Mobile Application (Design and) Development

20th class

Prof. Stephen Intille
s.intille@neu.edu

Today

- Q&A
- A bit on Bluetooth
- 3 presentations

Things you can do

- Manage/monitor BT settings
  - Control discoverability
  - Discover nearby BT devices
  - Use BT as a proximity-based peer-to-peer connection
- Use WiFi
  - Scan for hotspots
  - Creating and modify WiFi settings
  - Monitor Internet connectivity
  - Control and monitor Internet settings

Bluetooth

- Available since v2.0
- Not all devices will have BT (but nearly)

- Designed for short-range, low-bandwidth, peer-to-peer communication
- Only encrypted, pair communication supported on Android (as of 2.1)

BT components

- BluetoothAdapter (Local device)
- BluetoothDevice (Remote devices)
- BluetoothSocket
  - Call createRfcommSockettoServiceRecord on a remote BT Device object to initiate comm.
- BluetoothServerSocket
  - Listen for incoming connection requests from Bluetooth Sockets on remote devices

BT basics

```
// Right now can only get default (most devices only have one)
BluetoothAdapter bluetooth = BluetoothAdapter.getDefaultAdapter();


// (Manifest): Manifest permission nodes
<uses-permission android:name="android.permission.BLUETOOTH"/>
// Modify local device BT settings:
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
```

BT basics

```
// Reading Bluetooth Adapter properties
BluetoothAdapter bluetooth = BluetoothAdapter.getDefaultAdapter();

String toastText;
if (bluetooth.isEnabled()) {
  String address = bluetooth.getAddress();
  String name = bluetooth.getName();
  toastText = name + " : " + address;
}
else
  toastText = "Bluetooth is not enabled";

Toast.makeText(this, toastText, Toast.LENGTH_LONG).show();

bluetooth.setName("Blackfang");
```

BT basics

```
// Enabling Bluetooth and tracking the adapter state
BluetoothAdapter bluetooth = BluetoothAdapter.getDefaultAdapter();

BroadcastReceiver bluetoothState = new BroadcastReceiver() {
  @Override
  public void onReceive(Context context, Intent intent) {
    String prevStateExtra = BluetoothAdapter.EXTRA_PREVIOUS_STATE;
    String stateExtra = BluetoothAdapter.EXTRA_STATE;
    int state = intent.getIntExtra(stateExtra, -1);
    int previousState = intent.getIntExtra(prevStateExtra, -1);

    String tt = "";
    switch (state) {
      case (BluetoothAdapter.STATE_TURNING_ON) : {
        tt = "Bluetooth turning on"; break;
      }
      case (BluetoothAdapter.STATE_ON) : {
        tt = "Bluetooth on";
        unregisterReceiver(this);
        break;
      }
      case (BluetoothAdapter.STATE_TURNING_OFF) : {
        tt = "Bluetooth turning off"; break;
      }
      case (BluetoothAdapter.STATE_OFF) : {
        tt = "Bluetooth off"; break;
      }
      default: break;
    }

    Toast.makeText(this, tt, Toast.LENGTH_LONG).show();
  }
};
```

BT basics

```
// Enabling Bluetooth and tracking the adapter state (continued)

if (!bluetooth.isEnabled()) {
  String actionStateChanged = BluetoothAdapter.ACTION_STATE_CHANGED;
  String actionRequestEnable = BluetoothAdapter.ACTION_REQUEST_ENABLE;
  registerReceiver(bluetoothState, new IntentFilter(actionStateChanged));
  startActivityForResult(new Intent(actionRequestEnable), 0);
}
```

- Also possible to enable and disable directly if BLUETOOTH_ADMIN set in manifest
  - Use sparingly
  - Better to get user to agree

Discovery modes

- SCAN_MODE_CONNECTABLE_DISCOVERABLE
  - Device discoverable from any BT device
- SCAN_MODE_CONNECTABLE
  - Devices that have previously connected and bonded to the local device can find it during discovery; others cannot
- SCAN_MODE_NONE
  - Turned off

Discovery modes

- By default, discoverable for 2 min (You can change this with an EXTRA_DISCOVERABLE_DURATION)
- Discovering other devices can take 12 seconds
  - Impacts other performance of BT...

