Connectivity

GVHD: Trần Hồng Nghi
Các thành viên nhóm:
- **Huỳnh Ngọc Thạch-13520757**
- **Nguyễn Xuân Dưỡng- 13520147**

Developers
Connectivity

Giới thiệu

Nội dung chính:

- Wifi Peer-To-Peer
- Bluetooth
- Usb
- …..

Wifi Peer-To-Peer

Giới thiệu:

- Wifi peer-to-peer hỗ trợ các phiên bản android 4.0 trở về sau với các phần cứng thích hợp kết nối trực tiếp với nhau không cần thông qua AP.
- Tốc độ kết nối, khoảng cách kết nối tốt hơn so với Bluetooth.

Developers

Connectivity

Wifi Peer-To-Peer

Các thành phần chính của wifi P2P APIs:

- Phương thức cho phép tìm kiếm, yêu cầu và kết nối giữa 2 thiết bị được định nghĩa trong lớp **WifiP2pManager.**
- Các thành phần lắng nghe (Listeners) trong class **WifiP2pManager.**
- Các intent được sử dụng để xác định loại sự kiện mà wifi P2P framework phát hiện được.

Wifi Peer-To-Peer

Tổng quan về wifi P2P API:

Các phương thức	Mô tả
initialize()	Đăng kí ứng dụng với wifi framework, được gọi trước khi các phương thức khác của wifi P2P được gọi.
connect()	Bắt đầu một kết nối p2p với một thiết bị nào đó.
cancelConnect()	Hủy bỏ kết nối.
requestConnectInfo()	Yêu cầu thông tin kết nối của một thiết bị nào đó.
createGroup()	Tạo một P2P group.
removeGroup()	Xóa một group p2p đang có.
requestGroupInfo()	Yêu cầu thông tin của một group p2p nào đó.
discoverPeers()	Khởi tạo quá trình tìm kiếm.
requestPeers()	Yêu cầu danh sách các thiết bị hiện đã được tìm thấy.

Wifi Peer-To-Peer

Tổng quan về wifi P2P API:

- Wifi p2p Listeners:

Listener interface	Đi kèm với phương thức
WifiP2pManager.ActionListener	connect(), cancelConnect(), createGroup(), removeGroup(), discoverPeers()
WifiP2pManager.ChannelListener	initialize()
WifiP2pManager.ConnectionInfoListener	requestConnectInfo()
WifiP2pManager.GroupInfoListener	requestGroupInfo()
WifiP2pManager.PeerListListener	requestPeers()

Wifi Peer-To-Peer

Tổng quan về wifi P2P API:

- Wifi p2p Intent:

Intent	Mô tả
WIFI_P2P_CONNECTION_CHANGED_ACTION	Broadcast khi trạng thái kết nối wifi của thiết bị thay đổi
WIFI_P2P_PEERS_CHANGED_ACTION	Broadcast khi gọi discoverPeers()
WIFI_P2P_STATE_CHANGED_ACTION	Broadcast khi wifi p2p được bật hay bị tắt
WIFI_P2P_THIS_DEVICE_CHANGED_ACTION	Broadcast khi thông tin thiết bị thay đổi

Developers
Connectivity

Wifi Peer-To-Peer

Broadcast Receiver:

```
public class WiFiDirectBroadcastReceiver extends BroadcastReceiver {

    private WifiP2pManager mManager;
    private Channel mChannel;
    private MyWiFiActivity mActivity;

    public WiFiDirectBroadcastReceiver(WifiP2pManager manager, Channel channel,
            MyWifiActivity activity) {
        super();
        this.mManager = manager;
        this.mChannel = channel;
        this.mActivity = activity;
    }

    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();

        if (WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION.equals(action)) {
            // Check to see if Wi-Fi is enabled and notify appropriate activity
        } else if (WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION.equals(action)) {
            // Call WifiP2pManager.requestPeers() to get a list of current peers
        } else if (WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION.equals(action)) {
            // Respond to new connection or disconnections
        } else if (WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION.equals(action)) {
            // Respond to this device's wifi state changing
        }
    }
}
```

- Tạo một lớp mới kế thừa từ lớp **BroadcastReceiver()**
- Trong hàm **onReceive()** kiểm tra những intent cần thiết của API wifi p2p để xử lí khi có các sự kiện xảy ra

