UM10809

Step by Step guide for using MIFARE SDK to develop Android NFC Applications

Rev. 02.02 — 24 Feb 2015 **User Manual**

Document information

Info	Content
Keywords	MIFARE, Android, Eclipse, Sample Application, FAQ
Abstract	This document describes the MIFARE SDK and steps to setup the development environment to develop applications using SDK

Revision history

Rev	Date	Description
00.01	20130131	Initial version
00.02	20140214	Updated with SAM and other important content
00.03	20140221	Modified the template the standard NXP template
00.04	20140415	Update and sent for review
00.05	20140416	Review comments updated
00.06	20140506	Document Number updated to UM10809
00.07	20140520	Minor review comments closed
01.00	20140523	Updated the code structure. Baselined.
01.01	20140627	Added new sub-section for Lite library. Baselined for release with lib Ver 01.01.01.1427
01.02	20140930	Added new FAQ question, updated description for NTAG I2C. Baselined for release with lib Ver 01.02.00.1440
01.03	20141104	Added section for registering at mifare.net to get application key
01.04	20141105	Added comparison table, updated architecture diagram, closed review comments regarding branding, order of MIFARE products across various chapters in the document including appendix
02.00	20141114	Baselined for MIFARE SDK Release Ver 02.00.00.1446
02.01	20141215	Added filterIntent in abstract method in code snippet. Baselined for Ver 02.01.00.1451
02.02	20150224	Minor updates and baselined for Ver 02.02.00.1509

Contact information

For more information, please visit: http://www.mifare.net

For sales office addresses, please send an email to: salesaddresses@nxp.com

1. Introduction

1.1 Introduction to MIFARE SDK

This SDK consists of Libraries that can be used to develop applications which can run on devices with NFC hardware to communicate with MIFARE Cards and MIFARE SAM.

Fig 1. MIFARE cards, NTAG, ICODE and SAM

Please refer to Appendix-4 onwards at the end of the document for more details on the MIFARE products.

1.2 Lite & Advanced Versions

The SDK is available in two different versions – *Lite & Advanced*.

'Lite' version of SDK consists of APIs for simple use cases such as personalization of the cards and quick read/write operation and single NDEF operations on supported cards etc. This version is meant for light weight users to quickly create and deploy applications. To make the usage simple certain assumptions are made. Please refer to Appendix-2 for complete list of assumptions and limited features supported in MIFARE SDK *Lite*.

'Advanced' version APIs expose the complete features of any MIFARE card and support all operations. Support for SAM & KeyStore are provided in MIFARE SDK *Advanced* version only.

The following table summarizes the differences between *'Lite' & 'Advanced'* versions of MIFARE SDK. Partial means not all commands of the card are supported. Refer to Javadoc for list of supported APIs.

Component	Lite	Advanced
MIFARE Classic	✔	✔
MIFARE Plus	✔ Limited	✔
MIFARE DESFire	✔ Limited	✔
MIFARE Ultralight	✔	✔
NTAG	✔	✔
ICODE	✔ Partial	✔
MIFARE SAM	✕	✔
KeyStore	✕	✔
Crypto	✔	✔
Utilities	✔	✔

Fig 2. Comparison – Lite vs Advanced

1.3 Contents of MIFARE SDK - Advanced

The SDK contains the following items:

Release Note	-	ReleaseNotes.txt
Java library files	-	NxpNfcLib.jar & jsr268library.jar
Javadoc documentation	-	JavaDocNxpNfcLib.zip
User Manual	-	This document
Sample application (Sources + APK)	-	SampleNxpNfcLib.zip&SampleNxpNfcLibLite.zip
HID Card Reader Manager	-	CardReaderManager.apk

1.4 Architecture Diagram of Library

The library is designed for ease of use and quick development of applications. It consists of components for each type of card and SAM. As of now, SAM is supported using HID SAM reader and hence is interfaced via HID card reader manager library.

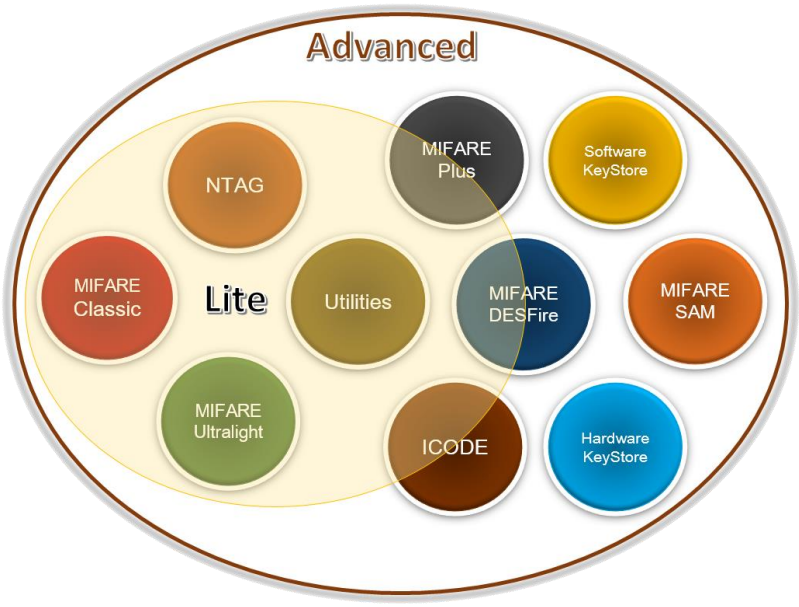


Fig 3. MIFARE SDK Architecture

*Lite – Limited support available for Plus, DESFire & ICODE

1.5 Required Software & Hardware for developing applications

For both 'Lite' & 'Advanced' versions, you need to register at www.mifare.net first.

1.5.1 Lite

1.5.1.1 Software

1. Download MIFARE SDK from http://www.mifare.net/en/products/mifare-sdk/ after registration.

Fig 4. MIFARE SDK Download Page

2. Android Development Tool environment from Google

   http://developer.android.com/sdk/index.html

1.5.1.2 Hardware

Any Android Mobile device with NFC Hardware with OS version 4.x (Ice Cream Sandwich and above) & MIFARE Cards.

1.5.2 Advanced

1.5.2.1 Software

- Android Development Tool environment from Google
  http://developer.android.com/sdk/index.html

- MIFARE SDK from Mifare website
  http://www.mifare.net/en/products/mifare-sdk/

- HID Omnikey Drivers for Android
  (omnikey_android_20130506_r1.0.0.0.zip)
  http://www.hidglobal.com/drivers/16553

1.5.2.2 Hardware

- Any Android Mobile device with NFC Hardware with OS version 4.x (Ice Cream Sandwich and above)

- Any one of the following HID SAM readers – 6121 OR 2061

  http://www.hidglobal.com/products/readers/omnikey/2061

  http://www.hidglobal.com/products/readers/omnikey/6121

USB OTG Cable to connect HID Dongle to mobile

Fig 5. Hardware needed for developing applications for SAM

1.6 Registering application to get App key

Every application that shall be developed and deployed using the MIFARE SDK 'Advanced' version library should be registered at https://inspire.nxp.com

2. Installing Eclipse and setting up an Android Project to use NxpNfcLib

2.1 ADT download page

2.1.1 Download Android Development Tools

The Android Development Tools (ADT) is a plugin for Eclipse including necessary library and debugging functionalities to develop Android applications. This plugin will be provided by Google including Eclipse and can be downloaded from the Android SDK homepage.

URL - http://developer.android.com/sdk/index.html

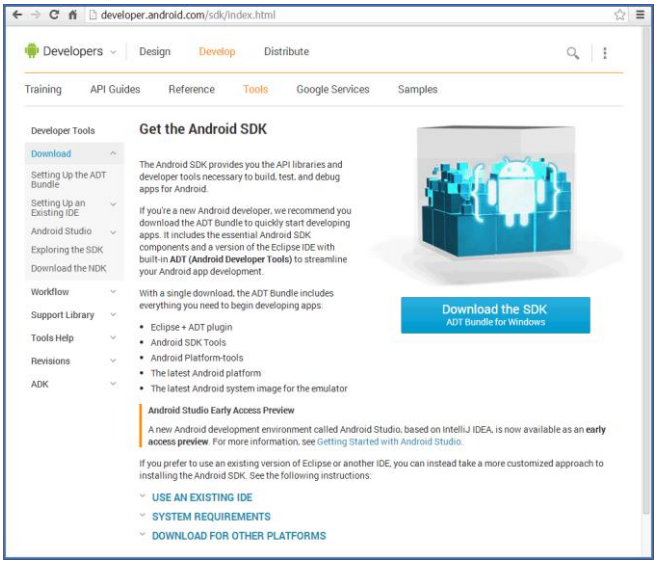


Fig 6. Android SDK Download Page

Select your desired processor architecture and accept the "Terms and Conditions".

