Android ADK Toolkit Documentation

Release 0.2.0

Emanuele Palazzetti

October 14, 2014

Contents

This toolkit helps beginners to be up and running with ADK 2012 without difficulties. ADK toolkit exposes an `AdkManager` to manage `UsbManager` and `UsbAccessory`. In this way you don't need to fully understand some background concept about how ADK works because you can simply access your ADK accessories with a `readSerial()` and `writeSerial()` methods.

Easy!

CHAPTER 1

ADK Toolkit

1.1 Overview

Accessory Development Kit allows you to build USB and Bluetooth accessories to extend the capabilities of your user's Android-powered devices. Android defines a standard protocol (AOA) that you can implement in your accessories and have it compatible with a wide range of Android devices.

1.1.1 Compatible Android devices

Android Open Accessory support is included in Android 3.1 (API Level 12) and higher, and supported through an Add-On Library in Android 2.3.4 (API Level 10) and higher. Check official documentation for more information.

> **Warning:** If your Android device uses API level 12+, it doesn't mean that it is ADK compatible. You should always check your smartphone/board specifications to see if they have this support.

1.1.2 Compatible Arduino devices

Your accessory should implement many features as described in building custom accessories section. Many Arduino devices have a built-in support for AOA protocol and there is a small list of what is supported:

- Arduino ADK
- Arduino Due
- UDOO board

Note: This list is community driven. There I will list **only** devices that me or other contributors have used. If you are pretty sure that other boards have this support, follow the contribution guidelines.

1.2 Usage

ADK 2012 is the latest version you can use to develop amazing accessories for Android-powered devices. This library only works as a wrapper of all ADK features for your Android app. If you want to see how ADK 2012 works, follow official documentation.

Note: Even if this library is a wrapper and supports Android API level 10, I will only target API level 12 in this

documentation as stated in cutting down backward support issue.

1.2.1 Install

This library is available in MavenCentral repository and its deployment flow is based on SonaType Nexus repository. To use the library simply configure your Gradle or Maven dependencies as follows:

Gradle

```
dependencies {
    compile 'me.palazzetti:adktoolkit:0.2.0'
}
```

Maven

```
<dependency>
    <groupId>me.palazzetti</groupId>
    <artifactId>adktoolkit</artifactId>
    <version>0.2.0</version>
    <type>aar</type>
</dependency>
```

Eclipse users

All published libraries in MavenCentral are in AAR format. Unfortunately, Eclipse seems to have a bug and AAR import will not work as expected. However there is an assemble task to produce a JAR library. To create the library simply launch in your root folder:

```
$ ./gradlew assembleJar
```

This will create a JAR library inside `adktoolkit/build/libs/` folder. Pre-assembled libraries are available in GitHub release section.

1.2.2 Android Manifest

Create `res/xml/usb_accessory_filter.xml` configuration file to identify your accessory:

```
<resources>
    <usb-accessory
        version="0.1.0"
        model="External-Droid"
        manufacturer="Example, Inc."/>
</resources>
```

Declare in your manifest that your application requires USB accessory support:

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

    <!-- ... -->
</manifest>
```

Then add in your activity block this ADK intent filter:

```
<manifest ...>
    <application ...>
        <activity ...>

            <!-- ... -->

            <!-- Adk Intent Filter -->
            <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/usb_accessory_filter"/>
        </activity>
    </application>
</manifest>
```

1.2.3 Android code

To use this toolkit initialize the `AdkManager` in your Activity during `onCreate()` method:

```
private AdkManager mAdkManager;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    mAdkManager = new AdkManager((UsbManager) getSystemService(Context.USB_SERVICE));
}
```

If you need to register a `BroadcastReceiver` to catch `UsbManager.ACTION_USB_ACCESSORY_DETACHED` action, you can use library default implementation as follows (always in your `onCreate()` method):

```
registerReceiver(mAdkManager.getUsbReceiver(), mAdkManager.getDetachedFilter());
```

1.2.4 Starting and stopping ADK listener

When you initialize an `AdkManager`, it just create a connection object between your device and your accessory. You need to start/stop AOA communication when you open/close your activity. Add these calls in your `onResume()` and `onPause()` methods:

```
@Override
protected void onPause() {
    super.onPause();
    mAdkManager.close();
}

@Override
protected void onResume() {
    super.onResume();
    mAdkManager.open();
}
```

Note: If you need to leave the activity without stopping the communication, you can avoid `mAdkManager.close()`. However don't forget to close the communication with a widget or a button in your activity so users can disable the accessory when they want. This avoid useless battery consumption.

1.2.5 Write and read serial text

As I write in my unittest, you can simply:

```
adkManager.writeSerial("Hello world!");
String readValue = adkManager.readSerial();
assertEquals("Hello world!", readValue);
// Not bad! :)
```

`writeSerial()` allows you to write a single char or a String object.

`readSerial()` read a single char or a String object until there are bytes to read in the buffer

Note: `readSerial()` could be a long-running task (ex: you want to continuously read data from a thermal sensor). In this case, put `readSerial()` call inside a `Service` or an `AsyncTask` and don't run this in your UI main thread.

1.3 Arduino communication

Your Android app needs an Arduino sketch as a counterpart. Here you can find a basic template:

```
#include <adk.h>

#define RCVSIZE 128

// Accessory descriptor. It's how Arduino identifies itself in Android.
char accessoryName[] = "Terminal echo";
char manufacturer[] = "Example, Inc.";
char model[] = "Terminal-echo";

char versionNumber[] = "0.1.0";
char serialNumber[] = "1";
char url[] = "http://www.example.com";