Wifi Peer-To-Peer

Khai báo các permission trong AndroidManifest.xml

```
<uses-sdk android:minSdkVersion="14" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Wifi Peer-To-Peer

Kiểm tra xem thiết bị có hỗ trợ wifi P2P không:

```
@Override
public void onReceive(Context context, Intent intent) {
    ...
    String action = intent.getAction();
    if (WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION.equals(action)) {
        int state = intent.getIntExtra(WifiP2pManager.EXTRA_WIFI_STATE, -1);
        if (state == WifiP2pManager.WIFI_P2P_STATE_ENABLED) {
            // Wifi P2P is enabled
        } else {
            // Wi-Fi P2P is not enabled
        }
    }
    ...
}
```

Wifi Peer-To-Peer

Lấy về thể hiện của lớp WifiP2pManager và đăng kí ứng dụng với wifi API framework:

```
WifiP2pManager mManager;
Channel mChannel;
BroadcastReceiver mReceiver;
...
@Override
protected void onCreate(Bundle savedInstanceState){
    ...
    mManager = (WifiP2pManager) getSystemService(Context.WIFI_P2P_SERVICE);
    mChannel = mManager.initialize(this, getMainLooper(), null);
    mReceiver = new WiFiDirectBroadcastReceiver(mManager, mChannel, this);
    ...
}
```

- Tạo ra một thể hiện của lớp **WifiP2pManager** trong hàm **onCreate()** của activity
- Đăng kí ứng dụng với wifi p2p framework thông qua cách gọi hàm **initialize()**
- Tạo mới một broadcast receiver để nhận các thông báo

Wifi Peer-To-Peer

Tạo các intent filter cho broadcast receiver của ứng dụng:

```
IntentFilter mIntentFilter;
...
@Override
protected void onCreate(Bundle savedInstanceState){
    ...
    mIntentFilter = new IntentFilter();
    mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION);
    mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION);
    mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION);
    mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION);
    ...
}
```

Wifi Peer-To-Peer

Đăng kí/hủy đăng kí các broadcast receiver:

```
/* register the broadcast receiver with the intent values to be matched */
@Override
protected void onResume() {
    super.onResume();
    registerReceiver(mReceiver, mIntentFilter);
}
/* unregister the broadcast receiver */
@Override
protected void onPause() {
    super.onPause();
    unregisterReceiver(mReceiver);
}
```

Wifi Peer-To-Peer

Dò tìm thiết bị kết nối:

- Gọi phương thức **discoverPeers()**
- Hệ thống sẽ phát ra intent **WIFI_P2P_PEERS_CHANGED_ACTION**
- **Gọi requestPeers()**

Wifi Peer-To-Peer

Dò tìm thiết bị kết nối:

```
mManager.discoverPeers(channel, new WifiP2pManager.ActionListener() {
    @Override
    public void onSuccess() {
        ...
    }

    @Override
    public void onFailure(int reasonCode) {
        ...
    }
});
```

```
PeerListListener myPeerListListener;
...
if (WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION.equals(action)) {

    // request available peers from the wifi p2p manager. This is an
    // asynchronous call and the calling activity is notified with a
    // callback on PeerListListener.onPeersAvailable()
    if (mManager != null) {
        mManager.requestPeers(mChannel, myPeerListListener);
    }
}
```

Wifi Peer-To-Peer

Kết nối đến thiết bị:

```
//obtain a peer from the WifiP2pDeviceList
WifiP2pDevice device;
WifiP2pConfig config = new WifiP2pConfig();
config.deviceAddress = device.deviceAddress;
mManager.connect(mChannel, config, new ActionListener() {

    @Override
    public void onSuccess() {
        //success logic
    }

    @Override
    public void onFailure(int reason) {
        //failure logic
    }
});
```

Wifi Peer-To-Peer

Truyền gửi dữ liệu

Sau khi thiết lập xong kết nối hai thiết bị có thể truyền gửi dữ liệu cho nhau bằng mô hình client-server.

Developers

Connectivity

Wifi Peer-To-Peer

Server:

```
@Override
protected String doInBackground(Void... params) {
    try {

        /**
         * Create a server socket and wait for client connections. This
         * call blocks until a connection is accepted from a client
         */
        ServerSocket serverSocket = new ServerSocket(8888);
        Socket client = serverSocket.accept();

        /**
         * If this code is reached, a client has connected and transferred data
         * Save the input stream from the client as a JPEG file
         */
        final File f = new File(Environment.getExternalStorageDirectory() + "/"
                + context.getPackageName() + "/wifip2pshared-" + System.currentTimeMillis()
                + ".jpg");