Note: There seems to be an issue on the site – FIRST you have to select your windows architecture version and SECOND you have to accept the "Terms and Conditions". Press the "Download the SDK ADT Bundle for Windows" button to start download.

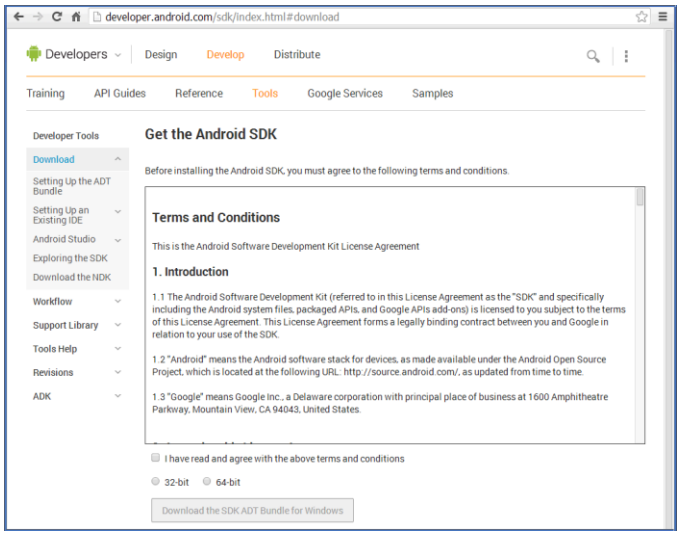


Fig 7. Get the android SDK

When the download dialog opens, select the directory where you want to store the archive. ***Note:*** The downloaded file contains a preconfigured standalone installation.

2.1.2 Extracting the files

Open the downloaded archive and save it to a “safe” place on your file system (e.g. C:\Program Files\ADT).

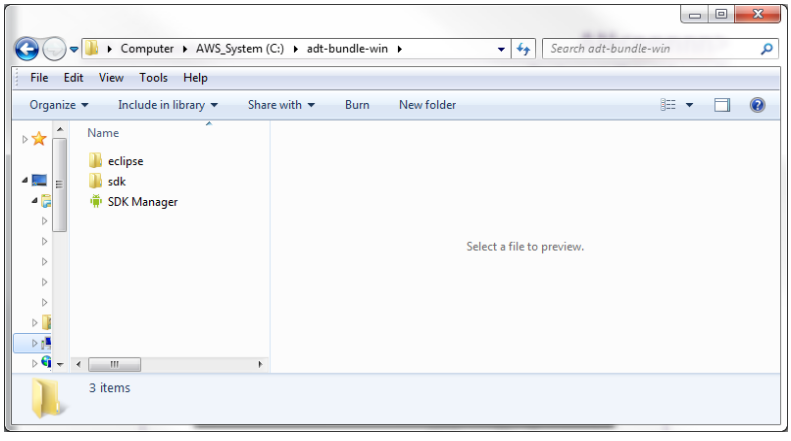


Fig 8. Contents of the archive

2.1.3 File structure after copying the content of the archive to the file system

The targeted folder should look like this:

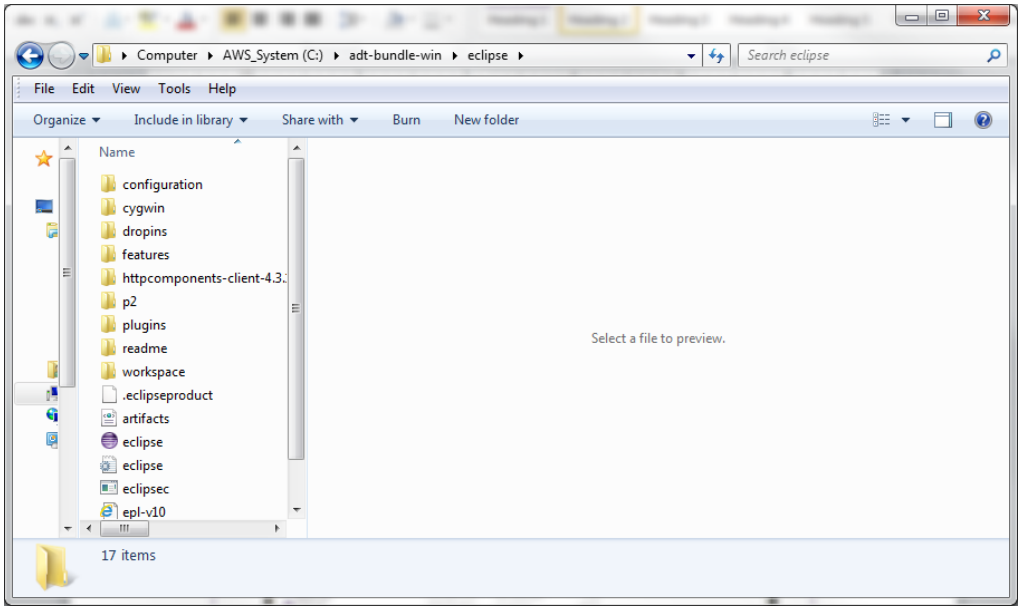


Fig 9. Eclipse

2.1.4 Launch Eclipse

Open the eclipse folder and start Eclipse by double-clicking on eclipse.exe

2.2 Select your workspace & Developing view

2.2.1 Select your workspace location

After starting eclipse you will be asked where you want to create workspace. The selected workspace will contain all projects you create. Press OK.

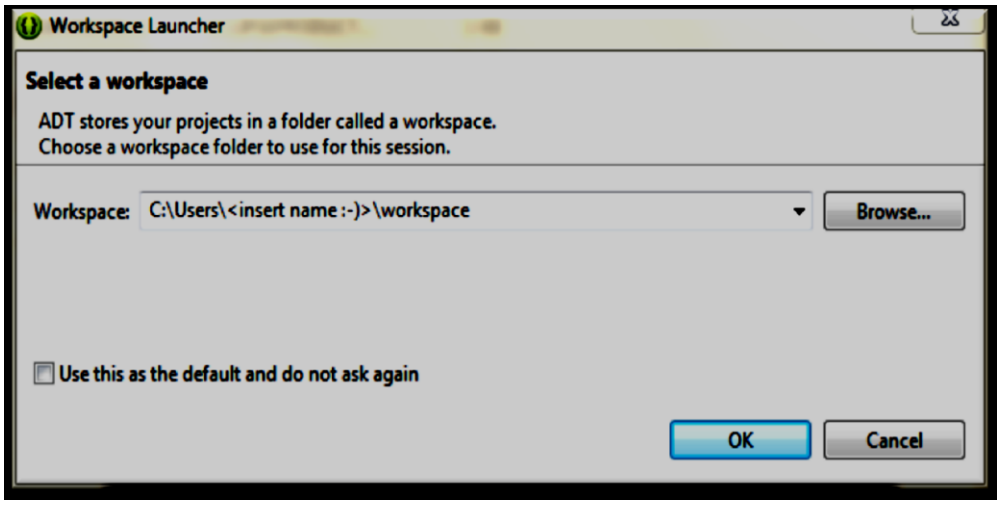


Fig 10. Select a workspace

2.2.2 Welcome dialog

Close the welcome dialog to get to the developing view. This manual is not intended to describe further Eclipse instructions.

