Bluetooth

Introduction

- The Android platform includes the support for working with the bluetooth connectivity and allows us to develop applications that wirelessly exchange data with applications running on other bluetooth devices.

The Capabilities

❖ The Android Bluetooth APIs allow the following:

+ Scan for other bluetooth devices.

+ Query for paired bluetooth devices.

+ Create RFCOMM channels connected with other device.

+ Transfer data to/from other devices.

+ Manage multiple connections.

The `BluetoothAdapter` Class

- This class represents the local bluetooth adapter. It is the entry point for every bluetooth interaction.
- Using this class we can discover other bluetooth devices, get the list of those devices ours is already paired with, instantiate `BluetoothDevice` based on a known MAC address and get a `BluetoothServerSocket` for listening to other communications coming from other devices.

The `BluetoothDevice` Class

- This class represents a remotely connected bluetooth device.

The `BluetoothSocket` Interface

- This interface describes the a bluetooth socket (similarly to TCP socket).
- This interface describes the connection point through which an application can exchange data with other bluetooth devices using `InputStream` and `OutputStream` objects.

The `BluetoothServerSocket` Class

- This class represents an open server socket that listens for incoming requests. It works similarly to the TCP/IP `ServerSocket` class.
- In order to connect two android devices, one must first open a server socket by instantiating this class.
- When a request for having a connection arrives this class returns a `BluetoothSocket` object.

The `BluetoothClass` Class

- This class describes the general characteristics and the general capabilities of a bluetooth device.
- It provides access to a set of read only properties.

Bluetooth Permissions

- In order to use the bluetooth in our application we must include at least one of the following two bluetooth permissions: `BLUETOOTH` and `BLUETOOTH_ADMIN`.

```
<manifest ... >
 <uses-permission android:name="android.permission.BLUETOOTH" />
 ...
</manifest>
```

Bluetooth Permissions

- The `BLUETOOTH` permissions is required for performing any bluetooth communication, such as requesting a connection, accepting a connection and transferring data.
- The `BLUETOOTH_ADMIN` permission is required in order to initiate device discovery or in order to manipulate Bluetooth settings. In order to get the `BLUETOOTH_ADMIN` permission we should get the `BLUETOOTH` permission as well.

Setting Up Bluetooth

❖ In order to set up a bluetooth connection we must first verify that our device supports bluetooth connectivity.

```
...
BluetoothAdapter adapter = BluetoothAdapter.getDefaultAdapter();
if (adapter == null)
{
    // bluetooth is not supported
}
...
```

Setting Up Bluetooth

- If the bluetooth is supported we can verify that the user has enabled it... and in case he hasn't we can ask him to.

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

In case the bluetooth is not enabled a dialog window will appear asking for the user permission to enable bluetooth. If the user responds "Yes," the system will enable the bluetooth and the focus will return back to our application. If enabling bluetooth succeeds, our activity will receive the RESULT_OK result code passed in the onActivityResult() callback method.

Finding Devices

- When the connection is made with a remote device for the first time, a pairing request is automatically presented to the user.
- When a device is paired, the basic information (e.g. MAC address) about that device can be read. Using the MAC address for a remote device a connection can be initiated.

Finding Devices

- Being paired (when two devices know about each other) and being connected are two different things.
- Being connected means the devices currently share an RFCOMM channel and are able to transmit data with each other.
- Two devices that want to connect with each other must first be paired. Once they are paired a connection between the two can be established.

Querying Paired Devices

- Before the application performs a 'device discovery' it worth querying the set of the devices that are already paired with our device and check if the desired device is one of them.

```
...
Set<BluetoothDevice> set =adapter.getBondedDevices();
if (set.size() > 0)
{
    for (BluetoothDevice device : set)
    {
        ...
    }
}
...
```

Discovering Devices

- Calling the `startDiscover()`, on our `BluetoothAdapter` object we use, will start an asynchronous process through which other bluetooth devices will be discovered. This method returns true if the discover process has successfully started.
- In order to get information about each one of the discovered devices we must register a `BroadcastReceiver` for the `ACTION_FOUND` intent.

Discovering Devices

- For each device the discovery process finds the `ACTION_FOUND` intent will be broadcast. This Intent carries the `EXTRA_DEVICE` and the `EXTRA_CLASS` extra fields, that respectively contain the `BluetoothDevice` and the `BluetoothClass` information.