Monitoring discovery modes

```
@Override
protected void onActivityResult(int requestCode,
                                int resultCode, Intent data) {
  if (requestCode == DISCOVERY_REQUEST) {
    boolean isDiscoverable = resultCode > 0;
    int discoverableDuration = resultCode;
  }
}

// Monitoring discoverability modes
registerReceiver(new BroadcastReceiver() {
  @Override
  public void onReceive(Context context, Intent intent) {
    String prevScanMode = BluetoothAdapter.EXTRA_PREVIOUS_SCAN_MODE;
    String scanMode = BluetoothAdapter.EXTRA_SCAN_MODE;
    int scanMode = intent.getIntExtra(scanMode, -1);
    int prevMode = intent.getIntExtra(prevScanMode, -1);
  }
}, new IntentFilter(BluetoothAdapter.ACTION_SCAN_MODE_CHANGED));
```

Monitoring discovery

```
// Monitoring discovery
BroadcastReceiver discoveryMonitor = new BroadcastReceiver() {

  String dStarted = BluetoothAdapter.ACTION_DISCOVERY_STARTED;
  String dFinished = BluetoothAdapter.ACTION_DISCOVERY_FINISHED;

  @Override
  public void onReceive(Context context, Intent intent) {
    if (dStarted.equals(intent.getAction())) {
      // Discovery has started.
      Toast.makeText(getApplicationContext(), "Started...", Toast.LENGTH_SHORT).show();
    }
    else if (dFinished.equals(intent.getAction())) {
      // Discovery has completed.
      Toast.makeText(getApplicationContext(),
                  "Discovery Completed...", Toast.LENGTH_SHORT).show();
    }
  }
};
registerReceiver(discoveryMonitor, new IntentFilter(dStarted));
registerReceiver(discoveryMonitor, new IntentFilter(dFinished));
```

Discovering remote BT devices

```
// Discovering remote Bluetooth Devices
BroadcastReceiver discoveryResult = new BroadcastReceiver() {
  @Override
  public void onReceive(Context context, Intent intent) {
    String remoteDeviceName =
      intent.getStringExtra(BluetoothDevice.EXTRA_NAME);
    BluetoothDevice remoteDevice;
    remoteDevice = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);

    Toast.makeText(getApplicationContext(),
                "Discovered: " + remoteDeviceName,
                Toast.LENGTH_SHORT).show();

    // TODO Do something with the remote Bluetooth Device.
  }
};
registerReceiver(discoveryResult,
                new IntentFilter(BluetoothDevice.ACTION_FOUND));

if (!bluetooth.isDiscovering())
  bluetooth.startDiscovery();
```

BT communcation

- BT communications API wrappers for RFCOMM (BT radio frequency communications protocol)
- RFCOMM supports RS232 serial communication
- In short, gives you communication sockets between two devices

RFCOMM

- BluetoothServerSocket
  - Listening socket for initiating link
- BluetoothSocket
  - Create a client socket to connect to a listening BT Server Socket (returned by ServerSocket once connection made)
  - Used on server and client side to transfer data

Listening BT socket conn. req.

```
private void requestBluetooth() {
  startActivityForResult(new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE),
                         DISCOVERY_REQUEST);}

protected void onActivityResult(int requestCode, int resultCode, Intent data) {
  if (requestCode == DISCOVERY_REQUEST) {
    boolean isDiscoverable = resultCode > 0;
    int discoverableDuration = resultCode;
    if (isDiscoverable) {
      UUID uuid = UUID.fromString("a60f35f0-b93a-11de-8a39-08002009c666");
      String name = "bluetoothserver";

      final BluetoothServerSocket btserver = bluetooth.listenUsingRfcommWithServiceRecord(name, uuid);

      Thread acceptThread = new Thread(new Runnable() {
        public void run() {
          try { // Block until client connection established.
            BluetoothSocket serverSocket = btserver.accept();
            // TODO Transfer data using the server socket
          } catch (IOException e) {
            Log.d("BLUETOOTH", e.getMessage());
        }}});
      acceptThread.start();
    }}}
```

To establish communication...

