



Code > Android SDK

Reading NFC Tags With Android

Ralf Wondratschek Last updated May 16, 2013

🕒 13 min | 💬 English



Are you curious about what NFC is and how it can be integrated into your own Android applications? This tutorial will quickly introduce you to the topic before diving in and teaching you how to build a simple NFC reader app!

What is NFC?

NFC is the abbreviation for *Near Field Communication*. It is the international standard for contactless exchange of data. In contrast to a large range of other technologies, such as wireless LAN and Bluetooth, the maximum distance of two devices is 10cm. The development of the standard started in 2002 by NXP Semiconductors and Sony. The [NFC Forum](#), a consortium of over 170 companies and members, which included Mastercard, NXP, Nokia, Samsung, Intel, and Google, has been designing new specifications since 2004.

There are various possibilities for NFC use with mobile devices; for example, paperless tickets, access controls, cashless payments, and car keys. With the help of NFC tags you can control your phone and change settings. Data can be exchanged simply by holding two devices next to each other.

In this tutorial I want to explain how to implement NFC with the Android SDK, which pitfalls exist, and what to keep in mind. We will create an app step by step, which can read the content of NFC tags supporting NDEF.

NFC Technologies

There are a variety of NFC tags that can be read with a smartphone. The spectrum ranges from simple stickers and key rings to complex cards with integrated cryptographic hardware. Tags also differ in their chip technology. The most important is NDEF, which is supported by most tags. In addition, Mifare should be mentioned as it is the most used contactless chip technology worldwide. Some tags can be read and written, while others are read-only or encrypted.

Only the NFC Data Exchange Format (NDEF) is discussed in this tutorial.



Adding NFC Support in an App

We start with a new project and a blank activity. It is important to select a minimum SDK version of level 10, because NFC is only supported after Android 2.3.3. Remember to choose your own package name. I've chosen *net.vrallev.android.nfc.demo*, because vrallev.net is the domain of my website and the other part refers to the topic of this application.

```

1 |         <uses-sdk
2 |             android:minSdkVersion="10"
3 |             android:targetSdkVersion="17" />

```

The default layout generated by Eclipse is almost sufficient for us. I've only added an ID to the TextView and changed the text.

```

1 |         <TextView
2 |             android:id="@+id/textView_explanation"
3 |             android:layout_width="wrap_content"
4 |             android:layout_height="wrap_content"
5 |             android:text="@string/explanation" />

```

To get access to the NFC hardware, you have to apply for permission in the manifest. If the app won't work without NFC, you can specify the condition with the uses-feature tag. If NFC is required, the app can't be installed on devices without it and Google Play will only display your app to users who own a NFC device.

```

1 |         <uses-permission android:name="android.permission.NFC" />
2 |
3 |     <uses-feature
4 |         android:name="android.hardware.nfc"
5 |         android:required="true" />

```

The MainActivity should only consist of the onCreate() method. You can interact with the hardware via the NfcAdapter class. It is important to find out whether the NfcAdapter is null. In this case, the Android device does not support NFC.

```

1 | package net.vrallev.android.nfc.demo;
2 |
3 | import android.app.Activity;
4 | import android.nfc.NfcAdapter;
5 | import android.os.Bundle;
6 | import android.widget.TextView;
7 | import android.widget.Toast;
8 |
9 | /**
10 |  * Activity for reading data from an NDEF Tag.
11 |  *
12 |  * @author Ralf Wondratschek
13 |  *
14 |  */
15 | public class MainActivity extends Activity {
16 |
17 |     public static final String TAG = "NfcDemo";
18 |
19 |     private TextView mTextView;

