/android/bluetooth/BluetoothDevice.html)

Represents a remote Bluetooth device. Use this to request a connection with a remote device through a [BluetoothSocket](http://developer.android.com/reference/android/bluetooth/BluetoothSocket.html) or query information about the device such as its name, address, class, and bonding state.

* [*BluetoothSocket*](http://developer.android.com/reference/android/bluetooth/BluetoothSocket.html)

Represents the interface for a Bluetooth socket (similar to a TCP [Socket](http://developer.android.com/reference/java/net/Socket.html)). This is the connection point that allows an application to exchange data with another Bluetooth device via InputStream and OutputStream.

Then, we will separate the whole client application into 5 parts: (1) Get the BluetoothAdapter. (2) Enable Bluetooth. (3) Get Paired Devices. (4) Connect as a Client. (5) Manage a Connection.

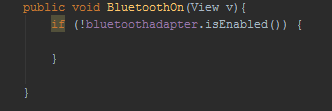
1. Get the BluetoothAdapter

The *BluetoothAdapter* is required for any and all Bluetooth activity. The static *getDefaultAdapter()* method will returns a *BluetoothAdapter* that represents the device's own Bluetooth adapter, and your application can interact with it using this object.



1. Enable Bluetooth

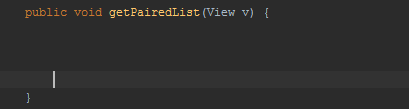
Next, you need to ensure that Bluetooth is enabled.



If the *isEnabled()* returns false, Bluetooth is disabled now. You can call *startActivityForResult()* to turn on the Bluetooth. A dialog will appear requesting user permission to enable Bluetooth.

1. Get Paired Devices

Following the Edison Bluetooth Setup procedures, you should have already paired Edison with the Android phone. For this reason, here we only explain how to get paired devices. Remember there is a difference between being paired and being connected. To be paired means that two devices are aware of each other's existence, have a shared link-key that can be used for authentication, and are capable of establishing an encrypted connection with each other. To be connected means that the devices currently share an RFCOMM channel and are able to transmit data with each other.



To do so, call *getBondedDevices().* This will return a set of *BluetoothDevices* representing paired devices . You can put them in an *ArrayAdapter* and show them using the *ListView*. In addition, *device.getName(), device.getAddress(), device.getBondState(), device.getBluetoothClass(), device.getClass()* can show the name, address, bond state, Bluetooth class and class of the device, respectively.



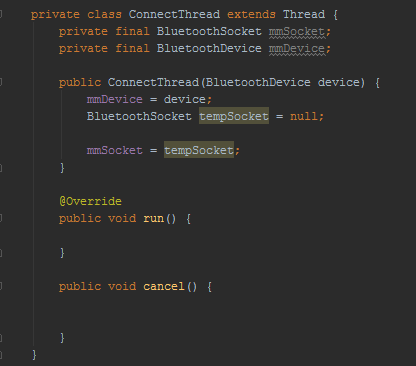
1. Connect as a Client

A Universally Unique Identifier (UUID) is a standardized 128-bit format for a string ID used to uniquely identify information. In this case, it's used to uniquely identify your application's Bluetooth service. For our project, you may use the following UUID to build a connection.



When the client attempts to connect with the server, it will carry a UUID that uniquely identifies the service with which it wants to connect. This UUID must be matched in order for the connection to be accepted. Now, we can continue to see how to initiate a connection with a remote device.

First, from the paired device list, you have to select one of them as objective remote device. Then, in order to get a *BluetoothSocket* to connect with the remote device, you can call *createRfcommSocketToServiceRecord(myUUID)* to initiate a *BluetoothSocket*.

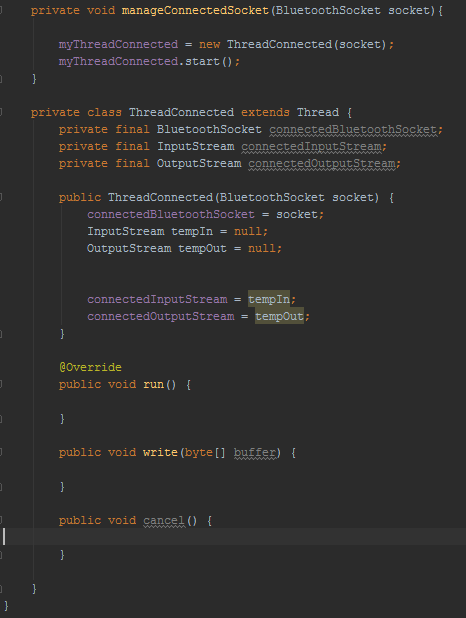


Second, after you get a *BluetoothSocket*, you can call *connect()* to initiate a connection. Upon this call, the system will perform a *Service Discovery Protocol (SDP)* lookup on the remote device in order to match the UUID. If the lookup is successful and the remote device accepts the connection, it will share the RFCOMM channel to use during the connection and [*connect*()](http://developer.android.com/reference/android/bluetooth/BluetoothSocket.html#connect()) will return.

Once the connection is built successfully, a “send” button and an EditText will be shown up using *setVisibility(View.VISIBLE)*. Of course, you don’t have to do it. It’s up to your own preference.

