AN11367

How to build a NFC Application on Android

Rev. 1.0 — 19 June 2013
270210

Application note
COMPANY PUBLIC

Document information

Info	Content
Keywords	How to build a NFC Application on Android, android
Abstract	This application note is related to the implementation procedures of Android applications handling NFC data transfer with the RC663 Blueboard and the NP512.

Revision history

Rev	Date	Description
1.0	20130619	First release

Contact information

For more information, please visit: http://www.nxp.com

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

1. Introduction

This application note explains how to develop Android applications on smart phones to work with the RC663 contactless reader.

The phone handles NDEF messages – image, text… - received from the RC663 and the PN512 and displays its content. The messages used in the example explained in this document are encapsulated into SNEP messages before being sent by the reader.

Further information can be found at [1] and [2] but with this document, one can be able to get started.

2. Requirements

2.1 Blue board p2p project

This project can be downloaded from the web site of the CLEV663B [3].

2.2 Android phone

The example shown in this document was made for the Google Galaxy Nexus GT-I9250 with Android version 4.2.2 (Jelly Bean) and NFC (Android beam) hardware.

Fig 1. Google Galaxy Nexus

2.3 Android SDK

The Android SDK offers the API libraries and developer tools essential to build, test and debug Android applications.

More information about Android SDK download and configuration can be found at [4].

3. About Android

This section is derived from [5] where it is treated in detail.

Android is an operating system for mobiles officially announced in November 2007 and originally developed by a startup called itself Android which has been bought by Google afterwards. Android attracted too many developers by the fact of being open source and Linux kernel based. Since the 3.0 version Honeycomb, Android is also available on tablets. Other versions of Android also exist on notebooks and televisions.

After an impressive evolution, Android became one of the most important Google brands, especially for smart phone. The most popular constructors implanting Android on various devices are: Samsung, Sony, HTC, LG, Motorola…

The different drafts of Google's operating system carry a version number and named one after the other by desert names in alphabetical order.

Thus, Google has already released around ten Android versions:

- Android 1.0 Apple Pie
- Android 1.1 Banana Bread
- Android 1.5 Cupcake
- Android 1.6 Donut
- Android 2.0 / 2.1 Éclair
- Android 2.2 Frozen Yogurt (FroYo)
- Android 2.3 Gingerbread (GB)
- Android 3.0 Honeycomb
- Android 4.0 Ice Cream Sandwich (ICS)
- Android 4.1 / 4.2 Jelly Bean (JB)

Each major version is subject to many improvements or corrections. Therefore, there are alternatives in each version like Gingerbread includes all Android versions from 2.3.1 to 2.3.7.

The operation system is now not developed only by Google but by a consortium, the Open Handset Alliance [6], that unites Google and mobile phones and semiconductors manufacturer and operators, as long as development units.

3.1 Android API

Android API consists of a set of packages defining classes that ease the use of Android features and the devices components.

These packages are constantly upgraded and added as the API level gets higher. Depending on the Android version installed on a device, some APIs won't be available. Minimum API levels to handle NFC hardware are defined in Chapter 3.3.

3.2 Android application development basics

The language used to write Android application is the Java programming language. The code, as well as eventual data and/or resource files, is compiled by the Android SDK tool into a single .apk file. This file is used by Android-powered devices to install the application.

Each Android application lives in its own Security Sandbox.

Every application has a unique Linux user ID assigned to it by the system and specific permissions on all of its files. Thereby they can only be accessed by the user ID assigned to that application.

Each application runs in its own Linux process that is started when any of the application components need to be executed and shut down when it is no longer needed or if the system has to recover memory for other applications.

The process itself has its own virtual machine so that an application's code can be executed separately from other applications.

By default, each application has only access to the components that are required to execute its work. This is called *the principle of least privilege* which creates a highly secure environment in which it is not possible for an application to access parts of the system for which it is not given permission.

If one needs to make applications share data, it is possible to arrange for two applications to share the same Linux user ID so that they can access each other's files. This can also make the applications share the same Linux process and the same virtual machine.

In order to access system services, an application can request permission to access device data such as SMS messages, camera, Bluetooth… Note that all application permissions must be granted by the user at install time.

3.2.1 Application components

These are the necessary structuring blocks of an Android application. The system can enter the application trough any of its component as they represent different entry points. Not all components are considered as entry point and some of them depend of each other but each one is its own entity and has a specific role to play.

Application components come in four types: Activities, which represent a single screen with a user interface; Services that run in the background to perform long-running operations or to perform work for remote processes; Content Providers which manage a shared set of application data and Broadcast Receivers which are components that respond to system-wide broadcast announcements. Each type has a distinct goal and lifecycle which defines how to create and destroy the components.

3.2.2 Activities

An Android application is often composed by several activities loosely bound to each other. Normally, the first activity that is presented to the user when lunching the application for the first time is specified as the "main" activity.

Each activity can start another one to perform different actions.