```

```
20     private NfcAdapter mNfcAdapter;
21
22     @Override
23     protected void onCreate(Bundle savedInstanceState) {
24         super.onCreate(savedInstanceState);
25         setContentView(R.layout.activity_main);
26
27         mTextView = (TextView) findViewById(R.id.textview_explai
28
29         mNfcAdapter = NfcAdapter.getDefaultAdapter(this);
30
31         if (mNfcAdapter == null) {
32             // Stop here, we definitely need NFC
33             Toast.makeText(this, "This device doesn't support
34             finish();
35             return;
36
37         }
38
39         if (!mNfcAdapter.isEnabled()) {
40             mTextView.setText("NFC is disabled.");
41         } else {
42             mTextView.setText(R.string.explanation);
43         }
44
45         handleIntent(getIntent());
46     }
47
48     private void handleIntent(Intent intent) {
49         // TODO: handle Intent
50     }
51 }
```

If we start our app now, we can see the text whether NFC is enabled or disabled.

How to Filter for NFC Tags

We have our sample app and want to receive a notification from the system when we attach an NFC tag to the device. As usual, Android uses its Intent system to deliver tags to the apps. If multiple apps can handle the Intent, the activity chooser gets displayed and the user can decide which app will be opened. Opening URLs or sharing information is handled the same way.

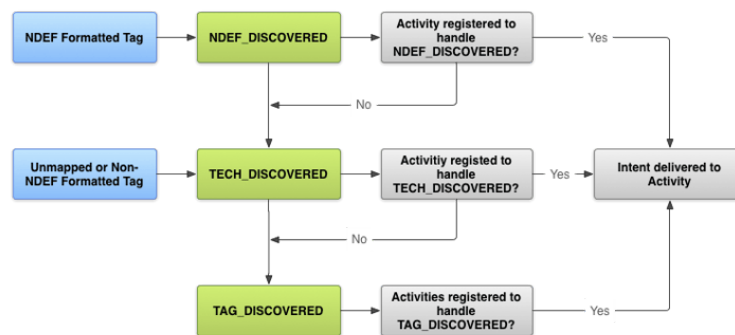
NFC Intent Filter

There are three different filters for tags:

1. ACTION_NDEF_DISCOVERED
2. ACTION_TECH_DISCOVERED
3. ACTION_TAG_DISCOVERED

The list is sorted from the highest to the lowest priority.

Now what happens when a tag is attached to the smartphone? If the system detects a tag with NDEF support, an Intent is triggered. An `ACTION_TECH_DISCOVERED` Intent is triggered if no Activity from any app is registered for the NDEF Intent or if the tag does not support NDEF. If again no app is found for the Intent or the chip technology could not be detected, then a `ACTION_TAG_DISCOVERED` Intent is fired. The following graphic shows the process:



In summary this means that each app needs to filter after the Intent with the highest priority. In our case, this is the NDEF Intent. We implement the `ACTION_TECH_DISCOVERED` Intent first to highlight the difference between priorities.

