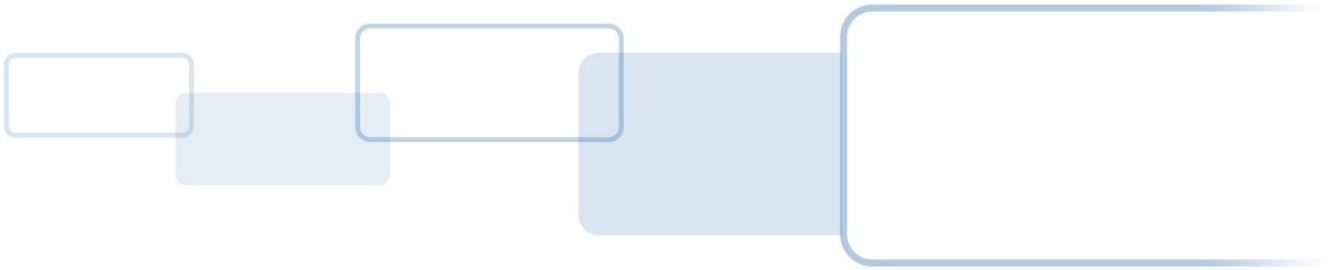
# OMNIKEY ANDROID DRIVER

## USER GUIDE

PLT-02492, Rev. 1.4

September 2016



hidglobal.com

## Copyright



## Trademarks

HID Global, HID, the HID logo, and OMNIKEY are the trademarks or registered trademarks of HID Global Corporation, or its licensors, in the U.S. and other countries.

## Revision History

| Date | Description | Version |
|---|---|---|
| September 2016 | New Part Number. OMNIKEY 5127 and OMNIKEY 5127 Mini readers added. | 1.4 |
| September 2012 | Adaptation of Bluetooth description due to storage of MAC addresses in App preferences. | 1.3 |
| September 2012 | Added notes on Android package installation from within another Android package. | 1.2 |
| September 2012 | Added notes on system requirements, setup of development environment, Android package creation. | 1.1 |
| August 2012 | Initial release. | 1.0 |

## Contacts

For additional offices around the world, see www.hidglobal.com corporate offices.


| North America | Asia Pacific |
|---|---|
| 611 Center Ridge Drive<br>Austin, TX 78753<br>USA<br>Phone: 866-607-7339<br>Fax: 949-732-2120 | 19/F 625 King's Road<br>North Point, Island East<br>Hong Kong<br>Phone: 852 3160 9833<br>Fax: 852 3160 4809 |
| **Europe, Middle East and Africa** | **Brazil** |
| Haverhill Business Park Phoenix Road<br>Haverhill, Suffolk CB9 7AE<br>England<br>Phone: 44 (0) 1440 711 822<br>Fax: 44 (0) 1440 714 840 | Condomínio Business Center<br>Av. Ermano Marchetti, 1435<br>Galpão A2 CEP 05038-001<br>Lapa - São Paulo/SP<br>Brazil<br>Phone: 55 11 5514-7100 |
| HID Global Customer Support: www.hidglobal.com/support | |

# Contents

# 1 Overview

## 1.1 Description

The OMNIKEY® Android driver package brings support for OMNIKEY smart card readers to the Android Operating System (OS). A management application, **CardReaderManager**, provides access to smart cards via the well-known JSR268 API.

This document describes the usage of smart cards and readers under the Android OS, using the **CardReaderManager** management app, as well as the smart card access via the JSR 268 Smart Card IO API [JSR 268 API].

## 1.2 Supported Smart Card Readers

The following smart card readers are supported:

- smart@link chipset
- OMNIKEY USB Readers: 1021, 3021, 3121, 3621, 3821, 6121, 5427, 5022, 5127, 5127 Mini
- OMNIKEY Bluetooth Reader: 2061

# 2 Getting Started

## 2.1 Minimum System Requirements

- Android device (tablet or smart phone) with On The Go (OTG) support.
- Android OS 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.

## 2.2 Supported Devices for USB Smart Card Readers

Android tablets and smart phones which support the USB host (OTG) mode and Android API level 13 or higher are supported. 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, adapters from micro-USB to USB are not always compatible between different vendors.

## 2.3 USB API Test

To check whether an Android device supports USB host (OTG) mode, a free app, *USB Device Info*, can be downloaded from the Play Store. This app can be installed directly via the Play Store on the device or downloaded directly from the Play Store web page:
https://play.google.com/store/apps/details?id=aws.apps.usbDeviceEnumerator.