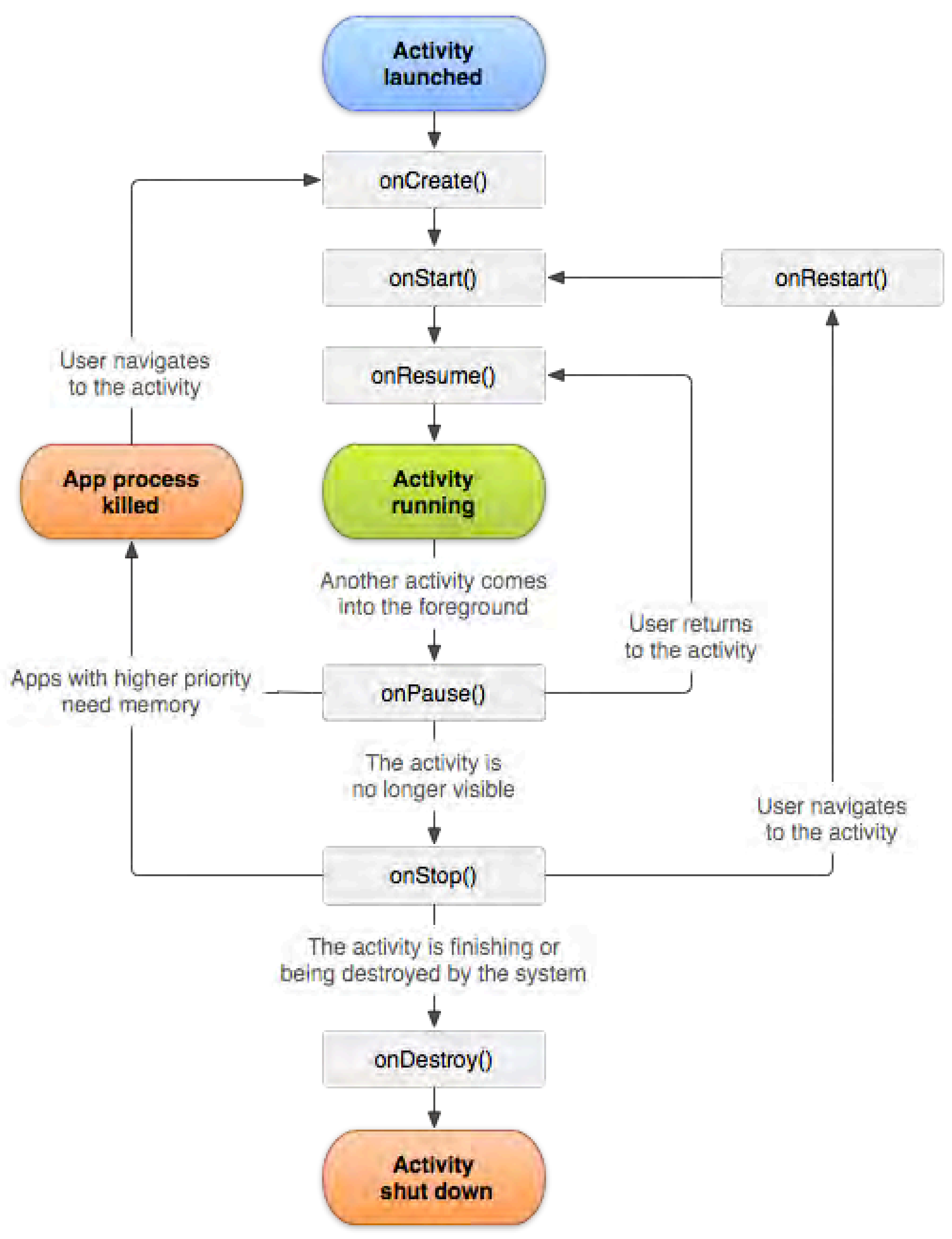
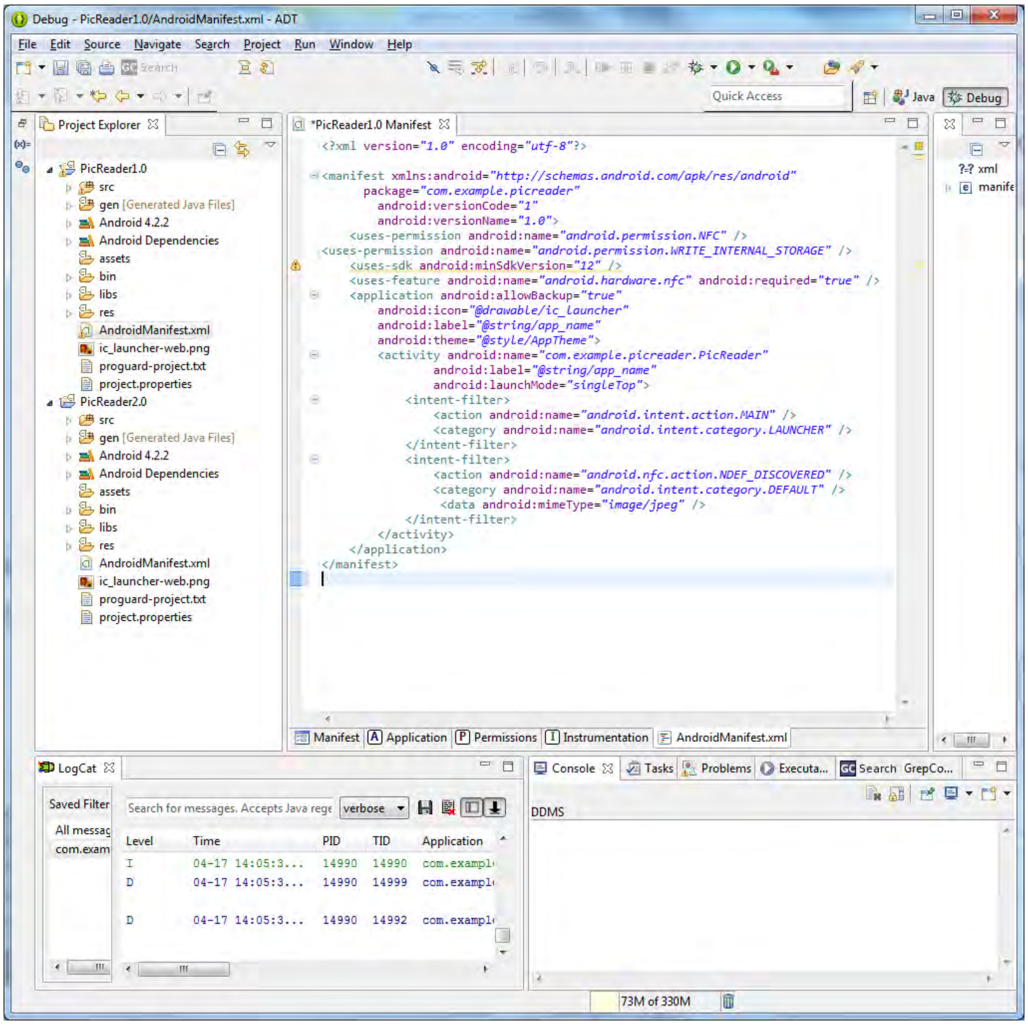

Activity launched
onCreate()
onStart()
onRestart()
onResume()
User navigates to the activity
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Fig 2. Activity Lifecycle [7]

3.2.3 Manifest file

The manifest file indicates to the system that the component it is trying to start already exists. The application must declare all its components in an `AndroidManifest.xml` file that should be at the root of the application project directory.