1. Manage a Connection

When you have successfully connected two devices, each one will have a connected *BluetoothSocket*. Then, you can transfer data between two devices easily using *read()* and *write().* Before that, you need to get the [InputStream](http://developer.android.com/reference/java/io/InputStream.html) and [OutputStream](http://developer.android.com/reference/java/io/OutputStream.html) that handle transmissions through the socket, via [*getInputStream()*](http://developer.android.com/reference/android/bluetooth/BluetoothSocket.html#getInputStream()) and [*getOutputStream()*](http://developer.android.com/reference/android/bluetooth/BluetoothSocket.html#getOutputStream()), respectively.



When *read()* returns with bytes from the InputStream, the data is sent to the UI activity using a *Handler* or *runOnUiThread().* Then, it will go back and wait for the upcoming stream. In addition, you can simply call *write()* to send the data to the remote device.

You can download BluetoothChat sample from the following link,

<https://developer.android.com/samples/BluetoothChat/index.html>

This sample allows two Android devices to carry out two-way text chat over Bluetooth.

The BluetoothChat sample has more functionality, and that is beyond the requirements of our project. What you need to do is to modify and simplify the sample source code so that Edison-Android communication can be built.

Notice that there is no standard answer and that you can add any functions or use any layouts that you want in your application.

Homework 3: Study the BluetoothChat application and build your own Android client application. Finally, once successfully connected, the Android client should be able to send messages to the Edison board. After receiving the “start” command from the Android client, the Edison board should send the sensor data constantly and the Android client should save the data to a file (.csv). We encourage you to learn the implementation way from BluetoothChat example (codebase of tutorial 4), also you can implement everything in one activity without service and broadcast.

You need to fill in the above empty functions and you can refer the official BluetoothChat example.

Try to get one type of sensor data first, then you should get multiple sensor data at the same time. Please preserve all types of sensor data available in the Grove Indoor Environment Kit in the smartphone side. You should design your own GUI. We hope to see your own design instead of figure 1.

The format of the CSV file is similar with the previous assignment.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Time Stamp | Sensor 1 | Sensor 2 | Sensor 3 | …….. |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Video: Take a video and show us the demo, including CSV file.

Report: In the report, there are two parts: Python part and Android part. Write clearly your logic for both. How you modify the loopback.py? How you design your Android Bluetooth Client? You should give us some screenshots and explain your code and logic to us. We also will check your code. Please add reasonable comment for your code carefully. If you cannot give us a video for demo, the report will be the only way for grading. Your thinking, your strategy, your coding, your logic, all of these will help us to grade.

Also, you still need to tell us what your contribution is and what your partner’ contribution is.

Finally, put all the videos, report, codes, CSV file to a zip file and submit the zip file to the blackboard.

Note that if your PC cannot detect Edison and you really don’t know how to solve it even you have tried lots of ways and links, try the following steps,

<https://communities.intel.com/thread/58775>

1. In linux: sudo apt-get install dfu-util

2. Download the latest Yocto image:https://software.intel.com/en-us/iot/hardware/edison/downloads

**Modify the flashal.sh:**

echo "Flashing U-Boot Environment Backup"

flash-command --alt u-boot-env1 -D "${VARIANT\_FILE}" –R

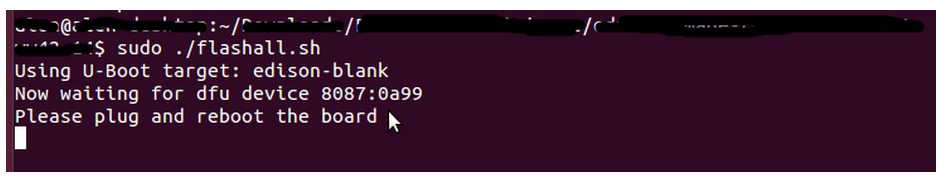
**To:**

echo "Flashing U-Boot Environment Backup and rebooting to apply partition changes"

flash-command --alt u-boot-env1 -D "${VARIANT\_FILE}" -R

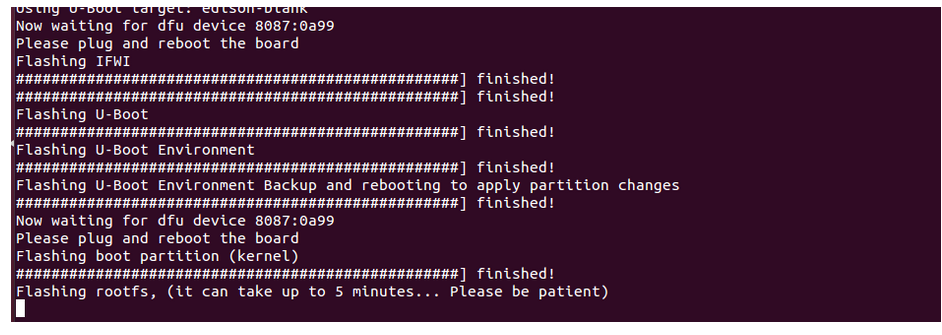
dfu-wait

3. sudo ./flashall.sh



4. Connect your board to the linux computer.

5. Wait.



TA can help you if you do not have a Linux machine.