

# Device Connect 1.0

Android デバイスプラグイン開発チュートリアル

1.0.0 版

2014 年 9 月 24 日

変更履歴

変更日付	変更内容	変更担当者
2014/09/24	初版作成。	畠山

## 目次

1. はじめに.....	4
1.1 本チュートリアルで開発するもの .....	5
1.2 用語定義 .....	6
1.3 Device Connect との連携 .....	7
2. 開発環境の構築 .....	8
2.1 Eclipse のダウンロード .....	8
2.1.1 dConnectDevicePluginSDK のインポート .....	8
2.1.2 dConnectSDKAndroid のインポート .....	11
3. デバイスプラグインの作成 .....	13
3.1 Android プロジェクトの作成.....	13
3.2 パッケージの作成.....	14
3.3 dConnectDevicePluginSDK ライブラリプロジェクトの追加.....	16
3.4 DConnectMessageServiceProvider の実装 .....	17
3.5 deviceplugin.xml のフォーマット.....	23
3.6 DConnectMessageService の実装.....	25
3.7 各プロファイルの実装と DConnectMessgeService への追加 .....	29
3.8 Network Service Discovery プロファイルの実装.....	31
3.9 System プロファイルの実装.....	34
3.10 プロファイルの追加 .....	37
3.11 設定画面の作成.....	38
3.12 Fragment の作成 .....	43
3.13 設定画面の動作確認 .....	48
4. デバイスプラグインの動作確認.....	49
4.1 デバイスの検索 .....	49
4.2 Profile の実行.....	53
5. APPENDIX.....	58
5.1 非同期の処理について.....	58
5.2 常駐 Service を使用しない軽量プラグイン .....	60

## 1. はじめに

Device Connect システムは、マルチ OS、マルチプラットフォームのランタイム環境上において、スマートデバイスと接続するための API (RESTful) を提供する。

これにより、スマートデバイスとの接続方法・連携方法の利便性を向上することを目的としている。

Device Connect システムが提供する機能一覧は以下の通りである。

- ・ 連携可能な周辺機器一覧を表示する機能を提供
- ・ 接続 I/F (Bluetooth, BLE, Wi-Fi, NFC) の違いのわかりにくさを解消
- ・ 機器プロファイルによる統一的な API の提供

本書は、Device Connect Manager とスマートデバイスを接続するためのデバイスプラグイン (Android 版) を作成するためのチュートリアルである。

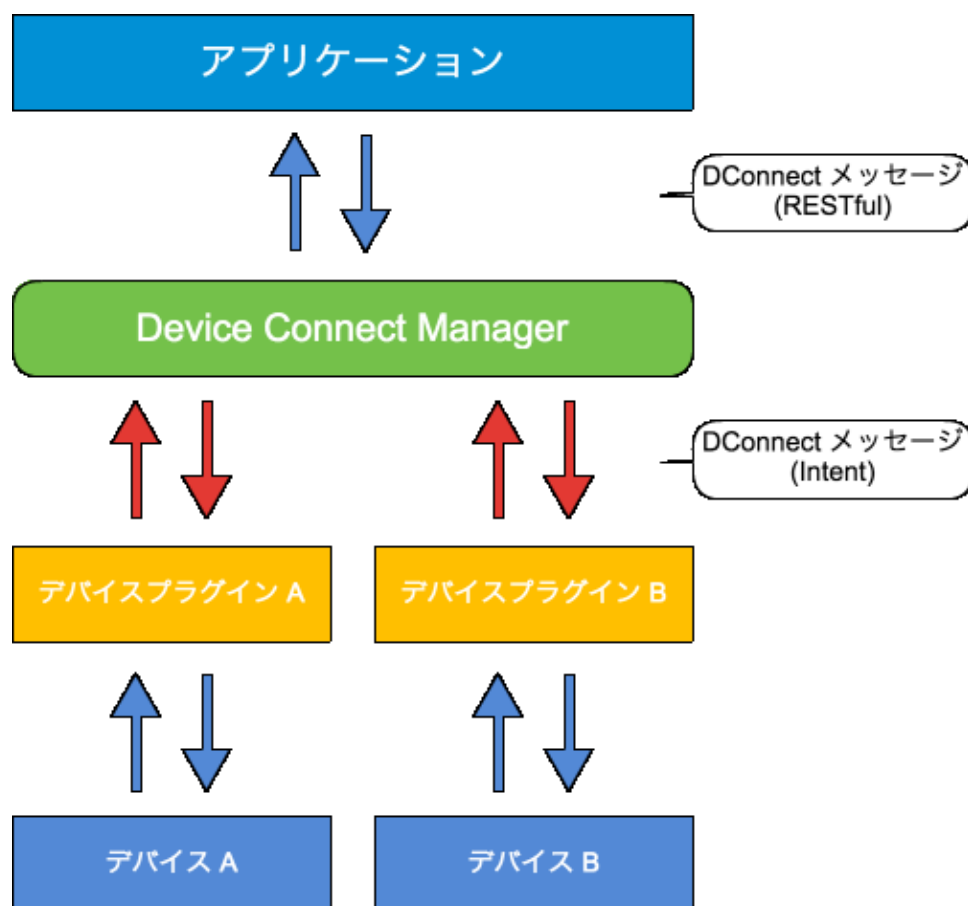


図.1 Device Connect システム概要図

## 1.1 本チュートリアルで開発するもの

本チュートリアルでは、

- ・ `NetworkServiceDiscoveryProfile`
- ・ `SystemProfile`

を用いた簡単なデバイスプラグインのサンプルを作成する。

2つのプロファイルを組み込んだサンプルに必要なソースコードは、以下により構成される。

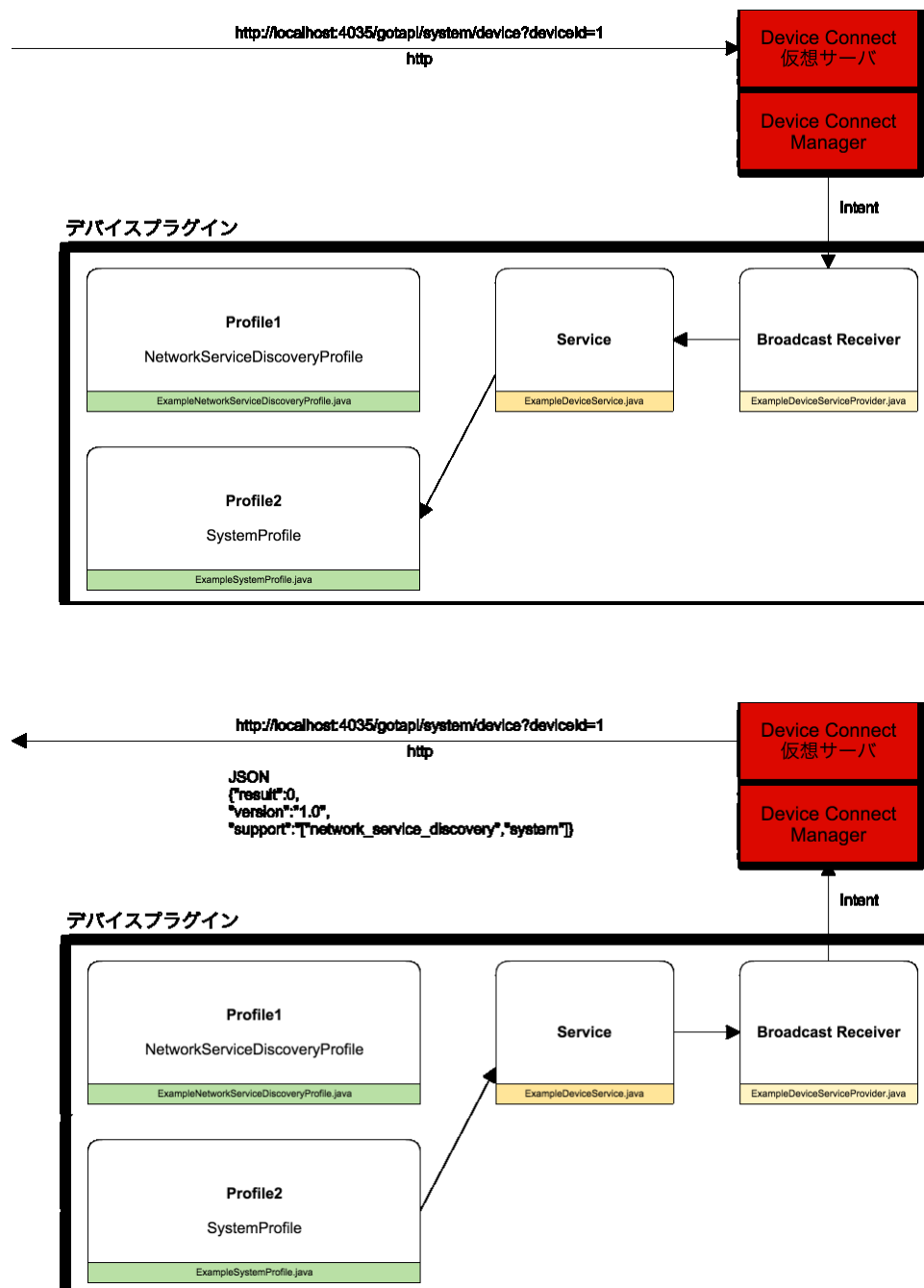
- ・ `ExampleDeviceProvider.java`  
(RESTful の処理を Device Connect 経由で受け取る Broadcast Receiver)
- ・ `ExampleDeviceService.java`  
(受け取った処理を行う Service)
- ・ `ExampleNetworkServiceDiscoveryProfile.java`  
(`NetworkServiceDiscoveryProfile` の処理を記述)
- ・ `ExampleSystemProfile.java`  
(`SystemProfile` の処理を記述)

## 1.2 用語定義

用語	説明
<b>Device Connect システム</b>	スマートフォン、周辺機器を P2P 通信(例: Bluetooth, Wi-Fi, NFC)で接続し、対向機の機器種別を意識せずシームレスに操作する機能を提供するアーキテクチャ。
<b>Device Connect API</b>	スマートフォン、周辺機器の機器種別を意識することなく、同一 I/F でスマートフォン、周辺機器を制御する機能を提供する API。
<b>Device Connect Manager</b>	Device Connect システムをスマートフォン上で実現するアプリケーション。 周辺機器の間に存在し、機器種別を意識する必要のない API を UI アプリケーション、周辺機器に提供する。
<b>スマートフォン</b>	Device Connect Manager を搭載したスマートフォン。 周辺機器と接続した状態で周辺機器を操作する Device Connect API を利用することが出来る。
<b>周辺機器</b>	Device Connect システムに対応した Device Connect Manager と P2P 接続する機器。(例: スマートウォッチ、おさいふリング、Bluetooth 体重計) Device Connect Manager と接続した状態でスマートフォンを操作する Device Connect API を利用することが出来る。
<b>デバイスプラグイン</b>	Device Connect Manager と連携して周辺機器を操作、および、イベントを送信するアプリケーション。 スマートフォン内で周辺機器毎の Android アプリケーションとして構成される。
<b>メッセージ</b>	送信先デバイス、データ種別、データ本体を含み、一方から他方へ送信することを目的として定義される。
<b>イベント</b>	状態変更(再生/一時停止/再開/停止)情報などの通知情報を指す。

### 1.3 Device Connect との連携

Android 版のデバイスプラグインと Device Connect Manager とは、Intent を通して連携する。Intent は、デバイスプラグインの Broadcast Receiver が取得し、Service に受け渡す。Service から、各種 Profile の処理を呼び出し、Intent で指示のあった処理を行い、Device Connect Manager に結果を返信する。



## 2. 開発環境の構築

Device Connect を使用するために必要な手順について説明する。

### 2.1 Eclipse のダウンロード

開発には、Eclipse を使用する。

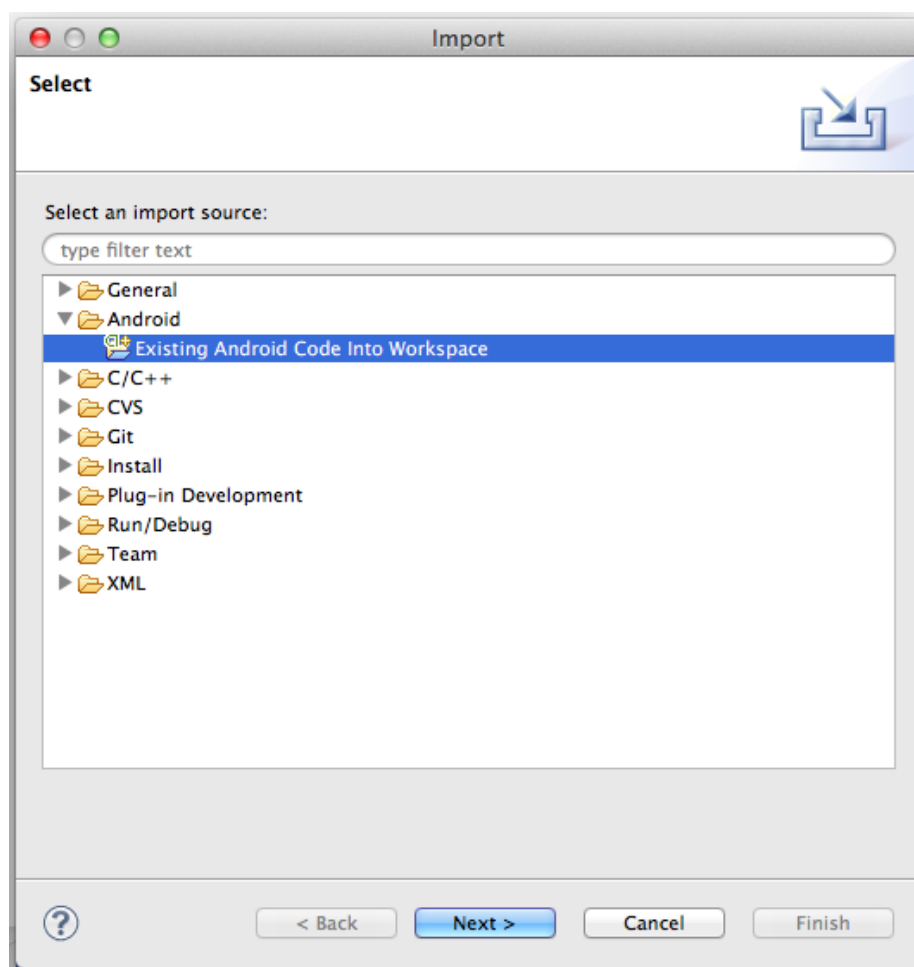
以下の URL から Android SDK が入った Eclipse をダウンロードする事が出来る。

