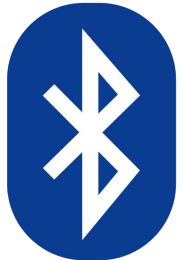


Android Bluetooth Support

Brandon Walker and James
Tobin
4/18/19



Contents

1. General Bluetooth Process / Android Specifics
2. Setting up workspace
3. Finding / Connecting to Devices (general)
4. Connecting as a Server / Client
5. Managing Connections
6. Overview
7. DEMO



Bluetooth process (general)



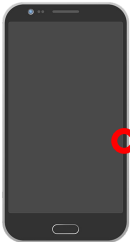
Service Discovery Process



Pairing Process



Bonding Process



Bond remains even when connection has been released/out of range

Android Bluetooth Support

- Using the Android Bluetooth APIs, an app can:
 - Scan for nearby bluetooth devices
 - Connect to nearby bluetooth devices through Service Directory
 - Query the local bluetooth adapter for paired android devices
 - Establish RFCOMM (Radio Frequency Communication) channels
 - Transfer data to and from other bluetooth devices
 - Manage multiple applications



First thing's first: Permissions

- BLUETOOTH
 - Must be enabled for any type of bluetooth communication
- BLUETOOTH_ADMIN
 - Must be enabled for more detailed bluetooth operation
 - Such as initiating device discovery or manipulating the device's bluetooth settings
- ACCESS_COARSE_LOCATION or ACCESS_FINE_LOCATION
 - Coarse allows GPS to access approximate location of device
 - Fine allows GPS to access a more detailed, precise location of device

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

Setting up Bluetooth


- Must first ensure that Bluetooth is supported and enabled on the device
- In order to do this, get a BluetoothAdapter object and call the getDefaultAdapter() method.
 - If this BluetoothAdapter object equates to null, the device doesn't support Bluetooth

```
BluetoothAdapter bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();  
if (bluetoothAdapter == null) {  
    // Device doesn't support Bluetooth  
}
```

Setting up Bluetooth

- Must first ensure that Bluetooth is supported and enabled on the device
- To check if it is enabled, call `.isEnabled()` method
 - If it's not, start a `BluetoothAdapter.ACTION_REQUEST_ENABLE` intent action with `startActivityForResult()`
 - A dialog will appear requesting user permission.

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



An app wants to turn Bluetooth ON for this device.

DENY ALLOW

Finding Devices

- Using the BluetoothAdapter, you can find remote Bluetooth devices either through device discovery or querying the list of already paired devices
- Before performing device discovery, it's worth querying the set of paired devices to see if the desired device is already known.
 - Call getBondedDevices() to retrieve a set of BluetoothDevice objects

```
Set<BluetoothDevice> pairedDevices = bluetoothAdapter.getBondedDevices();

if (pairedDevices.size() > 0) {
    // There are paired devices. Get the name and address of each paired device.
    for (BluetoothDevice device : pairedDevices) {
        String deviceName = device.getName();
        String deviceHardwareAddress = device.getAddress(); // MAC address
    }
}
```


Finding Devices

- Using the BluetoothAdapter, you can find remote Bluetooth devices either through device discovery or querying the list of already paired devices
- If the device was not found in the previous query, perform Device Discovery by calling startDiscovery():
 - Scanning procedure that searches for local area Bluetooth-enabled devices (*discoverable*) devices.
 - Need to access information such as device name, class, and unique MAC address in order to connect.
 - Start registering a BroadcastReceiver for the ACTION_FOUND intent

Finding Devices

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    ...

    // Register for broadcasts when a device is discovered.
    IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
    registerReceiver(receiver, filter);
}

// Create a BroadcastReceiver for ACTION_FOUND.
private final BroadcastReceiver receiver = new BroadcastReceiver() {
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if (BluetoothDevice.ACTION_FOUND.equals(action)) {
            // Discovery has found a device. Get the BluetoothDevice
            // object and its info from the Intent.
            BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
            String deviceName = device.getName();
            String deviceHardwareAddress = device.getAddress(); // MAC address
        }
    }
};

@Override
protected void onDestroy() {
    super.onDestroy();
    ...