Tech Discovered Intent

We must specify the technology we are interested in. For this purpose, we create a subfolder called `xml` in the `res` folder. In this folder we create the file `nfc_tech_filter.xml`, in which we specify the technologies.

```

1 | <?xml version="1.0" encoding="utf-8"?>
2 | <resources xmlns:xliff="urn:oasis:names:tc:xliff:document:1.2">
3 |     <tech-list>
4 |         <tech>android.nfc.tech.Ndef</tech>
5 |         <!-- class name -->
6 |     </tech-list>
7 | </resources>

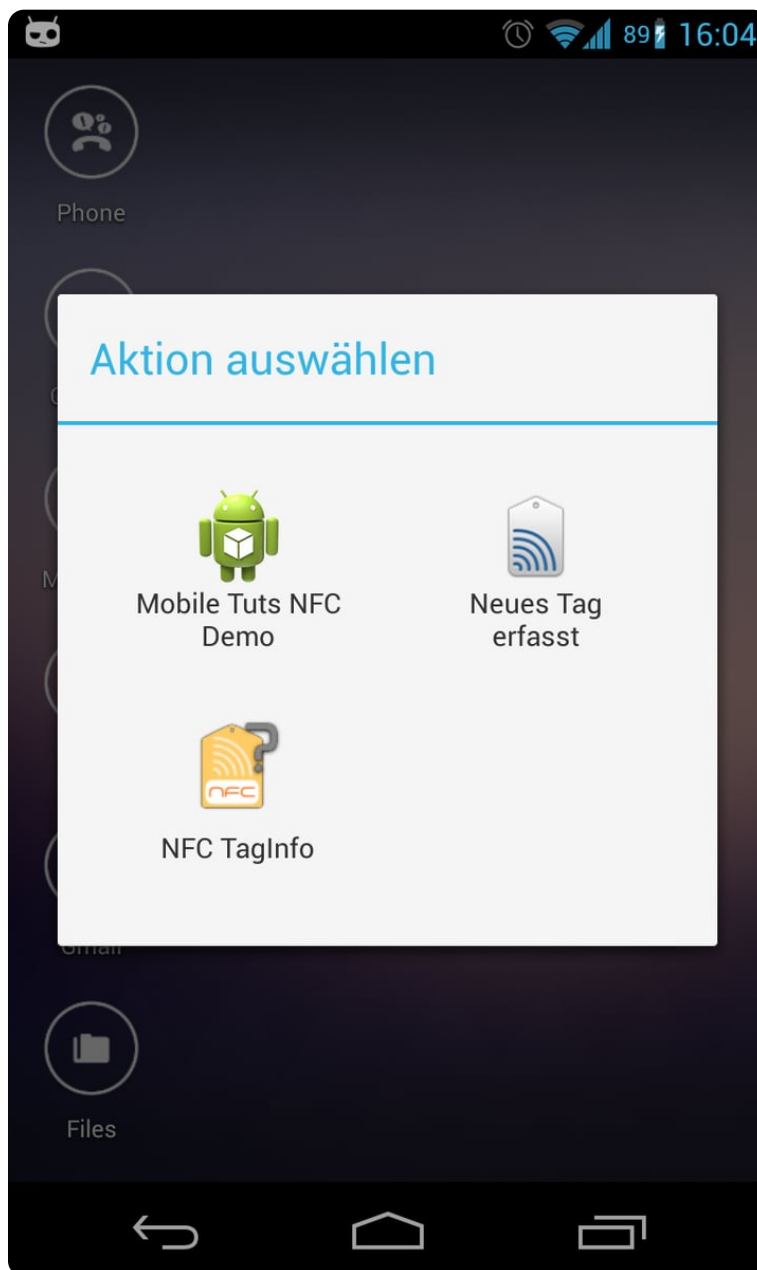
```

```
8 |
9 |     <!--
10 | <resources xmlns:xliff="urn:oasis:names:tc:xliff:document:1.2">
11 | <tech-list>
12 | <tech>android.nfc.tech.IsoDep</tech>
13 | <tech>android.nfc.tech.NfcA</tech>
14 | <tech>android.nfc.tech.NfcB</tech>
15 | <tech>android.nfc.tech.NfcF</tech>
16 | <tech>android.nfc.tech.NfcV</tech>
17 | <tech>android.nfc.tech.Ndef</tech>
18 | <tech>android.nfc.tech.NdefFormatable</tech>
19 | <tech>android.nfc.tech.MifareClassic</tech>
20 | <tech>android.nfc.tech.MifareUltralight</tech>
21 | </tech-list>
22 | </resources>
23 | -->
```

Now we must create an IntentFilter in the manifest, and the app will be started when we attach a tag.

```
1 | <?xml version="1.0" encoding="utf-8"?>
2 | <activity
3 |     android:name="net.vrallev.android.nfc.demo.MainActivity"
4 |     android:label="@string/app_name" >
5 |     <intent-filter>
6 |         <action android:name="android.intent.action.MAIN" />
7 |         <category android:name="android.intent.category.LAUNCHER" />
8 |     </intent-filter>
9 |
10 |     <intent-filter>
11 |         <action android:name="android.nfc.action.TECH_DISCOVERED" />
12 |     </intent-filter>
13 |
14 |     <meta-data
15 |         android:name="android.nfc.action.TECH_DISCOVERED"
16 |         android:resource="@xml/nfc_tech_filter" />
17 | </activity>
```

If no other app is registered for this Intent, our Activity will start immediately. On my device, however, other apps are installed, so the activity chooser gets displayed.