One can also find, in this file, information about user permissions required by the application, the minimum API level, hardware and software features, API libraries that the application should be linked against...



```
<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.example.picreader"
        android:versionCode="1"
        android:versionName="1.0">
    <uses-permission android:name="android.permission.NFC" />
<uses-permission android:name="android.permission.WRITE_INTERNAL_STORAGE" />
    <uses-sdk android:minSdkVersion="12" />
    <uses-feature android:name="android.hardware.nfc" android:required="true" />
    <application android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">
        <activity android:name="com.example.picreader.PicReader"
                android:label="@string/app_name"
                android:launchMode="singleTop">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.nfc.action.NDEF_DISCOVERED" />
                <category android:name="android.intent.category.DEFAULT" />
                 <data android:mimeType="image/jpeg" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Fig 3. AndroidManifest.xml

3.2.4 Application Resources

In addition to the code, an Android application also requires resources separated from the code like images, audio files….

For each resource included in the application, the SDK build tools define a unique integer ID that is used to reference the resource from the application code or other resources defined in XML.

3.2.5 Running an application

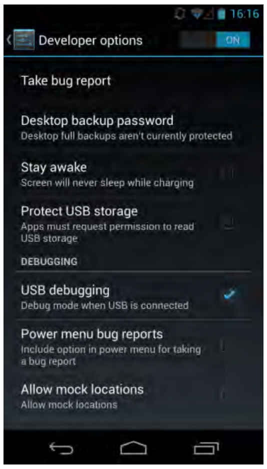
This section is derived from [8] where one can find more details and alternatives.

An Android application can be either run on the Eclipse emulator or on a real Android-powered device.

3.2.5.1 On real Android-powered device

In order to run the application on a real Android-powered device, these steps should be followed:

1. Plug in the device to the development machine with a USB cable. Appropriate USB driver for the device must be installed. More details about installing drivers can be found at [9].
2. Enable USB debugging on the device.


16:16

Developer options ON

Take bug report

Desktop backup password
Desktop full backups aren't currently protected

Stay awake
Screen will never sleep while charging

Protect USB storage
Apps must request permission to read USB storage

DEBUGGING

USB debugging ✓
Debug mode when USB is connected

Power menu bug reports
Include option in power menu for taking a bug report

Allow mock locations
Allow mock locations

Fig 4. USB debugging enabled

3. Under Eclipse, click on the project to run and click on the “Run” button from the toolbar.
4. A “Run as” window appears, select “Android Application” and click “OK”.

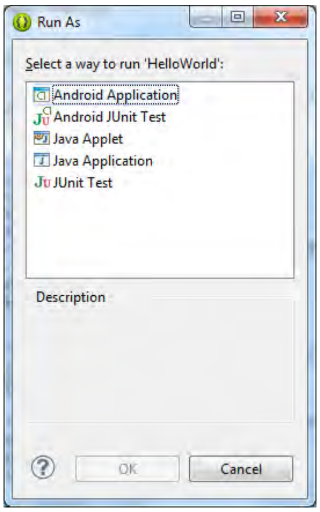


Fig 5. “Run as” window

5. The application will be installed on the connected device and started.

3.2.5.2 On the Emulator

If no real Android-powered device is available, one can also run the application on the emulator by following these steps:

1. Create an Android Virtual Device:
    a. Launch the Android Virtual Device Manager by clicking on “Android Virtual Device Manager” button from the toolbar.
    b. A panel will show up, click “New”.

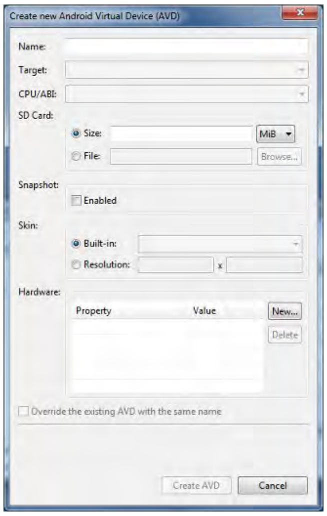


Fig 6. Android Virtual Device Manager

c. Fill in the details.

d. Click “Create AVD”.

e. Select the created item and click “Start”.

f. The emulator will boot up.

2. Under Eclipse, click on the project to run and click on the “Run” button from the toolbar.
3. A “Run as” window appears, select “Android Application” and click “OK”.
4. The application will be installed on the connected device and started.

3.2.6 HelloWorld Android application

A simple example is presented and explained at [10] .

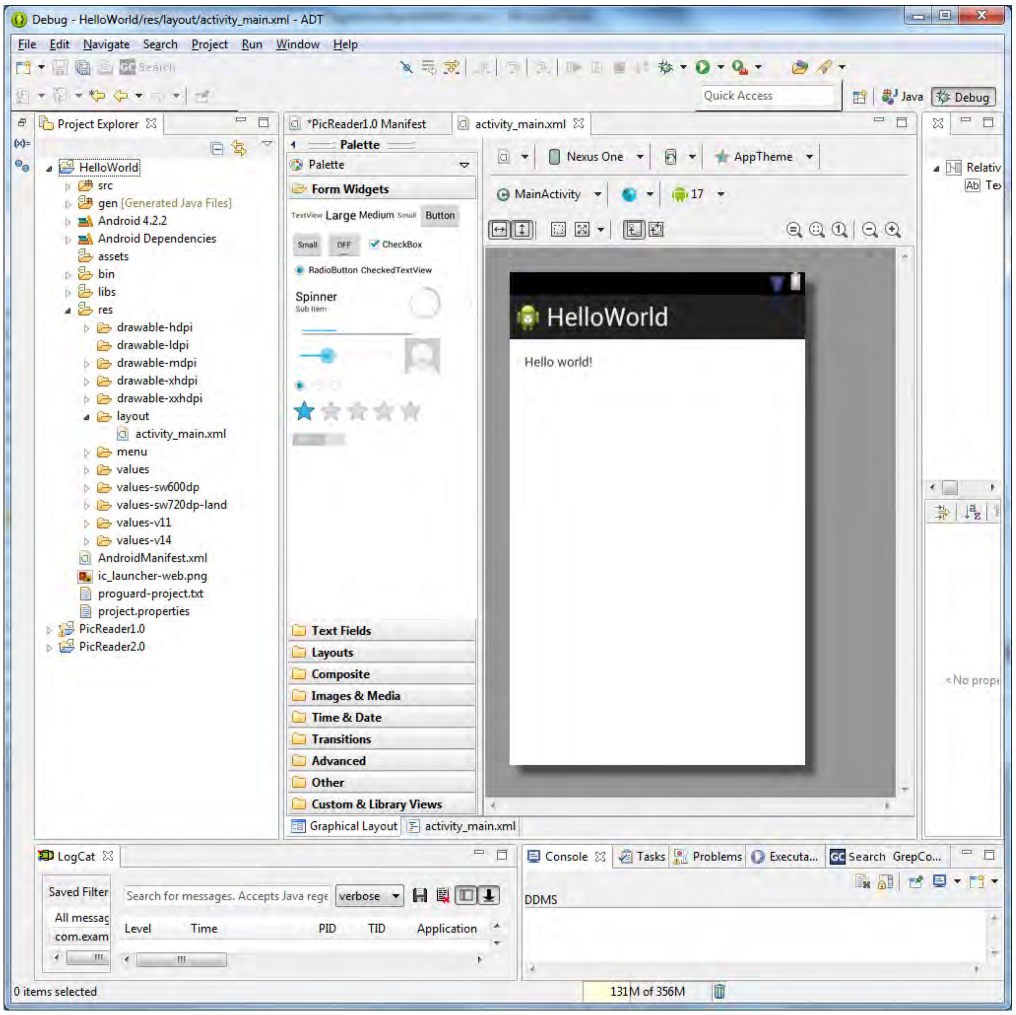


Fig 7. HelloWorld Android application

3.3 NFC Android API available

API level 9 only supports limited tag dispatch trough ACTION_TAG_DESCOVERED, and only gives access to NDEF messages through the EXTRA_NDEF_MESSAGES extra.

API level 10 includes reader/writer support as well as foreground NDEF pushing.

API level 14 provides easy way to push NDEF messages to other devices introducing Android Beam and simpler methods for NDEF records creation.

3.3.1 android.nfc

Offers access to Near Field Communication (NFC) functionality, letting applications read NDEF message in NFC tags. A "tag" may actually be another device that appears as a tag just like the CLEV663B in this case.

Classes' summary:

`NfcManager`

High level manager, used to get this device's `NfcAdapter`. An instance can be acquired using `getSystemService(String)`.

`NfcAdapter`

Represents the device's NFC adapter, which is the entry-point to NFC operations performing. An instance can be acquired using `getDefaultAdapter()`, or `getDefaultAdapter(android.content.Context)`.

`NdefMessage`

Represents an NDEF data message (the standard format in which "records" carrying data are transmitted between devices and tags). The application can receive these messages from an `ACTION_TAG_DISCOVERED` intent.

`NdefRecord`

Represents a record, which is delivered in an `NdefMessage`, carries the data being shared and describes its type.

3.3.2 android.nfc.tech

These classes offer access to the features of a tag technology, which vary by the type of tag that is scanned. A scanned tag can support multiple technologies that can be found by calling `getTechList()`.

4. Preparing the Manifest file

To be able to create an Android application that handles the phone’s NFC hardware, one should properly prepare the “AndroidManifest.xml” file.

4.1 Permission to use NFC

To use the NFC functionality of the phone, the following permission should be declared:

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

4.2 Android SDK min version

Since reader/writer support and foreground NDEF pushing are available only from API level 10, it should be used as the minimum version of Android SDK:

```
<uses-sdk android:minSdkVersion="10" />
```

4.3 NFC feature

In order to display the application in PlayStore only for NFC hardware equipped devices, the following feature should be declared:

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

4.4 Intent declaration

To make sure the application lunches on NDEF tag detection, the following intent should be used:

```
<intent-filter>
    <action android:name="android.nfc.action.NDEF_DISCOVERED" />
    <category android:name="android.intent.category.DEFAULT" />
       <data android:mimeType="image/*" /> // for image type message
       <data android:mimeType="text/*" />  // for text type message
</intent-filter>
```

5. Main part of the application

5.1 Import Library

To be able to work with NFC functionalities the most important libraries one can use are the following:

```
Import android.nfc.NdefMessage;
Import android.nfc.NfcAdapter;
```

5.2 Enable Foreground Dispatch

The figure below shows how the system reacts when detecting a tag.

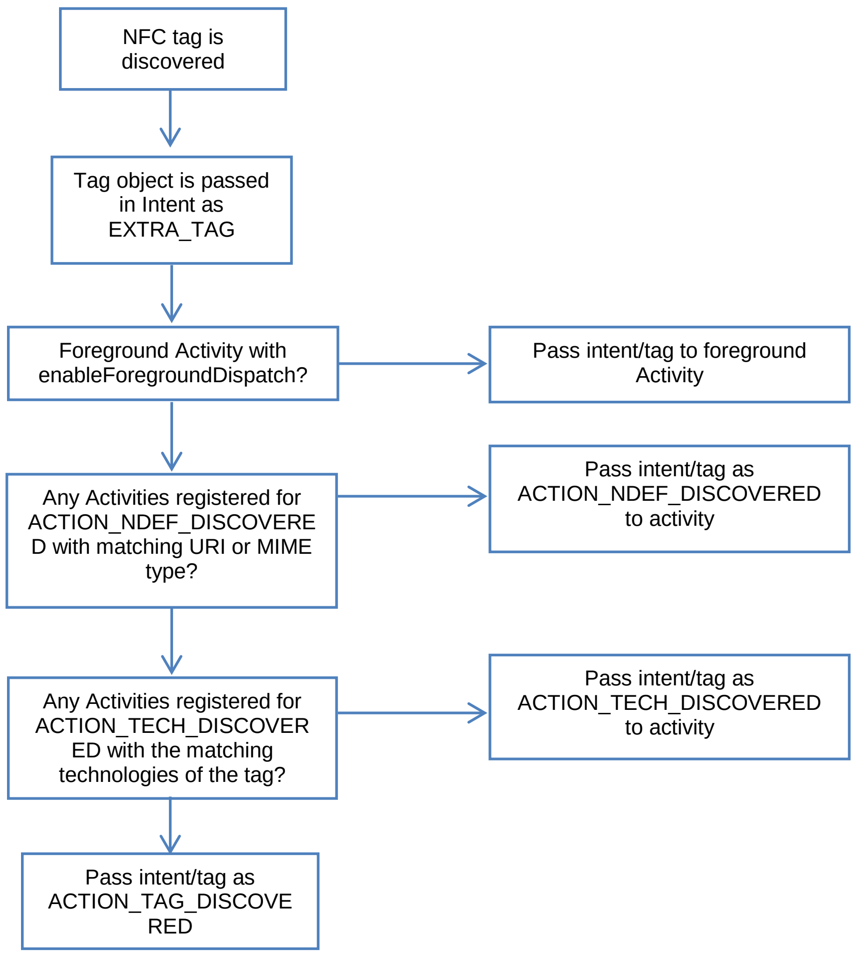

NFC tag is discovered
Tag object is passed in Intent as EXTRA_TAG
Foreground Activity with enableForegroundDispatch?
Pass intent/tag to foreground Activity
Any Activities registered for ACTION_NDEF_DISCOVERED with matching URI or MIME type?
Pass intent/tag as ACTION_NDEF_DISCOVERED to activity
Any Activities registered for ACTION_TECH_DISCOVERED with the matching technologies of the tag?
Pass intent/tag as ACTION_TECH_DISCOVERED to activity
Pass intent/tag as ACTION_TAG_DISCOVERED

Fig 8. NFC Tag flow in Android API level 10

As it appears in the figure, the first application to be called when a tag is detected is the one which is in a foreground status and has enabled the foreground dispatch system.

This is done by retrieving the NFC device adapter, and therefore the "`android.nfc.NfcAdapter`" library is needed.

```
mNfcAdapter = NfcAdapter.getDefaultAdapter();
```

And then, create a `PendingIntent` and pass it to the `enableForegroundDispatch()` method. This should be done in the `onResume()` method to make sure the application is running on a foreground status.

```
@Override
public void onResume() {
   super.onResume();
   PendingIntent intent = PendingIntent.getActivity(this, 0, new Intent(this,
getClass()).addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP), 0);
   NfcAdapter.getDefaultAdapter(this).enableForegroundDispatch(this,intent,
null, null);
}
```

One must not forget to disable the foreground dispatch when the application is running on background status, such as in `onPause()` method.

```
@Override
protected void onPause() {
   super.onPause();
   if (NfcAdapter.getDefaultAdapter(this) != null)
      NfcAdapter.getDefaultAdapter(this).disableForegroundDispatch(this);
   }
```

5.3 Retrieve NDEF Message’s payload

The data of an NDEF message received are wrapped in an array of `Parcelable` objects defined by the term “`android.nfc.extra.NDEF_MESSAGES`”. Each `NdefMessage` in that array is an array of NdefRecord objects itself. Each `NdefRecord` contains 3-bit TNF (type name format), the type of record, a unique ID, and the payload.

To get the `NdefRecord` array from the `NdefMessage` object, one should use the `getRecords()` method which returns a `NdefRecord[]` object.

To get the `NdefRecord` informations, the following methods are available:

```
byte[] getId()      // Returns the variable length ID
byte[] getPayload() // Returns the variable length payload
short  getTnf()     // Returns the 3-bit TNF
byte[] getType()    // Returns the variable length Type field
```

6. Example Project: PicReader

6.1 General presentation

Android application that receives an image sent by the RC663 board and displays it as well as its properties.

6.1.1 Workflow

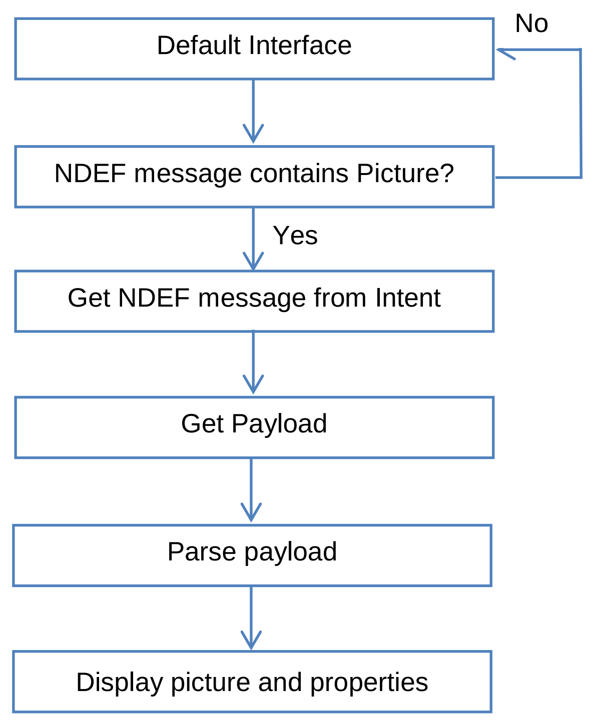

Default Interface
No
NDEF message contains Picture?
Yes
Get NDEF message from Intent
Get Payload
Parse payload
Display picture and properties

Fig 9. Workflow

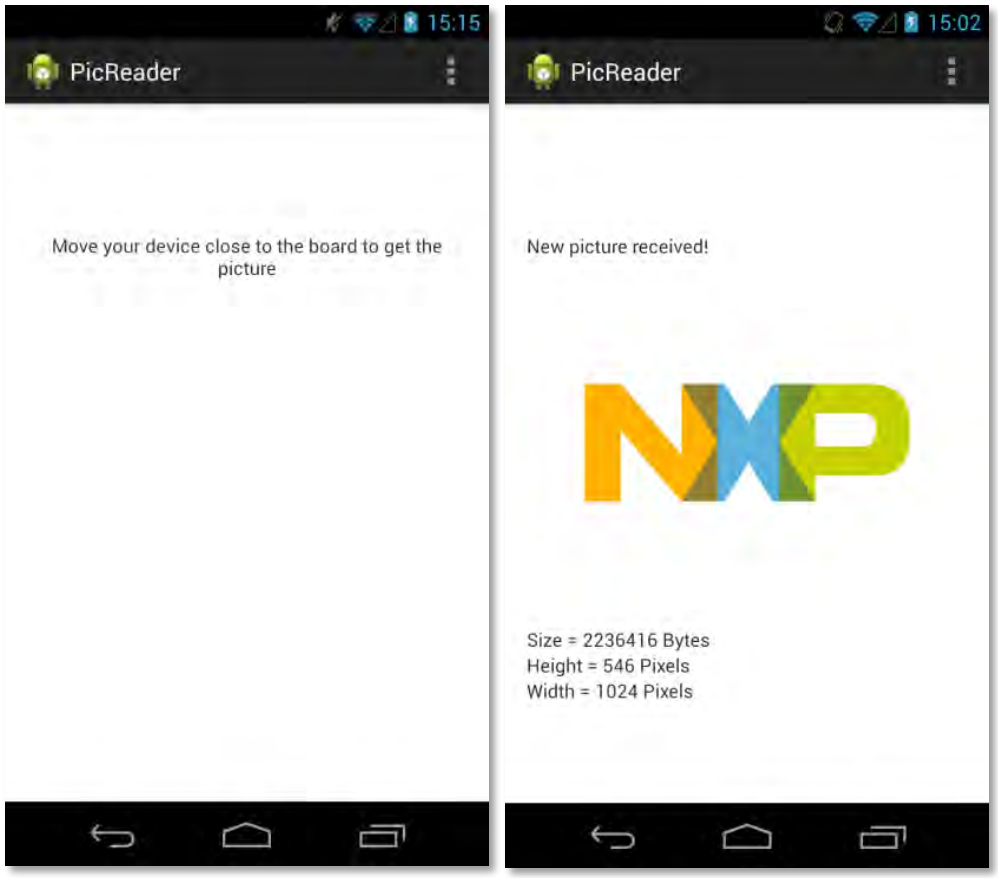
6.1.2 Graphical Interface

In this example, only one user interface is defined `activity_pic_reader.xml`.

By default, the content of this interface is a single indication text: `Move your device close to the board to get the picture`.

Once the NDEF message is received and processed, the same interface content changes to display:

- Text: “New picture received!”
- Image: the picture received.
- Text: Size, width, height of the picture.



Move your device close to the board to get the picture

New picture received!

Size = 2236416 Bytes
Height = 546 Pixels
Width = 1024 Pixels

Fig 10. Graphical Interface

6.2 AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.example.picreader"
        android:versionCode="1"
        android:versionName="1.0">
    <uses-permission android:name="android.permission.NFC" />
<uses-permission
android:name="android.permission.WRITE_INTERNAL_STORAGE" />
  <uses-sdk android:minSdkVersion="12" />
    <uses-feature android:name="android.hardware.nfc"
android:required="true" />
    <application android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">
        <activity android:name="com.example.picreader.PicReader"
```

```
                 android:label="@string/app_name"
                 android:launchMode="singleTop">
             <intent-filter>
                 <action android:name="android.intent.action.MAIN" />
                 <category
android:name="android.intent.category.LAUNCHER" />
             </intent-filter>
             <intent-filter>
                 <action
android:name="android.nfc.action.NDEF_DISCOVERED" />
                 <category
android:name="android.intent.category.DEFAULT" />
                  <data android:mimeType="image/jpeg" />
             </intent-filter>
         </activity>
     </application>

</manifest>
```

