Android ADK Toolkit Documentation

Release 0.2.0

Emanuele Palazzetti

Sep 27, 2017

Contents

This toolkit is a collections of utilities that 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 avoid any complex understanding of how Android ADK works. If you want to create quickly your first accessory as fast as possible, this toolkit is a great getting started.

CHAPTER 1

ADK Toolkit

Overview

The Accessory Development Kit (ADK) allows you building USB or Bluetooth accessories that extend the capabilities of your user's Android-powered devices. Android defines the Android Open Accessory Protocol (AOA) used in accessories to create a communication channel between your Android application and your ADK compatible device.

Compatible Android devices

Android Open Accessory support is included since Android 3.1 (API Level 12), but a porting through an Add-On Library was available even in Android 2.3.4 (API Level 10). Check the official documentation for more information.

> **Warning:** Even if your Android device uses an API level 12+, it doesn't mean that it is ADK compatible. The ADK support is related to some hardware specifications that your smartphone / board manufacturer should comply. Before you proceed, check if your smartphone or board have a full ADK support.

Compatible ADK devices

Your accessory should implement many features as described in building custom accessories section. Many Arduino devices have a built-in support for ADK (and so AOA protocol) and the following is a list of supported devices:

- 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.

Usage

ADK 2012 is the latest version that can be used during accessories development. This library works as a wrapper of ADK capabilities so your application development will be smoothest. If you want to see how ADK 2012 works, follow the official documentation.

Note: Even if ADK is supported since Android API level 10, I will only target API level 12 in this library as stated in cutting down backward support issue.

Installing the library

This library is available in MavenCentral and JCenter repositories. Adding the library dependency is pretty easy and you can configure your Gradle or Maven dependency file as follows:

Gradle

```
dependencies {
    compile 'me.palazzetti:adktoolkit:0.3.0'
}
```

Maven

```
<dependency>
    <groupId>me.palazzetti</groupId>
    <artifactId>adktoolkit</artifactId>
    <version>0.3.0</version>
    <type>aar</type>
</dependency>
```

Eclipse users

All published libraries in MavenCentral or JCenter are in AAR format. Unfortunately, Eclipse seems to have a bug and AAR import will not work as expected. I've created an assemble task in the gradle build script to produce a JAR library that you can easily import manually in your project. The pre-assembled libraries are available in the repository release section.

Note: If you are using Eclipse with ADT, be aware that Android Studio is now the official IDE for Android, so it's a good idea to migrate your projects to Android Studio. For help moving projects, see Migrating to Android Studio. Despite that, ADKToolkit library will continue to support JAR library releases.

Configuring the Android application

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>
```

Starting the ADK listener

To use this library, initialize the `AdkManager` in your `Activity` `onCreate()` callback like you can see in the following snippet:

```
private AdkManager mAdkManager;

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

    mAdkManager = new AdkManager(this);
}
```

If you need to register a `BroadcastReceiver` to catch `UsbManager.ACTION_USB_ACCESSORY_DETACHED` action, you can use the library default implementations as follows:

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

    mAdkManager = new AdkManager(this);
    registerReceiver(mAdkManager.getUsbReceiver(), mAdkManager.getDetachedFilter());
}
```

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 the following calls in your `onResume()` and `onPause()` callbacks to open and close ADK communication, when your Activity is resumed or paused:

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

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

Warning: Because of an internal ADK bug that is still not fixed, it's not possible to open the ADK again when the channel has been closed. This means that if you need to use the ADK between activities, you should not call the `close()` method otherwise the only way to open the communication again is to restart your hardware accessory.

Using the toolkit

The ADKToolkit library exposes an interface to write and read bytes in/from the internal ADK buffer. If you need to send some values to your accessory, you can use the following methods within your application code:

```
byte[] byteArray = {4, 2};
byte byteValue = 42;
int intValue = 42
float floatValue = 42.0f;
String stringValue = "Answer to The Ultimate Question of Life, the Universe, and␣
↪Everything"

adkManager.write(byteArray);
adkManager.write(byteValue);
adkManager.write(intValue);
adkManager.write(floatValue);
adkManager.write(stringValue);
```

On the other hand if you need to read a value from your accessory (for instance, a sensor value), you can use the `read()` method that returns an `AdkMessage` instance. This class, wraps the returned `byte[]` array from the buffer and exposes an API to parse retrieved value. The following is an example how to read accessory data from your Android application:

```
AdkMessage response = mAdkManager.read();
```

Then you can call the following methods according to sent data:

- `response.getBytes()`: returns the raw bytes array so you can manipulate it on your own
- `response.getString()`: returns a string applying a `(char)` typecasting for each byte
- `response.getByte()`: returns the first byte of the bytes array buffer
- `response.getFloat()`: expects that the content of the bytes array buffer is a string; it calls the `getString()` method and tries to parse the string in a float value
- `response.getInt()`: expects that the content of the bytes array buffer is a string; it calls the `getString()` method and tries to parse the string in an integer value
- `response.isEmpty()`: returns `true` if the received buffer is empty or not initialized

If for any reasons the parsing causes an exceptions, it will be caught from the AdkResponse's methods and the returned value will be `null`.