NDEF Discovered Intent

As I mentioned before, the Tech Discovered Intent has the second highest priority. However, since our app will support only NDEF, we can use the NDEF Discovered Intent instead, which has a higher priority. We can delete the technology list again and replace the IntentFilter with the following one.

```
1 | <intent-filter>
2 |   <action android:name="android.nfc.action.NDEF_DISCOVERED" />
3 |
4 |   <category android:name="android.intent.category.DEFAULT" />
```

```
5 |  
6 |         <data android:mimeType="text/plain" />  
7 |     </intent-filter>
```

When we attach the tag now, the app will be started like before. There is a difference for me, however. The activity chooser does not appear and the app starts immediately, because the NDEF Intent has a higher priority and the other apps only registered for the lower priorities. That's exactly what we want.

Foreground Dispatch

Note that one problem remains. When our app is already opened and we attach the tag again, the app is opened a second time instead of delivering the tag directly. This is not our intended behavior. You can bypass the problem by using a Foreground Dispatch.

Instead of the system having distributed the Intent, you can register your Activity to receive the tag directly. This is important for a particular workflow, where it makes no sense to open another app.

I've inserted the explanations at the appropriate places in the code.

```
1 | package net.vrallev.android.nfc.demo;  
2 |  
3 | import android.app.Activity;  
4 | import android.app.PendingIntent;  
5 | import android.content.Intent;  
6 | import android.content.IntentFilter;  
7 | import android.content.IntentFilter.MalformedMimeTypeException;  
8 | import android.nfc.NfcAdapter;  
9 | import android.os.Bundle;  
10 | import android.widget.TextView;  
11 | import android.widget.Toast;  
12 |  
13 | /**  
14 | * Activity for reading data from an NDEF Tag.  
15 | *  
16 | * @author Ralf Wondratschek  
17 | *  
18 | */  
19 | public class MainActivity extends Activity {  
20 |  
21 |     public static final String MIME_TEXT_PLAIN = "text/plain";  
22 |     public static final String TAG = "NfcDemo";  
23 |  
24 |     private TextView mTextView;
```



```

25     private NfcAdapter mNfcAdapter;
26
27     @Override
28     protected void onCreate(Bundle savedInstanceState) {
29         super.onCreate(savedInstanceState);
30         setContentView(R.layout.activity_main);
31
32         mTextView = (TextView) findViewById(R.id.textView_explai
33
34         mNfcAdapter = NfcAdapter.getDefaultAdapter(this);
35
36         if (mNfcAdapter == null) {
37             // Stop here, we definitely need NFC
38             Toast.makeText(this, "This device doesn't support
39             finish();
40             return;
41
42         }
43
44         if (!mNfcAdapter.isEnabled()) {
45             mTextView.setText("NFC is disabled.");
46         } else {
47             mTextView.setText(R.string.explanation);
48         }
49
50         handleIntent(getIntent());
51     }
52
53     @Override
54     protected void onResume() {
55         super.onResume();
56
57         /**
58 * It's important, that the activity is in the foreground (resumed). Otherwise
59 * an IllegalStateException is thrown.
60 */
61         setupForegroundDispatch(this, mNfcAdapter);
62     }
63
64     @Override
65     protected void onPause() {
66         /**
67 * Call this before onPause, otherwise an IllegalArgumentException is thrown as
68 */
69         stopForegroundDispatch(this, mNfcAdapter);
70
71         super.onPause();
72     }
73
74     @Override
75     protected void onNewIntent(Intent intent) {
76         /**
77 * This method gets called, when a new Intent gets associated with the current activity.
78 * Instead of creating a new activity, onNewIntent will be called. For more info
79 * at the documentation.
80 *
81 * In our case this method gets called, when the user attaches a Tag to the device.
82 */
83         handleIntent(intent);
84     }
85