6.3 PicReader.java

```
package com.example.picreader;

import java.io.BufferedOutputStream;
import java.io.ByteArrayInputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.app.Activity;
import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.nfc.NdefMessage;
import android.nfc.NfcAdapter;
import android.os.Bundle;
import android.os.Environment;
import android.os.Parcelable;
import android.util.Log;
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.TextView;
import android.widget.Toast;

public class PicReader extends Activity {
   NfcAdapter mNfcAdapter;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_pic_reader);
      // Check for available NFC Adapter
```

```
    if (!chkNFCAv()) {
      return;
    }

    if
(NfcAdapter.ACTION_NDEF_DISCOVERED.equals(getIntent().getAction())) {
      processIntent(getIntent());
    }

  }

  @Override
  public void onResume() {
    super.onResume();
    PendingIntent intent = PendingIntent.getActivity(this, 0, new
Intent(
        this, getClass()).addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP),
0);
    NfcAdapter.getDefaultAdapter(this).enableForegroundDispatch(this,
        intent, null, null);
  }

  @Override
  protected void onPause() {
    super.onPause();
    if (NfcAdapter.getDefaultAdapter(this) != null)

  NfcAdapter.getDefaultAdapter(this).disableForegroundDispatch(this);
  }

  @Override
  public void onNewIntent(Intent intent) {
    // onResume gets called after this to handle the intent
    setIntent(intent);

    if
(NfcAdapter.ACTION_NDEF_DISCOVERED.equals(getIntent().getAction())) {
        processIntent(getIntent());
    }

  }

  public boolean chkNFCAv() {
    mNfcAdapter = NfcAdapter.getDefaultAdapter(this);
    if (mNfcAdapter == null) {
      Toast.makeText(this, "NFC is not available", Toast.LENGTH_LONG)
          .show();
      finish();
      return false;
    }
    return true;
  }

  // Parses the NDEF Message from the intent
```

```
void processIntent(Intent intent) {
  Parcelable[] msgs = intent
      .getParcelableArrayExtra(NfcAdapter.EXTRA_NDEF_MESSAGES);
  NdefMessage[] nmsgs = new NdefMessage[msgs.length];
  for (int i = 0; i < msgs.length; i++) {
    nmsgs[i] = (NdefMessage) msgs[i];

  }
  byte[] payload = nmsgs[0].getRecords()[0].getPayload();
  if (payload != null) {
    TextView pageTitle = (TextView) findViewById(R.id.textView2);
    ImageView iv = new ImageView(this);
    TextView SizeIVTV = new TextView(this);
    TextView HeightIVTV = new TextView(this);
    TextView WidthIVTV = new TextView(this);
    LinearLayout ll1 = (LinearLayout) findViewById(R.id.ll1);
    if (nmsgs[0].getRecords()[0].getTnf() == 2) {
      Toast.makeText(this, "New picture received!",
          Toast.LENGTH_SHORT).show();
      ByteArrayInputStream imageStream = new ByteArrayInputStream(
          payload);
      Bitmap b = BitmapFactory.decodeStream(imageStream);
      iv.setImageBitmap(b);
      // saveImage(b);

      String sizeIV = "Size = " + b.getByteCount() + " Bytes";
      SizeIVTV.setText(sizeIV);
      ll1.addView(SizeIVTV);

      String heightIV = "Height = " + b.getHeight() + " Pixels";
      HeightIVTV.setText(heightIV);
      ll1.addView(HeightIVTV);
      pageTitle.setText("New picture received!\n\n");
      String widthIV = "Width = " + b.getWidth() + " Pixels";
      WidthIVTV.setText(widthIV);
      ll1.addView(WidthIVTV);
      ll1.addView(iv);
    }
  }

}


public static final String md5(final String s) {
  try {
    // Create MD5 Hash
    MessageDigest digest = java.security.MessageDigest
        .getInstance("MD5");
    digest.update(s.getBytes());
    byte messageDigest[] = digest.digest();

    // Create Hex String
    StringBuffer hexString = new StringBuffer();
```

```
      for (int i = 0; i < messageDigest.length; i++) {
        String h = Integer.toHexString(0xFF & messageDigest[i]);
        while (h.length() < 2)
          h = "0" + h;
        hexString.append(h);
      }
      return hexString.toString();

    } catch (NoSuchAlgorithmException e) {
      e.printStackTrace();
    }
    return "";
  }

  public Bitmap saveImage(Bitmap b) {

    String fileName = md5(b.getDensity() + "" + b.getHeight()
        + b.getWidth());

    String fname = fileName + ".jpeg";
    Toast.makeText(getApplicationContext(), fname, Toast.LENGTH_LONG)
        .show();

    File dir = getDir(Environment.DIRECTORY_PICTURES,
Context.MODE_PRIVATE);
    File internalFile = null;

    internalFile = new File(dir, fname);
    internalFile.setReadable(true);
    internalFile.setWritable(true);
    internalFile.setExecutable(true);
    try {
      internalFile.createNewFile();

      FileOutputStream fos = new FileOutputStream(internalFile);
      BufferedOutputStream bos = new BufferedOutputStream(fos);

      b.compress(Bitmap.CompressFormat.JPEG, 100, bos);

      bos.flush();
      bos.close();

    } catch (Exception e) {
      Log.e("eks", e.getMessage());
    }

    return (b);

  }

}
```

6.4 Derivative: receiving two NDEF messages

One may need to receive more than one NDEF message and display all the contents.

To do this, some changes must occur in the code. In this example, we will add one NDEF message of text type that should be sent before the picture.

The main changes are illustrated in the following figure.

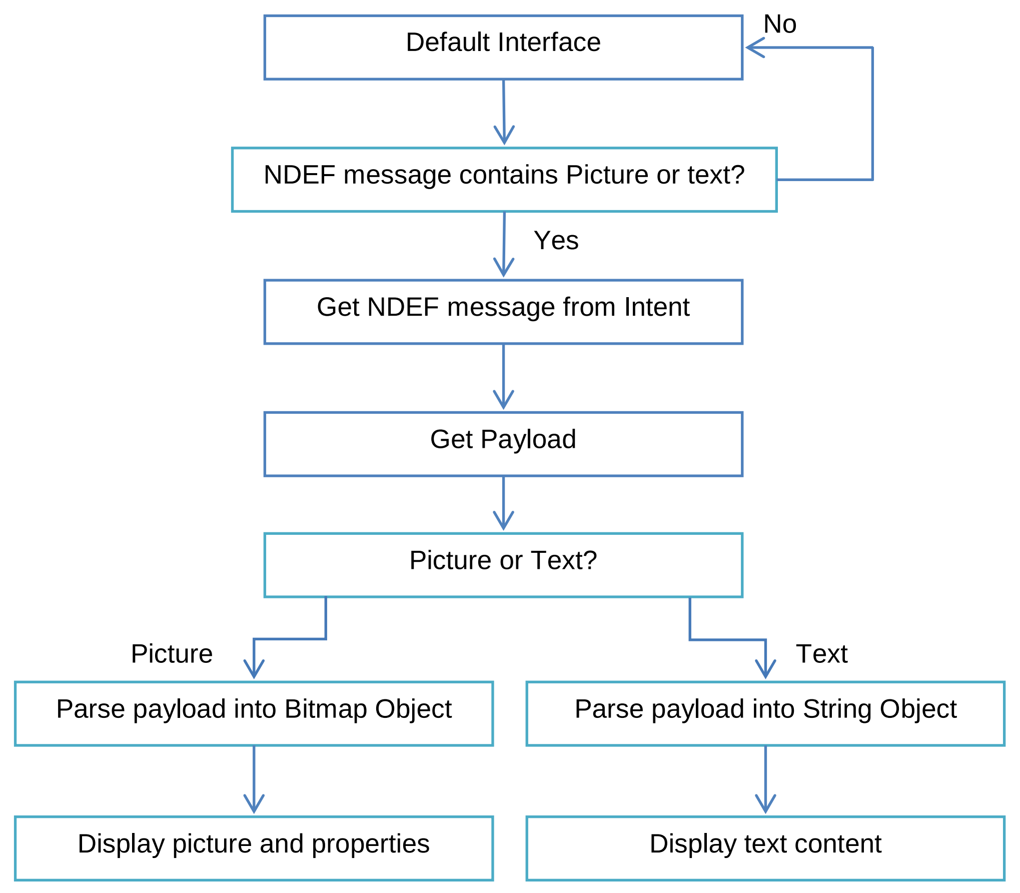

Default Interface
No
NDEF message contains Picture or text?
Yes
Get NDEF message from Intent
Get Payload
Picture or Text?
Picture
Text
Parse payload into Bitmap Object
Parse payload into String Object
Display picture and properties
Display text content