<http://developer.android.com/sdk/index.html>

#### 2.1.1 dConnectDevicePluginSDK のインポート

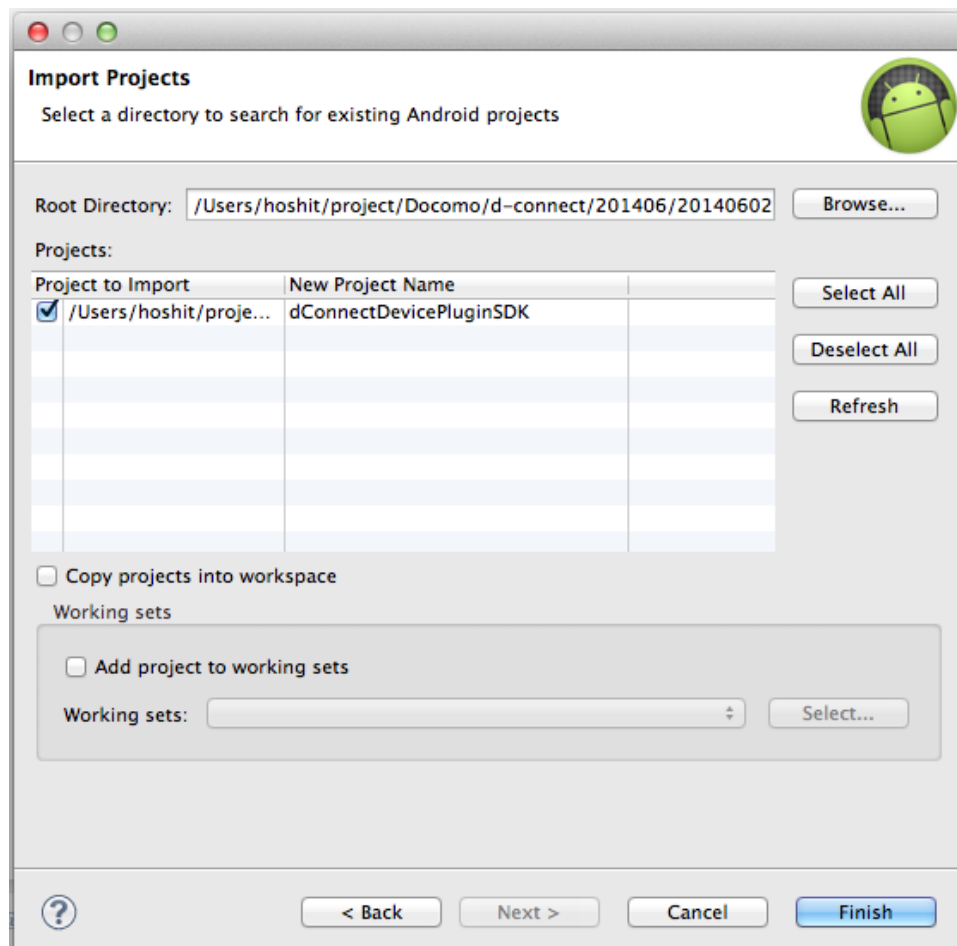
Eclipse の import 機能で、プロジェクト「dConnectDevicePluginSDK」をインポートする。

Android の下にある Existing Projects into Workspace を選択する。

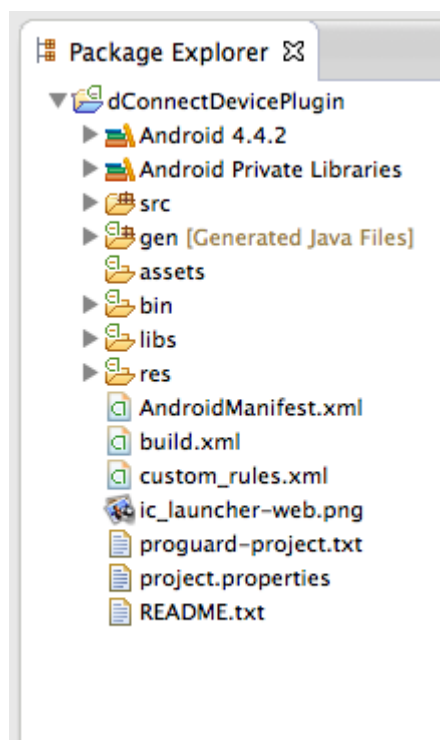




Browse...ボタンを押下して、dConnectDevicePluginSDK を選択すると、Project to import に dConnectDevicePluginSDK が表示されるので、チェックボックスにチェックをして Finish ボタンを押下する。



これで、dConnectDevicePluginSDK が Eclipse のプロジェクトとしてインポートされる。  
正常にインポートされた場合に以下のような Android ライブラリプロジェクトが Package Explorer に追加される。

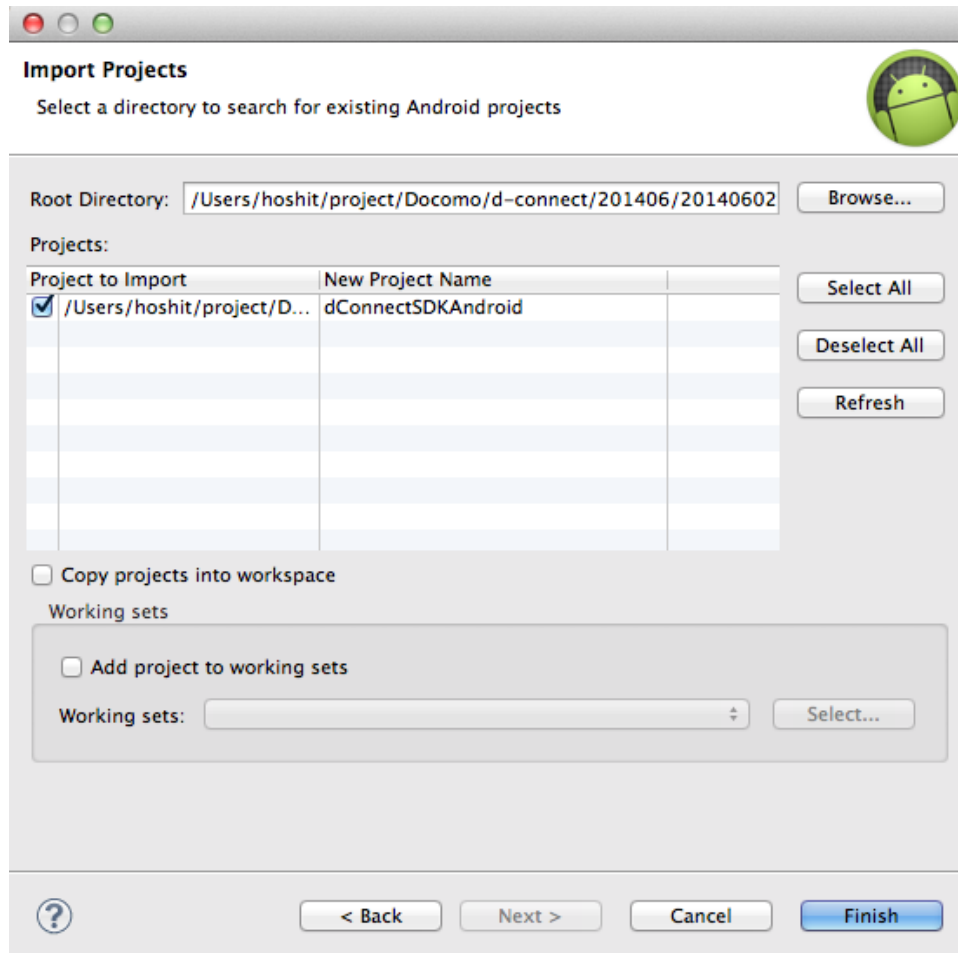


この方法にてエラーが発生してプロジェクトがインポートできない場合は、Import 画面において、General の下にある Existing Projects into Workspace でインポート処理を行うこと\*1。

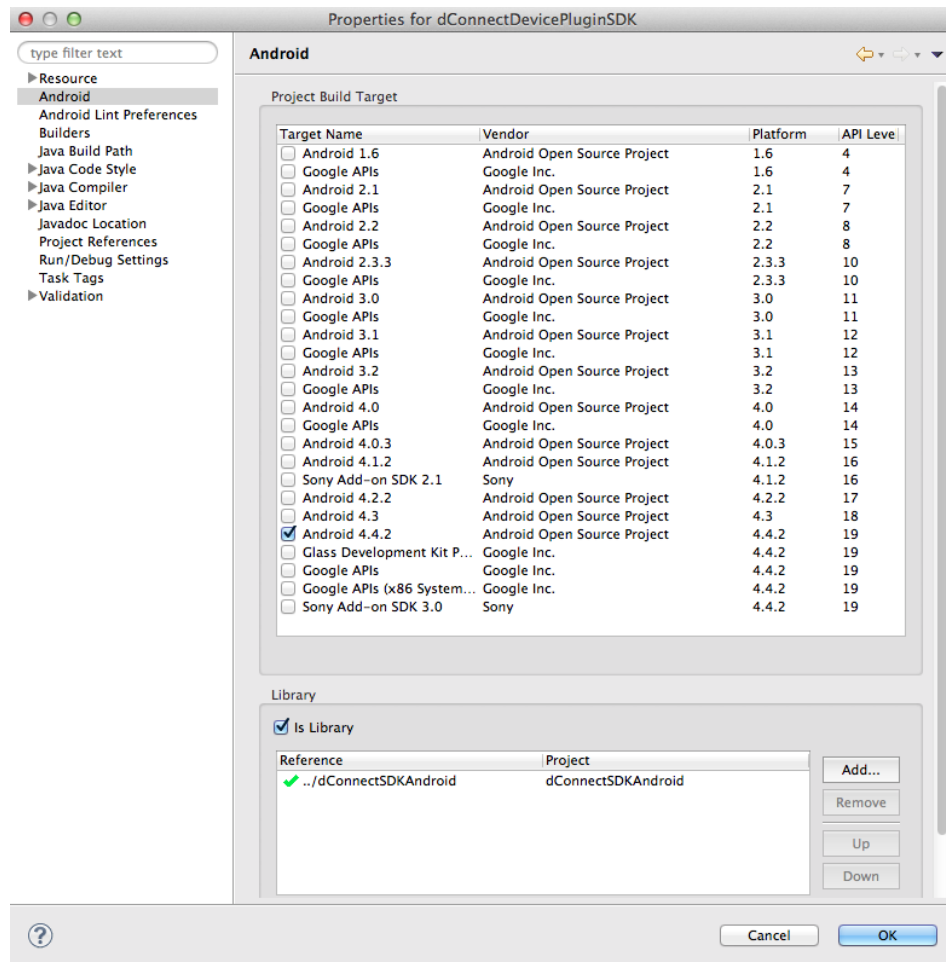
\*1: Device Connect のプロジェクト群の中には純 Java 実装であるような、Android プロジェクト以外の通常 Eclipse プロジェクトも含まれる。

### 2.1.2 dConnectSDKAndroid のインポート

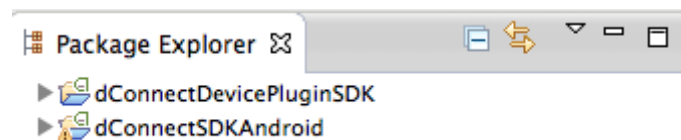
dConnectDevicePluginSDK では、Device Connect の UI アプリを簡単に作成するための SDK を使用している。そのライブラリのインポートを行う。



dConnectDevicePluginSDK の Properties を開き、以下の図のようにインポートした dConnectSDKAndroid を設定する。



ここまでの作業を終えると、プロジェクトのエラーが消える。それでもエラーが出る場合は、ライブラリのリンクをチェックするか、プロジェクトのクリーンを試みる事。



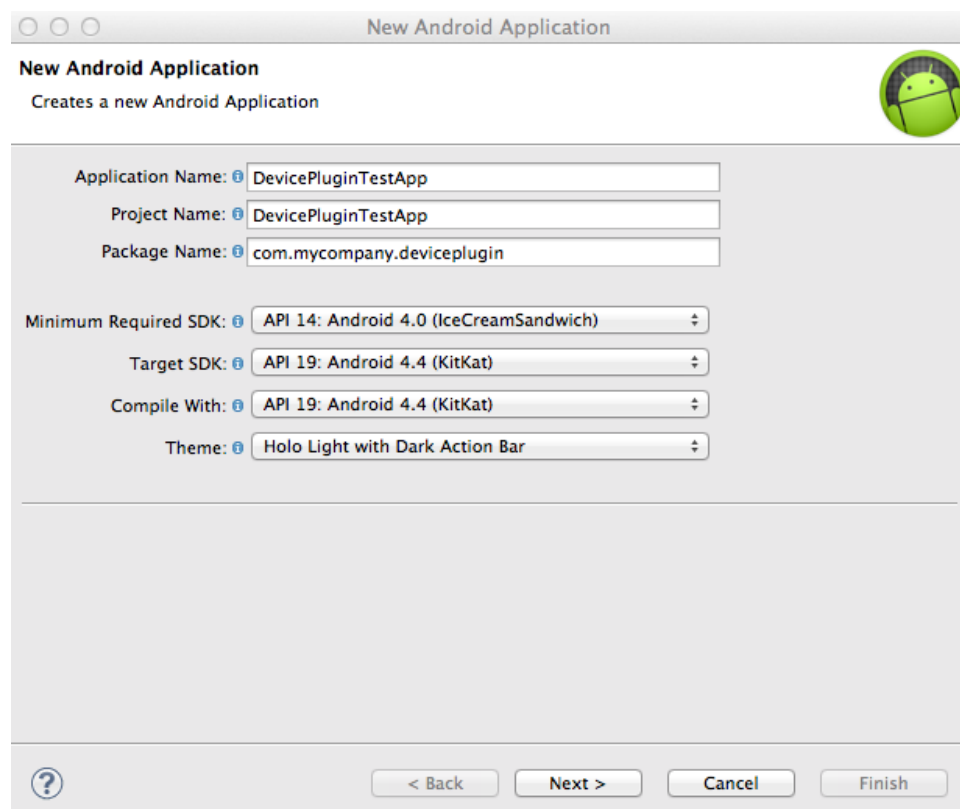
### 3. デバイスプラグインの作成

#### 3.1 Android プロジェクトの作成

Android プロジェクトの作成方法は、通常の Android プロジェクトの作成方法と同じ Eclipse の Android Project Wizard で作成する。

ここでは、サンプルとして以下のプロジェクトを作成する。

Application Name	DevicePluginTestApp
Project Name	DevicePluginTestApp
Package Name	com.mycompany.deviceplugin
Minimum Required SDK	Android 4.0



Android Application Wizard に従ってプロジェクトを作成すると、以下のように Package Explorer にプロジェクトが追加される。