# 3 Installation of Android Packages (*.apk)

To use the described features it is first necessary to install the package on the Android device. This installation process includes the installation of a Management App (CardReaderManager.apk), together with 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.

## 3.1 Installation via USB

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

- The Android SDK has to be installed on the host OS
- USB driver for the device has to be installed (Windows OS only)
- USB debugging has to be enabled on the device

### 3.1.1 Android SDK

Independent of the OS used, the SDK for Android can be downloaded from: 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/addingpackages.html and http://developer.android.com/tools/extras/support-library.html.

### 3.1.2 USB Drivers (Windows OS Only)

If the host OS is Windows Vista®, 7 or 8, further USB drivers for the device are necessary in order to be able to establish the device connection via Android Debug Bridge (adb). The Android Developer website lists various vendors and links to their support pages where the drivers can be downloaded, see http://developer.android.com/tools/extras/oem-usb.html.

### 3.1.3 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) and navigate to the **System** section, followed by **Developer options**. In this menu, the **USB debugging option** can be enabled.

After the Android SDK and tools together with the 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 OS and the installation path to be **C:\Program Files\Android\Android Studio**. 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:\Program Files\Android\Android Studio> cd platform-tools
C:\Program Files\Android\Android Studio\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:\Program Files\Android\Android Studio\platform-tools> adb.exe install
“C:\Users\<User Name>\My Documents\app.apk”
860 KB/s (220167 bytes in 0.250s)
pkg: /data/local/tmp/app.apk
Success

C:\Program Files\Android\Android Studio\platform-tools> adb.exe uninstall
com.example.app
Success
```

Once installed, the Android device can be unplugged and the App can be used immediately on the device, see *Section 4: Using the Management App*.

## 3.2 Installation via HTTP

An alternative installation option to USB Debugging is to directly download and install the package on the Android device via HTTP access. USB connection is not necessary for this procedure but the following prerequisites must be met:

- The Android device must have access to the HTTP server where the Android package(s) are located.
- The user has to allow the installation of Android Apps from sources other than the Play Store.

### 3.2.1 Allow Installation of Android Apps from Unknown Sources

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

- **Android 3.2:** Navigate to **Settings** and select **Applications**. Enable the **Unknown sources** option.
- **Android 4.x:** Navigate to **Settings** and select **Security** (under the **Personal** heading). Navigate to the **Device administration** section and enable the **Unknown sources** option.

Once the **Unknown sources** option is enabled, the *.apk can be downloaded and installed

- Download the Android package via the device’s web browser
- Navigate to the Download folder and tap on the downloaded package to begin installation. If the package has been previously installed, a message is shown that highlights this fact.
- The privileges required by the App are then displayed. Tap on **Install** to accept the requirements and continue the installation.

### 3.2.2 Uninstallation of Installed Apps on the Device

- **Android 3.2:** Navigate to **Settings** and select **Applications**. In this menu, select **Manage applications** and select the **Downloaded** tab. Tap on the required App in the list to display the **App info** screen. Tap **UNINSTALL** to remove the App.
- **Android 4.x:** Navigate to **Settings** and select **Apps**, followed by the **Downloaded** tab. Tap on the required App in the list to display the **App info** screen. Tap **UNINSTALL** to remove the App.

# 4 Using the Management App

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

## 4.1 USB Readers

When a supported USB reader is plugged into the Android device, a screen displays information about the connected USB device. To prevent this screen from displaying in the future, select the **Default** option. Tap **OK** to dismiss and launch the associated App which shows a list of currently managed devices. If the newly attached device is not listed, tap on **Refresh** to reload the reader list.

Further reader information (name, activity and usage status) is shown by tapping 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, and the user wants to expose specific reader to applications via the JSR268 interface, the readers not needed can be deactivated in the reader information screen.

## 4.2 Bluetooth Readers

Once a Bluetooth reader has been used with the Management App (i.e. successfully discovered and paired) it's MAC address is stored in the App preferences. See *Section 4.2.1: Device Discovery Process*. This allows the Management App to display a list of available, paired readers at launch (provided Bluetooth is enabled when the Management App is launched and at least one applicable Bluetooth device is in close proximity to the Android device).

### 4.2.1 Device Discovery Process

Tap **Bluetooth discovery** to search for nearby available Bluetooth devices. If Bluetooth is currently disabled on the device, the Management App will ask permission to enable it. Once Bluetooth has been enabled the device discovery process will start. This process lasts about 10-15 seconds and will display a list of available peripheral devices. Tap on the required reader to initiate pairing, during which the Android device may ask for the Bluetooth PIN (if this is the first time the Bluetooth device and the Android device are pairing). Follow the on screen instructions to enter the Bluetooth PIN. Once successfully paired, the MAC address of the reader will be stored in the Management App preferences.

**Note:** If no available devices are discovered a message will be displayed.

### 4.2.2 Bluetooth Activation

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

- By starting the Management App and tapping the **Bluetooth discovery** button. As described above, Bluetooth is then enabled on the device.
- Manually:
  - **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 tap on the **Bluetooth (Turn on Bluetooth)** button.
  - **Android 4.x:** Navigate to **Settings** (e.g. via the **Applications** menu) and switch the button next to **Bluetooth** from **Off** to **On**.

### 4.2.3 Unpair Paired Bluetooth Devices

Android stores the pairing information after initial pairing with a Bluetooth device. If this pairing needs be to deleted:

- **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 is 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**. The list of paired devices is shown and from this list the pairing for each device can be detached.

# 5 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. For further information, refer to: http://download.oracle.com/otndocs/jcp/jscio-4.0-fr-eval-oth-JSpec/.

## 5.1 Usage of the JSR268 API in Third-Party Apps

A library is provided (Jar1 file) which encapsulates the JSR268 interface for use in third-party Apps, allowing access to readers and smart cards.

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** and select 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;
```

