AT-UserGuide

Android-Treiber User Guide

Version 1.3
(13. September 2012)

Internal Document

Release Status:
DRAFT

Index

1 DOCUMENT INFORMATION 3

1.1 Change History 3

1.2 Distribution & Approval History 3

1.3 Related Documents 3

2 TERMS AND ABBREVIATIONS 4

3 PURPOSE OF THIS DOCUMENT 5

3.1 Structure of this document 5

4 MINIMUM SYSTEM REQUIREMENTS 6

4.1 Supported smart card readers 6

4.2 USB API test 6

4.3 Note on supported devices for USB smart card readers 6

5 INSTALLATION OF ANDROID PACKAGES (`*.apk`) 7

5.1 Installation via USB 7

5.1.1 Android SDK 7

5.1.2 USB drivers (Windows operating systems only) 7

5.1.3 USB debugging 8

5.2 Installation via HTTP 9

5.2.1 Allow installation of Android Apps from unknown sources 9

5.2.2 Uninstallation of installed Apps on the device 9

6 USAGE OF THE MANAGEMENT APP 10

6.1 USB readers 10

6.2 Bluetooth readers 10

6.2.1 Device discovery process 10

6.2.2 Bluetooth activation 11

6.2.3 Unpair paired Bluetooth devices 11

7 JSR268 API 12

7.1 Usage of the JSR268 API in individual Apps 12

7.2 Complete example Activity 13

8 INSTALLATION OF MANAGEMENT APP FROM WITHIN A THIRD-PARTY APP 15

8.1 Check installation status 15

8.2 Prepare Android package for installation 15

8.3 Package installation 16

1 Document Information

1.1 Change History

Version	Date	Author	Description
1.0	29.08.2012	HID Global	Initial revision
1.1	04.09.2012	HID Global	Add notes on system requirements, setup of development environment, Android package creation
1.2	05.09.2012	HID Global	Add notes on Android package installation from within another Android package
1.3	12.09.2012	HID Global	Adaptation of Bluetooth description due to storage of MAC addresses in App preferences

1.2 Distribution & Approval History

Version	Distributed to / approved by	Date distributed	Date approved

1.3 Related Documents

Abbrev	Description

2 Terms and Abbreviations

Abbrev	Description

3 Purpose of this document

The *Android-Treiber* project aimed at providing usage of the OK 1021, 3021 and 2061 USB and Bluetooth smart card readers on the Android operating system. For this, a management App[1] has been implemented and the access to the smart cards is revealed via the well-known JSR268 API[2].

This document describes the usage of smart cards and readers under the Android operating system, using the aforementioned management App, as well as the smart card access via the JSR268 API.

3.1 Structure of this document

In Section 4, the minimum system requirements as well as a simple test if the Android device supports the correct USB mode is outlined. Section 5 describes the installation of Android Apps via the Android package (`*.apk`) file. This installation process is shown via USB and via HTTP connection. The next section highlights the usage of the management App, where USB and Bluetooth transmission is highlighted. In the final sections the JSR268 API, and an example App that uses the JSR268 library to access card readers and smart cards, is exhibited, including the installation of the management App from within the JSR268 example App.

---

1. Android application
2. http://jcp.org/aboutJava/communityprocess/final/jsr268/index.html

4 Minimum system requirements

In order to use the components described in this document, the following minimum system requirements have to be met:

- Android device (tablet or smart phone) with Android operating system version 3.2 (*Honeycomb*) or higher. This corresponds to Android API level 13 or higher.
- API support for `android.hardware.usb`.
- API support for `android.bluetooth`.

The Apps can be installed without special privileges, but the user may have to grant access to USB devices and the Bluetooth stack during operation, if this is not already set.

4.1 Supported smart card readers

According to the project goals, the following USB and Bluetooth smart card readers are supported:

- OmniKey 1021 (USB),
- OmniKey 3021 (USB),
- OmniKey 2061 (Bluetooth).

4.2 USB API test

To check whether the Android devices supports USB host (OTG[1]) mode, the USB device enumerator, a free App from the Android Play Store, can be used, which can be installed directly via the Play Store App or downloaded from the Play Store web page[2].

4.3 Note on supported devices for USB smart card readers

Basically all Android tables and smart phones which support the USB host (OTG) mode and the Android API level 13 or higher are usable. However, not all devices may offer a USB port to fit the USB Type A plug of the smart card readers. Please check what kind of adapter cables are needed to properly connect the reader to the devices. Due to inconsistencies between device manufacturers, especially adapters from micro-USB to USB are not always compatible between different vendors.