- Remote device must be discoverable
- Remote device must accept connections using a Bluetooth Server Socket
- Local and remote devices must be paired (if not, user will be prompted to pair when you initiate connection request)

Checking remote devices

```
// Checking remote devices for discoverability and pairing
final BluetoothDevice device = bluetooth.getRemoteDevice("01:23:77:35:2F:AA");
final Set<BluetoothDevice> bondedDevices = bluetooth.getBondedDevices();

BroadcastReceiver discoveryResult = new BroadcastReceiver() {
@Override
public void onReceive(Context context, Intent intent) {
  BluetoothDevice remoteDevice = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);

    if ((remoteDevice.equals(device) &&
        (bondedDevices.contains(remoteDevice)) {
      // TODO Target device is paired and discoverable
    }};

registerReceiver(discoveryResult, new IntentFilter(BluetoothDevice.ACTION_FOUND));

if (!bluetooth.isDiscovering())
  bluetooth.startDiscovery();
```

Connecting

```
// Connecting to a remote Bluetooth server

try{
  BluetoothDevice device = bluetooth.getRemoteDevice("00:23:76:35:2F:AA");
  BluetoothSocket clientSocket =
    device.createRfcommSocketToServiceRecord(uuid);
  clientSocket.connect();
  // TODO Transfer data using the Bluetooth Socket
} catch (IOException e) {
  Log.d("BLUETOOTH", e.getMessage());
}
```

Sending strings

```
// Sending and receiving strings using Bluetooth Sockets
private void sendMessage(String message){
  OutputStream outStream;
  try {
    outStream = socket.getOutputStream();

    // Add a stop character.
    byte[] byteArray = (message + " ").getBytes();
    byteArray[byteArray.length - 1] = 0;

    outStream.write(byteArray);
  } catch (IOException e) { }
}

private String listenForMessage()
  String result  = "";
  int bufferSize = 1024;
  byte[] buffer = new byte[bufferSize];
```

Sending strings

```
  try {
    InputStream instream = socket.getInputStream();
    int bytesRead = -1;

    while (true) {
      bytesRead = instream.read(buffer);
      if (bytesRead != -1) {
        while ((bytesRead == bufferSize) && (buffer[bufferSize-1] != 0)){
          message = message + new String(buffer, 0, bytesRead);
          bytesRead = instream.read(buffer);
        }
        message = message + new String(buffer, 0, bytesRead - 1);
                return result;
      }
    }

  } catch (IOException e) {}

  return result;
}
```

Managing network connection

- Monitor intents for changes in network connectivity
- APIs provide control over settings
- Users can specify connectivity preferences

Managing network connection

- ConnectivityManager
  - Represents the Network Connectivity Service
  - Monitor, configure, and control radios
  - Need ACCESS_NETWORK_STATE and CHANGE_NETWORK_STATE permissions
  - Can get user pref for background data transfers (enforced at app level)

    (Settings > Accounts and sync > Background)
    - Beware ... Mobile data bills!

Conectivity Manager

```
// Accessing the Connectivity Manager
String service = Context.CONNECTIVITY_SERVICE;
ConnectivityManager connectivity = (ConnectivityManager)getSystemService(service);

Boolean backgroundEnabled = connectivity.getBackgroundDataSetting();

registerReceiver(
  new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
      // Do something when the background data setting changes.
    },
  new
IntentFilter(ConnectivityManager.ACTION_BACKGROUND_DATA_SETTING_CHANGED));
```

Tips from the field

- Tips from using Android and Bluetooth communicating with hardware (Wockets) and HR monitor

- If you are connecting to multiple radios at the same time (multithreaded), you might want to serialize the requests

Tips from the field

- It takes time (> 5 seconds) for Bluetooth to be enabled/disabled/change states. Make sure to verify (via polling or callbacks) that the radio is in the state you think it is in.
- Always verify the Bluetooth state is what you think it is. The user (or other apps) can change settings from underneath your app at any time.

