Using Android Hardware

Using Bluetooth

- The application framework provides access to the Bluetooth functionality
- Using the Bluetooth APIs, an Android application can perform the following:
  - Scan for other Bluetooth devices
  - Query the local Bluetooth adapter for paired Bluetooth devices
  - Establish RFCOMM channels
  - Connect to other devices through service discovery
  - Transfer data to and from other devices
  - Manage multiple connections through the Android Bluetooth APIs.

Bluetooth Basics

- All of the Bluetooth APIs are available in the android.bluetooth package.
- Few classes needed to use Bluetooth on Android
  - BluetoothAdapter
    - Represents the local Bluetooth adapter (Bluetooth radio).
    - The BluetoothAdapter 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 using a known MAC address, and create a BluetoothServerSocket to listen for communications from other devices.

- BluetoothDevice
  - Represents a remote Bluetooth device.
  - Use this to request a connection with a remote device through a BluetoothSocket or query information about the device such as its name, address, class, and bonding state.
- BluetoothSocket
  - Represents the interface for a Bluetooth socket (similar to a TCP Socket).
  - This is the connection point that allows an application to exchange data with another Bluetooth device via InputStream and OutputStream.

- BluetoothServerSocket
  - Represents an open server socket that listens for incoming requests (similar to a TCP ServerSocket).
  - In order to connect two Android devices, one device must open a server socket with this class.
  - When a remote Bluetooth device makes a connection request to the this device, the BluetoothServerSocket will return a connected BluetoothSocket when the connection is accepted.
- BluetoothClass
  - Describes the general characteristics and capabilities of a Bluetooth device.
  - This is a read-only set of properties that define the device's major and minor device classes and its services.

Bluetooth Permissions

- In order to use Bluetooth features in your application, you need to declare at least one of two Bluetooth permissions: BLUETOOTH and BLUETOOTH_ADMIN.
- You must request the BLUETOOTH_ADMIN permission in order to initiate device discovery or manipulate Bluetooth settings.

Note: If you use BLUETOOTH_ADMIN permission, then must also have the BLUETOOTH permission.

Setting Up Bluetooth

1. Get the BluetoothAdapter The BluetoothAdapter is required for any and all Bluetooth activity.
   - To get the BluetoothAdapter, call the static getDefaultAdapter() method.
     This returns a BluetoothAdapter that represents the device's own Bluetooth adapter (the Bluetooth radio). There's one Bluetooth adapter for the entire system, and your application can interact with it using this object.

```
BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
if (mBluetoothAdapter == null) {
   // Device does not support Bluetooth
}
```

Setting Up Bluetooth

2. Enable Bluetooth
    1. Ensure that Bluetooth is enabled. Call isEnabled() to check whether Bluetooth is currently enable.
    2. To request that Bluetooth be enabled, call startActivityForResult() with the ACTION_REQUEST_ENABLE action Intent.
    3. If enabling Bluetooth succeeds, your Activity will receive the RESULT_OK result code in the onActivityResult() callback. If Bluetooth was not enabled due to an error (or the user responded "No") then the result code will be RESULT_CANCELED.

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

Setting Up Bluetooth

3. Finding Devices

1. Using the BluetoothAdapter, you can find remote Bluetooth devices either through device discovery or by querying the list of paired (bonded) devices.
2. The current Android Bluetooth API's require devices to be paired before an RFCOMM connection can be established.

Querying paired devices

- getBondedDevices():
  This will return a Set of BluetoothDevices representing paired devices.

```
Set<BluetoothDevice> pairedDevices = mBluetoothAdapter.getBondedDevices();
// If there are paired devices
if (pairedDevices.size() > 0) {
    // Loop through paired devices
    for (BluetoothDevice device : pairedDevices) {
        // Add the name and address to an array adapter to show in a ListView
        mArrayAdapter.add(device.getName() + "\n" + device.getAddress());
    }
}
```

Discovering devices

- startDiscovery():
  - Called to start Bluetooth device discovery.
  - Returns a boolean value, indicating whether the discovery is started successfully or not.
- Your application must register a BroadcastReceiver for the ACTION_FOUND Intent in order to receive information about each device discovered.
- For each device, the system will broadcast the ACTION_FOUND Intent. This Intent carries the extra fields EXTRA_DEVICE and EXTRA_CLASS, containing a BluetoothDevice and a BluetoothClass, respectively.