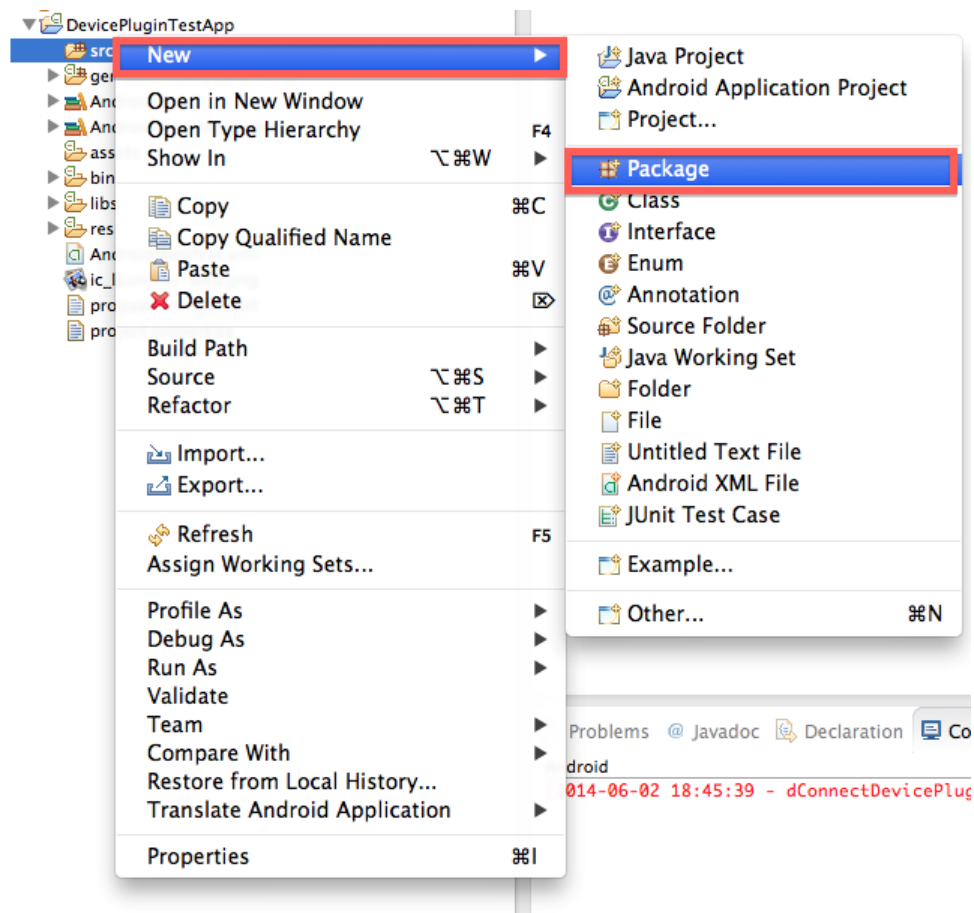


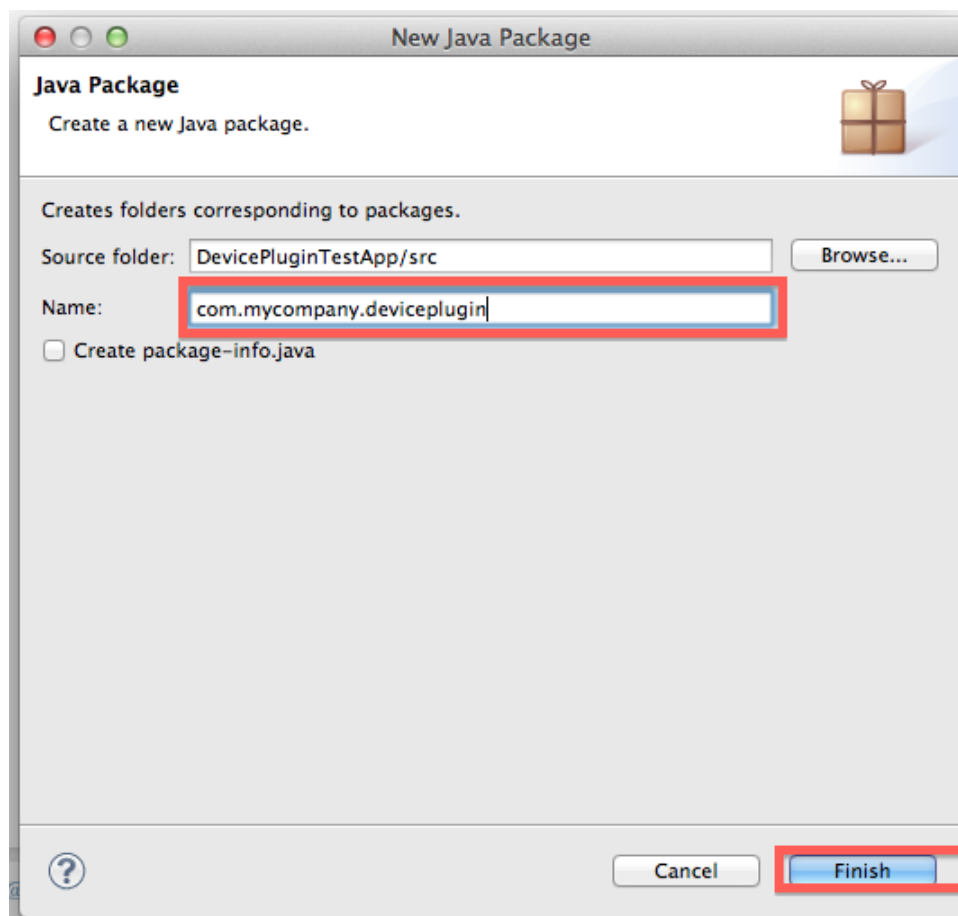
### 3.2 パッケージの作成

※既にパッケージが作成されている場合は、次章へ進む。

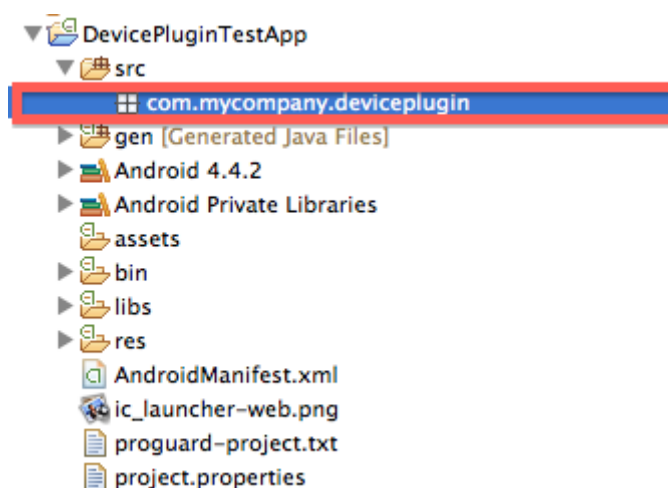
プロジェクトを作成するときに、Activity を作成しなかった場合は、src ディレクトリに何もファイルが存在しない状態になっている。その場合、最初に指定したパッケージを作成する必要がある。

作成するときは、src が選択された状態で右クリックから New>パッケージを選ぶ。





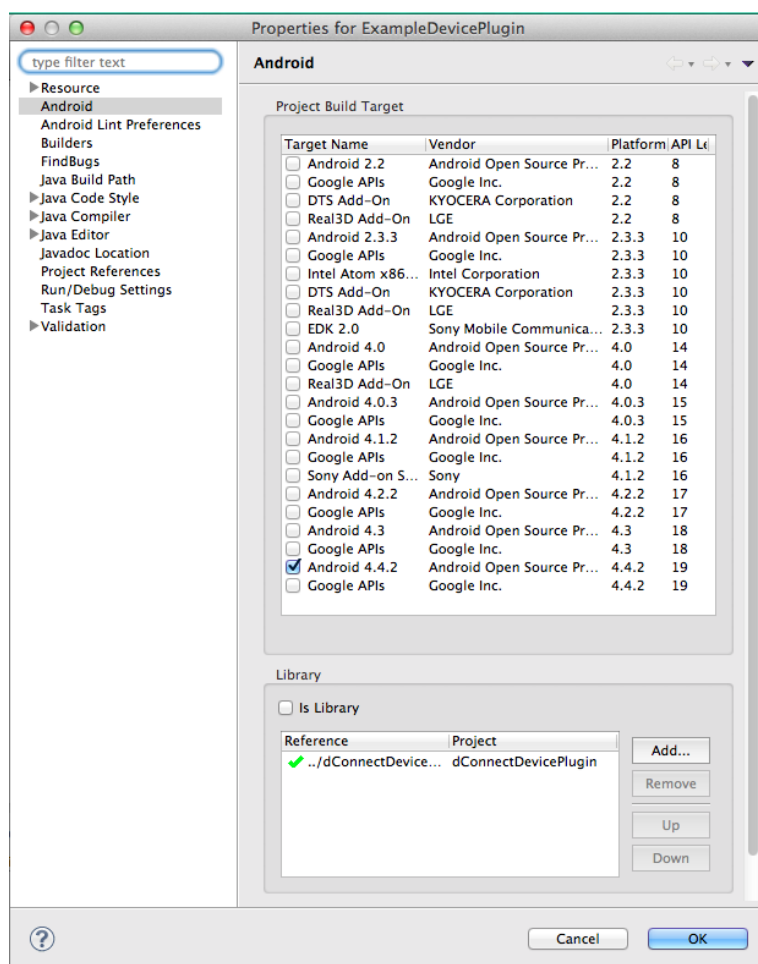
ここまでの作業が完了すると、以下のように空のパッケージが出来上がる。この中にソースコードを入れていく事になる。



### 3.3 dConnectDevicePluginSDK ライブラリプロジェクトの追加

デバイスプラグインを作成するには、dConnectDevicePlugin のライブラリプロジェクトを追加する必要がある。

Package Explorer で、DevicePluginTestApp のプロパティを開き、左の項目から Android を選択し、右下の参照ライブラリー一覧の脇にある Add ボタンを押下して dConnectDevicePlugin を追加する。



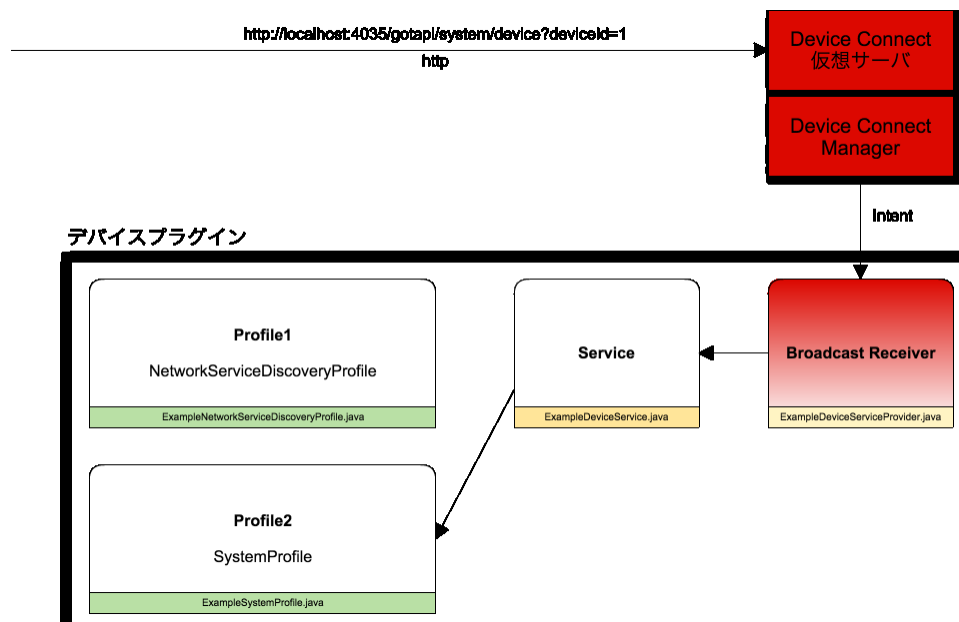
プロジェクトによっては、android-support-v4.jar で警告が発生する場合があります。その際には、android-support-v4.jar を削除する\*2。

\*2: オフィシャル Android SDK によって提供される Support Library は随時アップデートされるため、様々なバージョンが存在する。Device Connect のプロジェクト群の内にバージョンの異なる Support Library が共存してしまう際は、バージョン競合によってプロジェクトのビルドが失敗するので、いずれか 1 つのバージョンの Support Library のみ残す様に、参照ライブラリを含めプロジェクトで使われる Support Library を整理する必要がある。



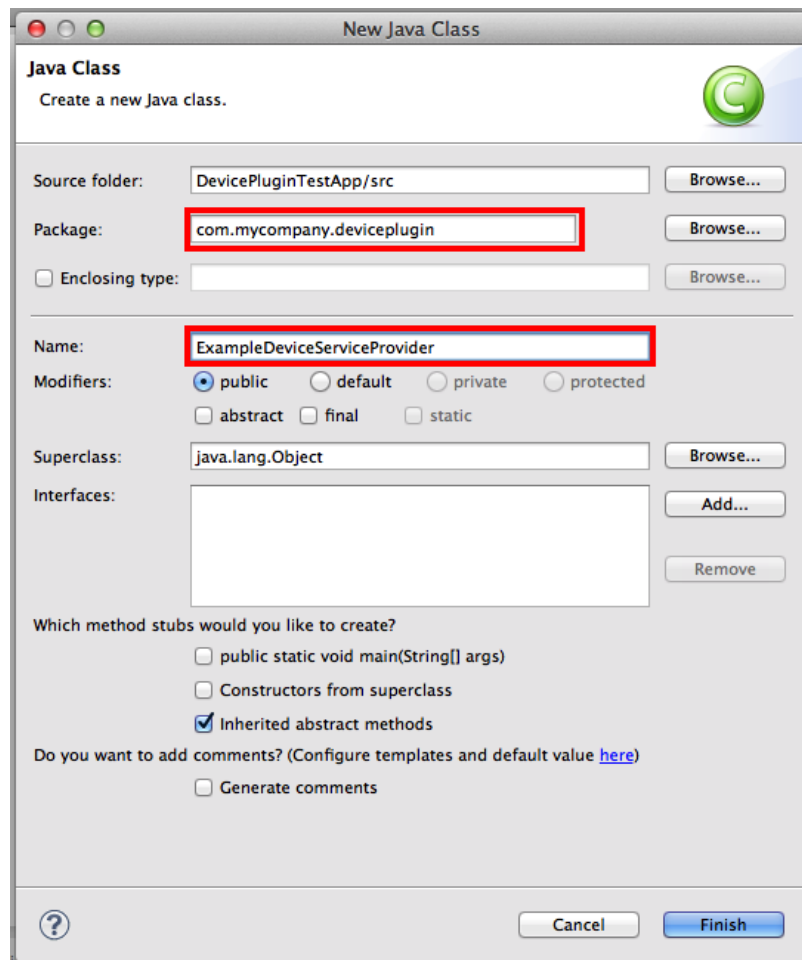
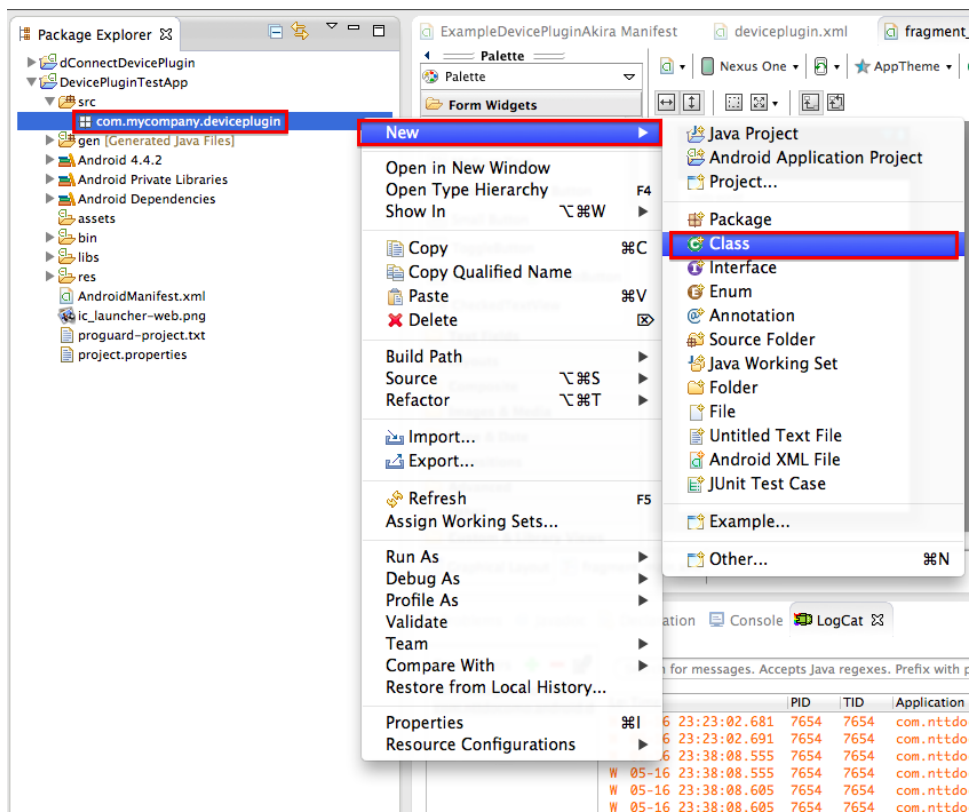
### 3.4 DConnectMessageServiceProvider の実装

DConnectMessageServiceProvider は、Device Connect Manager からの通知を受け付けるための機能になる。



今回、MainActivity は使用しないため、MainActivity.java および、AndroidManifest.xml 内の Activity タグを削除する。

パッケージを選択し(com.mycompany.deviceplugin)、ショートカットメニューで、[New]-[Class] を選択する。



DConnectMessageServiceProvider では、以下のメソッドを実装する必要がある。

```
Class<Service> getServiceClass()
```

このメソッドで返却したサービスに dConnectMessage から送られてきたメッセージが通知される。

#### ExampleDeviceServiceProvider.java

```
package com.mycompany.deviceplugin;

import android.app.Service;
import com.nttdocomo.android.dconnect.message.DConnectMessageServiceProvider;

public class ExampleDeviceReceiver<T extends Service> extends DConnectMessageServiceProvider<Service> {
    @SuppressWarnings("unchecked")
    @Override
    protected Class<Service> getServiceClass() {
        Class<? extends Service> clazz = (Class<? extends Service>) ExampleDeviceService.class;
        return (Class<Service>) clazz;
    }
}
```