```

```

86         private void handleIntent(Intent intent) {
87             // TODO: handle Intent
88         }
89
90         /**
91         * @param activity The corresponding {@link Activity} requesting the foreground
92         * @param adapter The {@link NfcAdapter} used for the foreground dispatch.
93         */
94         public static void setupForegroundDispatch(final Activity activity,
95             final Intent intent = new Intent(activity.getApplicationContext(),
96             intent.setFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP);
97
98             final PendingIntent pendingIntent = PendingIntent.getActivity(activity,
99
100            IntentFilter[] filters = new IntentFilter[1];
101            String[][] techList = new String[][]{};
102
103            // Notice that this is the same filter as in our manifest
104            filters[0] = new IntentFilter();
105            filters[0].addAction(NfcAdapter.ACTION_NDEF_DISCOVERED);
106            filters[0].addCategory(Intent.CATEGORY_DEFAULT);
107            try {
108                filters[0].addDataType(MIME_TEXT_PLAIN);
109            } catch (MalformedMimeTypeException e) {
110                throw new RuntimeException("Check your mime type.");
111            }
112
113            adapter.enableForegroundDispatch(activity, pendingIntent, filters, techList);
114        }
115
116        /**
117        * @param activity The corresponding {@link BaseActivity} requesting to stop the foreground dispatch.
118        * @param adapter The {@link NfcAdapter} used for the foreground dispatch.
119        */
120        public static void stopForegroundDispatch(final Activity activity,
121            final NfcAdapter adapter) {
122            adapter.disableForegroundDispatch(activity);
123        }

```

Now, when you attach a tag and our app is already opened, `onNewIntent` is called and no new Activity is created.

Reading Data From an NDEF Tag

The last step is to read the data from the tag. The explanations are inserted at the appropriate places in the code once again. The `NdefReaderTask` is a private inner class.

```

1     package net.vrallev.android.nfc.demo;
2
3     import java.io.UnsupportedEncodingException;
4     import java.util.Arrays;