        File dirs = new File(f.getParent());
        if (!dirs.exists())
            dirs.mkdirs();
        f.createNewFile();
        InputStream inputstream = client.getInputStream();
        copyFile(inputstream, new FileOutputStream(f));
        serverSocket.close();
        return f.getAbsolutePath();
    } catch (IOException e) {
        Log.e(WiFiDirectActivity.TAG, e.getMessage());
        return null;
    }
}
```

Developers

Connectivity

Wifi Peer-To-Peer

Client:

```
try {
    /**
     * Create a client socket with the host,
     * port, and timeout information.
     */
    socket.bind(null);
    socket.connect((new InetSocketAddress(host, port)), 500);

    /**
     * Create a byte stream from a JPEG file and pipe it to the output stream
     * of the socket. This data will be retrieved by the server device.
     */
    OutputStream outputStream = socket.getOutputStream();
    ContentResolver cr = context.getContentResolver();
    InputStream inputStream = null;
    inputStream = cr.openInputStream(Uri.parse("path/to/picture.jpg"));
    while ((len = inputStream.read(buf)) != -1) {
        outputStream.write(buf, 0, len);
    }
    outputStream.close();
    inputStream.close();
} catch (FileNotFoundException e) {
    //catch logic
} catch (IOException e) {
    //catch logic
}

/**
```

Bluetooth

Tổng quan:

- Bluetooth android cho phép các thiết bị di động có thể trao đổi dữ liệu không dây với nhau.

Các chức năng cơ bản của Bluetooth Api:

- Quét các thiết bị Bluetooth khác trong phạm vi cho phép
- Thiết lập kênh RFCOMM
- Kết nối đến thiết bị khác thông qua dịch vụ dò tìm
- Truyền dữ liệu với các thiết bị khác
- Quản lí kết nối

Bluetooth

Bluetooth permissions:

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

Bluetooth

Những cấu hình đầu tiên:

- Lấy về đối tượng **BluetoothAdapter**: đây là thành phần bắt buộc để có thể thực hiện các chức năng của Bluetooth.

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

Bluetooth

Những cấu hình đầu tiên:

Trước khi thực hiện các chức năng với Bluetooth chúng ta sẽ phải kiểm tra thiết bị có hỗ trợ bluetooth hay không, Bluetooth đã được bật hay chưa.

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

Bluetooth

- **Tìm kiếm các thiết bị.**
- **Truy vấn các thiết bị đã được ghép nối trước đó:**

Sử dụng phương thức **getBonedDevices()** để lấy về danh sách các thiết bị đã từng ghép đôi, sau đó hiển thị lên cho người dùng:

```
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());
    }
}
```

Bluetooth

Tìm kiếm các thiết bị trong phạm vi:

- Sử dụng hàm **startDiscovery()** để tiến hành tìm các thiết bị
- Đăng kí với broadcast receiver.

```
// 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());
        }
    }
};
// Register the BroadcastReceiver
IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
registerReceiver(mReceiver, filter); // Don't forget to unregister during onDestroy
```

Bluetooth

Bật tính năng có thể được tìm thấy:

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

Bluetooth

Kết nối đến các thiết bị:

- Cài đặt cả chức năng client và server.
- Client và server kết nối qua kênh **RFCOMM**
- Khi kết nối thành công trao đổi dữ liệu với nhau
- Cách lấy về đối tượng **BluetoothSocket** của client và server là khác nhau:
  - Server nhận được khi có một yêu cầu kết nối đến từ client được chấp nhận
  - Client nhận được khi nó mở kênh RFCOMM kết nối đến server

Bluetooth

Server:

- Server có nhiệm vụ lắng nghe, chấp nhận các kết nối đến từ client
- Các bước cài đặt một server socket lắng nghe:
  - Lấy về đối tượng BluetoothServerSocket bằng cách gọi hàm **listenUsingRfcommWithServiceRecord(String, UUID)**
  - Bắt đầu lắng nghe các yêu cầu từ client với phương thức **access()**
  - Giải phóng server:
    - ✓ Nếu không còn muốn nhận thêm kết nối nào khác
    - ✓ Sử dụng phương thức **close()**

Bluetooth