---

[1] On-the-go, USB interface feature

[2] https://play.google.com/store/apps/details?id=aws.apps.usbDeviceEnumerator

5 Installation of Android packages (`*.apk`)

To use to described features it is first necessary to install the package on the Android device. This installation process encompasses the delivered management App (CardReaderManager.apk) as well as third-party Apps that use the JSR268 library for card reader communication.

The installation can be done in two ways, via USB and via HTTP. This section will describe the steps necessary to complete the installation for both variants.

5.1 Installation via USB

When installing Android packages via the USB connection, the following prerequesits have to be met:

1. The Android SDK[1] has to be installed on the host operating system.
2. A USB driver for the device has to be installed (Windows operating systems only).
3. USB debugging has to be enabled on the device.

5.1.1 Android SDK

Independent of the operating system used, the SDK for Android can be downloaded under http://developer.android.com/sdk/index.html.

After installation, the *SDK Manager* can be used to install required packages. For package installation, the *Android SDK Tools* as well as the *Android SDK Platform-tools* have to be installed. Further, at least one of the platforms supporting API level greater of equal to 13 shall be installed, which is necessary for Android development. To enable specific APIs, like Fragment or AsyncTaskLoader, the current version of the *Android Support Library* also has to be installed. For more information on package installation via the SDK Manager, see http://developer.android.com/sdk/installing/adding-packages.html and http://developer.android.com/tools/extras/support-library.html.

5.1.2 USB drivers (Windows operating systems only)

If the host operating system is Windows (XP, 7 or Vista), further USB drivers for the device are necessary in order to be able to establish the device connection via `adb`. On the Android web page (see http://developer.android.com/tools/extras/oem-usb.html), a table lists various vendors and links to their support pages, where the drivers can be downloaded.

---

[1] Software development kit

5.1.3 USB debugging

On the Android device itself, USB debugging has to be enabled so that the device accepts the `adb` connection and grants the reception of commands like package removal and installation. The USB debugging configuration option is found in different places under Android 3.2 and Android 4.x.

- Android 3.2: Navigate to *Settings* (e.g. via the *Applications* menu) and select *Applications* > *Development*. In this menu, the *USB debugging* option can be enabled.
- Android 4.x: Navigate to *Settings* (e.g. via the *Applications* menu). Then an entry *Developer options*, shown under the *System* heading, has to be selected, where the *USB debugging* option can be found.

After the Android SDK and the tools and necessary USB drivers are installed and USB debugging is activated on the device, the device can be plugged to the host via a USB device cable. The Android device shall then be automatically detected, which can be verified via the `adb` tool that is part of the tools installed via the SDK Manager.

In a console (Linux or Windows), navigate to the directory where the Android SDK has been installed. The following example assumes a Windows operating system and the installation path to be `C:\Programme\Android\android-sdk`. The `adb` tool is installed in the `platform-tools` directory of the Android SDK. When called with the `devices` parameter, it shows all Android devices that are currently connected via USB.

```
C:\> cd Programme\Android\android-sdk
C:\Programme\Android\android-sdk> cd platform-tools
C:\Programme\Android\android-sdk\platform-tools> adb.exe devices
List of devices attached
0149CCCF0501300B        device
```

When the device is shown in the above output, the USB connection has been set up successfully and Android packages can be installed and removed via `adb`. For this, the command line parameters `install` and `uninstall` can be used. With `install`, the path to the `*.apk` file has to be given while `uninstall` takes the fully-qualified package name, for example:

```
C:\Programme\Android\android-sdk\platform-tools>   adb.exe   install
"C:\Dokumente und Einstellungen\user\Eigene Dateien\app.apk"
860 KB/s (220167 bytes in 0.250s)
        pkg: /data/local/tmp/app.apk
Success

C:\Programme\Android\android-sdk\platform-tools>  adb.exe  uninstall
com.example.app
Success
```

Once installed, the USB connected can be unplugged and the App can be used right away on the Android device (see Section 6).

5.2 Installation via HTTP

If the Android package that shall be installed is provided on a mirror with HTTP access, another installation option is to directly download and install the package on the Android device. Therefore, no USB connection is necessary for this procedure, but the following prerequesits have to be met:

1. The Android device must have access to the HTTP server where the Android package(s) are located.
2. The user has to allow the installation of Android Apps that are not coming from the Android Play Store

5.2.1 Allow installation of Android Apps from unknown sources

Normally, Android Apps are installed via the Android Play Store, where the packages that are provided passed certain security criteria. Therefore the security policy of Android intends to have to user explicitly allow the installation from unknown sources. The appropriate configuration option is found in different places under Android 3.2 and Android 4.x.