USBHost Usb;
ADK adk(&Usb, manufacturer, model, accessoryName, versionNumber, url, serialNumber);
uint8_t buffer[RCVSIZE];
uint32_t readBytes = 0;

void setup() {
    Serial.begin(115200);
    Serial.println("Ready to listen!");
    delay(2000);
}
```

1.3.1 Arduino and Android Manifest

Accessory descriptor defines how your accessory identifies itself in your Android app. These values should be the same as ones in `res/xml/usb_accessory_filter.xml` file otherwise your accessory will not find a suitable app to communicate with:

- `versionNumer`
- `model`
- `manufacturer`

You can use `url` variable to open an external link when any suitable application is found. There you can provide more information about how to configure your accessory. You can also target an apk or a Google Play Store URL where users can download and install your app.

1.3.2 Write and read serial text

You can use this snippet function to read text sent by your Android device:

```
void readFromAdk() {
    Usb.Task();

    if (adk.isReady()){
        adk.read(&readBytes, RCVSIZE, buffer);
        if (readBytes > 0){
            // Do something with buffer
        }
    }
}
```

If you want to write something to your Android device, use this snippet:

```
void writeToAdk(char textToSend[]) {
    adk.write(sizeof(textToSend), (uint8_t*)textToSend);
}
```

Then you can write text to Android device like this:

```
void loop() {
    Usb.Task();

    if (adk.isReady()){
        writeToAdk("Hello world!");
        delay(1000);
    }
}
```

1.3.3 Simple echo sketch

You can use this sketch to create an echo accessory which resend to Android every received characters:

```
void setup() {
    Serial.begin(115200);
    Serial.println("Ready to listen!");
    delay(2000);
}
```

```
void loop() {
    Usb.Task();

    if (adk.isReady()){
        adk.read(&readBytes, RCVSIZE, buffer);
        if (readBytes > 0){
            adk.write(readBytes, buffer);
        }
    }
}
```

CHAPTER 2

Project Info

2.1 Contributing

If you want to contribute you need to follow these guidelines. Otherwise your pull request will **not be accepted**.

2.1.1 Setup

Fork adk-toolkit repository on GitHub and follow these steps:

- Clone your repository locally
- Pull upstream changes into your fork regularly

It's a good practice to pull upstream changes from master into your fork on a regular basis, infact if you work on outdated code and your changes diverge too far from master, the pull request has to be rejected.

To pull in upstream changes:

```
git remote add upstream https://github.com/palazzem/adk-toolkit
git fetch upstream
```

Then merge the changes that you fetched:

```
git merge upstream/master
```

For more info, see http://help.github.com/fork-a-repo/

Note: Please be sure to rebase your commits on the master when possible, so your commits can be fast-forwarded: I'm trying to avoid merge commits when they are not necessary.

2.1.2 Issues

You can find the list of bugs, enhancements and feature requests on the issue tracker. If you want to fix an issue, pick up one and add a comment stating you're working on it. If the resolution implies a discussion or if you realize the comments on the issue are growing pretty fast, move the discussion to the Google Group.

2.1.3 How to get your pull request accepted

All Android ADK community want your code, so please follow these simple guidelines to make the process as smooth as possible.

Run the tests!

The first thing the core committers will do is to run all tests. Any pull request that fails this test suite will be **immediately rejected**.

Add the tests!

Even if the code coverage is not a good metric for code quality, it's better to add tests when you add code. If you find an issue that could be reproduced with a test, just add this test and solve the problem with a bugfix.

Code conventions matter

There are no good nor bad conventions, just follow official code style guidelines and nobody will argue. Try reading the code and grasp the overall philosophy regarding method and variable names, avoid black magics for the sake of readability, keep in mind that simple is better than complex.

2.2 Authors

2.2.1 Main developer

Emanuele Palazzetti <emanuele.palazzetti@gmail.com>

2.2.2 Contributors

No one... be the first!

2.3 Changelog

2.3.1 0.2.1 [2014-10-14]

- `writeSerial` now accept both `byte` and `String` values
- `readSerial` is now **deprecated** and default to `readString` method
- Added `readString` and `readByte` so you can read `String` and `byte` values from the serial port

Bugfixes

- Fixed documentation: #9

2.3.2 0.2.0 [2014-03-24]

- `FileInputStream` and `FileOutputStream` are `protected` so they can be mocked easily during testing
- Testing with Mockito

Bugfixes

- Better input/output stream management to avoid NullPointerException on Accessory loading

Backwards incompatible changes in 0.2.0

- Some class/method names are misleading so readText/sendText become readSerial/writeSerial and closeAdk/resumeAdk become close/open
- `AdkReceiver` has been removed because the actual implementation of read/write can handle multiple char

2.3.3 0.1.0 [2014-02-05]

- ADK fast constructor
- Simple default implementation of Broadcast receiver and IntentFilter
- Writing and reading features available
- Simple AsyncTask support

2.4 Migrate from previous versions

2.4.1 Migrate from 0.1.0 to 0.2.0

Incompatibility

Some class/method names are misleading so readText/sendText become readSerial/writeSerial and closeAdk/resumeAdk become close/open.

Rename in your project:

- `readText` to `readSerial`
- `sendText` to `writeSerial`
- `closeAdk` to `close`
- `resumeAdk` to `open`

Incompatibility

`AdkReceiver` has been removed because the actual implementation of read/write can handle multiple char.

If you have some `AsyncTask` which extend `AdkReceiver`, simply extend a regular AsyncTask and add a valid doInBackground method as follows:

```
public class MyAsyncTask extends AsyncTask<AdkManager, String, Void> {

    @Override
    protected Void doInBackground(AdkManager... params) {
        AdkManager adkManager = params[0];
        publishProgress(adkManager.readSerial());

        return null;
    }

    // Follows your implementation of MyAsyncTask
}
```