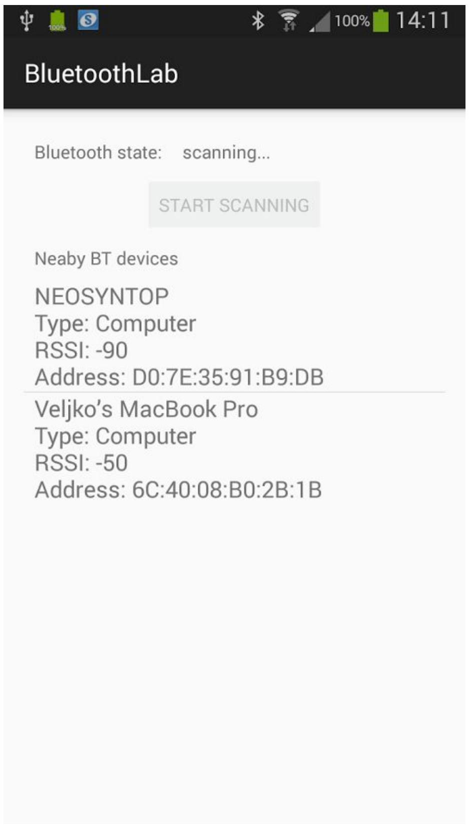
Lab 8 - Bluetooth Sensing

Bluetooth (BT) is a short-distance wireless communication standard, created for so-called personal area networks (PANs). Thus, in mobile sensing we can use BT for sensing nearby devices, and consequently, for sensing nearby people carrying these devices. In this lab we will work on sensing the BT environment and displaying the results to the user. In addition, we will very briefly discuss data transfer using BT.

Sensing Nearby Devices

We will begin by setting the device's BT for scanning. To use BT connection on Android you need to request "`android.permission.BLUETOOTH`" in the Manifest. To do anything more advanced, such as scan a device's BT environment, you need to request "`android.permission.BLUETOOTH_ADMIN`". So, start by creating a new project with a blank **Activity** and add the above permissions to the Manifest.



NEOSYNTOP
Type: Computer
RSSI: -90
Address: D0:7E:35:91:B9:DB

Veljko's MacBook Pro
Type: Computer
RSSI: -50
Address: 6C:40:08:B0:2B:1B

The layout should be simple, composing of a **TextView** that will show the current state of BT (let's call it `bt_state`), a **Button** (call it `scan_btn`) to initiate BT scanning, and a **ListView** for showing the results (call it `device_list`). You can see the one that I've made on the left (note that the list of nearby BT devices is populated dynamically, every time a user initiates scanning).

Enabling BT

BluetoothAdapter is the key class that enables the interaction with the BT interface. Through this class we can enable/disable the adapter, initiate BT scanning, and setting the name of the device. Bluetooth is present in almost every phone, however, it is often disabled by default. Let's check if the chip is present and if it is enabled.

Create a public field of type **BluetoothAdapter** in **MainActivity**, and in its `onCreate` method initialise the field:

```
mBluetoothAdapter =
BluetoothAdapter.getDefaultAdapter();
```

If `mBluetoothAdapter` is still equal "`null`" after the above call, that means a BT chip is not available on the device. Set the **TextView** indicating the state of BT to "`unavailable`", else you may set it to "`available`".

Even if the device is available, that does not mean it is enabled (i.e. BT is "off"). Unless the app is in the foreground we don't really need to manipulate BT, so only in onStart check if BT is enabled, and if not turn it on. To check whether it is enabled use:

```
mBluetoothAdapter.isEnabled()
```

If it is enabled, set `bt_state` **TextView** to “`enabled`”, else set it to “`enabling…`” and proceed with enabling BT on the phone. There are two ways to do this:

1. call `mBluetoothAdapter.enable();`
2. send an **Intent** with the action set to `BluetoothAdapter.ACTION_REQUEST_ENABLE`. Both cases are asynchronous, meaning, the adapter will not be turned on immediately.

The main difference between the two approaches is that 2) ask the user to explicitly allow the app to turn on BT. On the other hand, 1) silently turns on BT. For a large majority of applications 2) is the preferred way to turn the interface on. People don’t expect that their applications randomly turn BT on/off without their consent.

In `onStart`, let’s enable BT (if not enabled already), by starting an Intent as described in 2):

```
Intent enableIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
```

To capture the exact moment when BT is enabled, we can listen to `BluetoothAdapter.ACTION_STATE_CHANGED` via a **BroadcastReceiver**. This will also be triggered if the user decides to switch BT on/off. However in our app, we can use a more simple approach. Simply start the above intent with:

```
startActivityForResult(enableIntent, REQUEST_ENABLE_BT);
```

Where `REQUEST_ENABLE_BT` is an integer code you defined.

The intent to enable BT will return eventually. Capture this even in `onActivityResult` (overridden in MainActivity), and check whether `requestCode` matches `REQUEST_ENABLE_BT,` and whether `resultCode` is equal to `Activity.RESULT_OK`. If so, BT is enabled, and you can change the state in the **TextView**. Else (i.e. if the result is not equal to `RESULT_OK`), there was a problem and enabling BT failed, so set the status to “`enabling BT failed`”.