    // Don't forget to unregister the ACTION_FOUND receiver.
    unregisterReceiver(receiver);
}
```

Connecting Devices

- In order to create a connection, both the server and client side mechanisms must be implemented.
 - Each side must obtain a `BluetoothSocket`, and are considered *connected* when they each have a connected `BluetoothSocket` on the same RFCOMM channel.
- One technique involves automatically preparing each device as a server so that a server socket is open and listening for connections on both ends.
 - Either device can initiate a connection and the other becomes the client.
 - Alternatively, one can explicitly act as the server holding the server socket and the other only initiates the connection

Connecting as a server

- Must hold an open BluetoothServerSocket

```
private final BluetoothServerSocket mmServerSocket;
```

- Purpose of this socket is to listen for incoming connection requests and provide a connected BluetoothSocket

```
BluetoothSocket socket = null;  
// Keep listening until exception occurs or a socket is returned.  
while (true) {  
    try {  
        socket = mmServerSocket.accept();
```

- Once the BluetoothSocket is acquired, the BluetoothServerSocket should be discarded to avoid more other connections occurring.

```
if (socket != null) {  
    // A connection was accepted. Perform work associated with  
    // the connection in a separate thread.  
    manageMyConnectedSocket(socket);  
    mmServerSocket.close();
```

Connecting as a server

```
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 = bluetoothAdapter.listenUsingRfcommWithServiceRecord(NAME, MY_UUID);
        } catch (IOException e) {
            Log.e(TAG, "Socket's listen() method failed", e);
        }
        mmServerSocket = tmp;
    }
}
```

Connecting as a server

```
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) {
            Log.e(TAG, "Socket's accept() method failed", e);
            break;
        }

        if (socket != null) {
            // A connection was accepted. Perform work associated with
            // the connection in a separate thread.
            manageMyConnectedSocket(socket);
            mmServerSocket.close();
            break;
        }
    }
}

// Closes the connect socket and causes the thread to finish.
public void cancel() {
    try {
        mmServerSocket.close();
    } catch (IOException e) {
        Log.e(TAG, "Could not close the connect socket", e);
    }
}
}
```

Connecting as a client

- In order to actually connect with a remote device on an open server socket, you must obtain a BluetoothDevice object that will represent the device.
 - Get a BluetoothSocket by using the BluetoothDevice

```
private final BluetoothSocket mmSocket;  
private final BluetoothDevice mmDevice;  
BluetoothSocket tmp = null;  
// Get a BluetoothSocket to connect with the given BluetoothDevice.  
// MY_UUID is the app's UUID string, also used in the server code.  
tmp = device.createRfcommSocketToServiceRecord(MY_UUID);
```

- Initiate connection by using connect(). SHOULD PERFORM IN SEPARATE THREAD

```
// Connect to the remote device through the socket. This call blocks  
// until it succeeds or throws an exception.  
mmSocket.connect();
```

- Should always cancel the discovery (cancelDiscovery()) process before the connection attempt occurs, however.

Connecting as a client

```
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;

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


Connecting as a client

```
public void run() {
    // Cancel discovery because it otherwise slows down the connection.
    bluetoothAdapter.cancelDiscovery();

    try {
        // Connect to the remote device through the socket. This call blocks
        // until it succeeds or throws an exception.
        mmSocket.connect();
    } catch (IOException connectException) {
        // Unable to connect; close the socket and return.
        try {
            mmSocket.close();
        } catch (IOException closeException) {
            Log.e(TAG, "Could not close the client socket", closeException);
        }
        return;
    }

    // The connection attempt succeeded. Perform work associated with
    // the connection in a separate thread.
    manageMyConnectedSocket(mmSocket);
}

// Closes the client socket and causes the thread to finish.
public void cancel() {
    try {
        mmSocket.close();
    } catch (IOException e) {
        Log.e(TAG, "Could not close the client socket", e);
    }
}
}
```