4. Add the following variables to your Activity:

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

5. 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);
```

6. 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:

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

7. 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();
```

8. 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 JSR 268 documentation [JSR 268 Doc], 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:** 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.

## 5.2 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;
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) {
          // ...
        }
    }
}
```

# 6 Installation of Management App from within a Third-Party App

It is possible to allow a third-party App to automatically install the Management App when that third-party App first attempts to interact with card readers and smart cards (assuming that the Management App is not currently installed).

Before proceeding, consider the following:

1. Check whether the Management App is already installed.
2. If not, copy the Management App's *.apk file (e.g. CardReaderManager.apk) to the assets folder of the Android App project (via Windows Explorer).
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) has either to be initialized at App startup or after the Management App has been installed. See step 4 of *Section 5.1: Usage of the JSR268 API in Third-Party Apps*. Otherwise no connection is established and the JSR268 API calls will fail, as the library would guess that there is no backend available.

## 6.1 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.
}
```

## 6.2 Prepare Android

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, the 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();
```

## 6.3 Package installation

Once prepared, the package installation is an easy 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, and 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 screen that asks whether the App shall be installed and also lists the permissions needed by this App.

# Appendix: A Definitions, Abbreviations and Symbols

| | |
|---|---|
| **APDU** | Application Protocol Data Unit |
| **API** | Application Programming Interface |
| **OID** | Object Identifier |
| **PC/SC** | Personal Computer/Smart Card |
| **IFD** | Interface Device (for accessing ICC card) |
| **OTG** | On The Go, USB interface feature |
| **OS** | Operating System |
| **SDK** | Software Development Kit |

# Appendix: B References

| | |
|---|---|
| **ISO 7816-3** | ISO 7816-3<br>Identification cards — Integrated circuit cards - Part 3: Cards with contacts — Electrical interface and transmission protocols<br>Third edition - 2006-11-01 |
| **ISO 7816-4** | ISO 7816-4<br>Identification cards — Integrated circuit cards - Part 4: Organization, security and commands for Interchange<br>Second edition - 2005-01-15 |
| **PCSC-3** | Interoperability Specification for ICCs and Personal Computer Systems<br>Part 3. Requirements for PC-Connected Interface Devices<br>Revision 2.01.09 |
| **CCID** | Specification for Integrated Circuit(s) Cards Interface Devices<br>Revision 1.1<br>April 22rd, 2005 |
| **USB Dev Enum** | https://play.google.com/store/apps/details?id=aws.apps.usbDeviceEnumerator |
| **JSR 268 Doc** | http://docs.oracle.com/javase/6/docs/jre/api/security/smartcardio/spec/javax/smartcardio/package-summary.html |