```
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 = mBluetoothAdapter.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;
            }
        }
    }
```

Bluetooth

Client:

- Các client có nhiệm vụ khởi tạo các yêu cầu kết nối đến server nào đó
- Các bước cơ bản:
  - Sử dụng đối tượng **BluetoothDevice** (là đối tượng đại diện cho client) lấy về đối tượng socket BluetoothSocket bằng cách gọi phương thức **createRfcommSocketToServiceRecord(UUID)**
  - Khởi tạo kết nối đến server bằng cách dùng phương thức connect()

Developers

Connectivity

Bluetooth

```
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
        mBluetoothAdapter.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);
    }
```

Bluetooth

Quản lí các kết nối:

Sau khi hoàn thành xong thao tác kết nối chúng ta có thể chia sẻ dữ liệu giữa hai thiết bị với nhau

- Lấy về các đối tượng **InputStream** hoặc **OutputStream** thông qua hai phương thức tương ứng là **getInputStream** và **getOutputStream**
- Đọc và ghi dữ liệu thông qua 2 stream đó

Bluetooth

```
private class ConnectedThread extends Thread {
    private final BluetoothSocket mmSocket;
    private final InputStream mmInStream;
    private final OutputStream mmOutStream;

    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();
            tmpOut = socket.getOutputStream();
        } catch (IOException e) { }

        mmInStream = tmpIn;
        mmOutStream = tmpOut;
    }

    public void run() {
        byte[] buffer = new byte[1024];  // buffer store for the stream
        int bytes; // bytes returned from read()

        // Keep listening to the InputStream until an exception occurs
        while (true) {
            try {
                // Read from the InputStream
                bytes = mmInStream.read(buffer);
                // Send the obtained bytes to the UI activity
                mHandler.obtainMessage(MESSAGE_READ, bytes, -1, buffer)
                        .sendToTarget();
            } catch (IOException e) {
                break;
            }
        }
    }
```

```
    /* Call this from the main activity to send data to the remote device */
    public void write(byte[] bytes) {
        try {
            mmOutStream.write(bytes);
        } catch (IOException e) { }
    }

    /* Call this from the main activity to shutdown the connection */
    public void cancel() {
        try {
            mmSocket.close();
        } catch (IOException e) { }
    }
}
```

Usb

Tổng quan:

Android hỗ trợ nhiều loại kết nối usb thông qua hai chế độ:

- Chế độ **usb accessory**
- Chế độ **usb host**

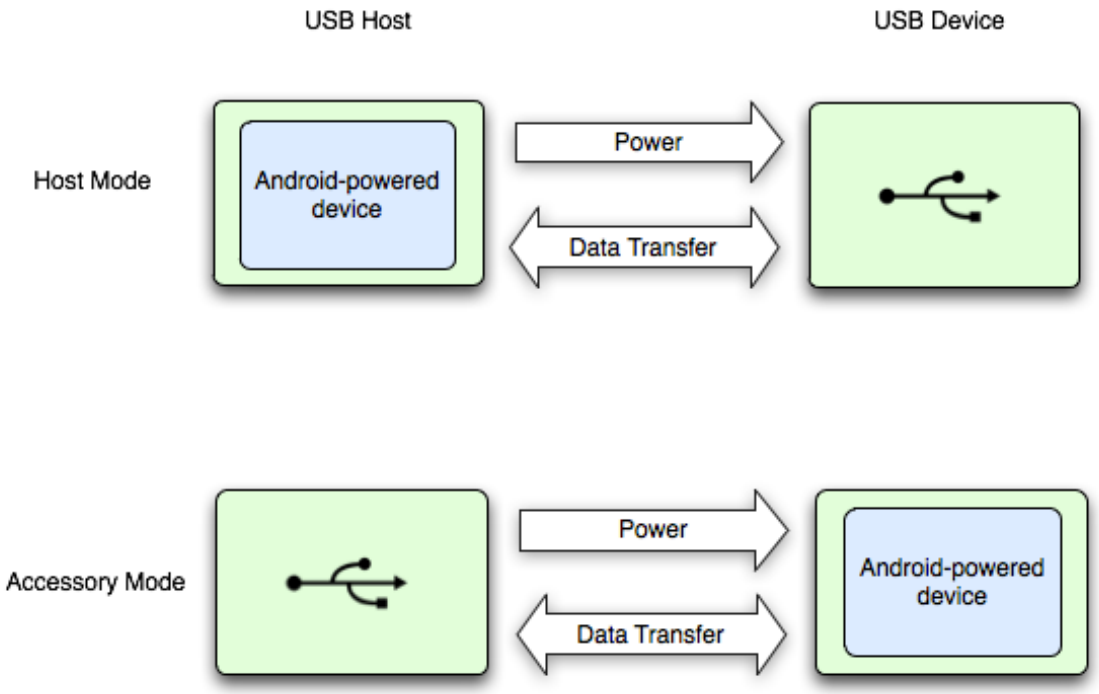


Figure 1. USB Host and Accessory Modes

USB Accessory

Accessory Api:

Accessory api được giới thiệu chính thức ở phiên bản android 3.1, nhưng chúng ta vẫn có thể sử dụng được ở phiên bản 2.3.4.

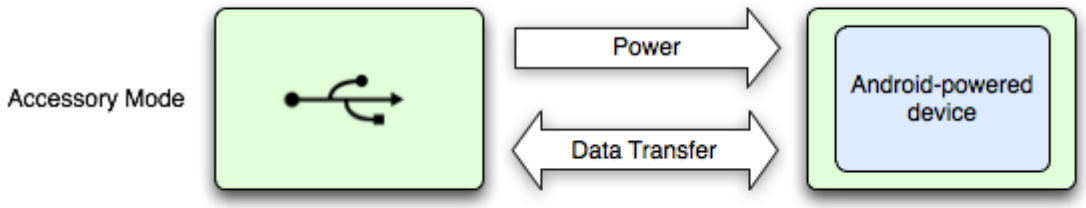
Có hai cách để hỗ trợ chế độ usb accesscory: **com.android.future.usb và android.hardware.usb**

USB Accessory

Tổng quan về API:

Hai lớp hỗ trợ usb accessory mode:

- **UsbManager**
- **UsbAccessory**



USB Accessory

Khác nhau giữa add-on library và platform api trong thao tác với UsbManager và UsbAccessory:

- Lấy về đối tượng **UsbManager** sử dụng add-on library