Discovering Devices

```
...
private final BroadcastReceiver mReceiver = new BroadcastReceiver()
{
    public void onReceive(Context context, Intent intent)
    {
        String action = intent.getAction();
        if (BluetoothDevice.ACTION_FOUND.equals(action))
        {
            BluetoothDevice device = intent.getParcelableExtra(
                BluetoothDevice.EXTRA_DEVICE);
            adapter.add(device.getName()+" "+device.getAddress());
        }
    }
};
...
```

the address of the other device will be later used for connecting it

```
...
IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
registerReceiver(mReceiver, filter);
...
```

Enabling Discoverability

- In order to be discoverable by other devices we should call the `startActivityForResult(Intent,int)` passing over the `ACTION_REQUEST_DISCOVERABLE` action intent object.
- Doing so will issue a request to enable a discoverable mode through which the device will become discoverable for 120 seconds. We can define a different duration by adding the `EXTRA_DISCOVERABLE_DURATION` extra data.

Enabling Discoverability

```
...
Intent intent = new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
intent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 300);
startActivityForResult(intent);
...
```

300 seconds is the maximum possible discoverable duration

The result of this code is a pop-up dialog window that requests the user permission to turn on the device discoverability. If the user approves then the device will become discoverable for the specified amount of time.

Our activity will then receive a call to the onActivityResult() callback method.

If the user approves becoming discoverable and the bluetooth is still not enabled it will be automatically enabled.

We can set a broadcast receiver through which we will be notified when the discoverable mode changes.

Enabling Discoverability

- Enabling the device discoverability isn't necessary when our application is the one that initiates a connection to another remote device.
- Enabling the device discoverability might be necessary when we want our application to host a server socket for accepting incoming connections only. Being discoverable will allow the remote device to discover our device before initiating the connection itself.

Devices Connection

- One device should function as a server hosting a small server activity that uses the `BluetoothServerSocket` class. The other should function as a client hosting a small activity that uses the `BluetoothSocket` class.
- The client will use the MAC address of the server in order to initiate the connection.

Devices Connection

- When both the client and the server are connected to each other each one of them has a `BluetoothSocket` object that underneath is connected with the `BluetoothSocket` object on the other device.
- Implementing both the client and the server mechanism in one application and have that application installed on two separated devices will allow each one of them to function both as a client and as a server.

Devices Connection (Server)

```
private class BluetoothServerThread extends Thread
{
    private BluetoothServerSocket serverSocket;

    public BluetoothServerThread()
    {
        try
        {
            serverSocket = adapter.
                listenUsingRfcommWithServiceRecord(NAME,APP_UUID);
        }
        catch (IOException e) { }
    }

    public stopServer()
    {
        try

        {
            serverSocket.close();
        }
        catch(IOException e) {}

    }
```

The NAME is an identifiable name of our service.

A connection will be created only if the remote device has sent a connection request with a UUID matching the one registered with this listening server socket

Calling close() on the BluetoothServerSocket object from another thread will stop its blocked waiting for a connection request to arrive. Calling close() releases the server socket and the resources it occupied.

Devices Connection (Server)

```
    public void run()
    {
        BluetoothSocket socket = null;
        while (true)
        {
            try
            {
                socket = serverSocket.accept();
            }
            catch (IOException e) { }
            break;
        }

        if (socket != null)
        {
            manageSocket(socket);
            serverSocket.close();
            break;
        }
    }
}
```

The accept method shouldn't be called within the main thread which is also the UI thread, in order to avoid blocking it.

Once we get a BluetoothSocket we can pass it over to another method to handle it. This is an imaginary method.

Unlike TCP/IP, RFCOMM allows one connected client per channel at a time. In most cases it makes sense to call close() on the BluetoothServerSocket immediately after accepting the connected socket.

Devices Connection (Client)

```
private class ConnectThread extends Thread
{
    private BluetoothSocket socket;
    private BluetoothDevice device;

    public ConnectThread(BluetoothDevice device)
    {
        this.device = device;
        try
        {
            socket = device.createRfcommSocketToServiceRecord(APP_UUID);
        }
        catch (IOException e) { }
    }

    public void stopClient()
    {
        try
        {
            socket.close();
        }
        catch (IOException e) { }
    }
```

Must be identical with the one been used on the server side.

This method will allow other threads to stop the client trying to get a connection. This method will also indirectly allows other threads to free the resources when there is no more any need in this connection.

Devices Connection (Client)

```
    public void run()
    {
        adapter.cancelDiscovery();
        try
        {
            socket.connect();
        }
        catch (IOException connectException)
        {
            try
            {
                socket.close();
            }
            catch (IOException closeException) { }
            return;
        }
        manageSocket(socket);
    }
}
```

We should call the `cancelDiscovery()` method before the connection is made.

This is where the actual connection is created.

This is an imaginary method that will handle the socket we succeeded to get.

Google Chat Sample

- Within the code samples of SDK 2.1 you can find the Chat Sample application that allows two users that hold android handsets to communicate with each other using this application.