Tips from the field

- Make sure you have the Bluetooth permissions (there are 2, one for using Bluetooth and 1 for administrating Bluetooth for adding new devices, etc)

Tips from the field

- Make sure to restore Bluetooth state to where you found it before changing any settings. If the user left it on for a hands free device, then used it to do something with your app, you want to make sure you don't just switch it off when your app is done so the user can no longer use their hands free device. Same thing in reverse, if the user had it off to save power, make sure you turn if off when done.

Tips from the field

- Android uses BlueZ (BT stack for Linux) and under Ubuntu Linux, Fahd has found that if you write an application with multiple threads to manage 14+ connections, it does not work as well as forking processes to manage the 14+ connections: a case for actually having multiple processes managing connections ... ugly but was more robust on Linux

Tips from the field

- At some point, I got better luck on some phones with reflection rather than using the standard API... specifically data arrived more smoothly not sure why...

Tips from the field

- When allocating and deallocating objects, be very cautious with memory leaks...
  - Use the heap features in Eclipse to track how your heap is doing
  - Especially when you are disconnecting/reconnecting etc.
- Log everything to be able to identify the problems.

Tips from the field

- When connecting to 1 radio, worked fine on all tested phone. When connecting to 2 radios, on some phones (GalaxyS 2.1) the connections alternated slowly and sometimes created data loss.
- Make sure to verify that you are not losing data (e.g. send consecutive numbers and look for gaps).

Tips from the field

- Pay special attention to accurate timestamping. Run tests and make sure you can recover the signal structure as accurately as possible

- RSSI noisy but does work well enough to consider doing some interesting things with indoor location.

Tips from the field

- If you run into trouble, or discover something great, email it and share it with everyone...

Accessing network info

```
// Get the active network information.
NetworkInfo activeNetwork = connectivity.getActiveNetworkInfo();
int networkType = networkInfo.getType();
switch (networkType) {
  case (ConnectivityManager.TYPE_MOBILE) : break;
  case (ConnectivityManager.TYPE_WIFI) : break;
  default: break;
}

// Get the mobile network information.
int network = ConnectivityManager.TYPE_MOBILE;
NetworkInfo mobileNetwork = connectivity.getNetworkInfo(network);
NetworkInfo.State state = mobileNetwork.getState();
NetworkInfo.DetailedState detailedState = mobileNetwork.getDetailedState();
```

Get/set network

```
// Find and set preferred network
int networkPref = connectivity.getNetworkPreference();
Connectivity.setNetworkPreference(NetworkPreference.PREFER_WIFI);


// Control the availability of network types  (turn off WiFi, turn on mobile)
connectivity.setRadio(NetworkType.WIFI, false);
connectivity.setRadio(NetworkType.MOBILE, true);
```

- Using a BroadcastReceiver, you can monitor for
  - Failover from preferred network, no connection, connection failure messages, reasons for possible failover

WiFi manager

```
// Accessing the Wi-Fi Manager
String service = Context.WIFI_SERVICE;
WifiManager wifi = (WifiManager)getSystemService(service);

// ********************************************************************
// (Manifest): Wi-Fi Manager WiFi Permissions
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>

// ********************************************************************
// Monitoring and changing Wi-Fi state
if (!wifi.isWifiEnabled())
  if (wifi.getWifiState() != WifiManager.WIFI_STATE_ENABLING)
    wifi.setWifiEnabled(true);
```

WiFi

- You can monitor
  - Wi-Fi hardware status changes
  - Connection point changes
    - Mac address, IP address, etc.
  - Connectivity state changes
  - **Signal strength changes**