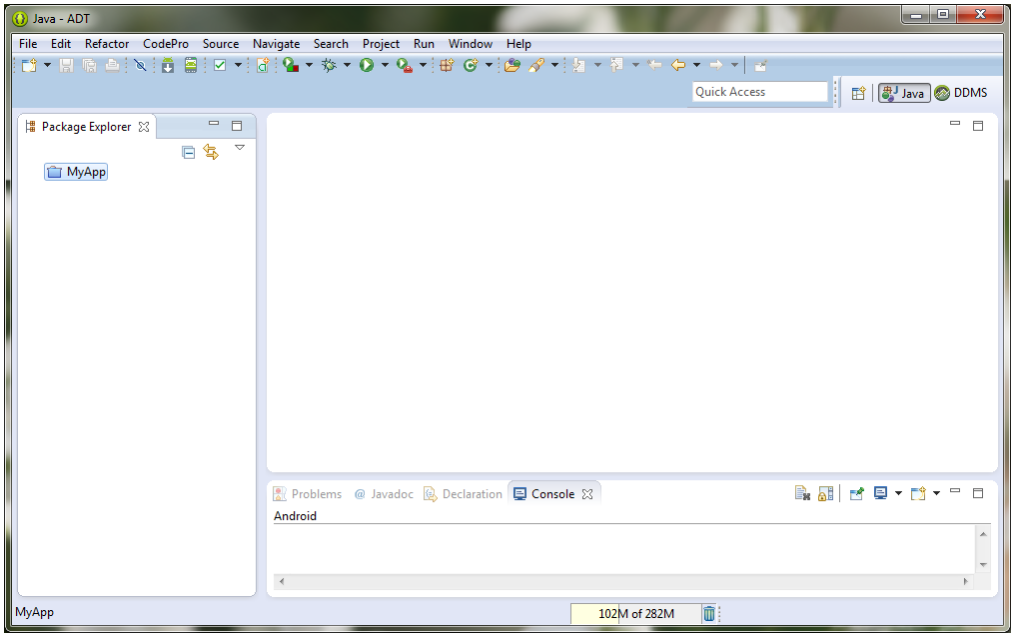


Fig 11. Eclipse developing view

2.3 Create an Android Application Project

To create an Android Application Project click File > New > Android Application Project.

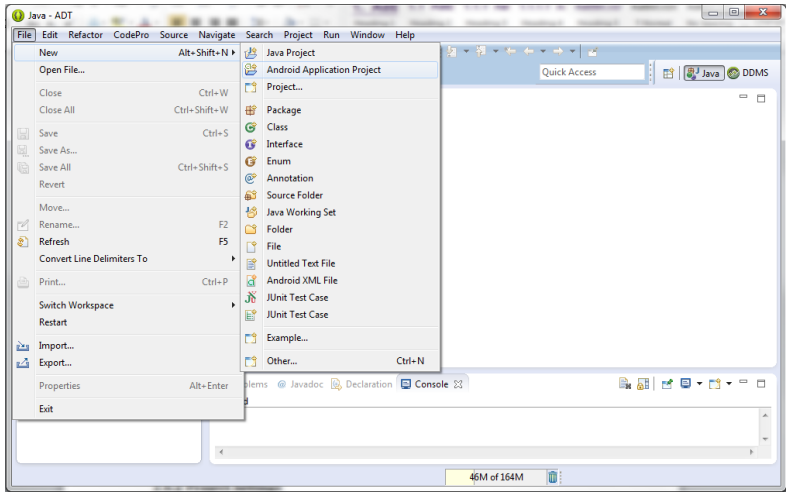


Fig 12. Eclipse menu view for creating new project

2.3.1 Project Settings

Enter an Application name, Project name, Package name and the SDK requirements. The minimum required SDK for developing projects using NFC is level 10, but NDEF message support has been added in SDK version 14. But it is recommended to use 16 for minimum SDK and for 'Target SDK' & 'Compile with' select the latest available.

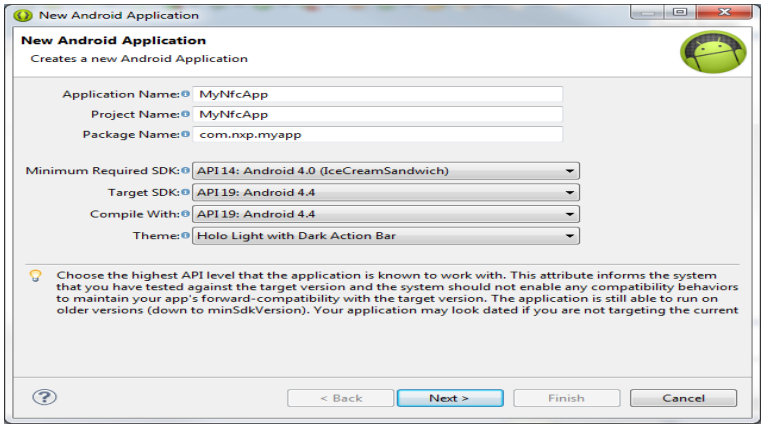


Fig 13. Project setting

2.3.2 Configure Project

Make sure that “Mark this Project as a library” is not set. You can leave other settings as they are.

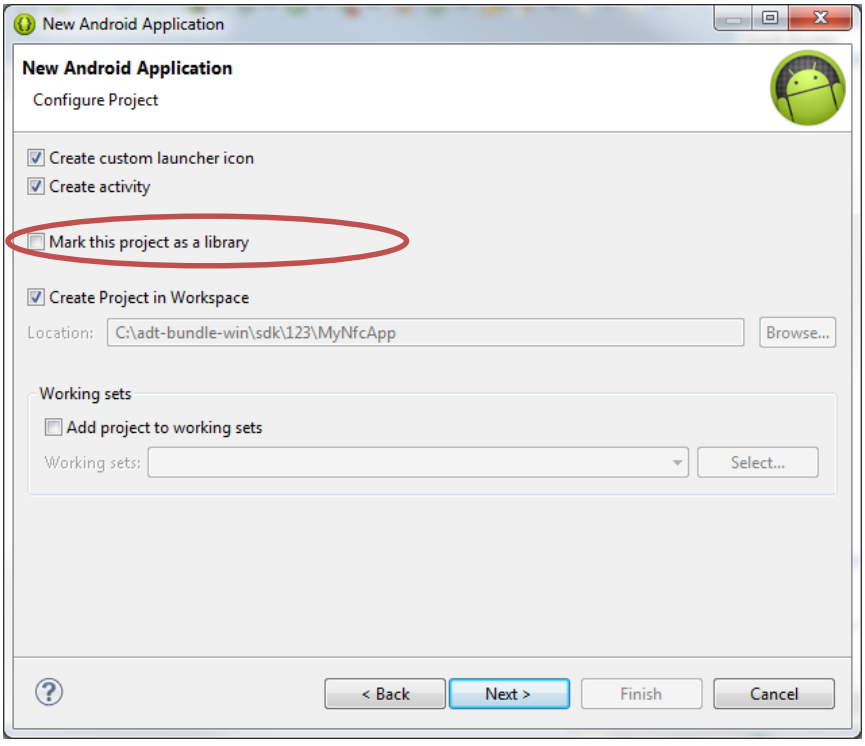


Fig 14. Configure project

2.3.3 Configure Launcher Icon

If you have a custom icon you want to use, you can set it here. Click Next.

2.3.4 Create Activity

You can leave these settings as they are. Click Next.

2.3.5 Blank Activity

If you want to change the name of the main activity you can do this here. Click Finish. Now the project has been created. You can see your project files on the left-hand side. The main activity has already been opened in the editor.

2.3.6 Extracting the library

Open the MIFARE SDK archive and extract the content to “safe” place on your file system.

2.4 Adding the library

2.4.1 Adding the library to your workspace

To add the library to your project right click onto your project and click “Properties”. Select “Java Build Path” on the left side. Select the tab “Libraries”. Click “Add External JARs…”. Locate the extracted bundle on your file system. Select the file “nxpnfclib.jar” and click open.