Fig 11. Derivate Workflow

The changes in the code will be like following:

- Add Text Type detection in the `AndroidManifest.xml` file:

```
<intent-filter>
      <action android:name="android.nfc.action.NDEF_DISCOVERED" />
      <category android:name="android.intent.category.DEFAULT" />
      <data android:mimeType="image/jpeg" />
      <data android:mimeType="text/*" />
</intent-filter>
```

- Add content verification (text/image):

```
if (nmsgs[0].getRecords()[0].getTnf() == 2) // 2 = image
```

In this case, we perform the same code as for the first example. Else, we display the text contained in the NDEF message received.

AN11367 All information provided in this document is subject to legal disclaimers. © NXP B.V. 2013. All rights reserved.

7. References

[1] **Programming Android**
Z. Mednieks, L. Dornin, G. B. Meike and M. Nakamura | 2011

[2] www.nxp.com/redirect/cdn.oreillystatic.com/oreilly/booksamplers/9781449389697_sampler.pdf

[3] **Near Field Communication | Android Developers**
www.nxp.com/redirect/developer.android.com/guide/topics/connectivity/nfc/index.html

[4] **RC663 Blueboard documentation**
http://www.nxp.com/demoboard/CLEV663B.html#documentation

[5] **Android SDK download**
www.nxp.com/redirect/developer.android.com/sdk/index.htmlhttp://developer.android.com/sdk/index.html

[6] **Android Developer App Components**
www.nxp.com/redirect/developer.android.com/guide/components/index.htmlhttp://developer.android.com/guide/components/index.html

[7] **Open Handset Alliance**
www.nxp.com/redirect/openhandsetalliance.com/http://www.openhandsetalliance.com/

[8] **Android Developer Activities**
www.nxp.com/redirect/developer.android.com/guide/components/activities.html http://developer.android.com/guide/components/activities.html

[9] **Android Developer Running Your App**
www.nxp.com/redirect/developer.android.com/training/basics/firstapp/running-app.htmlhttp://developer.android.com/training/basics/firstapp/running-app.html

[10] **Android Developer OEM USB Drivers**
www.nxp.com/redirect/developer.android.com/tools/extras/oem-usb.htmlhttp://developer.android.com/tools/extras/oem-usb.html

[11] **Android Developer Building Your First App**
www.nxp.com/redirect/developer.android.com/training/basics/firstapp/index.html http://developer.android.com/training/basics/firstapp/index.html

8. Legal information

8.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, 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.

8.2 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.

8.1 Licenses

Purchase of NXP ICs with ISO/IEC 14443 type B functionality

This NXP Semiconductors IC is ISO/IEC 14443 Type B software enabled and is licensed under Innovatron's Contactless Card patents license for ISO/IEC 14443 B.

The license includes the right to use the IC in systems and/or end-user equipment.

RATP/Innovatron Technology

8.2 Trademarks

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

MIFARE — is a trademark of NXP B.V.

DESFire — is a trademark of NXP B.V.

MIFARE Ultralight — is a trademark of NXP B.V.

MIFARE Plus — is a trademark of NXP B.V

9. List of figures

Fig 1. Google Galaxy Nexus 3
Fig 2. Activity Lifecycle [7]...6
Fig 3. AndroidManifest.xml 7
Fig 4. USB debugging enabled 8
Fig 5. “Run as” window 9
Fig 6. Android Virtual Device Manager 10
Fig 7. HelloWorld Android application.......................11
Fig 8. NFC Tag flow in Android API level 10.............14
Fig 9. Workflow 16
Fig 10. Graphical Interface..17
Fig 11. Derivate Workflow...22

10. Contents

1. Introduction 3
2. Requirements 3
2.1 Blue board p2p project 3
2.2 Android phone 3
2.3 Android SDK 3
3. About Android 4
3.1 Android API 4
3.2 Android application development basics 4
3.2.1 Application components 5
3.2.2 Activities 5
3.2.3 Manifest file 6
3.2.4 Application Resources 7
3.2.5 Running an application 7
3.2.5.1 On real Android-powered device 8
3.2.5.2 On the Emulator 9
3.2.6 HelloWorld Android application 10
3.3 NFC Android API available 11
3.3.1 android.nfc 11
3.3.2 android.nfc.tech 12
4. Preparing the Manifest file 13
4.1 Permission to use NFC 13
4.2 Android SDK min version 13
4.3 NFC feature 13
4.4 Intent declaration 13
5. Main part of the application 13
5.1 Import Library 13
5.2 Enable Foreground Dispatch 13
5.3 Retrieve NDEF Message’s payload 15
6. Example Project: PicReader 16
6.1 General presentation 16
6.1.1 Workflow 16
6.1.2 Graphical Interface 16
6.2 AndroidManifest.xml 17
6.3 PicReader.java 18
6.4 Derivative: receiving two NDEF messages 22
7. References 24
8. Legal information 25
8.1 Definitions 25
8.2 Disclaimers 25
8.1 Licenses 25
8.2 Trademarks 25
9. List of figures 26
10. Contents 27

Please be aware that important notices concerning this document and the product(s) described herein, have been included in the section 'Legal information'.

© NXP B.V. 2013. **All rights reserved.**

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

Date of release: 19 June 2013 **270210**

Document identifier: AN11367