Discovering device

```
// Create a BroadcastReceiver for ACTION_FOUND
    private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
        public void onReceive(Context context, Intent intent) {
            String action = intent.getAction();
            // When discovery finds a device
    if (BluetoothDevice.ACTION_FOUND.equals(action)) {
      // Get the BluetoothDevice object from the Intent
      BluetoothDevice device =
    intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
      // Add the name and address to an array adapter to show in a ListView
    mArrayAdapter.add(device.getName() + "\n" + device.getAddress());
            }
       }
    };
    //continued
```

Discovering device

```
// Register the BroadcastReceiver
    IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
    registerReceiver(mReceiver, filter);
```

Note: All that's needed from the BluetoothDevice object in order to initiate a connection is the MAC address.

Enabling discoverability

- to make the local device discoverable to other devices, call startActivityForResult(Intent, int) with the ACTION_REQUEST_DISCOVERABLE action Intent.

```
Intent discoverableIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
discoverableIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURA
TION, 300);
startActivity(discoverableIntent);
```

Note: If Bluetooth has not been enabled on the device, then enabling device discoverability will automatically enable Bluetooth.

Connecting as a server

- To connect two devices, one must act as a server by holding an open BluetoothServerSocket.
- The purpose of the server socket is to listen for incoming connection requests and when one is accepted, provide a connected BluetoothSocket.

Procedure for connecting as a server

1. Get a BluetoothServerSocket by calling the listenUsingRfcommWithServiceRecord(String, UUID).
2. Start listening for connection requests by calling accept().
3. Unless you want to accept additional connections, call close().

- Unlike TCP/IP, RFCOMM only allows one connected client per channel at a time
- BluetoothServerSocket or BluetoothSocket are thread-safe.

```
private class AcceptThread extends Thread {
        private final BluetoothServerSocket mmServerSocket;

        public AcceptThread() {
            // Use a temporary object that is later assigned to mmServerSocket,
            // because mmServerSocket is final
            BluetoothServerSocket tmp = null;
            try {
                // MY_UUID is the app's UUID string, also used by the client code
                tmp = mAdapter.listenUsingRfcommWithServiceRecord(NAME,
    MY_UUID);
            } catch (IOException e) { }
            mmServerSocket = tmp;
        }
```

```
public void run() {
        BluetoothSocket socket = null;
  // Keep listening until exception occurs or a socket is returned
        while (true) {
            try {
                socket = mmServerSocket.accept();
            } catch (IOException e) {
                break;
            }
            // If a connection was accepted
            if (socket != null) {
                // Do work to manage the connection (in a separate thread)
                manageConnectedSocket(socket);
                mmServerSocket.close();
                break;
            }
        }
    }
```

```
    /** Will cancel the listening socket, and cause the thread to finish */
    public void cancel() {
        try {
            mmServerSocket.close();
        } catch (IOException e) { }
    }
}
```

Connecting as a client

- In order to initiate a connection with a remote device, you must first obtain a BluetoothDevice object that represents the remote device.
- You must then use the BluetoothDevice to acquire a BluetoothSocket and initiate the connection.
- Following are the steps to obtain a connection:
  1. Using the BluetoothDevice, get a BluetoothSocket by calling createRfcommSocketToServiceRecord(UUID).
  2. Initiate the connection by calling connect().

Example

```
private class ConnectThread extends Thread {
        private final BluetoothSocket mmSocket;
        private final BluetoothDevice mmDevice;

        public ConnectThread(BluetoothDevice device) {
            // Use a temporary object that is later assigned to mmSocket,
            // because mmSocket is final
            BluetoothSocket tmp = null;
            mmDevice = device;

            // Get a BluetoothSocket to connect with the given BluetoothDevice
            try {
                // MY_UUID is the app's UUID string, also used by the server code
                tmp = device.createRfcommSocketToServiceRecord(MY_UUID);
            } catch (IOException e) { }
            mmSocket = tmp;
        }
```

```
public void run() {
        // Cancel discovery because it will slow down the connection
        mAdapter.cancelDiscovery();

        try {
            // Connect the device through the socket. This will block
            // until it succeeds or throws an exception
            mmSocket.connect();
        } catch (IOException connectException) {
            // Unable to connect; close the socket and get out
            try {
                mmSocket.close();
            } catch (IOException closeException) { }
            return;
        }

        // Do work to manage the connection (in a separate thread)
        manageConnectedSocket(mmSocket);
    }
```

```
    /** Will cancel an in-progress connection, and close the socket */
    public void cancel() {
        try {
            mmSocket.close();
        } catch (IOException e) { }
    }
}
```