Sensing BT devices

One BT device cannot sense the existence of another BT device, unless this other device is set to be “discoverable”. In this lab we want to sense as many devices as possible, so let’s ensure that each phone is discoverable. When you are sure that the BT adapter is enabled (in `onStart` and in `onActivityResult`), make a device discoverable. You do that by sending an intent (`startActivity(intent)`) with action set to `BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE`. You can set for how many minutes your device is discoverable by putting an integer extra with a tag `BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION`, and an integer value (minutes). The maximum you can have a device visible for is 300 minutes, so use this value. You should make device discoverable only if it is not already discoverable. You can check whether a device is discoverable by comparing `mBluetoothAdapter.getScanMode()` with `BluetoothAdapter.SCAN_MODE_CONNECTABLE_DISCOVERABLE)`. If they are *not equal*, then the device is not discoverable.

Finally, let’s do the sensing. Set and implement **OnClickListener** for the button you have in the layout. Here, check if BT is enabled, and if so, proceed with with scanning by calling `mBluetoothAdapter.startDiscovery();` Before, however, check whether a discovery is already in progress, by checking `mBluetoothAdapter.isDiscovering()`. You can cancel the existing sensing session by calling `mBluetoothAdapter.cancelDiscovery()`. Once you called `startDiscovery`, you may disable the button, as you don't want an impatient user restarting sensing before the results are in. Also, set the status **TextView** to "`scanning`"

`startDiscovery()` is an asynchronous call. Sensing results will be returned via `ACTION_FOUND` intent, and you will be notified that the scanning process is finished via `ACTION_DISCOVERY_FINISHED` intent. We need a **BroadcastReceiver** to capture the results. In **MainActivity** create a field `private final BroadcastReceiver mReceiver` and initialise it to a new **BroadcastReceiver**. Override its `onReceive` method. We will first capture all device-found events. Check whether the received intents action field equals "`BluetoothDevice.ACTION_FOUND`". If so, the sensed device information is stored in the intent's extra bundle:

```
BluetoothDevice device =
intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
```

From device we can easily get name, MAC address and the type of device. Figure out how to get the first two by reading BluetoothDevice documentation. For device type, use `device.getBluetoothClass().getMajorDeviceClass()` to get a coarse type of the device. This will tell you whether a device is a computer or a phone, for example. However, the returned value is an integer code. You can use the following function to convert the code to a human-readable string:

```
public static String lookupBTType(int type) {
    switch (type) {
        case BluetoothClass.Device.Major.AUDIO_VIDEO:
            return "Audio/Video";
        case BluetoothClass.Device.Major.COMPUTER:
            return "Computer";
        case BluetoothClass.Device.Major.HEALTH:
            return "Health related";
        case BluetoothClass.Device.Major.IMAGING:
            return "Imaging device";
        case BluetoothClass.Device.Major.MISC:
            return "Misc device";
        case BluetoothClass.Device.Major.NETWORKING:
            return "Networking device";
        case BluetoothClass.Device.Major.PERIPHERAL:
            return "Peripheral";
        case BluetoothClass.Device.Major.PHONE:
            return "Phone";
        case BluetoothClass.Device.Major.TOY:
            return "Toy";
        case BluetoothClass.Device.Major.UNCATEGORIZED:
            return "Uncategorised device";
        case BluetoothClass.Device.Major.WEARABLE:
            return "Wearable";
    }
    return "Unknown";
}
```

Concatenate device name, type info and MAC address into a string, and separate the fields with a new line. This should be put in our device list. **ListView** is best populated with an **Adapter**. In **MainActivity** create a private field `ArrayAdapter<String> mNewDevicesArrayAdapter;` In `onCreate` initialise the **Adapter**:

```
mNewDevicesArrayAdapter = new ArrayAdapter<String>(this,
android.R.layout.simple_list_item_1);
```

and set it to be your ListView's Adapter (using `.setAdapter()`).
Now, back in **BroadcastReceiver**, use `mNewDevicesArrayAdapter.add()` to add the string describing a newly-found device to the list.

In **BroadcastReceiver**, also capture `BluetoothAdapter.ACTION_DISCOVERY_FINISHED` intent. Then, re-enable the scanning button, set **TextView** status back to "`enabled`", and check how many results you have (`mNewDevicesArrayAdapter.getCount()`). If the answer is zero, add a string "`no devices found`" to the list.

Our app is almost ready. We have implemented **BroadcastReceiver**, however, we haven't registered for the Intents of interest. Do that in `onCreate`. Create two **IntentFilters** with `BluetoothDevice.ACTION_FOUND` and `BluetoothAdapter.ACTION_DISCOVERY_FINISHED` actions and register the receiver with each of the filters: e.g. `this.registerReceiver(mReceiver, filter);` Don't forget to unregister the receiver in `onDestroy`. In the same method, ensure that an discovery process is cancelled (in case it is still running).

You should test your application now, check if scanning works, and if the results are as expected.

Finally, let's add the information about the received signal strength (RSSI) of the detected bluetooth devices. This information is passed as another extra with the intent through which we have detected a new scan result. `intent.getShortExtra(BluetoothDevice.EXTRA_RSSI, Short.MIN_VALUE)` ensures you get the RSSI value (or min short int, in case it is not available). Put this information into the result string shown to the user.

One final tweak to our app: when a user initiates scanning we should erase old results. `mNewDevicesArrayAdapter.clear();` takes care of that.

Data Transfer Using BT

Android Bluetooth communication is executed through **BluetoothSockets**. One side has to implement a **BluetoothServerSocket** and listen to incoming communication requests, while the other initiates the communication. When the server and the client establish a connection, it should be moved to a separate thread. In addition, BluetoothServerSocket should work in a separate thread, so that it does not block the app while waiting for the client to initiate the connection. A very detailed example of BT communication is available through Google sample projects:
https://github.com/googlesamples/android-BluetoothChat

Extend the above sensing app with the ability to start a chat session between two phones.

Happy coding!