```
UsbManager manager = UsbManager.getInstance(this);
```

- Lấy về đối tượng **UsbManager** sử dụng api platform

```
UsbManager manager = (UsbManager) getSystemService(Context.USB_SERVICE);
```

- Lấy về đối tượng **UsbAccessory** sử dụng add-on library

```
UsbAccessory accessory = UsbManager.getAccessory(intent);
```

- Lấy về đối tượng **UsbAccessory** sử dụng api platform

```
UsbAccessory accessory = (UsbAccessory) intent.getParcelableExtra(UsbManager.EXTRA_ACCESSORY);
```

USB Accessory

AndroidManifest.xml:

- Bởi vì không phải thiết bị nào cũng hỗ trợ usb accessory nên chúng ta sẽ sử sụng khai báo <uses-feature android:name="android.hardware.usb.accessory" />
- Nếu sử dụng add-on library thì thay bằng

<uses-library android:name="com.android.future.usb.accessory" />

Developers

Connectivity

USB Accessory

AndroidManifest.xml

```
<manifest ...>
    <uses-feature android:name="android.hardware.usb.accessory" />

    <uses-sdk android:minSdkVersion="<version>" />
    ...
    <application>
      <uses-library android:name="com.android.future.usb.accessory" />
        <activity ...>
            ...
            <intent-filter>
                <action android:name="android.hardware.usb.action.USB_ACCESSORY_ATTACHED" />
            </intent-filter>

            <meta-data android:name="android.hardware.usb.action.USB_ACCESSORY_ATTACHED"
                android:resource="@xml/accessory_filter" />
        </activity>
    </application>
</manifest>
```

Accessory_filter.xml:

```
<?xml version="1.0" encoding="utf-8"?>

<resources>
    <usb-accessory model="DemoKit" manufacturer="Google" version="1.0"/>
</resources>
```

USB Accessory

Dò tìm các thiết bị usb accessory:

Có thể dò tìm bằng cách sử dụng các intent filter có nhiệm vụ thông báo khi có thiết bị kết nối hoặc thực hiện dò tìm các thiết bị hiện đang kết nối

- **Sử dụng intent filter:**
  - ✓ file AndroidManifest.xml

```
<activity ...>
    ...
    <intent-filter>
        <action android:name="android.hardware.usb.action.USB_ACCESSORY_ATTACHED" />
    </intent-filter>

    <meta-data android:name="android.hardware.usb.action.USB_ACCESSORY_ATTACHED"
        android:resource="@xml/accessory_filter" />
</activity>
```

Developers
Connectivity

USB Accessory

- **sử dụng intent filter:**
  - ✓File usbAcccessory_filter.xml:

```
<?xml version="1.0" encoding="utf-8"?>

<resources>
    <usb-accessory manufacturer="Google, Inc." model="DemoKit" version="1.0" />
</resources>
```

  - ✓Ở fle activity cần lấy về đối tượng UsbAccessory bằng 1 trong 2 cách sau:

```
UsbAccessory accessory = UsbManager.getAccessory(intent);
```

Hoặc:

```
UsbAccessory accessory = (UsbAccessory)intent.getParcelableExtra(UsbManager.EXTRA_ACCESSORY);
```

USB Accessory

- **Dò tìm các thiết bị đang kết nối:**

Bằng cách sử dụng phương thức getAccessoryList() ta có thể lấy về mảng các thiết bị đang kết nối:

```
UsbManager manager = (UsbManager) getSystemService(Context.USB_SERVICE);
UsbAccessory[] accessoryList = manager.getAcccessoryList();
```

USB Accessory

Đạt được quyền giao tiếp với các thiết bị:

- Trước khi có thể dùng ứng dụng giao tiếp với các thiết bị usb accessory chúng ta phải hỏi người sử dụng có cho phép hay không.
- Khi ứng dụng sử dụng các intent filter để thông báo khi có thiết bị kết nối thì ứng dụng sẽ tự động được cấp quyền bởi user, nếu không sử dụng thì ta phải yêu cầu quyền bằng cách tường minh

Developers
Connectivity

USB Accessory

broadcastReceiver:

```
private static final String ACTION_USB_PERMISSION =
    "com.android.example.USB_PERMISSION";
private final BroadcastReceiver mUsbReceiver = new BroadcastReceiver() {

    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if (ACTION_USB_PERMISSION.equals(action)) {
            synchronized (this) {
                UsbAccessory accessory = (UsbAccessory) intent.getParcelableExtra(UsbManager.EXTRA_ACCESSORY);

                if (intent.getBooleanExtra(UsbManager.EXTRA_PERMISSION_GRANTED, false)) {
                    if(accessory != null){
                        //call method to set up accessory communication
                    }
                }
                else {
                    Log.d(TAG, "permission denied for accessory " + accessory);
                }
            }
        }
    }
};
```

Developers
Connectivity

USB Accessory

Đăng kí broadcast receiver trong hàm onCreate của activity:

```
UsbManager mUsbManager = (UsbManager) getSystemService(Context.USB_SERVICE);
private static final String ACTION_USB_PERMISSION =
    "com.android.example.USB_PERMISSION";
...
mPermissionIntent = PendingIntent.getBroadcast(this, 0, new Intent(ACTION_USB_PERMISSION), 0);
IntentFilter filter = new IntentFilter(ACTION_USB_PERMISSION);
registerReceiver(mUsbReceiver, filter);
```

Gọi phương thức requestPermission

```
UsbAccessory accessory;
...
mUsbManager.requestPermission(accessory, mPermissionIntent);
```

USB Accessory

Giao tiếp với các usb accessory:

```
UsbAccessory mAccessory;
ParcelFileDescriptor mFileDescriptor;
FileInputStream mInputStream;
FileOutputStream mOutputStream;

...

private void openAccessory() {
    Log.d(TAG, "openAccessory: " + accessory);
    mFileDescriptor = mUsbManager.openAccessory(mAccessory);
    if (mFileDescriptor != null) {
        FileDescriptor fd = mFileDescriptor.getFileDescriptor();
        mInputStream = new FileInputStream(fd);
        mOutputStream = new FileOutputStream(fd);
        Thread thread = new Thread(null, this, "AccessoryThread");
        thread.start();
    }
}
```

USB Accessory

Sự kiện ngắt kết nối:

```
BroadcastReceiver mUsbReceiver = new BroadcastReceiver() {
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();

        if (UsbManager.ACTION_USB_ACCESSORY_DETACHED.equals(action)) {
            UsbAccessory accessory = (UsbAccessory)intent.getParcelableExtra(UsbManager.EXTRA_ACCESSORY);
            if (accessory != null) {
                // call your method that cleans up and closes communication with the accessory
            }
        }
    }
};
```

USB Host

Tổng quan về API:

Class	Mô tả
UsbManager	Cho phép thu thập thông tin, giao tiếp với các thiết bị usb
UsbDevice	Đại diện cho một usb
UsbInterface	Đại diện cho interface của thiết bị usb, 1 thiết bị có thể có nhiều interface
UsbDeviceConnection	Đại diện cho một kết nối đến thiết bị usb
UsbRequest	Chứa các phương thức bất đồng bộ giao tiếp với thiết bị
……	

USB Host

Dò tìm các thiết bị kết nối:

- **Sử dụng intent filter tự động thông báo các thiết bị đang kết nối:**
  - ✓ AndroidManifest.xml

