DustyTuba Bluetooth Library Documentation

April 26, 2011

Abstract

This document outlines the three basic steps needed to incorporate the DustyTuba Bluetooth Library in your Android application

1 Step one - Include necessary files

Firstly, you need to copy the three folders (`libs`, `res` and `src`)-folder included in the archive to the root folder of the Android project in which you want to use the library.

Secondly, you must include the libraries in the `libs` folder into your project. In Eclipse, this can be done by right-clicking your project and selecting "Properties", then selecting "Java Build Path", and in the tab "Libraries" clicking the "Add JARs..." button and selecting the two included `.jar` files in the `libs` folder.

As is also described in the Bump™ documentation, you must import your project's `R` class in the `com.bumptech.bumpapi.BumpResources` source file – i.e. if your project has package name `com.example.test`, you must insert the following line into the `BumpResources.java` file:

```
import com.example.test.R;
```

2 Step two - Modify manifest

Now you must expand the Android manifest to include the activities provided and to use the permissions needed. The activities you must declare goes in the `<application>`-element and are the following:

```
<activity android:name="dk.hotmovinglobster.dustytuba.id.GenericIPActivity" />
<activity android:name="dk.hotmovinglobster.dustytuba.id.FakeIPActivity" />
<activity android:name="dk.hotmovinglobster.dustytuba.id.ManualIPActivity" />
<activity android:name="dk.hotmovinglobster.dustytuba.id.BumpIPActivity"
    android:configChanges="keyboardHidden|orientation"/>
<activity android:name="com.bumptech.bumpapi.BumpAPI"
    android:configChanges="keyboardHidden|orientation"
    android:theme="@style/BumpDialog" />
<activity android:name="com.bumptech.bumpapi.EditTextActivity"
    android:configChanges="keyboardHidden|orientation"
    android:theme="@style/BumpDialog" />
```

The permissions to be declared goes in the main `<manifest>`-element and are the following:

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

See appendix A for a complete sample manifest file.

3 Step three - Call the library

In order to call the library, you will use the `BtAPI.getIntent()` method (see section 3.1) to generate an `Intent` which must be passed on to Android's `startActivityForResult()` method.

When the activity contained in the `Intent` finishes, you must override the `onActivityResult()` method of your main activity to handle the result and extract the Bluetooth connection object.

To receive data from the other end of the Bluetooth connection, you need to have a class implement the `BtAPIListener` interface (usually this class would be your main activity).

For example, to invoke the manual identity provider[1], insert the following code where you want it to be invoked:

```
Intent i = BtAPI.getIntent(MainActivity.this, BtAPI.IDENTITY_PROVIDER_MANUAL);
startActivityForResult(i, REQUEST_DUSTYTUBA);
```

where `REQUEST_DUSTYTUBA` is an integer constant chosen to distinguish between the result of this activity and others you may be using. Also make your main activity implement the `BtAPIListener` interface and override the `onActivityResult()` method as follows:

```
@Override
protected void onActivityResult (int requestCode, int resultCode, Intent data) {
  if (requestCode == REQUEST_DUSTYTUBA) {
    if (resultCode == RESULT_CANCELED ) {
      // User canceled
    } else if (resultCode == RESULT_OK) {
      BtConnection conn = (BtConnection)data.getParcelableExtra(BtAPI.EXTRA_BT_CONNECTION);
    conn.setListener(this);
    }
  }
}
```

3.1 The `getIntent()` method

The `BtAPI` provides the two static methods `getIntent()` used to generate an `Intent` to launch a given identity provider:

```
public static Intent getIntent(Context context, String idProvider);
public static Intent getIntent(Context context, String idProvider, Bundle extras);
```

Both methods need a `Context` object and an identity provider to use (see section 3.2 for a list). Furthermore, the second method allows for passing on a bundle of information to the identity provider – which is used e.g. for providing the Bump™ with an API key.

[1] Se section 3.2 for a description of an identity provider

3.2 Identity providers

An identity provider is a way of pairing two bluetooth devices and exchanging their identities in order to make a bluetooth connection. As of now, three identity providers exist:

`BtAPI.IDENTITY_PROVIDER_BUMP` Use the Bump™ service to physically bump two phones together and exchange connection information[2].

The Bump™ service requires an API key, which can be obtained from their website[3]. This key must be provided to the identity provider as follows:

```
Bundle b = new Bundle();
b.putString(BumpAPI.EXTRA_API_KEY, BUMP_API_KEY);
Intent i = BtAPI.getIntent(this, BtAPI.IDENTITY_PROVIDER_BUMP, b);
startActivityForResult(i, REQUEST_DUSTYTUBA);
```

`BtAPI.IDENTITY_PROVIDER_FAKE` The fake identity provider simply returns the MAC address you supply to it. This enables you to use a MAC address obtained through other means. As with the Bump™ identity provider, we supply the MAC address through a `Bundle`:

```
Bundle b = new Bundle();
b.putString(BtAPI.EXTRA_IP_MAC, "00:00:00:00:00:00");
Intent i = BtAPI.getIntent(this, BtAPI.IDENTITY_PROVIDER_FAKE, b);
startActivityForResult(i, REQUEST_DUSTYTUBA);
```

`BtAPI.IDENTITY_PROVIDER_MANUAL` is an identity provider allowing a user to manually enter a MAC address through a dialog. The manual identity provider is started as shown in section 3. An optional default MAC address can be provided with the `Bundle` exactly as the fake MAC address was provided in the fake identity provider.

[2]The DustyTuba project is in no way affiliated with Bump™, we are merely using their service to obtain a bluetooth connection

[3]`http://bu.mp`

A Sample manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="dk.hotmovinglobster.dustytuba.apitest"
  android:versionCode="1"
  android:versionName="1.0">
  <uses-sdk android:minSdkVersion="4" />

  <application android:icon="@drawable/icon" android:label="@string/app_name">
    <activity android:name=".MainActivity"
      android:label="@string/app_name">
    <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
  </activity>
  <activity android:name="dk.hotmovinglobster.dustytuba.id.GenericIPActivity" />
  <activity android:name="dk.hotmovinglobster.dustytuba.id.FakeIPActivity" />
  <activity android:name="dk.hotmovinglobster.dustytuba.id.ManualIPActivity" />
  <activity android:name="dk.hotmovinglobster.dustytuba.id.BumpIPActivity"
    android:configChanges="keyboardHidden|orientation"/>
  <activity android:name="com.bumptech.bumpapi.BumpAPI"
    android:configChanges="keyboardHidden|orientation"
    android:theme="@style/BumpDialog" />
  <activity android:name="com.bumptech.bumpapi.EditTextActivity"
    android:configChanges="keyboardHidden|orientation"
   android:theme="@style/BumpDialog" />
  </application>

  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
  <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
  <uses-permission android:name="android.permission.INTERNET" />
  <uses-permission android:name="android.permission.VIBRATE" />
  <uses-permission android:name="android.permission.READ_PHONE_STATE" />
</manifest>
```