また、デバイスプラグインが Device Connect Manager から送られてくる Intent を取得できるように、AndroidManifest.xml の application タグ内に receiver タグを追加する。

#### AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.mycompany.deviceplugin"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="14"
        android:targetSdkVersion="19" />

    <application
        android:allowBackup="true"
```

```

        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

<!-- Device Connect Example Device Plugin Provider. -->
<receiver android:name=".ExampleDeviceServiceProvider" >
    <meta-data
        android:name="com.nttdocomo.android.dconnect.deviceplugin"
        android:resource="@xml/deviceplugin" />

    <intent-filter>
        <action android:name="org.deviceconnect.action.GET" />
        <action android:name="org.deviceconnect.action.PUT" />
        <action android:name="org.deviceconnect.action.POST" />
        <action android:name="org.deviceconnect.action.DELETE" />
    </intent-filter>
</receiver>

<!-- LocalOAuth ユーザ認可ダイアログ用 Activity -->
<activity android:name="com.nttdocomo.android.dconnect.localoauth.activity.ConfirmAuthActivity" >
</activity>

<!-- LocalOAuth ユーザ認可ダイアログ用 Service -->
<service
    android:name="com.nttdocomo.android.dconnect.localoauth.LocalOAuth2Service"
    android:exported="false" >
    <intent-filter>
        <action android:name="com.nttdocomo.android.dconnect.localoauth.LocalOAuth2Service" />
    </intent-filter>
</service>
</application>

</manifest>

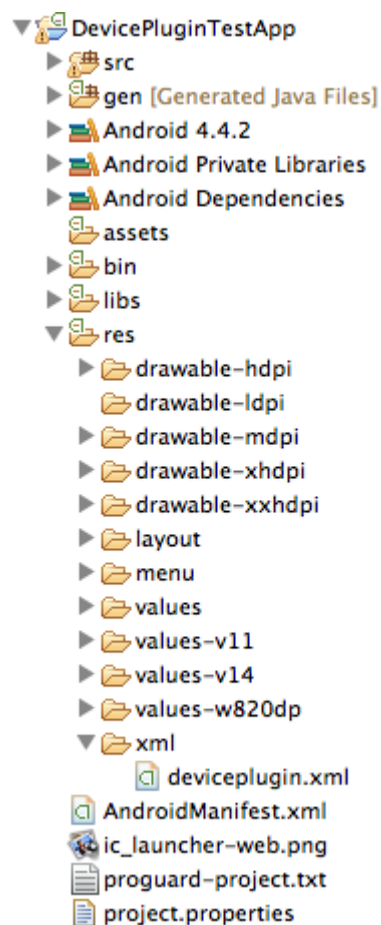
```

タグ内の `meta-data` タグの `name` 属性に『`com.nttdocomo.android.dconnect.deviceplugin`』が設定されている場合にデバイスプラグインとして処理が行われる。この定義がない場合には **Device Connect Manager** は処理を行わないので、十分に留意する事。

また、タグ内の Intent-filter タグに記述された Action それぞれが HTTP メソッドの GET, PUT, POST, DELETE に対応しており、これらの項目の有無によって、特定の HTTP メソッドのメッセージのみ受け取る様に設定できる：

- org.deviceconnect.action.GET
- org.deviceconnect.action.PUT
- org.deviceconnect.action.POST
- org.deviceconnect.action.DELETE

サービスを動作させるための deviceplugin.xml を res/xml 以下に作成する。



#### deviceplugin.xml

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

  <deviceplugin-provider xmlns:android="http://schemas.android.com/apk/res/android">

    <profile name="network_service_discovery" />

    <profile name="system" />

  </deviceplugin-provider>
```

deviceplugin.xml には、デバイスプラグインで使用するプロファイルの一覧を指定する。  
ここに宣言されているプロファイルは使用することができないので、注意すること。

### 3.5 deviceplugin.xml のフォーマット

deviceplugin.xml は、デバイスプラグインで使用するプロファイルを指定する。

ここで指定したプロファイル以外はアクセスできない。

また、**name** タグや **description** タグでは、プロファイルの名前や詳細説明を定義する。

独自拡張プロファイルを追加した場合には、これらのタグの追加も行う事。

タグが無かった場合には、認証画面においてプロファイルの名前がそのまま表示されてしまい分かりづらくなる。

独自拡張していない **Device Connect** で定義されているプロファイルについては、システムがタイトルや詳細説明をもっているため、新たに定義する必要はない。

タグ	属性	例	子タグ	説明
<b>profile</b>			<b>name</b> <b>description</b>	デバイスプラグインで使用するプロファイルを定義する。 独自プロファイルなどを定義したときもこのタグを追加する必要がある。
	<b>name</b>	<b>system</b>		プロファイル名を定義する。
<b>name</b>				認証ダイアログに表示するタイトル名を定義する。 ただし、 <b>Device Connect</b> で標準化しているプロファイルについては、システム側でタイトル名や説明が定義されているので、ここで追加する必要はない。
	<b>lang</b>	<b>ja</b>		言語指定を行う。BCP-47にある言語タグ 多言語対応を行うために、言語ごとにタイトルを設定することができる。 端末側が要求する言語が定義されていなかった場合は、最初に見つかった <b>name</b> タグのタイトルを表示する。
<b>description</b>				プロファイルの詳細説明を定義する。 ただし、 <b>Device Connect</b> で標準化しているプロファイルについては、システム側でタイトル名や説明が定義されているので、ここで追加する必要はない。
	<b>lang</b>	<b>ja</b>		言語指定を行う。BCP-47にある言語タグ 多言語対応を行うために、言語ごとに詳細説明を設定することができる。 端末側が要求する言語が定義されていなかった場合は、最初に見つかった <b>description</b> タグの詳細を表示する。

## サンプル

```
<?xml version="1.0" encoding="UTF-8"?>
<deviceplugin-provider xmlns:android="http://schemas.android.com/apk/res/android" >

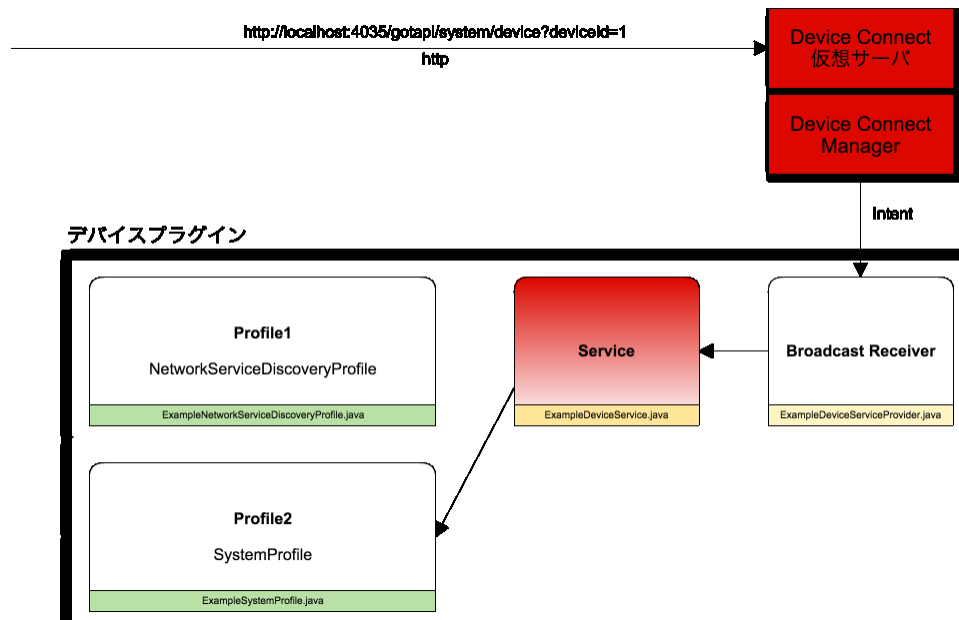
    <profile name="my_profile">
        <name lang="ja-JP">日本語</name>
        <description lang="ja-JP">日本語の説明</description>
        <name lang="en-US">English</name>
        <description lang="en-US">English explain</description>
    </profile>
    <profile name="network_service_discovery" />
    <profile name="system" />
    <profile name="file" />
    <profile name="mediastream_recording" />
    <profile name="setting" />
    <profile name="camera" />

</deviceplugin-provider>
```

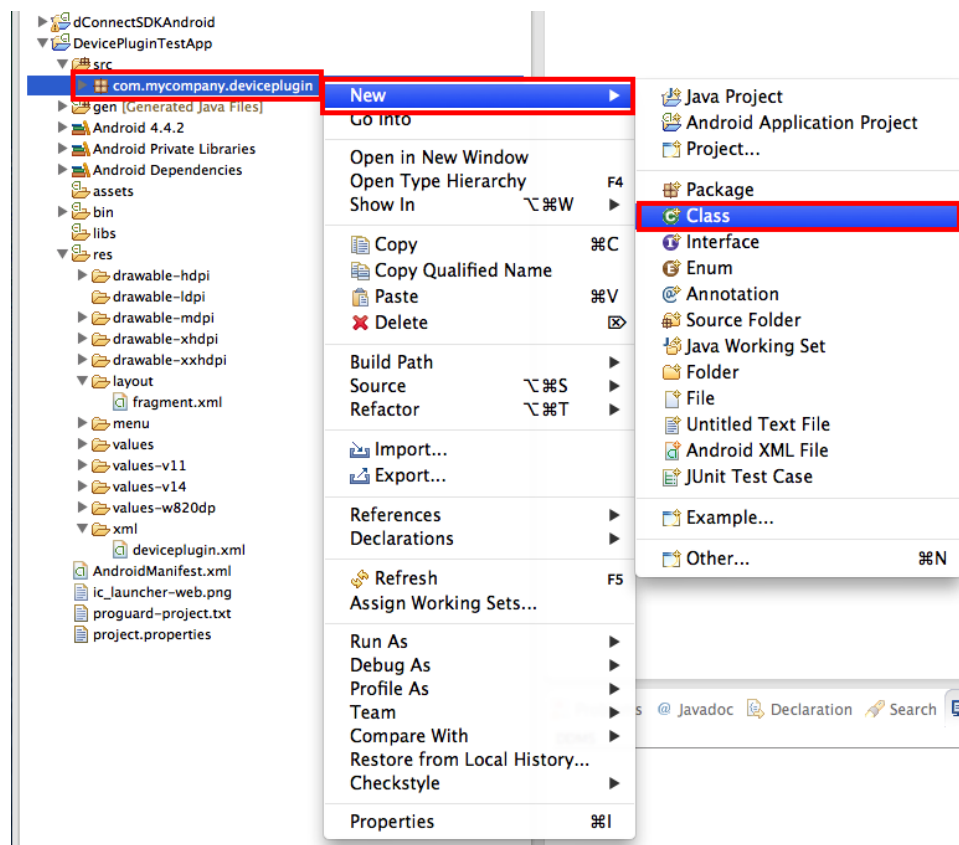


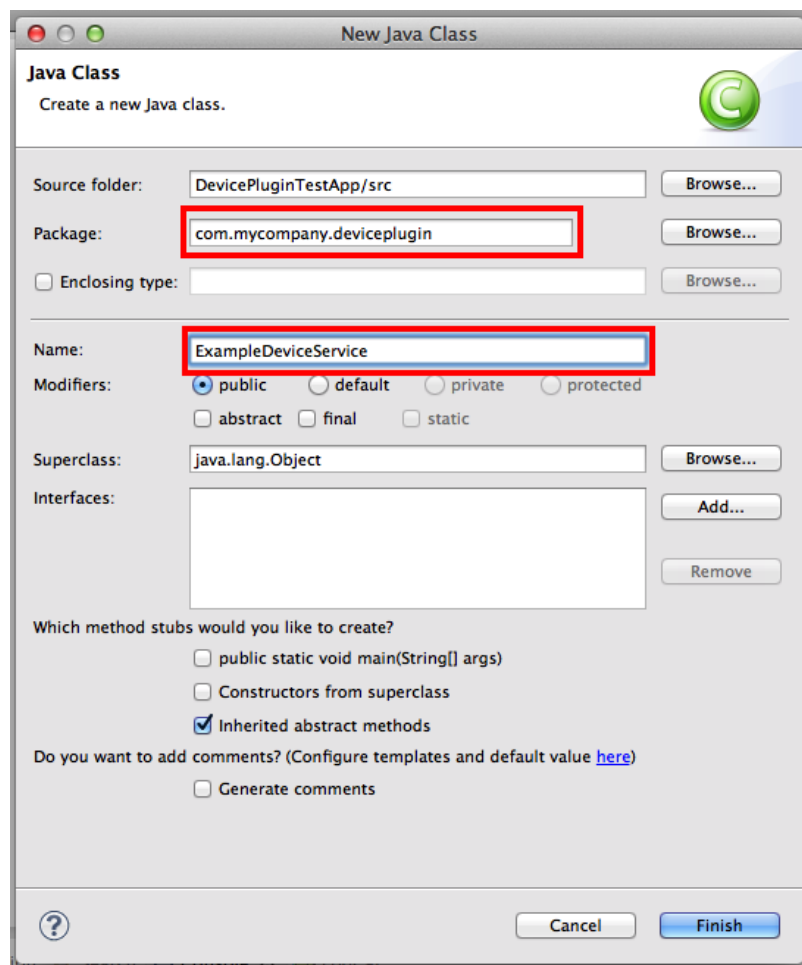
### 3.6 DConnectMessageService の実装

DConnectMessageService は、Device Connect Manager から送られてきた DConnect メッセージを処理するためのサービスである。



新規で java のクラスを作成する。





このクラスを実装することでデバイスプラグインを動作させる事が出来る。

### ExampleDeviceService.java

```
package com.mycompany.deviceplugin;

import com.nttdocomo.android.dconnect.message.DConnectMessageService;

/**
 * サンプルの dConnect メッセージサービス実装クラス.
 * @author docomo
 */
public class ExampleDeviceService extends DConnectMessageService {

    @Override
    public void onCreate() {
        super.onCreate();
    }
}
```

```

@Override
protected SystemProfile getSystemProfile() {
    return new ExampleSystemProfile(this);
}

@Override
protected NetworkServiceDiscoveryProfile getNetworkServiceDiscoveryProfile() {
    return new ExampleNetworkServiceDiscoveryProfile();
}
}

```

そして、**ExampleDeviceService** を動作させるために、**application** タグ内に **service** タグを追加する。

## AndroidManifest.xml

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.mycompany.deviceplugin"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="14"
        android:targetSdkVersion="19" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <!-- Device Connect Example Device Plugin Provider. -->
        <receiver android:name=".ExampleDeviceServiceProvider" >
            <meta-data
                android:name="com.nttdocomo.android.dconnect.deviceplugin"
                android:resource="@xml/deviceplugin" />

```

```

        <intent-filter>

            <action android:name="org.deviceconnect.action.GET" />

            <action android:name="org.deviceconnect.action.PUT" />

            <action android:name="org.deviceconnect.action.POST" />

            <action android:name="org.deviceconnect.action.DELETE" />

        </intent-filter>
    </receiver>

    <service
        android:name=".ExampleDeviceService"
        android:exported="false" >
    </service>

    <!-- LocalOAuth ユーザ認可ダイアログ用 Activity -->
    <activity android:name="com.nttdocomo.android.dconnect.localoauth.activity.ConfirmAuthActivity" >
    </activity>

    <!-- LocalOAuth ユーザ認可ダイアログ用 Service -->
    <service
        android:name="com.nttdocomo.android.dconnect.localoauth.LocalOAuth2Service"
        android:exported="false" >

        <intent-filter>

            <action android:name="com.nttdocomo.android.dconnect.localoauth.LocalOAuth2Service" />