Note: The `read()` method could be a long-running task, in particular if you want to read continuously data from a your accessory. In this case, call the `read()` method outside your main thread otherwise it will cause the Application Not Responding (ANR) error. A good approach is to use an IntentService, an AsyncTask, or a Runnable implementation together with an ExecutorService.

Arduino communication

Your Android application needs an Arduino sketch as a counterpart. Here you can find a basic template that shows how to initialize your **accessory descriptor** to let Android knows that an accessory is connected to the system:

```
#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() {
    // Some setup operations
}
```

Arduino and the Android Manifest

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

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

You can use `url` variable to open an external link when any application capable to manage this accessory is found. In this web page you can provide more information about how to configure your accessory.

Note: You can also target a `.apk` package or a Google Play Store URL where users can download and install your app.

Interacting with Android

When you're connecting your accessory to Android you can have the following scenarios:

- You want to collect data from your accessory (maybe some sensors) and send them back to your

Android application * You want to provide to users an interaction, using the Android user interface, and do some actions with accessory actuators

Despite what is the scope of your accessory, the following are brief examples how you can read and write data from your Arduino sketch.

Reading

Within your `loop()` function, add the following code to read data from the ADK buffer:

```
// readBytes, RCVSIZE and buffer are declared in the first snippet

void loop() {
    Usb.Task();

    if (adk.isReady()){
        adk.read(&readBytes, RCVSIZE, buffer);
        if (readBytes > 0){
            // Do something with buffer
        }
    }
    // Don't forget a delay in your sketch :)
}
```

Writing

If you want to send data to your Android device, use the following snippet:

```
// buffer is already declared in the first snippet

void loop() {
    Usb.Task();

    if (adk.isReady()){
        adk.write(sizeof(buffer), buffer);
    }
    // Don't forget a delay in your sketch :)
}
```

Remember that if the accessory needs both the writing and the reading phase, it could be a good idea to use two different `buffer` objects; for this reason you may want to declare two `uint8_t` buffers.

Simple echo sketch

You can use this sketch to create an echo accessory which sends received characters back to Android:

```
uint8_t readingBuffer[RCVSIZE];
uint8_t writingBuffer[RCVSIZE];

void setup() {
    // Nothing to do with this sketch
}

void loop() {
    Usb.Task();

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

CHAPTER 2

Project Info

Contributing

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

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.

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.

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.

Authors

Main developer

Emanuele Palazzetti <emanuele.palazzetti@gmail.com>

Contributors

No one... be the first!

Changelog

0.3.0 [2015-01-10]

New features

- Updated to latest gradle version `1.0.0`
- Added `AdkMessage` class, which exposes the raw `byte[]` array with some utility methods to get string, byte, int and float representations

- Issue #13: refactoring `AdkManager` to expose a common interface for `read()` and `write()`
- Issue #16: `AdkManager` constructor now accept an `Activity` context to initialize the accessory

Backwards incompatible changes from 0.2.x

- removed `writeSerial(String text)`
- removed `writeSerial(int value)`
- removed `readSerial()`
- removed `readString()`
- removed `readByte()`

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

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

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

Migrate from previous versions

Migrate from 0.2.x to 0.3.0

Improvement

The old initialization uses the following code:

```
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));
}
```

Now you can also create the AdkManager instance passing the Activity `Context` like you see in the following code:

```
private AdkManager mAdkManager;

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

    mAdkManager = new AdkManager(this);
}
```

Incompatibility

The following methods have been deleted:

- `writeSerial(String text)`
- `writeSerial(int value)`
- `readSerial()`
- `readString()`
- `readByte()`

For this reason you should rename in your project:

- `writeSerial()` to `write()`
- `readSerial()` to `read().getString()`
- `readString()` to `read().getString()`
- `readByte()` to `read().getByte()`

Note: Remember that `read()` returns an `AdkMessage` instance and you may want to cache this response in a variable.

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
}
```