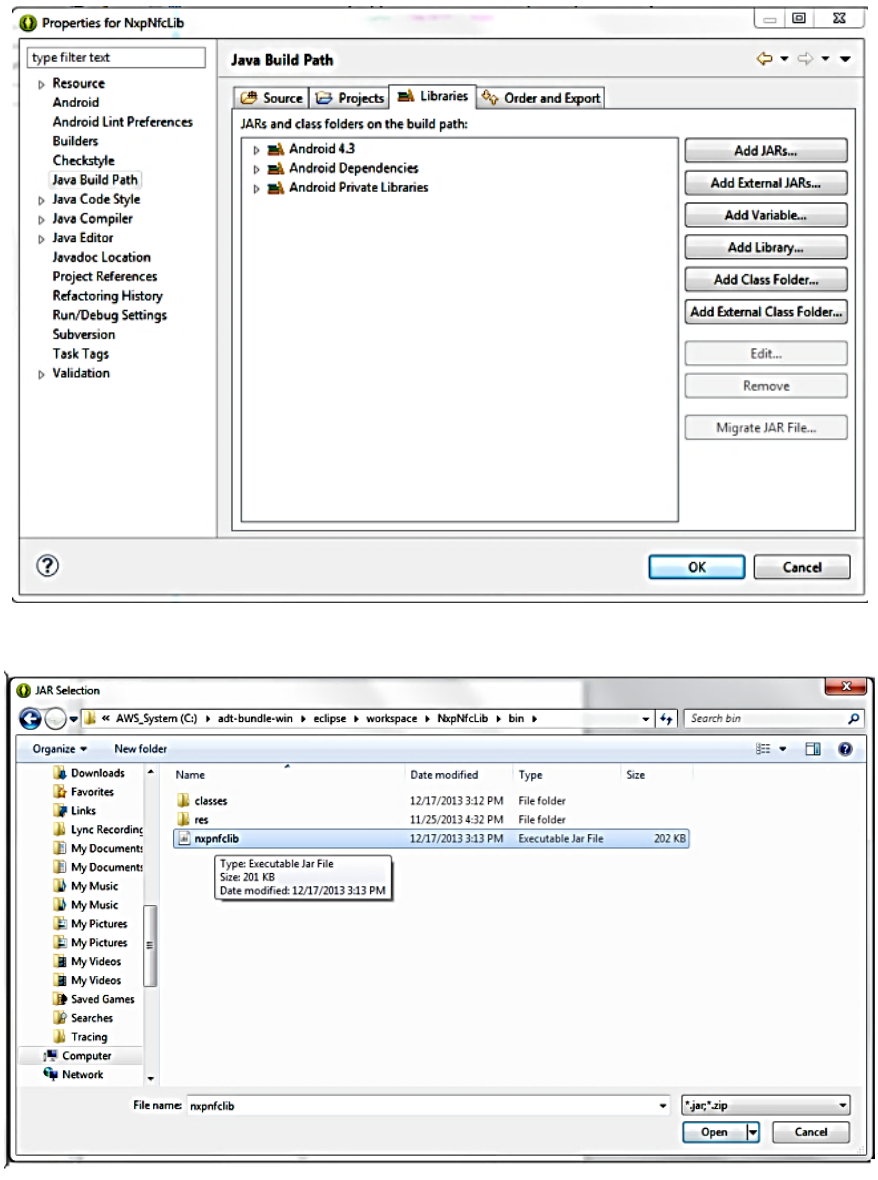


Fig 15. To add library

Select the TAB “Order and Export”; 1. Select checkbox “nxpnfclib.jar” 2. Click on Top button 3. Click OK

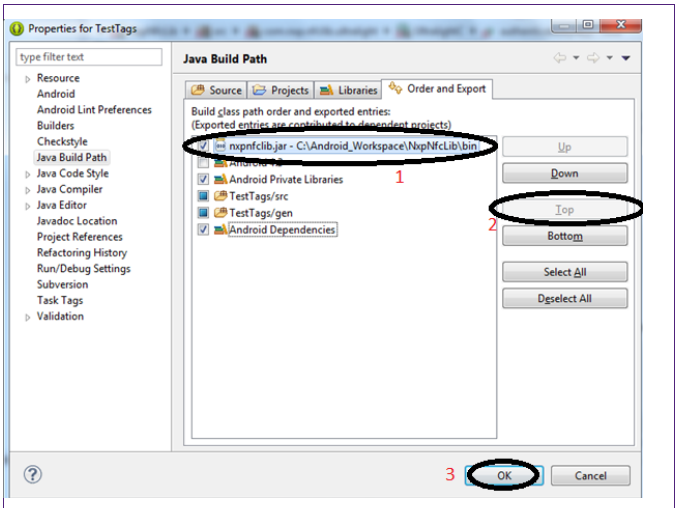


Fig 16. Set library into top position

Unzip the zip to a lib folder. Switch to "Libraries Tab",

1. Click the Tree view of nxpnfclib.jar
2. Click the Javadoc location
3. Click edit
4. Browse and select the folder where NxpNfcLibJavaDoc.zip is unzipped

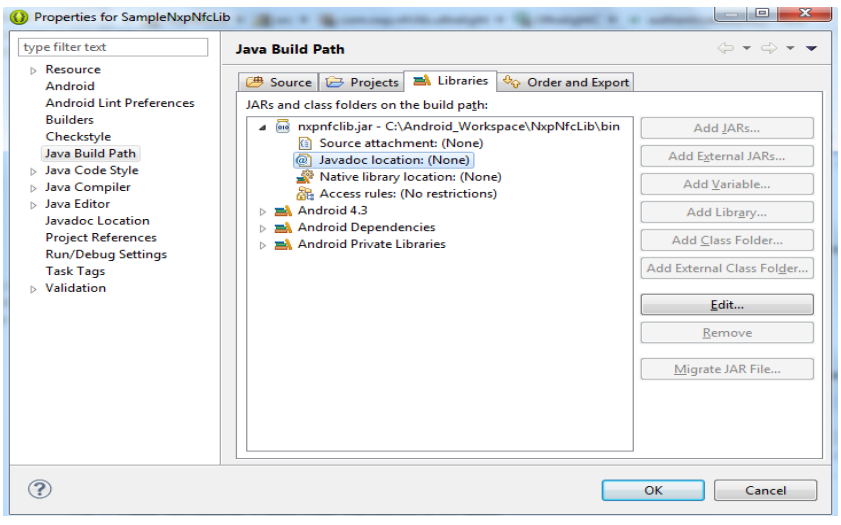


Fig 17. Setting Javadoc path

❖ ***NOTE** – If your application is required to communicate with SAM; then jsr268library.jar also need to be added as above.*

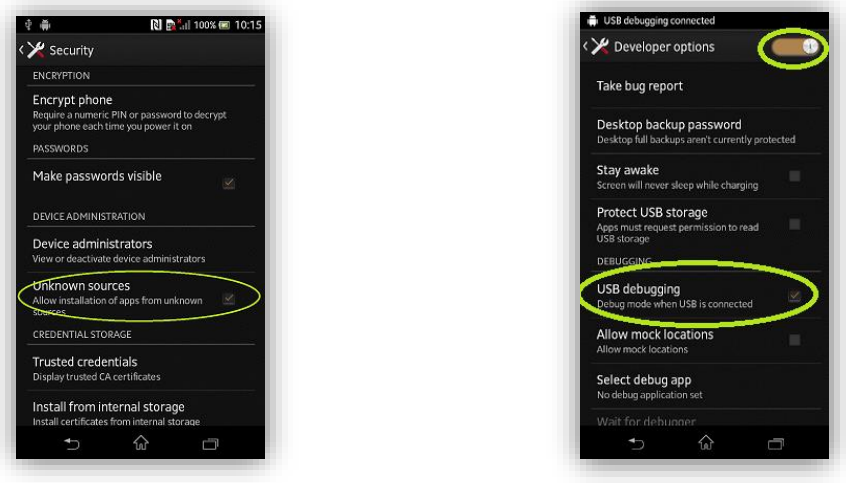
3. Starting the development

3.1 Pre-requisites for pushing/installing your application onto the device

Normally, Android Apps are installed via the Android Play Store, where the packages that are provided passed certain security criteria. To install from your desktop; you need to do the following two steps:

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. (Settings -> Security -> Device Administration).

Navigate to Settings, touch **{} Developer Options**. Enable by switching the slider to ON top-right corner of the screen and check/select **USB Debugging**.



a. Enable unknown sources

b. Enable debug mode

Fig 18. Enabling the developer options in device

3.2 Request permissions

Open the AndroidManifest.xml of your application and click on Permissions tab and Add permission for NFC and WRITE_EXTERNAL_STORAGE (as logs get stored in external storage location by default).