- Android 3.2: Navigate to *Settings* (e.g. via the *Applications* menu) and select *Applications*. In this menu, the *Unknown sources* option can be enabled.
- Android 4.x: Navigate to *Settings* (e.g. via the *Applications* menu). Then an entry *Security*, shown under the *Personal* heading, has to be selected, where the *Unknown sources* option can be found under the heading *Device administration*.

Once the unknown sources option is enabled, the `*.apk` can be downloaded and installed easily. First download the Android package with the the browser. By default, Android downloads its files to a certain directory whose contents can be listed via the *Downloads* menu, which is found under the *Applications* menu in both, Android 3.2 and Android 4.x.

The installation is started by clicking on the downloaded package in the *Downloads* menu. If the package that shall be installed has already been installed before, a message is shown that highlights this fact. In a next dialog the privileges are outlined that are claimed by the App to be installed. After a click on the Install button, the Android App will be installed and can be used right away (see Section 6).

5.2.2 Uninstallation of installed Apps on the device

Besides usage of `adb` (see Section 5.1), installed Apps can also be uninstalled directly on the device.

- Android 3.2: Navigate to *Settings* (e.g. via the *Applications* menu) and select *Applications*. In this menu, select *Manage applications* and switch to the *Downloaded* tab. On click of an App from the list, a dialog shows up where the App can be uninstalled via the *Uninstall* button.
- Android 4.x: Navigate to *Settings* (e.g. via the *Applications* menu) and select *Apps*. From there switch to the *Downloaded* tab. On click of an App from the list, a dialog shows up where the App can be uninstalled via the *Uninstall* button.

6 Usage of the management App

Dependent on the actual reader type (USB or Bluetooth), there are small usage differences.

6.1 USB readers

When one of the two supported USB readers is plugged into the USB interface of the Android device, a dialog and afterwards the management App appears. The dialog informs the user about the plugged in USB device and whether the associated App shall be started, which has to be answered with `OK`. If the associated App shall be started by default, a checkbox can be checked and on the next USB plug-in event the App will start automatically without the before-appearing dialog. Then the management App shows a list of currently managed devices, where the newly attached device shall be included. If not, a click on the *Refresh* button at the right corner of the App screen will reload the reader list.

Further reader information (name, activity and usage status) are shown when clicking on a reader name from the list. This screen is also used to activate or deactivate a reader. For example, if multiple readers are attached to a device, but the user only wants to expose some of them to applications via the JSR268 interface, those which shall not be propagated can be deactivated in the reader information screen.

6.2 Bluetooth readers

Once a Bluetooth reader has been used for the first time with the management App (i.e. it has been discovered and paired with the Android device, see Section 6.2.1), its MAC address is stored in the App preferences. If Bluetooth is enabled on the Android device, the management App reads its preferences on startup and adds all those Bluetooth devices to the list of available readers, whose MAC addresses are found in the App preferences and likewise in the list of paired devices that is fetched via the Android API. This way a Bluetooth reader is made available immediatly after startup of the management App on the second and following usage, without the necessity of doing another discovery process.

This procedure implies that listed Bluetooth readers might not be available all the time. As a consequence, only for actual Bluetooth operations (i.e. data transmission), as well as for the discovery of Bluetooth devices (see Section 6.2.1), at least one applicable Bluetooth device (i.e. a OK 2061 reader) has to be near the Android device.

6.2.1 Device discovery process

To initiate the Bluetooth discovery process, the management App provides a *Bluetooth discovery* button at the right corner of the screen. Once clicked, Android will ask the user if Bluetooth shall really be activated (if it is not yet activated on the device) and afterwards the Bluetooth device discovery process starts. This process lasts about 10-15 seconds and at the end either provides a list of names of found peripheral devices or a message telling that no applicable devices have been discovered.

If at least one applicable device has been discovered, a list is shown and the user has to choose the device according to the MAC address and the name (if one is set for the device) in the list. After clicking on the desired entry in the list, the pairing process will be initiated, during which Android asks for the Bluetooth PIN (if this is the first time the Bluetooth device and the Android device are pairing). Afterwards, the management App adds the Bluetooth device to the list of readers and the information screen can be invoked in the same way as described above for USB readers.