Managing a connection

- Now that we are connected, each device has a connected BluetoothSocket and can begin sharing information.
- To transfer data (generally):
 - Get the InputStream and OutputStream

```
tmpIn = socket.getInputStream();  
tmpOut = socket.getOutputStream();
```

- Read and write data using read(byte[]) and write(byte[]) (EACH ON DEDICATED THREADS)

```
mmBuffer = new byte[1024];  
// Read from the InputStream.  
numBytes = mmInStream.read(mmBuffer);  
mmOutputStream.write(bytes);
```

Managing a connection

```
public ConnectedThread(BluetoothSocket socket) {
    mmSocket = socket;
    InputStream tmpIn = null;
    OutputStream tmpOut = null;

    // Get the input and output streams; using temp objects because
    // member streams are final.
    try {
        tmpIn = socket.getInputStream();
    } catch (IOException e) {
        Log.e(TAG, "Error occurred when creating input stream", e);
    }
    try {
        tmpOut = socket.getOutputStream();
    } catch (IOException e) {
        Log.e(TAG, "Error occurred when creating output stream", e);
    }

    mmInStream = tmpIn;
    mmOutStream = tmpOut;
}
```

Managing a connection

```
public void run() {  
    mmBuffer = new byte[1024];  
    int numBytes; // bytes returned from read()  
  
    // Keep listening to the InputStream until an exception occurs.  
    while (true) {  
        try {  
            // Read from the InputStream.  
            numBytes = mmInStream.read(mmBuffer);  
            // Send the obtained bytes to the UI activity.  
            Message readMsg = handler.obtainMessage(  
                MessageConstants.MESSAGE_READ, numBytes, -1,  
                mmBuffer);  
            readMsg.sendToTarget();  
        } catch (IOException e) {  
            Log.d(TAG, "Input stream was disconnected", e);  
            break;  
        }  
    }  
}
```

Managing a connection

```
// Call this from the main activity to send data to the remote device.
public void write(byte[] bytes) {
    try {
        mmOutputStream.write(bytes);

        // Share the sent message with the UI activity.
        Message writtenMsg = handler.obtainMessage(
            MessageConstants.MESSAGE_WRITE, -1, -1, mmBuffer);
        writtenMsg.sendToTarget();
    } catch (IOException e) {
        Log.e(TAG, "Error occurred when sending data", e);

        // Send a failure message back to the activity.
        Message writeErrorMsg =
            handler.obtainMessage(MessageConstants.MESSAGE_TOAST);
        Bundle bundle = new Bundle();
        bundle.putString("toast",
            "Couldn't send data to the other device");
        writeErrorMsg.setData(bundle);
        handler.sendMessage(writeErrorMsg);
    }
}
```

Process Overview

- In order to connect and transfer data from an android app to a bluetooth device:
 - Set up correct permissions
 - Ensure that Bluetooth is supported and enabled
 - Query list of paired devices or do Device Discovery
 - Connect devices by designing a server and client relationship
 - Transfer data by building an InputStream and OutputStream and using read() and write()
- A bit of a complicated process, but aligns with most server/client data transfer behaviors

More Specific: Bluetooth Profiles

- A *Bluetooth profile* is a wireless interface specification for communication between devices
 - **BluetoothHeadset**
 - BluetoothA2dp
 - BluetoothHealth Device
- BluetoothHeadset and vendor specific AT commands
 - Create a broadcast receiver for the ACTION_VENDOR_SPECIFIC_HEADSET_EVENT intent
 - Handle hardware-specific events like volume up/down, mute, alert for low battery, etc.
- **DEMO**

Questions??