NOTE - If HID library is being used, please add the following permission

```
<uses-permission
android:name="com.hidglobal.ia.omnikey.service.premission.SMARTCARDIO"/
>
```

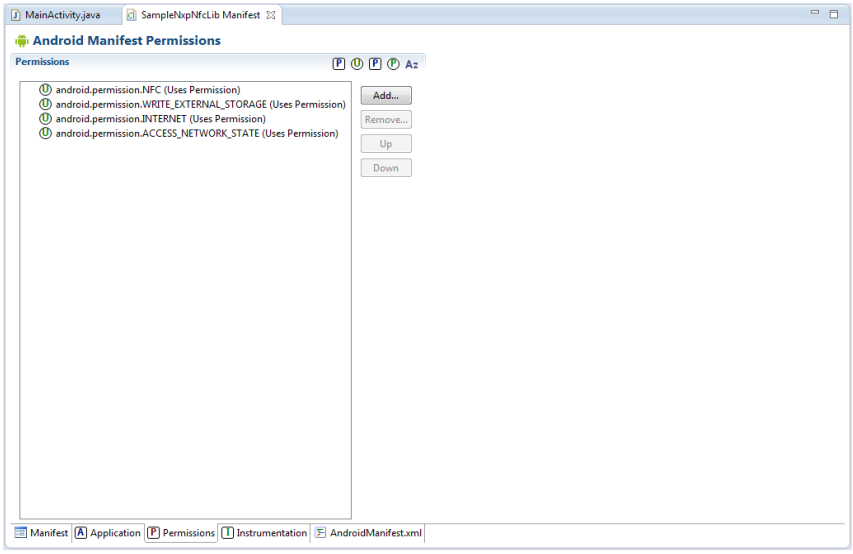


Fig 19. Add request permission

Or add the following lines manually in AndroidManifest.xml file.

```
<uses-permission android:name="android.permission.NFC"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

3.3 Restrict app to be used on devices with NFC hardware only

Open the AndroidManifest.xml of your application and click on Manifest tab and Click on Add.. Enter android.hardware.nfc in Name and 'true' in Required fields as shown below:

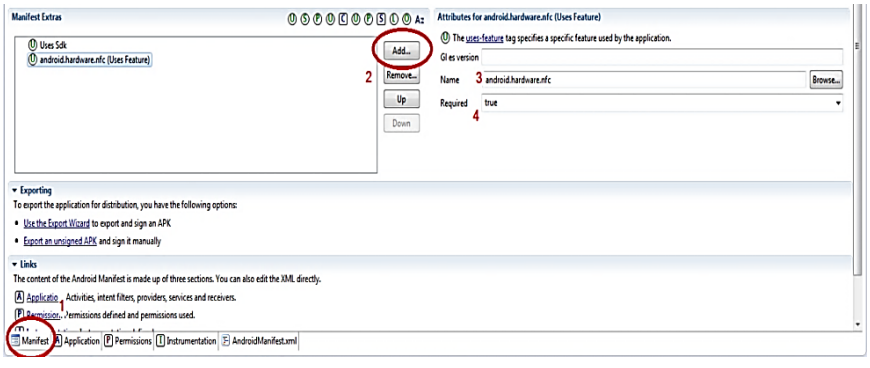


Fig 20. Add NFC feature in AndroidManifest.xml

Or add the following lines manually in AndroidManifest.xml file.

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

3.4 Add intent filters to get notification whenever a tag is discovered

Open the AndroidManifest.xml of your application and click on Application tab and Click on Add.. to add NDEF_DISCOVERED, TAG_DISCOVERED & TECH_DISCOVERED as shown below:

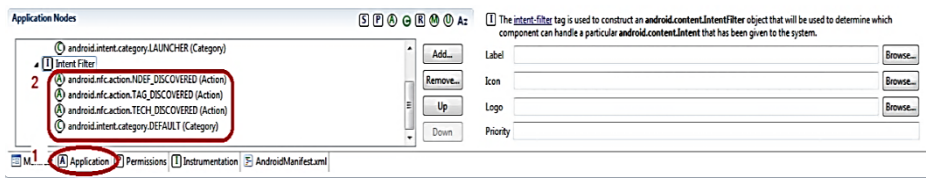


Fig 21. Add intent filters

Or add the following lines manually in AndroidManifest.xml file.

```
<intent-filter>
  <action    android:name="android.nfc.action.NDEF_DISCOVERED"/>
  <action    android:name="android.nfc.action.TAG_DISCOVERED" />
  <action    android:name="android.nfc.action.TECH_DISCOVERED"/>
  <category android:name="android.intent.category.DEFAULT"   />
</intent-filter>
```

3.5 Coding your application

There is a slight change in the way the library and card/tag handle is used in Lite and Advanced version libraries. Ensure to import correct classes when using Lite and Advanced version libraries. All the card objects of Lite version are to imported from *com.nxp.nfcliblite.cards.x* and in case of Advanced version, the card objects are to be imported from *com.nxp.nfclib.x*

3.5.1 First steps in using the Lite Library - NxpNfcLibLite

1. In the MainActivity class; initialize the library object as null. See below:

```
private NxpNfcLibLite libInstance = null;
```

2. In OnCreate method; get instance of NxpNfcLib and register this activity in library instance.

```
libInstance = NxpNfcLibLite.getInstance();

// Call registerActivity function before using other functions of the library.
libInstance.registerActivity(this);
```

3.5.2 First steps in using the Advanced library – NxpNfcLib

1. In the MainActivity class; initialize two objects as null. See below:

```
private NxpNfcLib libInstance = null;
private IKeyStore ks = null;
```

Instantiate the Software Keystore initialize the keys as shown below:

```
ks = KeyStoreFactory.getInstance().getSoftwareKeyStore();
ks.formatKeyEntry(0, IKeyConstants.KeyType.KEYSTORE_KEY_TYPE_MIFARE);
ks.setKey(0,  (byte)0, IKeyConstants.KeyType.KEYSTORE_KEY_TYPE_MIFARE,
     KEY_DEFAULT_1, (byte)0);
libInstance.loadKeyStore(ks);
```

2. In OnCreate method; get instance of NxpNfcLib & Software KeyStore. Initialize the Software KeyStore and load. Register this activity in library instance.

```
libInstance = NxpNfcLib.getInstance();
// Call registerActivity function before using other functions of the library.
```

Note that the `registerActivity()` API signature is different in the Lite & Advanced versions of the library.

Call this API along with key obtained from NXP license server(Refer to Section 1.6 for procedure to get key) as below:

```
libInstance.registerActivity(this, < key obtained from NXP >);
```

3.5.3 Common for both Lite and Advanced Libraries

Start and Stop foregroundDispatch in onResume & onPause respectively as shown below:

```
@Override
     protected void onPause() {
   libInstance.stopForeGroundDispatch();
   super.onPause();
}

@Override
protected void onResume() {
   libInstance.startForeGroundDispatch();
   super.onResume();
}
```

3.5.4 Adding your application logic

Your application logic can be added in *on<your card>Detected (card object)*{} in *onNewIntent(Intent intent)* function. This can be done as

```
protected void onNewIntent(Intent intent) {
<callback or filter intent>
<your app logic>
super.onNewIntent(intent);
}
```

Library provides two methods to add application logic i.e., either by using abstract class callback or filterintent. Advantage of first method is that you would not have auto generated code for unused cards in your application.

a. Method 1 – Use NxpNfcLib abstract class callback where you need to add manually the methods for required cards detected as shown below:

```
protected void onNewIntent(Intent intent) {
Nxpnfcliblitecallback callback = new Nxpnfcliblitecallback()
{
                    OR
Nxpnfclibcallback callback = new Nxpnfclibcallback() {
public void onUltraLigthCardDetected(MifareUL objUlCard) {
// TODO Auto-generated method stub
Log.i(TAG,"UltraLight Card Detected" );
objUlCard.connect();
<your app logic>
}
libInstance.filterIntent(intent, callback);
}
```

Please refer to the Javadoc documentation for the callbacks provided for each card/tag.

a. Method 2 – Auto handlers using the *libInstance.filterIntent (..) in onNewIntent* () as shown below:

```
protected void onNewIntent(Intent intent) {

libInstance.filterIntent(intent, new Inxpnfclibcallback() {

@Override
public void onUltraLigthCardDetected(MifareUL objUlCard) {
   Log.i(TAG,"UltraLight Card Detected" );
   objUlCard.connect();
}

@Override
```

```
   public void onUltraLigthEV1CardDetected(MifareUL objUlCardEV1) {
      Log.i(TAG,"UltraLight Card EV1 Detected" );
      objUlCardEV1.connect();
 }

 @Override
 public void onUltraLigthCCardDetected(MifareUL objULCardC) {
      Log.i(TAG,"UltraLight C Card Detected" );
      objULCardC.connect();
 }

 @Override
 public void onMifareClassicCardDetected(MFClassic objMFCCard) {
      Log.i(TAG,"Classic Card Detected" );
      objMFCCard.connect();
 }

 @Override
 public void onDESFireCardDetected(DESFire objDESFire) {
      Log.i(TAG,"DESFire Card Detected" );
      objDesfire.connect();
 }