6.2.2 Bluetooth activation

By default Bluetooth might not be enabled on the Android device. It can be enabled in the following way:

- By starting the management App and clicking the *Bluetooth discovery* button. As described above, during this process Bluetooth is enabled on the device.
- Manually:
    - Android 4.x: Navigate to *Settings* (e.g. via the *Applications* menu) and switch the button next to *Bluetooth* from *Off* to *On*.
    - Android 3.2: Navigate to *Settings* (e.g. via the *Applications* menu) and select *Wireless and Network*. In this menu select *Bluetooth settings* and then click the *Bluetooth (Turn on Bluetooth)* button.

6.2.3 Unpair paired Bluetooth devices

Android stores the pairing information after initial pairing with a Bluetooth device. If for any reason this pairing shall be deleted, this can be done in the following way:

- Android 3.2: Navigate to *Settings* (e.g. via the *Applications* menu) and select *Wireless and Network*. In this menu select *Bluetooth settings*, where a list of paired devices are shown. From this list, for each device the pairing can be detached.
- Android 4.x: Navigate to *Settings* (e.g. via the *Applications* menu) and select *Bluetooth*. There a list of paired devices are shown and from this list the pairing for each device can be detached.

7 JSR268 API

Access to the readers and inserted smart cards is provided from the management tool to user Apps via the public JSR268 API. This API provides several classes and methods for reader listing, card presence detection as well as data transmission. It is thoroughly documented under http://download.oracle.com/otndocs/jcp/jscio-4.0-fr-eval-oth-JSpec/.

7.1 Usage of the JSR268 API in individual Apps

As one main purpose of this project was to allow any App to access readers and smart cards via the JSR268 API, besides management and testing Apps a library (as Jar[1] file) is provided which encapsulates the JSR268 interface and can be included by any App to access the readers and smart cards via the management tool.

If Android development, either with or without the Eclipse framework, is set up according to http://developer.android.com/sdk/installing/index.html, the following steps are necessary in order to successfully use the JSR268 API in a new App (the delivered Jar file is called `jsr268library.jar` in the following how-to).

1. Copy the Jar file to the `libs` folder of the Android project (via Windows Explorer).
2. Using the Eclipse framework:
   (a) Right click on the project, select Properties.
   (b) Select Java Build Path and activate the Libraries tab.
   (c) Click the Add JARs button and choose the jsr268library.jar from the `libs` folder.
   (d) The library shall then be visible under the `Referenced Libraries` folder of the project.
3. Import the following two packages from the library:

```
import android.smartcardio.ipc.ICardService;
import android.smartcardio.ipc.CardService;
```

3. Add the following variables to your Activity:

```
private ICardService mService;
private TerminalFactory mFactory;
```

4. Initialize the CardService, e.g. in the Activity's `onCreate` method. The CardService will bind to the management tool (backend) automatically.

   For the initialization, the CardService needs the current object to be passed as the application context:

```
mService = CardService.getInstance(this);
```

5. The CardService object shall also be told to release the binding at the end of an operation, e.g. in the Activity's `onDestroy` method:

---

[1] Java archive

```
@Override
public void onDestroy() {
  super.onDestroy();
  mService.releaseService();
}
```

6. After initializing the CardService, the TerminalFactory can also be initialized. This object is the actual entry point for the JSR268 API methods and classes.

   ```
   mFactory = mService.getTerminalFactory();
   ```

7. The TerminalFactory object is actually an object of a subclass of `javax.smartcardio.TerminalFactory`. Therefore the operation then is confined to using the JSR268 API according to its documentation.

   A simple example, taken from the official JSR268 documentation[1], may look like this:

   ```
   List<CardTerminal> terminals = mFactory.terminals().list();
   // Get the first terminal and establish a card connection.
   CardTerminal terminal_1 = terminals.get(0);
   Card card = terminal_1.connect(“T=0”);
   CardChannel channel = card.getBasicChannel();
   // Send a command to the card and receive the response.
   byte[] command = { 0x00, 0xA4, 0x00, 0x0C, 0x02, 0x3F, 0x00 };
   ResponseAPDU response = channel.transmit(new CommandAPDU(command));
   // Interpret response, do further work.
   // ...
   // At the end, release card connection.
   card.disconnect(true);
   ```

Note that all CardService operations may throw an exception, as documented in the JSR268 API. Therefore the above example code shall be embedded in a `try-catch` block.