        </intent-filter>

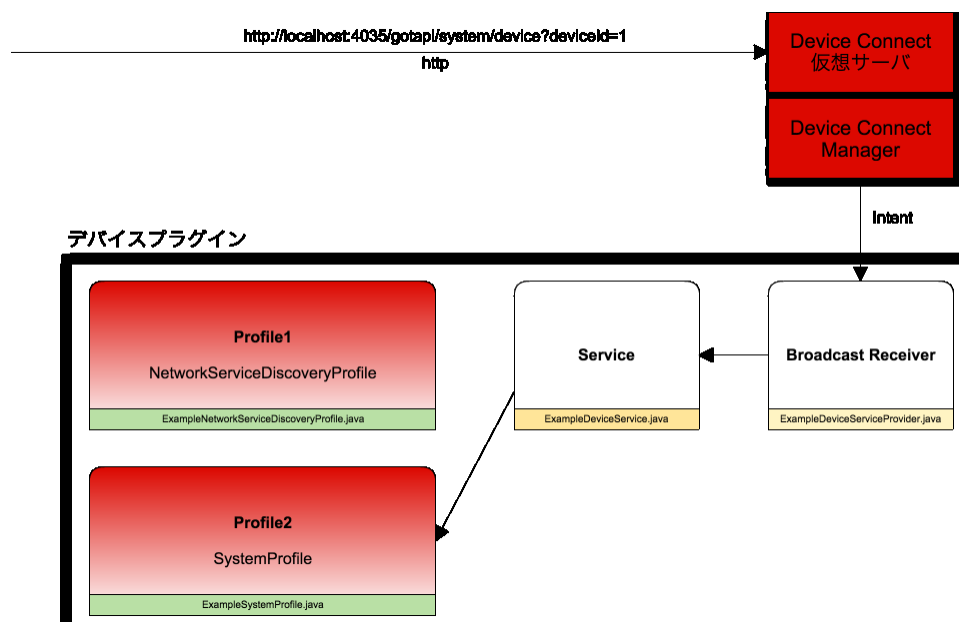
    </service>
</application>

</manifest>

```

### 3.7 各プロファイルの実装と DConnectMessgeService への追加

各デバイスプラグインで対応するプロファイルを実装して、DConnectMessageService に追加することでデバイスと Device Connect Manager の間の通信を行うことができるようになる。



各プロファイルの実装は、DConnectProfile を継承して作成することになる。

ベースクラスがこれらのメソッドの実装しており、サブクラスでは各 API に対応した "on + HTTP メソッド名 + Attribute 名" の関数を必要に応じてオーバーライドするようになっている。

Attribute の判定などは全てベースクラスが行う。

また、戻り値の **boolean** が **true** の場合には、引数にある **response** を即座に **Device Connect Manager** に返却するが、**false** の場合には返却処理は行わないので、デバイスプラグイン側で明示的に返却する必要がある。

非同期の処理を行いたい場合などは **false** を返して、別スレッドで処理した後に **response** を返却する。

DConnect で定義してあるプロファイルは、DConnectProfile を継承した以下のクラスがある。

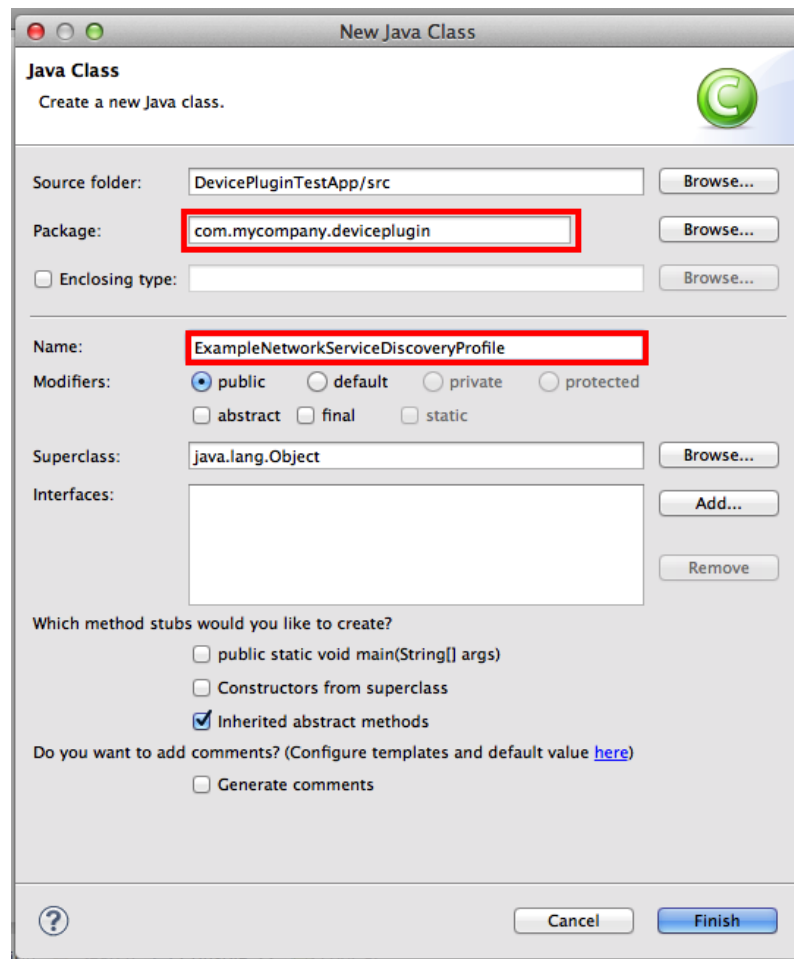
- ・ BatteryProfile
- ・ ConnectProfile
- ・ DeviceOrientationProfile
- ・ FileDescriptorProfile
- ・ FileProfile
- ・ MediaPlayerProfile
- ・ MediaStreamRecordingProfile
- ・ NetworkServiceDiscoveryProfile
- ・ NotificationProfile
- ・ PhoneProfile
- ・ ProximityProfile
- ・ SettingsProfile
- ・ SystemProfile
- ・ VibrationProfile

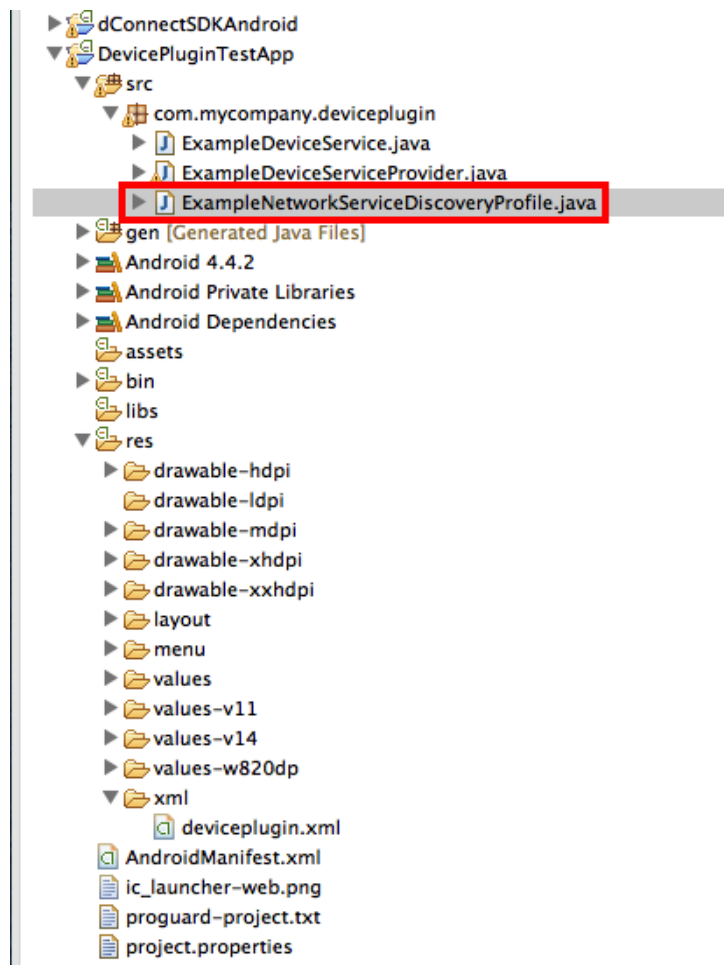
これらのクラスを継承して、各デバイスプラグインの処理を実装する。

### 3.8 Network Service Discovery プロファイルの実装

Network Service Discovery プロファイルは、Device Connect Manager にデバイスの有無を通知するためのプロファイルになる。

このプロファイルが実装されていないデバイスプラグインは動作しない。





このプロファイルは、NetworkServiceDiscoveryProfile クラスを継承して以下のようなクラスを実装する。

#### ExampleNetworkServiceDiscoveryProfile.java

```
package com.mycompany.deviceplugin;

import java.util.ArrayList;
import java.util.List;

import android.content.Intent;
import android.os.Bundle;

import com.nttdocomo.android.dconnect.profile.NetworkServiceDiscoveryProfile;
import com.nttdocomo.dconnect.message.intent.message.IntentDConnectMessage;

/**
```



```

* Network Service Discovery プロファイルの実装クラス.
* @author docomo
*/

public class ExampleNetworkServiceDiscoveryProfile extends NetworkServiceDiscoveryProfile {

    /** サンプルのデバイス ID を定義する. */
    private static final String DEVICE_ID = "example_device_id";

    /** サンプルのデバイス名を定義する. */
    private static final String DEVICE_NAME = "Example Device";

    /**
     * Network Service Discovery プロファイル.
     * [/network_service_discovery/getnetworkservices]に対応するメソッド.
     * @param request リクエスト
     * @param response レスポンス
     * @return 即座に返却する場合は true, 非同期に返却する場合は false
     */
    @Override
    public boolean onGetGetNetworkServices(final Intent request, final Intent response) {

        List<Bundle> services = new ArrayList<Bundle>();

        Bundle service = new Bundle();

        setId(service, DEVICE_ID);

        setName(service, DEVICE_NAME);

        setType(service, NetworkType.WIFI);

        setOnline(service, true);

        services.add(service);

        setServices(response, services.toArray(new Bundle[services.size()]));

        setResult(response, IntentDConnectMessage.RESULT_OK);

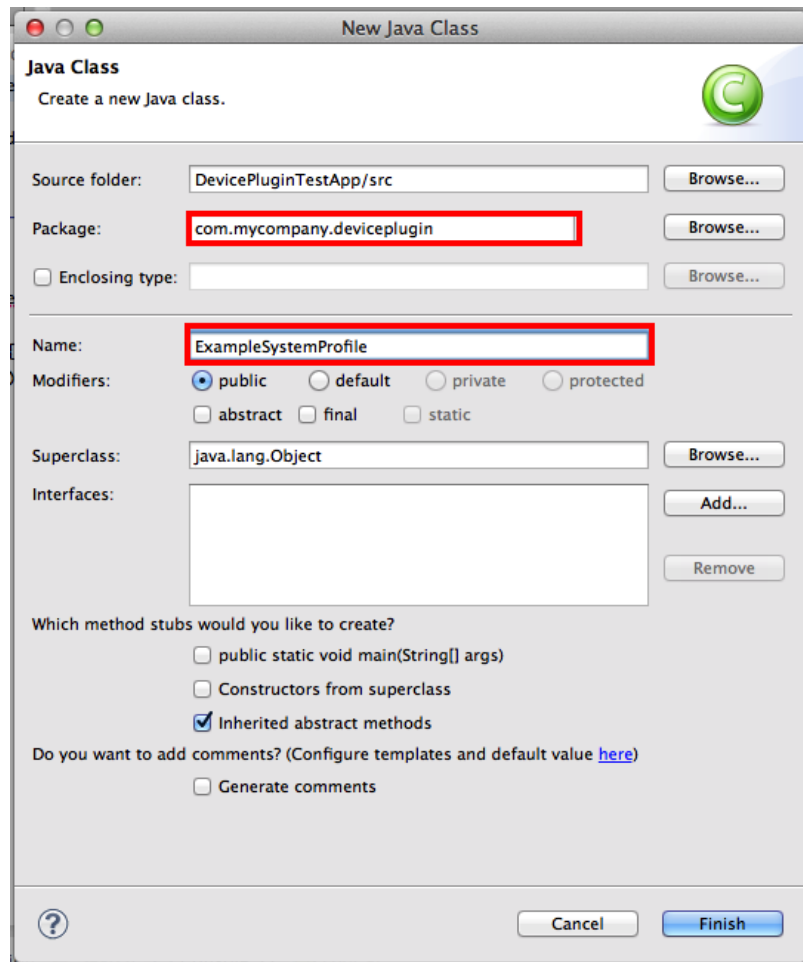
        return true;
    }
}

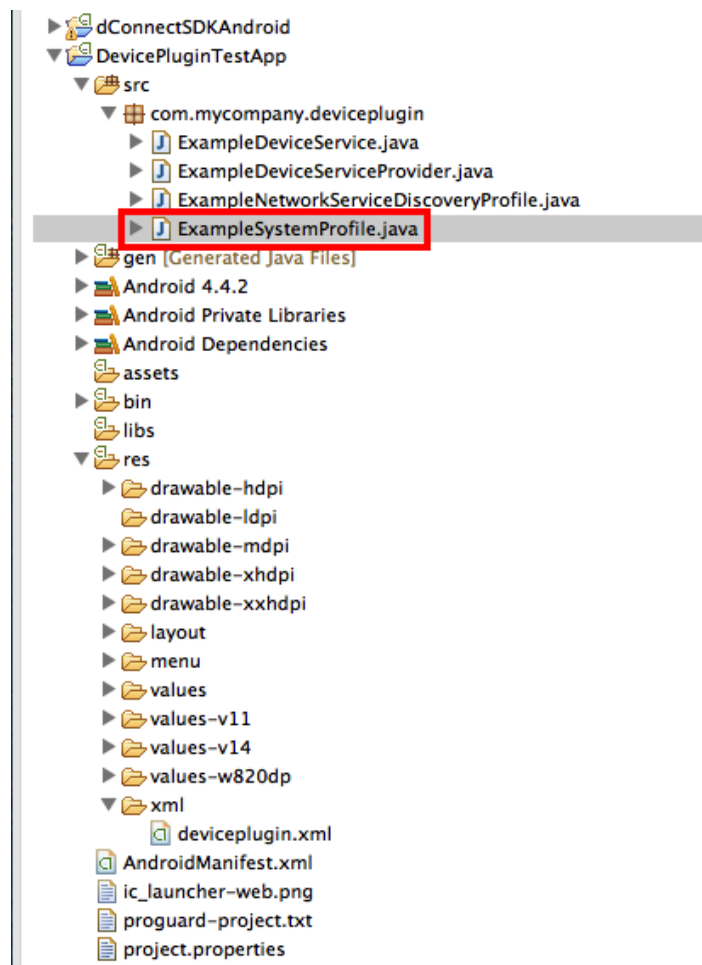
```

デバイス ID は、デバイスプラグインの中でデバイスを識別・管理するための ID となる。  
 このサンプルでは、ID を文字列「example\_device\_id」としているが実際のデバイスプラグイン  
 では、デバイスを一意に識別できるような ID を指定すること。

### 3.9 System プロファイルの実装

System Profile では、サポートしているプロファイルの一覧などを Device Connect Manager に返却する。





サポートしているプロファイルは、次の章の `addProfile` 操作を行う場所で追加した **Profile** の名前を自動で追加する。そのため、`ExampleSystemProfile` では、`addProfile()` した情報を受け取るためにコンストラクタで `DConnectProfileProvider` を渡さなければいけない。

そのプロファイルを返却する処理はベースクラスである `SystemProfile` クラスが行うため、このクラスではコンストラクタを用意する以外、特にすることはない。

なお、`DConnectMessageService` は `DConnectProfileProvider` のインタフェースを実装している。`getSettingActivity()` の `SettingActivity` クラスは、この時点でエラーが出るが、のちの章で作成するためそのままとしておく。

#### ExampleSystemProfile.java

```
package com.mycompany.deviceplugin;

import android.app.Activity;

import com.nttdocomo.android.dconnect.profile.DConnectProfileProvider;
import com.nttdocomo.android.dconnect.profile.SystemProfile;
```