 @Override
 public void onMifarePlusCardDetected(Plus objPlus) {
      Log.i(TAG,"Plus Card Detected" );
      objPlus.connect();

 }

 @Override
 public void onICodeSLIDetected(ICodeSLI arg0) {
      Log.i(TAG,"Icode SLI Detected" );
      arg0.connect();
 }

 @Override
 public void onICodeSLILDetected(ICodeSLIL arg0) {
      Log.i(TAG,"Icode SLIL Detected" );
      arg0.connect();
 }

 @Override
 public void onICodeSLISDetected(ICodeSLIS arg0) {
      Log.i(TAG,"Icode SLIS Detected" );
      arg0.connect();
 }


 @Override
 public void onICodeSLIXDetected(ICodeSLIX arg0) {
      Log.i(TAG,"Icode SLIX Detected" );
      arg0.connect();
 }

 @Override
 public void onICodeSLIXLDetected(ICodeSLIXL arg0) {
      Log.i(TAG,"Icode SLIXL Detected" );
      arg0.connect();
```

```
        }

        @Override
        public void onICodeSLIXSDetected(ICodeSLIXS arg0) {
              Log.i(TAG,"Icode SLIXS Detected" );
              arg0.connect();
        }

        @Override
        public void onNTag203xCardDetected(NTag203x arg0) {
              Log.i(TAG,"NTAG 203x card Detected" );
              arg0.connect();
        }

        @Override
        public void onNTag210CardDetected(NTag210 arg0) {
              Log.i(TAG,"NTAG 210 card Detected" );
              arg0.connect();
        }

        @Override
        public void onNTag212CardDetected(NTag212 arg0) {
              Log.i(TAG,"NTAG 212 card Detected" );
              arg0.connect();
        }

        @Override
        public void onNTag213215216CardDetected(NTag213215216 arg0) {
              Log.i(TAG,"NTAG 213215216 card Detected" );
              arg0.connect();
        }

        @Override
        public void onNTag213F216FCardDetected(NTag213F216F arg0) {
              Log.i(TAG,"NTAG 213F216F card Detected" );
              arg0.connect();
        }

        @Override
        public void onNTagI2CCardDetected(NTagI2C arg0) {
              Log.i(TAG,"NTAG I2C card Detected" );
              arg0.connect();
        }
        });
        super.onNewIntent(intent);
    }
    });
```

4. Starting the development using SAM

4.1 Use Cases

Applications using MIFARE SDK can be used to program Secure Application Module (SAM) and also in Hardware key store mode. SDK supports SAM AV2 only but for legacy purposes; an API(*switchFrom AV1To AV2Mode()*) is provided to change SAM AV1 to AV2.

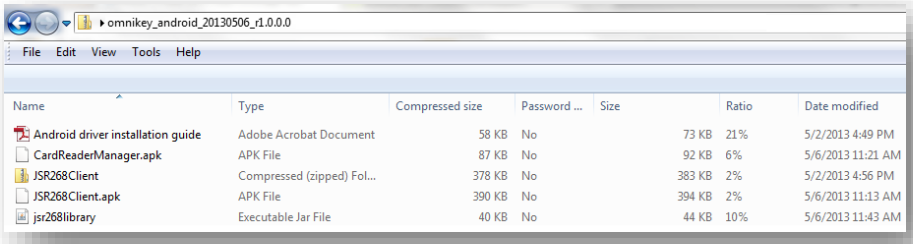
4.2 Prerequisites for developing Android Application with SAM

The NXP's SAM AV2 is to be connected to the Android device using the HID Omnikey reader. Please refer to Section 1.5.2 for required hardware. Insert the SAM module in to HID 6121 and connect to the device using the USB OTG cable. For bluetooth mode; insert the SAM into HID 2061 and start power ON.

Note - Refer to the HID reader product pages for more details:
http://www.hidglobal.com/products/readers/omnikey/2061
http://www.hidglobal.com/products/readers/omnikey/6121

Following are the contents of the zip downloaded from omnikey website - http://www.hidglobal.com/drivers/16553

- `CardReaderManager.apk` – Card reader manager. Mandatory to be installed on android device
- `jsr268library.jar` – Java library. Do not install on the phone but include in Project as shown in Section 4.10.
- `JSR268Client.apk – simple demo application. Do not install.`
- `JSR268Client.zip – source code for JSR268Client. Do not install.`



4.3 Coding your SAM based application

1. In the Activity class; initialize following objects as below:

```
private NxpNfcLib libInstance = null;
private SamAV1 mSamAV1 = null;
private SamAV1 mSamAV2 = null;
```

2. In OnCreate method; get instance of NxpNfcLib. Register this activity in library instance.

```
        // Call registerActivity function before using other functions of the library.
            libInstance.registerActivity(this, < key obtained from NXP >);
    try {
                SAMFactory.getInstance().getSAM(this, new ISamConnectedCallback() {

    @Override
    public void onSAMAv2Connected(SamAV2 arg0) {
        mSamAv2 = arg0;
        // TODO Auto-generated method stub
        NxpLogUtils.i(TAG, "Av2 Connected...");
        Toast.makeText(SamActivity.this, "Av2 Connected...",Toast.LENGTH_LONG).show();
                                    mSamAv2.connect();
    }

    @Override
    public void onSAMAv1Connected(SamAV1 arg0) {
        mSamAv1 = arg0;
        NxpLogUtils.i(TAG, "Av1 Connected...");
        Toast.makeText(SamActivity.this, "Av1 Connected...",Toast.LENGTH_LONG).show();
                                    mSamAv1.connect();
    }
  });
} catch (SAMException e1) {
  e1.printStackTrace();
  Toast.makeText(getApplicationContext(), "Connect Sam! ",
        Toast.LENGTH_SHORT).show();
    }
```

4.4 Coding using HardwareKeyStore

Coding in hardware key store is similar to software key store, apart from the fact that the hardware key store will keep the keys in SAM. Its usage is similar to that of software key store as follows:

```
   KeyStoreFactory.getInstance().getHardwareKeyStore(MainActivity.context, new
IHardwareKeystoreConnectedCallback() {
   @Override
   public void onHardwareKeyStoreConnected(HardwareKeyStore objHardware) {
   try {
         objHardware.authenticateHost(KEY_AES128_DEFAULT,
         IKeyConstants.KeyType.KEYSTORE_KEY_TYPE_AES128, CommunicationType.Plain);
         objHardware.formatKeyEntry(2, IKeyConstants.KeyType.KEYSTORE_KEY_TYPE_AES128);
         objHardware.setKey(6, (byte) 0x00, IKeyConstants.KeyType.KEYSTORE_KEY_TYPE_2K3DES,
      KEY_2KTDES);
         libInstance.loadKeyStore(objHardware);
   } catch (SmartCardException e) {
         e.printStackTrace();
   } catch (GeneralSecurityException e) {
         e.printStackTrace();
   } catch (IOException e) {
         e.printStackTrace();
   }
   }
```

5. Appendix 1 – SampleNxpNfcLib Application

NOTE – To edit the sample application sources, the first step is adding the nxpnfclib.jar as shown in Section 2.4

The Sample application that is supplied with SDK demonstrates basic capabilities of the library. It demonstrates use of

- Software & Hardware Keystore
- Utilities Logger
- Utilities Crypto/Java Cipher
- DESFire
  - Authenticate with default key
  - Create an application
  - Create standard data file
  - Write
  - Read back
- Plus / NTAG / ICODE
  - Read data
- Ultralight & Ultralight C
  - Read and display all pages
- Classic & Ultralight EV1
  - Authenticate
  - Write encrypted data to default sector / page
  - Read and decrypt
- SAM
  - Preconditions
    - Install CardManagerReader.apk
    - Connect the HID Dongle with micro USB cable
    - In case of Bluetooth dongle
      - Power on the device
      - Launch CardReaderManager application, press 'BLUETOOTH DISCOVERY' and choose this device if BT dongle or USB device and click OK.
  - Authentication
  - GetVersion
  - GetKey Entry
  - ChengeKey Entry
  - Activate Offline key
  - Enciphered & Deciphered mode of operations

6. Appendix 2 – NxpNfcLibLite API assumptions

6.1 Classic

- Factory Default classic card should be used at First while personalizing.
- Read and Write is limited to 45 bytes per application where First 3 bytes of every sector are reserved for application id.
- 0th sector is not recommended to be used.
- Total number application supported is equal to the Total number of Sectors minus one i.e. (Total Number of APP = (No of Sectors - 1))
- For First time use personalization is required, before reading and writing and authentication is required for subsequent use.
- Formatting the complete card is not possible but formatting particular application is possible where Sector key is reset to factory default and all the blocks are filled with 0xFF.

6.2 Plus

- Only Plus SL3 Card is supported
- Read and Write is limited to 45 bytes per application where First 3 bytes of every sector are reserved for application id.
- 0th sector is not recommended to be used.

6.3 DESFire

- Factory Default DESFire card should be used at First while personalizing.
- Length of the key must be 16 bytes, for all application keys and PICC Master key.
- Read and Write is limited to 256 bytes per application.
- Up to 28 applications are supported per card (this number depends on type of the card ex: for 512 bytes card only one applications supported.)
- For First time use personalization is required, before reading and writing and authentication is required for subsequent use.
- Formatting the card requires the PICC Master key which is used during personalization. (Format Resets the card to Factory Default Settings.)

6.4 Ultralight / Ultralight C / Ultralight EV1

- Only Reading and Writing of NDEF Message is Supported
- Size of the NDEF Message is card dependent.

6.5 ICODE

Some commands are only supported. Please refer to Lite Javadoc for supported APIs.

7. Appendix 3 – Frequently Asked Questions

1. Do we need anything else from Google ADT for developing NFC applications?

   Answer: No. ADT is just fine.

2. What is the Minimum Required SDK & other version to be chosen while creating the application?

   Answer: Choose the minimum to be android version with NFC. Remember this is the Minimum SDK required so it means that if you have a device with lower versions; your application will not get installed or will not work correctly.

   Minimum required SDK : API16

   Target SDK : API19 <your device or target version of SDK>

   Compile with : API19

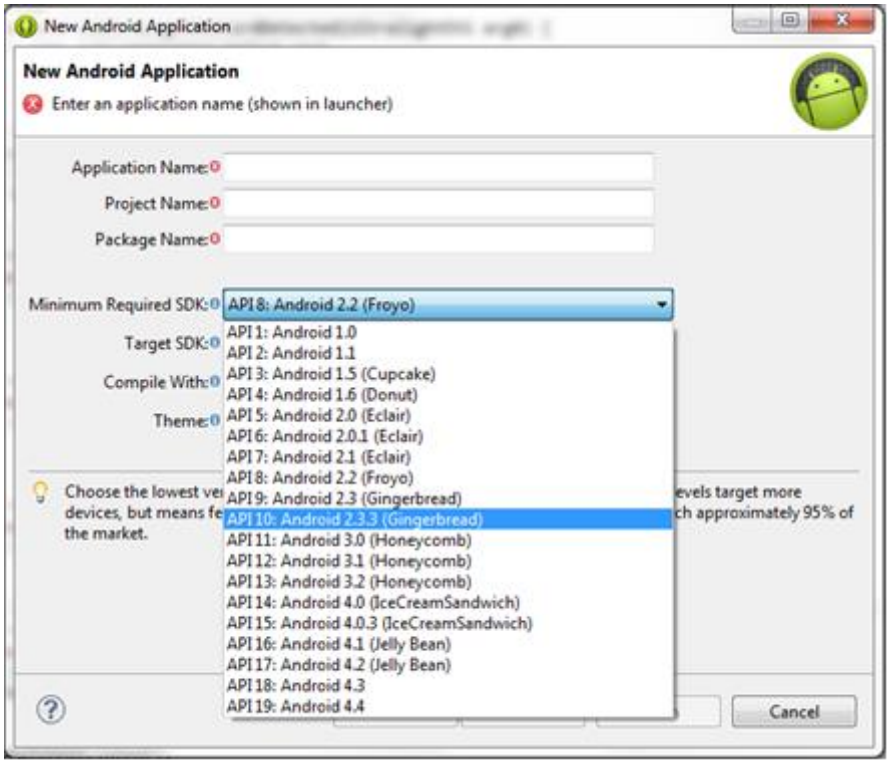


Fig 22. New Android Application settings

3. What are the Target SDK & Compile with options?

   Answer: It is suggested to choose your device version for both Target SDK and compile with. Otherwise you could choose the latest android versions. It is up to the developer choice.

4. Why do you need ACCESS_NETWORK_STATE & INTERNET permissions for an NFC application?

   Answer: We want to get one-time notification of the library usage to the NXP server. So accessing network state and internet permissions are required. Whenever internet or data connection is available; the data containing android version, first usage timestamp is sent. This is done only once.

5. Why do you need WRITE_EXTERNAL_STORAGE permission?

   Answer: This is required to save the log in external storage location and is optional. As part of the library, there is log utility(NxpLogUtils). If the customer wants to save the log to a file; he could just use the API NxpLogUtils.save(). The log will be saved as logdump.xml in NxpLogdump folder in external storage location. This is optional and need not be requested if it is not required to store the log in file.

6. Why are we choosing many intent actions such as NDEF_DISCOVERED, TAG_DISCOVERED & TECH_DISCOVERED ?

   Answer: Please refer to How NFC Tags are Dispatched to Applications section of http://developer.android.com/guide/topics/connectivity/nfc/nfc.html

7. Why are we calling startForeGroundDispatch & stopForeGroundDispatch in onResume() & onPause() ?

   Answer: This allows that foreground activity to have priority when an NFC tag is discovered. This will be very much useful when you have other NFC related applications installed on the device. With this starting the foreground(); only your application gets the tag detected notification as long as it is in launched state.

8. Nothing happens when I try to use any API with arg0 object in intent handler. For example, when I use arg0.write() as first API, What to do?

   Answer: Before using any API of the library; you should use connect() first such as arg0.connect() and then any other card transaction APIs.

9. Does MIFARE SDK support any other SAM reader, apart from HID?

   Answer: Not currently.

8. Appendix 4 - MIFARE Smartcard IC Portfolio

MIFARE includes the broadest product portfolio tailored for more than 40 different applications. It includes ICs for limited-use paper tickets as well as microcontrollers for dual interface smart cards with PKI crypto engines capable of hosting multiple applications. The leading product families available for smart card solutions are MIFARE Classic, MIFARE Plus, MIFARE DESFire and SmartMX. You can find an overview of all MIFARE card and ticket ICs as well as the whole identification portfolio here.

MIFARE Classic

Launched in 1995, the MIFARE Classic 1K IC was the first product, which could be fitted into an ISO contactless smart card and with its slim coil, allowed very high volume production. Today, millions of MIFARE Classic 1K ICs are in use around the globe in all different applications – from public transport, road-tolling and parking meters, to accessing car parks and paying at fuel pumps.

Expanding on the functionality of the existing MIFARE Classic 1K ICs, the MIFARE Classic 4K IC was developed. It provides significant increased memory size making it a suitable solution for multi-application cards. It allows end-users to be more flexible and to experience improved convenience.

MIFARE Plus

The MIFARE Plus offers breakthrough security and performance for the cost-sensitive automated fare collection (AFC) and access control markets. MIFARE Plus is the latest addition to NXP's MIFARE portfolio and features multiple levels of security, including Advanced Encryption Standard (AES) encryption, and a migration path from existing MIFARE Classic implementations.

MIFARE DESFire

MIFARE DESFire EV1 is ideal for service providers wanting to use multi-application smart cards in transport schemes, e-government or identity applications. It fully complies with the requirements for fast and highly secure data transmission, flexible memory organization and interoperability with existing infrastructure.

For more details, please visit - http://www.mifare.net/en/products/mifare-smartcard-ic-s/

9. Appendix 5 - MIFARE SmartTicket IC Portfolio

MIFARE Ultralight, Ultralight C & Ultralight EV1

MIFARE offers the broadest product portfolio tailored to the automatic fare collection market. It includes ICs for limited-use paper tickets as well as microcontrollers for dual interface smart cards with PKI crypto engines capable of hosting multiple applications. The leading product family available is MIFARE Ultralight. You can find an overview of all MIFARE card and ticket ICs as well as the whole identification portfolio here.

MIFARE Ultralight

Tickets based on MIFARE Ultralight ICs can act as single trip tickets in public transportation networks, loyalty cards or even day passes at big events. They are the ideal replacement for conventional ticketing solutions such as paper tickets, magnetic-stripe tickets or coins.

As the usage of contactless proximity smart cards becomes more and more common, transport operators are beginning to switch to completely contactless solutions. The introduction of the new contactless MIFARE Ultralight IC for limited-use tickets will lead to a reduction of system installation and maintenance costs. Terminals will be less vulnerable to damages and mechanical failures caused by ticket jams. MIFARE Ultralight can easily be integrated into existing schemes and even standard paper ticket vending equipment can be upgraded. In addition, this solution for low cost tickets helps transport operators to reduce fraud and the circulation of cash within the system.