7.2 Complete example Activity

The following example uses the management tool (backend) to interact with readers and smart cards and incorporates the card communication mentioned above.

```
import android.smartcardio.ipc.ICardService;
import android.smartcardio.ipc.CardService;


import android.os.Bundle;
import android.app.Activity;
import android.util.Log;
import android.view.View;
```

---

[1] see for example http://docs.oracle.com/javase/6/docs/jre/api/security/smartcardio/spec/javax/smartcardio/package-summary.html, or the downloadable documentation mentioned above

```
import android.smartcardio.Card;
import android.smartcardio.CardChannel;
import android.smartcardio.CardException;
import android.smartcardio.CardTerminal;
import android.smartcardio.CommandAPDU;
import android.smartcardio.ResponseAPDU;
import android.smartcardio.TerminalFactory;

public class MainActivity extends Activity {
  private ICardService mService;
  private TerminalFactory mFactory;

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

    mService = CardService.getInstance(this);
  }

  @Override
  public void onDestroy() {
    super.onDestroy();
    mService.releaseService();
  }

  // Here, the card communication is initiated with a Button click.
  public void onClick(View view) {
    try {
      if (mFactory == null) {
        mFactory = mService.getTerminalFactory();
      }

      List<CardTerminal> terminals = mFactory.terminals().list();
      // Get the first terminal and establish a card connection.
      CardTerminal terminal_1 = terminals.get(0);
      Card card = terminal_1.connect("T=0");
      CardChannel channel = card.getBasicChannel();
      // Send a command to the card and receive the response.
      byte[] command = { 0x00, 0xA4, 0x00, 0x0C, 0x02, 0x3F, 0x00 };
      ResponseAPDU response = channel.transmit(
                                new CommandAPDU(command));
      // Interpret response, do further work.
      // ...
      // At the end, release card connection.
      card.disconnect(true);
    } catch (CardException e) {
      // ...
    }
  }
}
```

8 Installation of management App from within a third-party App

When a third-party App using the JSR268 API is implemented which attempts to interact with card readers and smart cards, it is also useful to only have to install one `*.apk` (the one for the third-party App), which automatically installs the management App on the first startup (if not already installed).

In order to accomplish this task, a few things have to be kept in mind:

1. Copy the management App’s `*.apk` file (e.g. `CardReaderManager.apk`) to the `assets` folder of the Android App project (via Windows Explorer),
2. check whether the management App is already installed,
3. copy the `*.apk` file from the assets folder to an external storage (e.g. cache),
4. install the `*.apk` and check if the installation was successful.

Further the card service connection to the management App (i.e. backend), whose setup is described under list item 4. of Section 7.1, has either to be initialized at App startup or after the management App has been installed. Otherwise no connection is established and the JSR268 API calls will fail, as the library would guess that there is no backend available.

8.1 Check installation status

The current installation status of an application can be queried using the PackageManager class and the full-qualified package name of the desired application.

```
try {
  PackageManager pm = getPackageManager();
  pm.getPackageInfo(“com.sample.app”, PackageManager.GET_ACTIVITIES);
  // App already installed.
} catch (PackageManger.NameNotFoundException e) {
  // App not yet installed.
}
```

8.2 Prepare Android package for installation

Unfortunately, Android does not support the installation of packages directly from the `assets` folder of an `*.apk`. This means that the `*.apk` file has to be copied from the `assets` folder to an external storage, for example the external cache directory.

In order to copy the `*.apk` file to your App needs the permission to write to the external storage. This permission can be claimed in the `AndroidManifest.xml` file via an entry

```
<uses-permission
 android:name=”android.permission.WRITE_EXTERNAL_STORAGE” />
```

which has to be included directly as child-tag of the `<manifest>` tag.

The following example assumes that the *.apk file in the assets folder is called `CardReaderManager.apk`.

```
// Copy the .apk file from the assets directory to the external
// cache.
File temp = File.createTempFile(“CardReaderManager”, “apk”,
            getExternalCacheDir());
temp.setWriteable(true);
FileOutputStream out = new FileOutputStream(temp);
InputStream in = getResources().getAssets().open(
                 “CardReaderManager.apk”);
byte[] buffer = new byte[1024];
int bytes = 0;

while ((bytes = in.read(buffer)) != -1) {
  out.write(buffer, 0, bytes);
}
in.close();
out.close();
// This path is important for the following installation.
String cachePath = temp.getPath();
```

8.3 Package installation

Once prepared, the package installation is an easy and straightforward task which calls a predefined Android App that handles the installation procedure. The only information necessary is the path to the `*.apk` file which has to be installed, which has been stored in the `cachePath` variable in the above example.

```
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setDataAndType(Uri.fromFile(new File(cachePath)),
                      “application/vnd.android.package-archive”);
startActivity(intent);
```

This above code prompts the user with a dialog that asks whether the App shall be installed and also lists the permissions needed by this App.