```

/**
 * System プロファイル実装クラス.
 *
 * @author docomo
 */
public class ExampleSystemProfile extends SystemProfile {

    /**
     * デバイスプラグインのプロファイルを設定するコンストラクタ.
     * @param provider Profile の Provider
     */
    public ExampleSystemProfile(DConnectProfileProvider provider) {
        super(provider);
    }

    @Override
    protected Class<? extends Activity> getSettingPageActivity(Intent request, Bundle param) {
        Class<? extends Activity> clazz = (Class<? extends Activity>) SettingActivity.class;
        return clazz;
    }
}

```

### 3.10 プロファイルの追加

デバイスプラグインがサポートするプロファイルは、`DConnectMessageServiceProvider#addProfile(DConnectProfile)`で明示的に追加する必要がある。

重い処理を行う場合には `DConnectMessageService` を使用し、別スレッドを使用するべきで、また上記で説明したようにデバイスプラグインの中でデータを使い回したい場合なども `DConnectMessageService` を使用するべきである。

#### ExampleDeviceService.java

```
package com.mycompany.deviceplugin;

import com.nttdocomo.android.dconnect.message.DConnectMessageService;

/**
 * サンプルの dConnect メッセージサービス実装クラス.
 * @author docomo
 */
public class ExampleDeviceService extends DConnectMessageService {

    @Override
    public void onCreate() {
        super.onCreate();

        //ここで、DConnectMessageServiceProvider にプロファイルを追加することができる。
        addProfile(new ExampleNetworkServiceDiscoveryProfile());
        addProfile(new ExampleSystemProfile(this));
    }

    @Override
    protected SystemProfile getSystemProfile() {
        return new ExampleSystemProfile(this);
    }

    @Override
    protected NetworkServiceDiscoveryProfile getNetworkServiceDiscoveryProfile() {
        return new ExampleNetworkServiceDiscoveryProfile();
    }
}
```

### 3.11 設定画面の作成

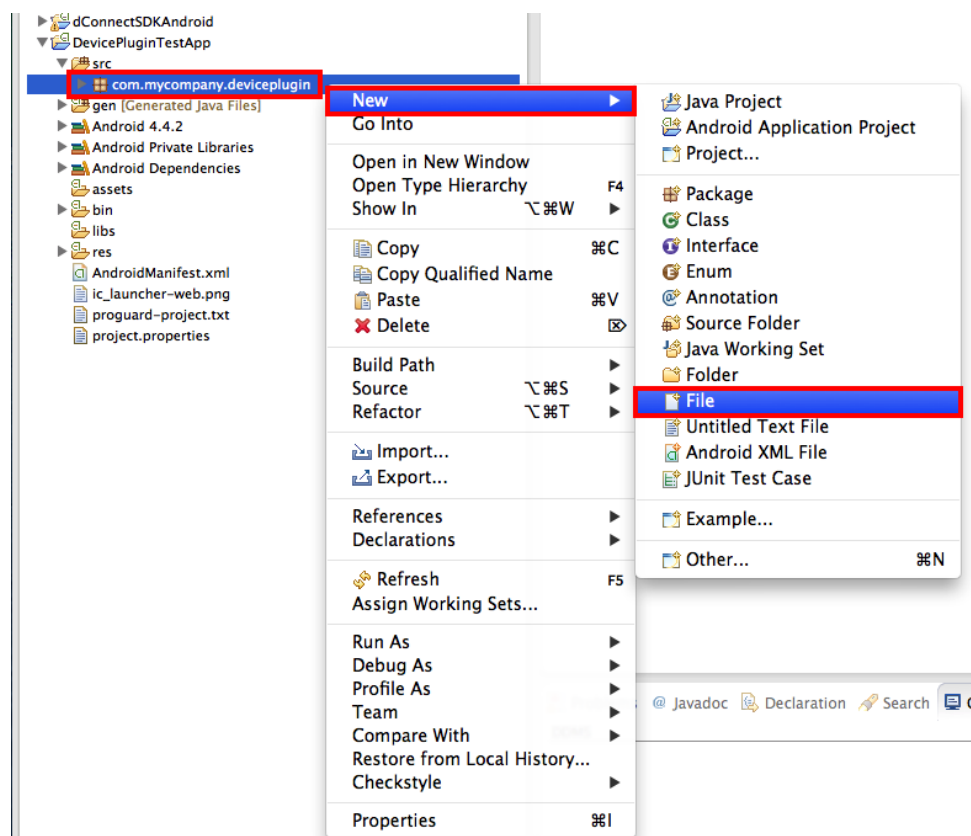
Device Connect で対応する周辺機器は、スマートフォンと接続する作業だけでも手間のかかる物がある。

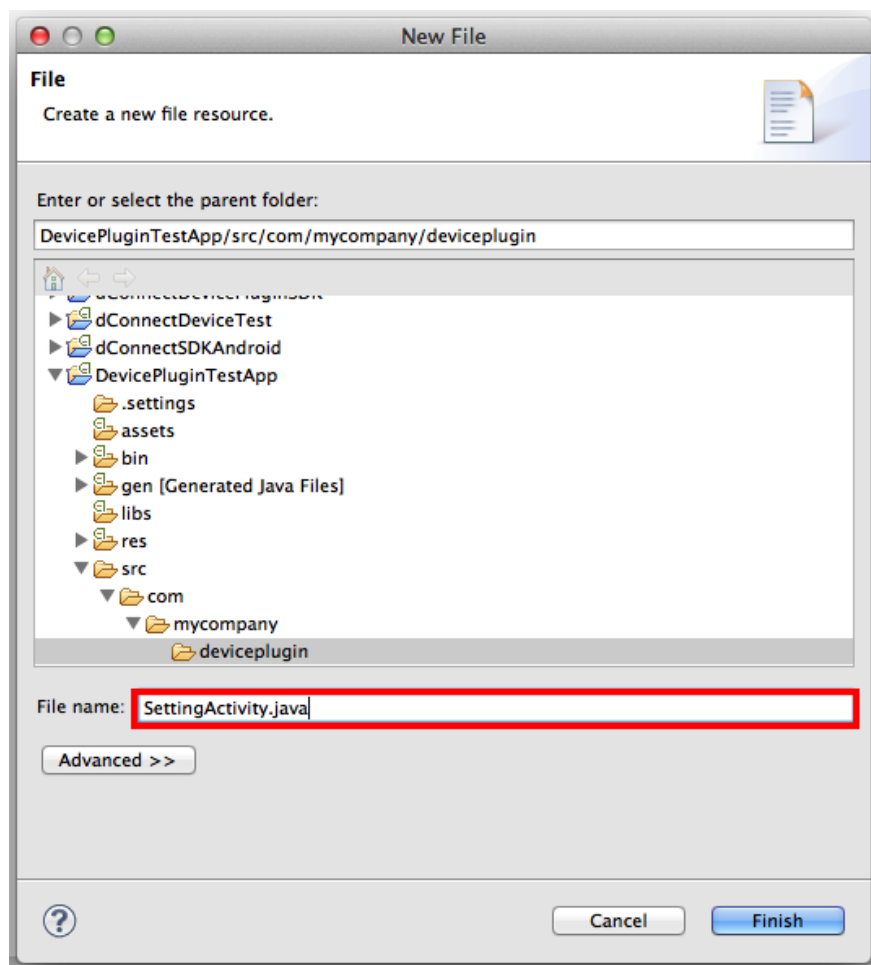
そのため、Device Connect では、それらの操作をサポートするページを作成する必要がある。

ここでは、そのページを簡単に生成するサンプルの作成方法について説明する。

#### 4.4.1 Activity の作成

「SettingActivity.java」を作成する。





作成するクラスで、DConnectSettingPageFragmentActivity というクラスを継承し、実装しなければいけないメソッドを実装する。

そのメソッドの中に、以下のような記述をする。

- ・ getPageCount()
  - 作成する設定画面のページ数を指定する。
- ・ createPage()
  - 表示する Fragment を設定する。
  - position には現在表示しているページ数が渡されてくるため、それを Bundle などによって Fragment へ渡す。

この状態で、まだ MyFragment を作成していないためにエラーメッセージが表示されるが、そのままとしておく。

#### SettingActivity.java

```
package com.mycompany.deviceplugin;
```

```

import android.os.Bundle;

import android.support.v4.app.Fragment;

import com.nttdocomo.android.dconnect.ui.activity.DConnectSettingPageFragmentActivity;

public class SettingActivity extends DConnectSettingPageFragmentActivity {

    /**
     * 設定画面のページ数.
     * @return ページ数
     */

    @Override
    public int getPageCount() {

        return 3;

    }

    /**
     * 設定画面のページ.
     * @param position 表示するページ
     * @return 表示する Fragment
     */

    @Override
    public Fragment createPage(int position) {

        Bundle b = new Bundle();

        b.putInt("position", position);

        MyFragment f = new MyFragment();

        f.setArguments(b);

        return f;

    }

}

```

この設定画面用 Activity を AndroidManifest.xml に追加する。

### AndroidManifest.xml

```

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

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

    package="com.mycompany.deviceplugin"

    android:versionCode="1"

    android:versionName="1.0" >

```



```

<uses-sdk
    android:minSdkVersion="14"
    android:targetSdkVersion="19" />

<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >

    <!-- Device Connect Example Device Plugin Provider. -->
    <receiver android:name=".ExampleDeviceServiceProvider" >
        <meta-data
            android:name="com.nttdocomo.android.dconnect.deviceplugin"
            android:resource="@xml/deviceplugin" />

        <intent-filter>
            <action android:name="org.deviceconnect.action.GET" />
            <action android:name="org.deviceconnect.action.PUT" />
            <action android:name="org.deviceconnect.action.POST" />
            <action android:name="org.deviceconnect.action.DELETE" />
        </intent-filter>
    </receiver>

    <service
        android:name=".ExampleDeviceService"
        android:exported="false" >
    </service>

    <!-- LocalOAuth ユーザ認可ダイアログ用 Activity -->
    <activity android:name="com.nttdocomo.android.dconnect.localoauth.activity.ConfirmAuthActivity" >
    </activity>

    <!-- LocalOAuth ユーザ認可ダイアログ用 Service -->
    <service
        android:name="com.nttdocomo.android.dconnect.localoauth.LocalOAuth2Service"
        android:exported="false" >

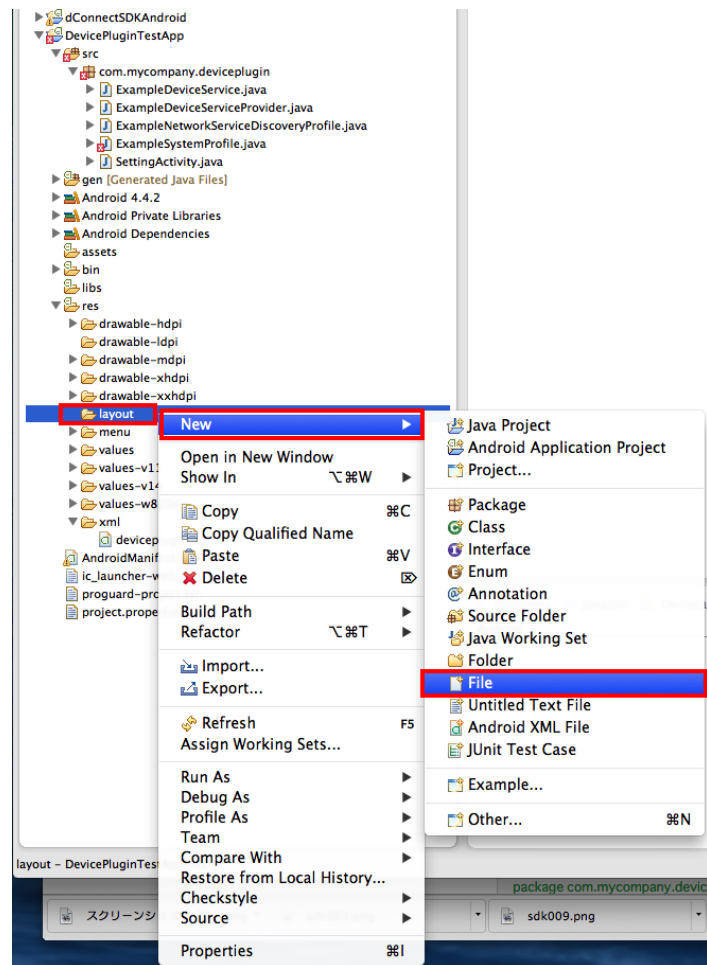
```

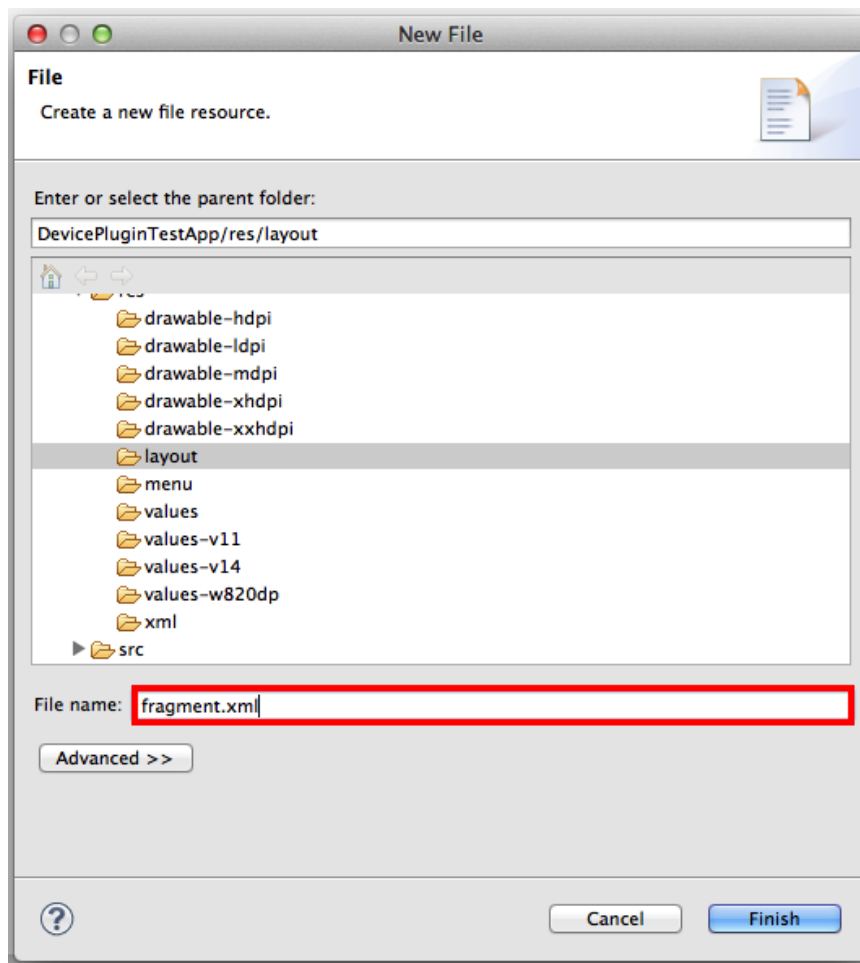
```
<intent-filter>
    <action android:name="com.nttdocomo.android.dconnect.localoauth.LocalOAuth2Service" />
</intent-filter>
</service>
<activity
    android:name=".SettingActivity"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
</application>