As frequent travelers and commuters normally use a high end contactless smart card (e.g.: based on MIFARE DESFire) for their regular trips, MIFARE Ultralight enables occasional travelers to benefit from the same advantages. It significantly improves boarding times and helps to experience quicker travel and easier movement between buses, trains and other means of transportation.

The mechanical and electronic specifications of MIFARE Ultralight are tailored to meet the requirements of paper ticket manufacturers. Issuing smart paper tickets based on MIFARE Ultralight only requires a minor upgrade to standard Edmonson / Eurosize ticket vending terminals. This can be achieved by fitting a simple contactless reader for ticket initialization.

MIFARE Ultralight operates according to the ISO 14443A standard. Meaning cards or tickets based on MIFARE Ultralight can be used at a distance of up to 10 cm with true anti-collision properties and without the need for a battery. Last but not least, MIFARE Ultralight is fully compatible with all existing MIFARE infrastructures and can therefore be easily integrated in current transportation schemes.

Key applications

- Limited-use tickets in public transport (e.g.: single trip tickets, multiple trip tickets, tourist weekend passes)
- Event ticketing (stadiums, exhibitions, leisure parks, etc.)

MIFARE Ultralight C

MIFARE Ultralight C is the newest member of the MIFARE product portfolio, complementing the offering in the low-cost segment.

It is the first smart card IC for limited-use applications that offers solution developers and providers the benefits of an open cryptography. With 3DES, MIFARE Ultralight C uses a widely adopted standard, enabling easy integration in existing infrastructures. The integrated authentication command set provides an effective anti-cloning functionality that helps to prevent counterfeit of tickets.

Tickets, vouchers or tags based on NXP MIFARE Ultralight C ICs can act as single trip tickets in public transportation networks or loyalty cards, day passes at big events or smart poster tags in NFC applications. They are the ideal replacement for conventional ticketing solutions such as paper tickets, magnetic-stripe tickets or coins.

MIFARE Ultralight C operates according to the ISO /IEC 14443 A standard, allowing an operating distance of up to 10cm with true anti-collision support. MIFARE Ultralight C is fully compatible with existing MIFARE infrastructure and can therefore be easily integrated in current contactless solutions. Especially existing solutions utilizing MIFARE Ultralight and MIFARE DESFire EV1 can benefit, as the command set is compatible to MIFARE Ultralight and the authentication commands are the same as the ones used in MIFARE DESFire EV1.

Key applications

- Limited-use tickets in public transport (e.g.: single trip tickets, multiple trip tickets, tourist weekend passes)
- Event ticketing (stadiums, exhibitions, leisure parks, etc.)
- Loyalty and pre-paid
- NFC Forum Tag Type 2

MIFARE Ultralight EV1

MIFARE Ultralight EV1 is the next generation of paper ticketing smart card IC for limited-use applications that offers solution developers and operators the maximum flexibility for their ticketing schemes and additional security options.

Enabling easy integration in existing infrastructures is guaranteed by compatibility with MIFARE based systems. The integrated originality checker is an effective cloning protection that helps to prevent counterfeit of tickets.

MIFARE Ultralight EV1 reflects the trend for enhanced security in contactless applications. Nowadays many solution providers eliminate double infrastructure where MIFARE Ultralight EV1 provides the perfect solution for a complete contactless system. It can easily be integrated in existing MIFARE installations.

Key applications

- Limited-use tickets in public transport
- Event ticketing (stadiums, exhibitions, leisure parks and many more)
- Loyalty

For more details, please visit http://www.mifare.net/en/products/mifare-smartticket-ics/

10. Appendix 6 - NTAG

NFC Forum compliant NTAG ICs are the ideal choice for mass market deployment of NFC proximity marketing and electronics pairing applications.

Based on ISO14443A technology, NTAG combines ease of integration, high RF sensitivity and anti-cloning features to provide benefits for the whole value chain up to the end consumer.

With the NTAG family, NXP is further expanding its NFC portfolio, already comprising NFC radio ICs, contactless reader / writers, embedded secure elements, contactless and dual interface smart cards (MIFARE and SmartMX), and authentication solutions.

NTAG I2C - In addition to the passive NFC Forum compliant contactless interface, the IC features an I²C contact interface, which can communicate with a microcontroller if the NTAG I²C is powered from an external power supply.

Key features and benefits

- ISO / IEC 14443A 2-3 compliant
- NFC Forum compliant
- Worldwide proven and reliable technology, together with worldwide support and third party equipment
- Unique identifier for each IC
- Re-programmable user memory from 48 bytes up to 32 kbytes with data retention up to 10 years
- Cloning protection
- Optional read-only locking function, crypto authentication & Field Detection pin

For more details, please visit -
http://www.nxp.com/products/identification_and_security/smart_label_and_tag_ics/ntag/ AND
http://nxp-rfid.com/ntag-i2c/

Key Applications

- Smart advertisements
- Connection Handovers
- Bluetooth simple pairing
- Wi-Fi protected setups
- Call & SMS requests
- Loyalty (vouchers, coupons)

11. Appendix 7 - ICODE

ICODE is the industry standard for high-frequency (HF) smart label solutions with billions of ICs in the field and thousands of successful installations. This proven technology supports the ISO 15693 / ISO 18000-3 compliant infrastructure.

Key features and benefits

- Best-in-class RF performance
- Worldwide proven and reliable technology, together with worldwide support and third party equipment
- Unique identifier for each IC
- Features optimally aligned with user requirements
- Re-programmable user memory up to 1280 bits with data retention up to 50 years
- Compliant with world-wide harmonized regulations
- Optional password protection for EAS, AFI and user memory
- Best performance-to-cost ratio
- Privacy functionalities
- Comprehensive portfolio of compatible products with strong roadmap on future innovations

Key applications

- Library and rental services
- Healthcare
- Ski ticketing
- Asset management and smart shelf solutions
- Factory automation

For more details, please visit -
http://www.nxp.com/products/identification_and_security/smart_label_and_tag_ics/icode/

12. Appendix 8 - MIFARE SAM

For a growing number of smart card applications, service providers need to ensure ever higher security standards, NXP MIFARE SAM AV2 is the perfect solution. It turns even the simplest reader into a high security transmission device. With 3DES and AES capabilities, our hardware solution offers superior cryptographic features for a variety of secure infrastructures.

Key benefits

- No need for high-end cryptography features in a smart card reader
- New, exceptional performance with direct connection to the contactless reader
- Supports crypto authentication and data encryption / decryption

Key features

- Compatible with MIFARE portfolio
- Supports MIFARE, 3DES and AES cryptography
- Key diversification
- Secure download and storage of keys
- 128 key entries
- ISO 7816 Baud rate up to 1.5 Mbit/s
- Will be available in PCM 1.1 module and HVQFN package

MIFARE SAM Product sheet

MIFARE, MIFARE Ultralight, MIFARE Plus, DESFire and SmartMX are trademarks of NXP Semiconductors N.V.

For more details, please visit http://www.mifare.net/en/products/mifare-sam-av21/

13. Legal information

13.1 Definitions

The content in this document is under internal review and improvement, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

13.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors accepts no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Evaluation products — This product is provided on an "as is" and "with all faults" basis for evaluation purposes only. NXP Semiconductors, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this product remains with customer.

In no event shall NXP Semiconductors, its affiliates or their suppliers be liable to customer for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out the use of or inability to use the product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages.

Notwithstanding any damages that customer might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP Semiconductors, its affiliates and their suppliers and customer's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by customer based on reasonable reliance up to the greater of the amount actually paid by customer for the product or five dollars (US$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose.

13.3 Licenses

Purchase of NXP MIFARE SDK components
Please visit MIFARE SDK page of www.mifare.net

13.4 Patents

Notice is herewith given that the subject software uses one or more of the patents and that each of these patents may have corresponding patents in other jurisdictions.

13.5 Trademarks

Notice: All referenced brands, product names, service names and trademarks are property of their respective owners.

MIFARE & MIFARE SDK — is a trademark of NXP B.V.

14. List of figures

15. Contents

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 24 Feb 2015
Document identifier: UM10809