```
<activity ...>
...
    <intent-filter>
        <action android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED" />
    </intent-filter>

    <meta-data android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED"
        android:resource="@xml/device_filter" />
</activity>
```

Developers
Connectivity

USB Host

- **Sử dụng intent filter:**
  - ✓File device_filter.xml

```xml
<?xml version="1.0" encoding="utf-8"?>

<resources>
    <usb-device vendor-id="1234" product-id="5678" />
</resources>
```

  - ✓Lấy đối tượng usb đang kết nối:

```
UsbDevice device = (UsbDevice) intent.getParcelableExtra(UsbManager.EXTRA_DEVICE);
```

USB Host

- **Dò tìm các thiết bị đang kết nối:**

```
UsbManager manager = (UsbManager) getSystemService(Context.USB_SERVICE);
...
HashMap<String, UsbDevice> deviceList = manager.getDeviceList();
UsbDevice device = deviceList.get("deviceName");
```

USB Host

Đạt được quyền giao tiếp với các thiết bị:

- Trước khi có thể dùng ứng dụng giao tiếp với các thiết bị usb chúng ta phải hỏi người sử dụng có cho phép hay không
- Khi ứng dụng sử dụng các intent filter để thông báo khi có thiết bị kết nối thì ứng dụng sẽ tự động được cấp quyền bởi user, nếu không sử dụng thì ta phải yêu cầu quyền bằng cách tường minh

USB Host

BroadcastReceiver:

```
private static final String ACTION_USB_PERMISSION =
    "com.android.example.USB_PERMISSION";
private final BroadcastReceiver mUsbReceiver = new BroadcastReceiver() {

    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if (ACTION_USB_PERMISSION.equals(action)) {
            synchronized (this) {
                UsbDevice device = (UsbDevice)intent.getParcelableExtra(UsbManager.EXTRA_DEVICE);

                if (intent.getBooleanExtra(UsbManager.EXTRA_PERMISSION_GRANTED, false)) {
                    if(device != null){
                      //call method to set up device communication
                   }
                }
                else {
                    Log.d(TAG, "permission denied for device " + device);
                }
            }
        }
    }
};
```

Developers
Connectivity

USB Host

Đăng kí broadcastReceiver:

```
UsbManager mUsbManager = (UsbManager) getSystemService(Context.USB_SERVICE);
private static final String ACTION_USB_PERMISSION =
    "com.android.example.USB_PERMISSION";
...
mPermissionIntent = PendingIntent.getBroadcast(this, 0, new Intent(ACTION_USB_PERMISSION), 0);
IntentFilter filter = new IntentFilter(ACTION_USB_PERMISSION);
registerReceiver(mUsbReceiver, filter);
```

Yêu cầu quyền:

```
UsbDevice device;
...
mUsbManager.requestPermission(device, mPermissionIntent);
```

USB Host

Giao tiếp với thiết bị usb:

Để giao tiếp tốt với các thiết bị usb chúng ta có thể thực hiện theo trình tự sau:

- Kiểm tra đối tượng **UsbDevice** các thông số về product ID, vendor ID, device class để xác định xem ta có muốn thực hiện giao tiếp hay không?
- Khi đã chắc chắn muốn giao tiếp với thiết bị tiến hành tìm **UsbInterface** thích hợp kèm với một **UsbEndpoint** thích hợp
- Thực hiện trao đổi dữ liệu dựa vào phương thức **Transfer()** hay **controlTransfer()**

USB Host

Kết thúc giao tiếp với thiết bị:

```
BroadcastReceiver mUsbReceiver = new BroadcastReceiver() {
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();

      if (UsbManager.ACTION_USB_DEVICE_DETACHED.equals(action)) {
            UsbDevice device = (UsbDevice)intent.getParcelableExtra(UsbManager.EXTRA_DEVICE);
            if (device != null) {
                // call your method that cleans up and closes communication with the device
            }
        }
    }
};
```

Thank you for listening!