</manifest>
```

### 3.12 Fragment の作成

次に、説明画面として表示する Fragment である MyFragment を作成する。





Fragment の作成は、通常の Fragment 作成と同じため、ここでは省略する。  
ここでは、単純に View の文字列を切り替えるだけのサンプルを作成する。  
layout ファイルとして、fragment.xml を作成する。

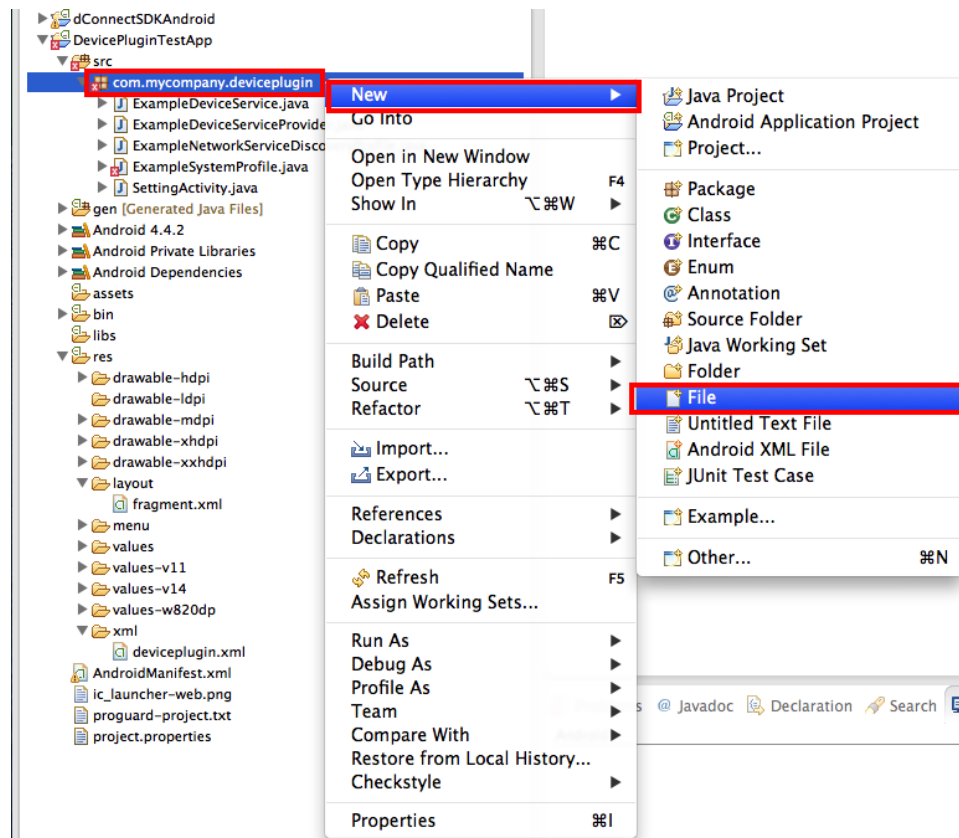
#### fragment.xml

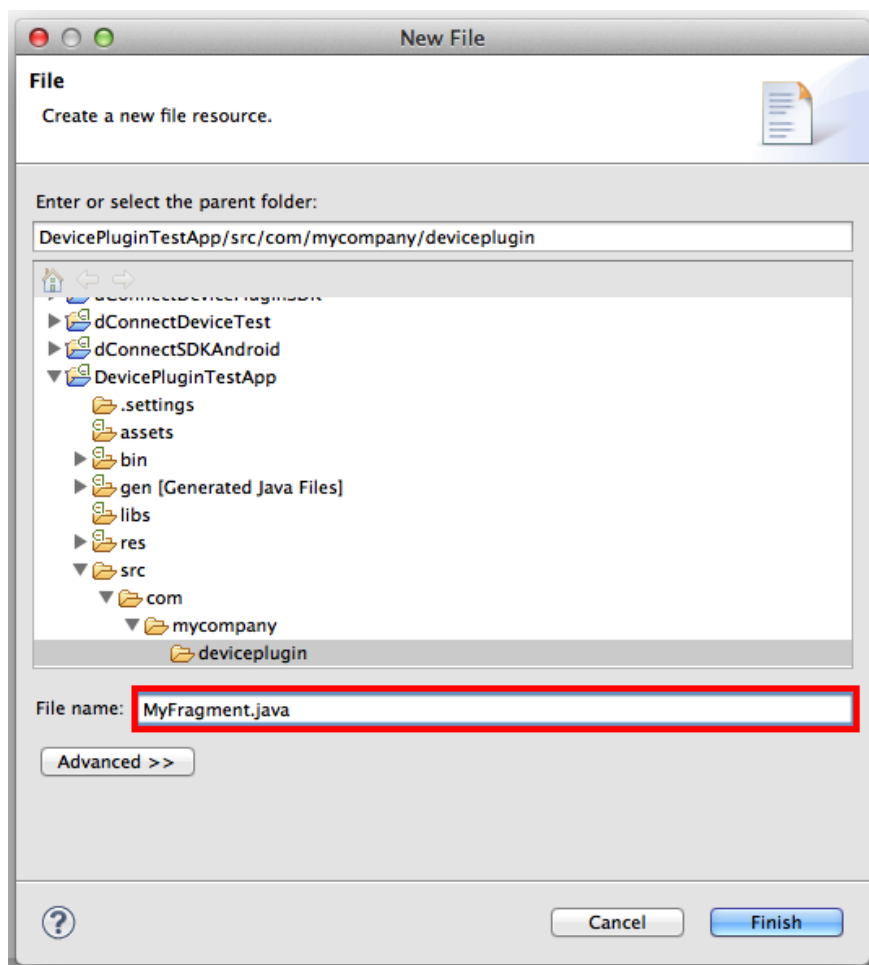
```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="30sp"
    android:id="@+id/text"/>
```

</LinearLayout>





## MyFragment.java

```
package com.mycompany.deviceplugin;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

public class MyFragment extends Fragment {

    @Override

    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {

Bundle args = getArguments();

        int position = args.getInt("position", -1);
```

```
View root = inflater.inflate(R.layout.fragment, container, false);

TextView view = (TextView) root.findViewById(R.id.text);

position++;

view.setText("Fragment : " + position);

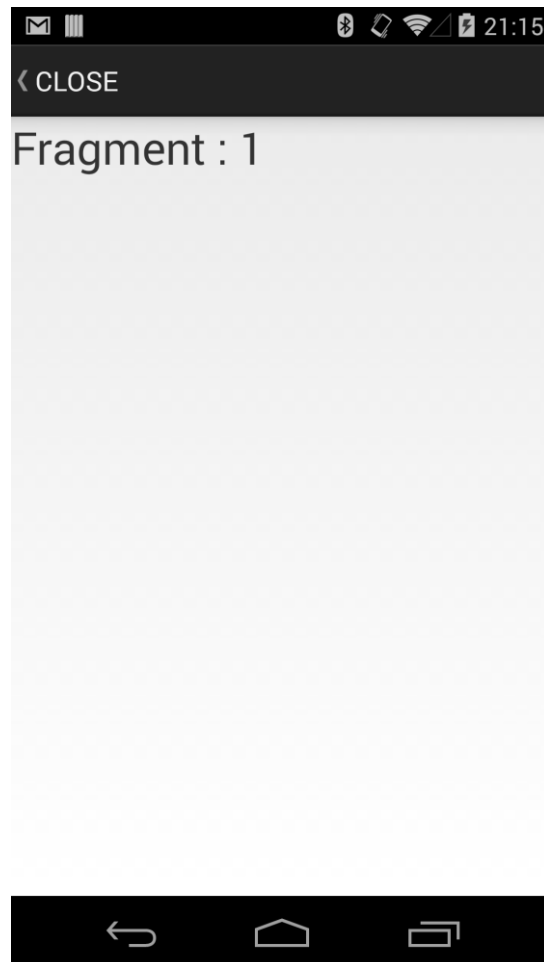
return root;

}

}
```

### 3.13 設定画面の動作確認

ここまで作成を終えた後、このデバイスプラグインをインストールすると設定画面が表示される。ページをフリックすると画面が横にスライドする事を確認できる。





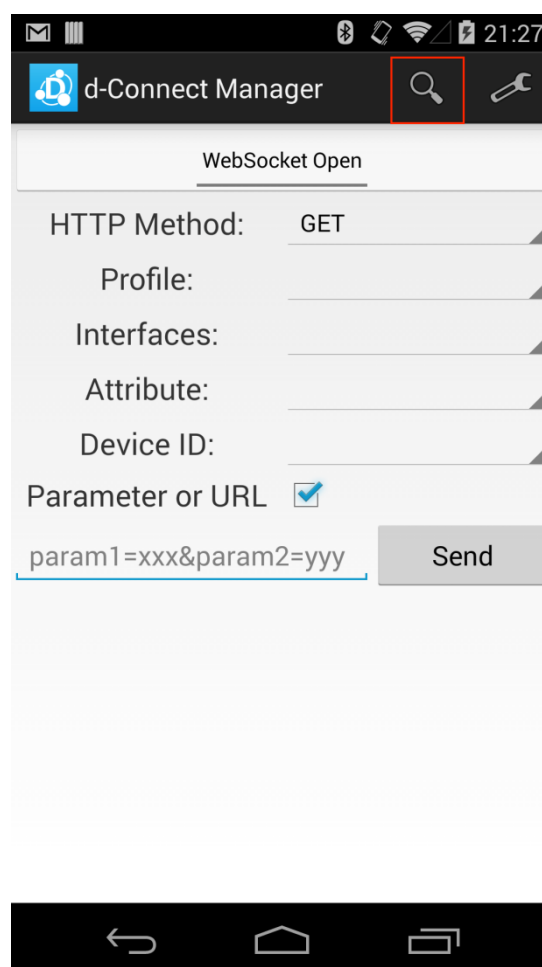
## 4. デバイスプラグインの動作確認

### 4.1 デバイスの検索

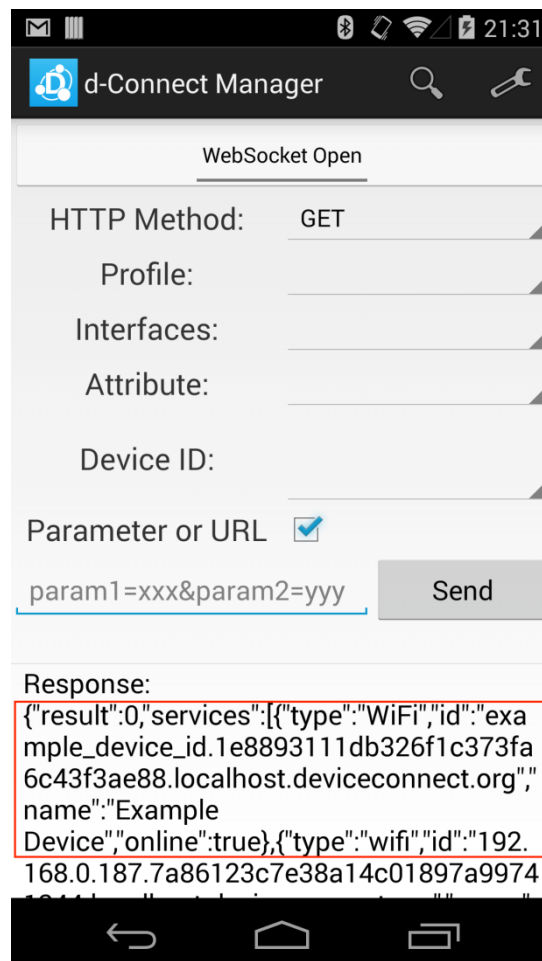
Device Connect Manager がインストールされている端末に DevicePluginTestApp をインストールする。



アイコンを押下することで Device Connect Manager のテストアプリが起動する。  
Device Connect Manager を起動して、虫眼鏡ボタンを押下する。



Response に "id": example\_device\_id.\*\*\*\*\* のデータが含まれていれば、Device Connect Manager とデバイスプラグインが接続されたことが確認できる。



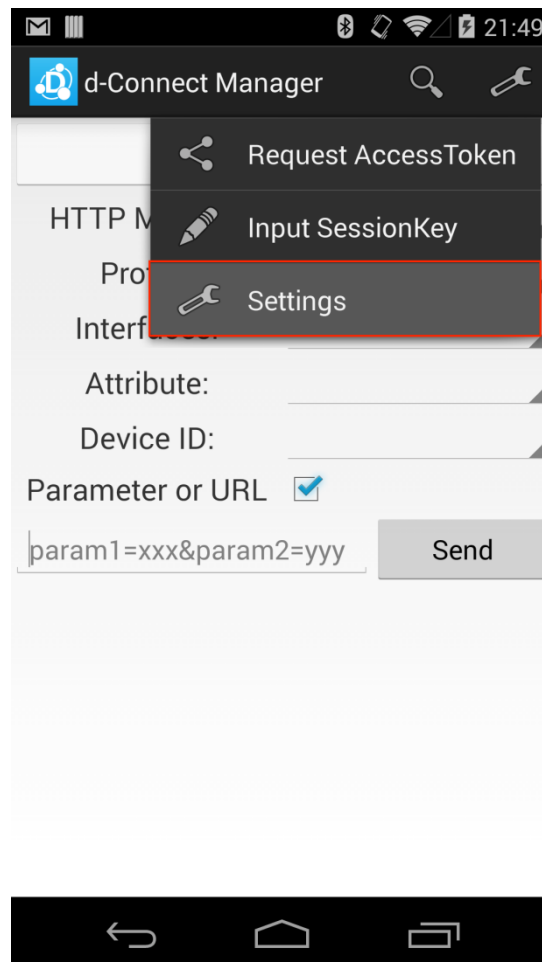
The screenshot shows the d-Connect Manager app interface. At the top, there's a header with the app name and icons for search and settings. Below the header, there's a section titled "WebSocket Open". Under this section, there are several input fields: "HTTP Method:" with a dropdown menu showing "GET", "Profile:", "Interfaces:", "Attribute:", and "Device ID:". Below these fields, there's a checkbox labeled "Parameter or URL" which is checked. Under the checkbox, there's a text input field containing "param1=xxx&param2=yyy" and a "Send" button. Below the "Send" button, there's a section titled "Response:" which contains a JSON response. The JSON response is highlighted with a red box. The JSON response is: {"result":0,"services":[{"type":"WiFi","id":"example\_device\_id.1e8893111db326f1c373fa6c43f3ae88.localhost.deviceconnect.org","name":"Example Device","online":true},{"type":"wifi","id":"192.168.0.187.7a86123c7e38a14c01897a997418414b"}]}

```
{
  "result": 0,
  "services": [
    {
      "type": "WiFi",
      "id": "example_device_id.1e8893111db326f1c373fa6c43f3ae88.localhost.deviceconnect.org",
      "name": "Example Device",
      "online": true
    },
    {
      "type": "wifi",
      "id": "192.168.0.187.7a86123c7e38a14c01897a997418414b"
    }
  ]
}
```

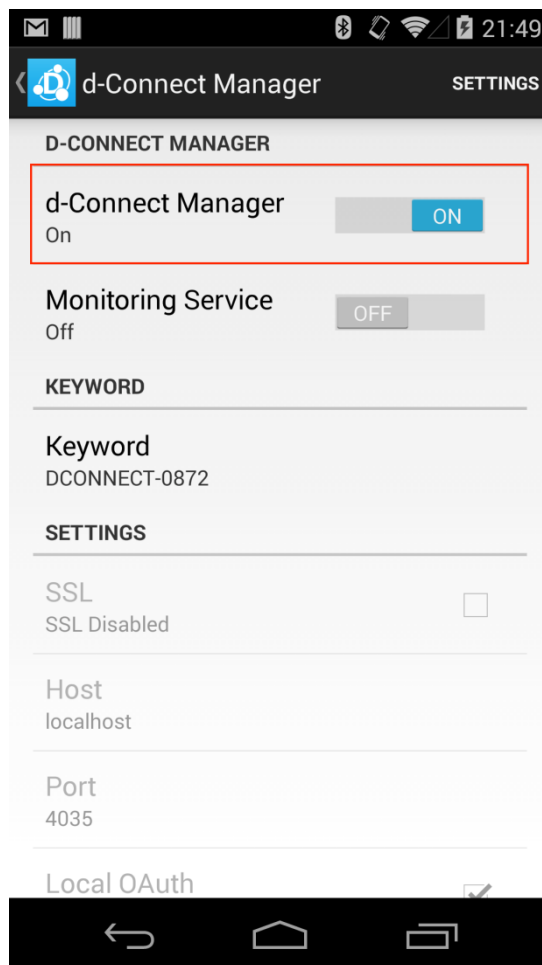
接続が正常に出来なかった場合には、デバイスプラグインが正常に Device Connect Manager に登録されていない場合もある。

その場合には、Device Connect Manager を再起動すること。

右上の **SETTINGS** ボタンを押下することで設定画面に遷移することができる。



ここで、Device Connect Manager の On/Off スイッチで、再起動することができる。



再起動ができたなら、前の画面に戻り再度虫眼鏡ボタンを押下して正しいレスポンスが返ってくるかを確認すること。

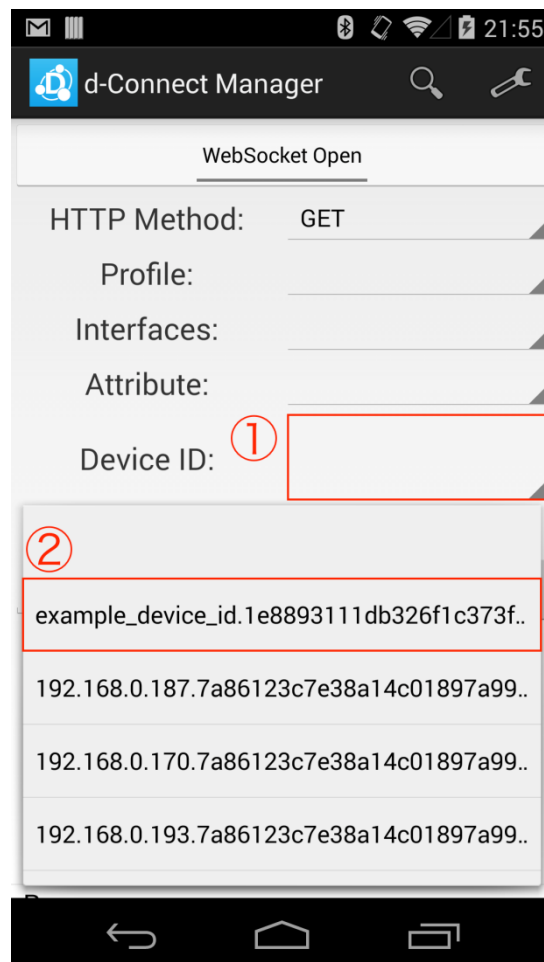
それでも表示されない場合には、デバイスプラグインの実装に問題があるので、プログラムを確認する。

## 4.2 Profile の実行

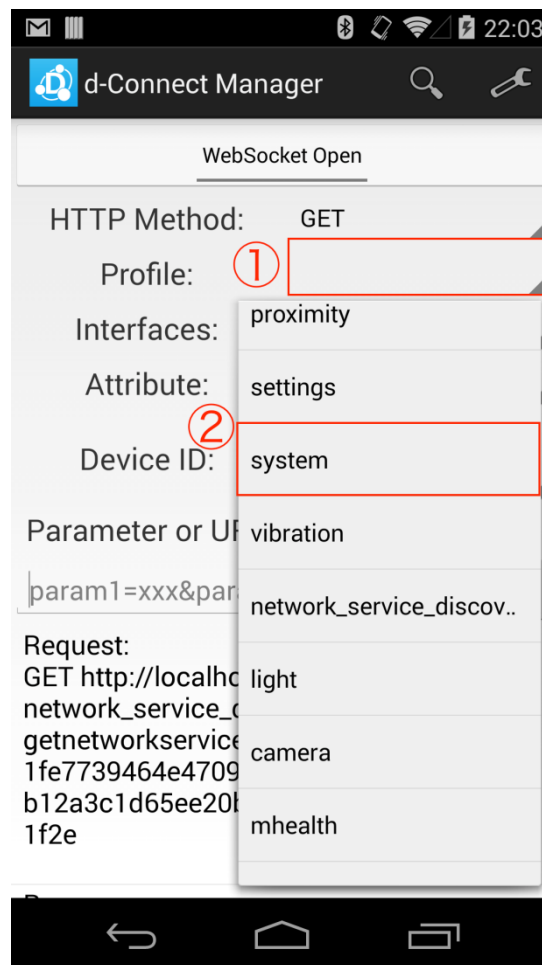
ここまで出来たのならば、次に実装しておいた **SystemProfile** を実行してみる。

まず、**SystemProfile** の実行の前にデバイスの検索を行うと、**DeviceId** の指定をすることができるようになる。

**DeviceId** の選択は以下の順で赤枠を選択していく。



HTTP Method は GET なのでそのまま。次に、Profile を以下の順番で選択する。



次にデバイスプラグインがサポートしている Profile 一覧を取得するためには、Interface を指定しなければいけないため、Interface を以下の手順で選択する。

他の Profile では、Interface がなく、Attribute を指定する Profile がほとんどである。その場合は Interface を指定しなくてよい。

The screenshot shows the 'd-Connect Manager' app interface. At the top, there's a status bar with icons for mail, signal, Bluetooth, location, Wi-Fi, and battery, along with the time 22:08. Below the app title, there's a 'WebSocket Open' section. The 'HTTP Method' is set to 'GET'. The 'Profile' is set to 'system'. The 'Interfaces' field is highlighted with a red box and a circled '1'. The 'Attribute' field is empty. The 'Device ID' is set to 'device', also highlighted with a red box and a circled '2'. Below these fields, there's a 'Parameter or URL' section with a checked checkbox and a text input field containing 'param1=xxx&param2=yyy'. A 'Send' button is next to the input field. At the bottom, there's a 'Request:' section showing the full GET request URL: 'GET http://localhost:4035/gotapi/network\_service\_discovery/getnetworkservices?accessToken=681ae51fe7739464e4709ff915ba1e9f60bdd200cb12a3c1d65ee20b0979218cf49dc68f0f1d1f2e'. The bottom of the screen shows standard Android navigation icons.

ここまでを選択すると以下になる。この状態で **Send** を選択すると以下のような RESTful なリクエストが送信される事になる。

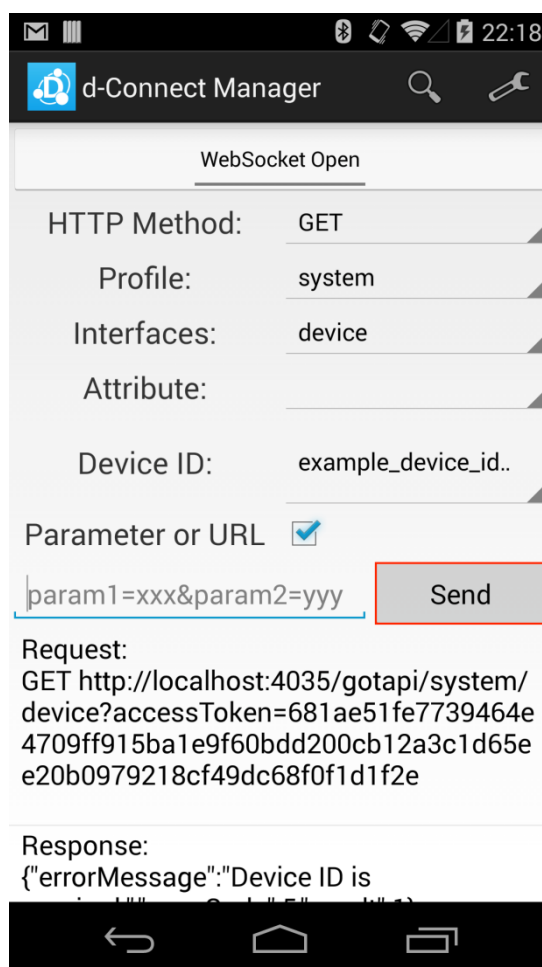
<http://localhost:4035/gotapi/system/device?deviceId=.....>

DeviceConnectAPI の URL の構造は以下のようにになっている。

<http://localhost:4035/gotapi/{Profile名}/{Interface名}/{Attribute名}?パラメータ>

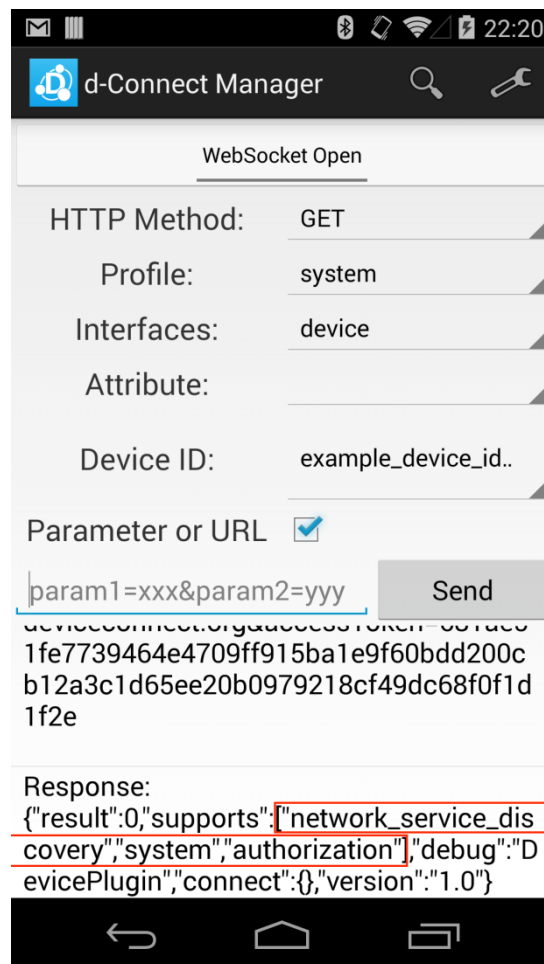
もしくは、

<http://localhost:4035/gotapi/{Profile名}/{Attribute名}?パラメータ>





ここまで実装する事によって、デバイスプラグインがサポートしている Profile の一覧が以下のように表示される。



The screenshot shows the 'd-Connect Manager' app interface. At the top, there's a status bar with icons for mail, signal, Bluetooth, Wi-Fi, and battery, along with the time 22:20. Below the app title, there's a 'WebSocket Open' section. It contains several fields: 'HTTP Method:' set to 'GET', 'Profile:' set to 'system', 'Interfaces:' set to 'device', 'Attribute:' (empty), and 'Device ID:' set to 'example\_device\_id..'. Below these is a 'Parameter or URL' section with a checked checkbox and a text input field containing 'param1=xxx&param2=yyy'. A 'Send' button is next to the input field. Below the input field, a long alphanumeric string is displayed: 'deviceconnect.org&access\_token=3b1fe7739464e4709ff915ba1e9f60bdd200cb12a3c1d65ee20b0979218cf49dc68f0f1d1f2e'. At the bottom, there's a 'Response:' section showing a JSON object: `{"result":0,"supports":["network_service_discovery","system","authorization"],"debug":"DevicePlugin","connect":{},"version":"1.0"}`. The 'network\_service\_discovery' value is highlighted with a red box. The bottom of the screen shows standard Android navigation icons.

WebSocket Open

HTTP Method: GET

Profile: system

Interfaces: device

Attribute:

Device ID: example\_device\_id..

Parameter or URL ☒

param1=xxx&param2=yyy Send

deviceconnect.org&access\_token=3b1fe7739464e4709ff915ba1e9f60bdd200cb12a3c1d65ee20b0979218cf49dc68f0f1d1f2e

Response:

`{"result":0,"supports":["network_service_discovery","system","authorization"],"debug":"DevicePlugin","connect":{},"version":"1.0"}`

## 5. APPENDIX

### 5.1 非同期の処理について

デバイスによっては、処理に時間のかかる命令が存在する。その場合は次のコーディングポリシーに従う：例えばプロファイルで実装する `onGetReceive()` でそういった命令を処理する場合には、別のスレッドで処理を行わせ最終的にレスポンスを返させる様にし、`onGetReceive()` では `false` を返却して早々に処理を終える。

例えば、デバイスの検索に時間がかかる場合には、以下のサンプルのように別スレッドとして検索を行うこともできる。

処理が完了した後に、明示的に `sendBroadcast` を行い、レスポンスを返却する必要がある

ただし、`Device Connect Manager` も永遠にはレスポンスを待つことはしない。

タイムアウトが存在し、その時間内に返却がない場合には、リクエストはエラーとして処理されてしまう。

現実装では、30 秒でタイムアウトするので、この時間内にレスポンスを返却しなければならない。

#### ExampleNetworkServiceDiscoveryProfile.java

```
/** サンプルのデバイス ID を定義する. */
private static final String DEVICE_ID = "example_device_id";

/** サンプルのデバイス名を定義する. */
private static final String DEVICE_NAME = "Example Device";

/**
 * Network Service Discovery プロファイル.
 * [/network_service_discovery/getnetworkservices] に対応するメソッド.
 * @param request リクエスト
 * @param response レスポンス
 * @return 即座に返却する場合は true, 非同期に返却する場合は false
 */
@Override
public boolean onGetGetNetworkServices(final Intent request, final Intent response) {

    new Thread(new Runnable() {
        @Override
        public void run() {
            List<Bundle> services = new ArrayList<Bundle>();
```

```
        Bundle service = new Bundle();
        setId(service, DEVICE_ID);
        setName(service, DEVICE_NAME);
        setType(service, NetworkType.WIFI);
        setOnline(service, true);
        services.add(service);
        setServices(response, services.toArray(new Bundle[services.size()]));
        setResult(response, IntentDConnectMessage.RESULT_OK);
        getContext().sendBroadcast(response);
    }
    }).start();
    return false;
}
```

## 5.2 常駐 Service を使用しない軽量プラグイン

ここまでで説明してきた DConnectMessageServiceProvider による実装は、Android の Service を用いてデバイスとのセッションを保持し続けている。

しかし、デバイスの中には上記の様にセッションを保持する必要が無いものも存在する。

その場合に Service を起動しておくのはリソースの無駄遣いになる。それを回避するために DConnectMessageService を用いずに DConnectMessageProvider のみを使用する。

しかし、サービスを起動する分のリソースを節約できる分、セッションを保持できないという制約が有る事を十分留意してプラグイン制作を行わなければならない\*3。

\*3 例えば、DConnectMessageProvider を実装したクラスのメンバフィールドを使用して一時的にデータを格納しようとしても、DConnectMessageProvider 派生クラスのインスタンスのライフサイクルは受け取った DConnect メッセージの処理の間だけであり、それ以降は破棄される。そして新たな DConnect メッセージ受け取り時には全く別物の DConnectMessageProvider 派生クラスの新規インスタンスが作成されるので、前回そして新たに DConnect メッセージを受け取った DConnectMessageProvider 同士におけるデータを格納したメンバフィールドは全く別物となり、データを長期に渡って取り回すことができない。

### ExampleDeviceProvider.java (DConnectMessageProvider 版)

```
package com.mycompany.deviceplugin;

import com.nttdocomo.android.dconnect.message.DConnectMessageProvider;

/**
 * サンプルの DConnectMessageProvider 実装クラス.
 * @author docomo
 */
public class ExampleDeviceReceiver extends DConnectMessageProvider {

    /**
     * コンストラクタ.
     * ここで、DConnectMessageServiceProvider にプロファイルを追加することができる。
     */
    public ExampleDeviceReceiver() {
        addProfile(new ExampleNetworkServiceDiscoveryProfile());
        addProfile(new ExampleSystemProfile(this));
    }
}
```

このとき、AndroidManifest.xml は、以下のようになる。

## AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.mycompany.deviceplugin"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="19" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <!-- Device Connect Example Device Plugin Provider. -->
        <receiver android:name=".ExampleDeviceReceiver" >
            <meta-data
                android:name="com.nttdocomo.android.dconnect.deviceplugin"
                android:resource="@xml/deviceplugin" />

            <intent-filter>
                <action android:name="org.deviceconnect.action.GET" />
                <action android:name="org.deviceconnect.action.PUT" />
                <action android:name="org.deviceconnect.action.POST" />
                <action android:name="org.deviceconnect.action.DELETE" />
            </intent-filter>
        </receiver>
    </application>

</manifest>
```