```

```

5
6     import android.app.Activity;
7     import android.app.PendingIntent;
8     import android.content.Intent;
9     import android.content.IntentFilter;
10    import android.content.IntentFilter.MalformedMimeTypeException;
11    import android.nfc.NdefMessage;
12    import android.nfc.NdefRecord;
13    import android.nfc.NfcAdapter;
14    import android.nfc.Tag;
15    import android.nfc.tech.Ndef;
16    import android.os.AsyncTask;
17    import android.os.Bundle;
18    import android.util.Log;
19    import android.widget.TextView;
20    import android.widget.Toast;
21
22    /*
23    * ... other code parts
24    */
25
26    private void handleIntent(Intent intent) {
27        String action = intent.getAction();
28        if (NfcAdapter.ACTION_NDEF_DISCOVERED.equals(action)) {
29
30            String type = intent.getType();
31            if (MIME_TEXT_PLAIN.equals(type)) {
32
33                Tag tag = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
34                new NdefReaderTask().execute(tag);
35
36            } else {
37                Log.d(TAG, "Wrong mime type: " + type);
38            }
39        } else if (NfcAdapter.ACTION_TECH_DISCOVERED.equals(action)) {
40
41            // In case we would still use the Tech Discovered Intent
42            Tag tag = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
43            String[] techList = tag.getTechList();
44            String searchedTech = Ndef.class.getName();
45
46            for (String tech : techList) {
47                if (searchedTech.equals(tech)) {
48                    new NdefReaderTask().execute(tag);
49                    break;
50                }
51            }
52        }
53    }
54
55    /**
56    * Background task for reading the data. Do not block the UI thread while reading
57    *
58    * @author Ralf Wondratschek
59    *
60    */
61    private class NdefReaderTask extends AsyncTask<Tag, Void, String> {
62
63        @Override
64        protected String doInBackground(Tag... params) {

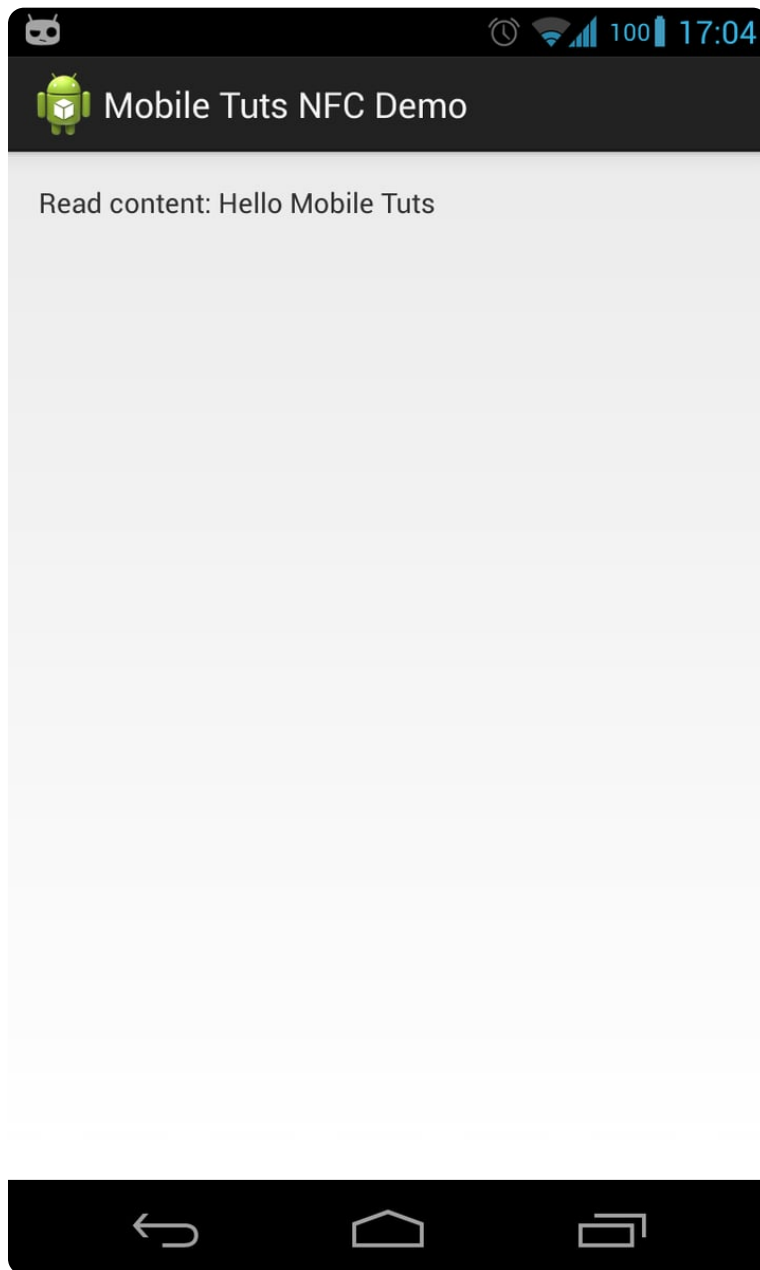
```

```

11     Tag tag = params[0];
12
13     Ndef ndef = Ndef.get(tag);
14     if (ndef == null) {
15         // NDEF is not supported by this Tag.
16         return null;
17     }
18
19     NdefMessage ndefMessage = ndef.getCachedNdefMessage();
20
21     NdefRecord[] records = ndefMessage.getRecords();
22     for (NdefRecord ndefRecord : records) {
23         if (ndefRecord.getTnf() == NdefRecord.TNF_WELL_KNOWN) {
24             try {
25                 return readText(ndefRecord);
26             } catch (UnsupportedEncodingException e) {
27                 Log.e(TAG, "Unsupported Encoding");
28             }
29         }
30     }
31
32     return null;
33 }
34
35     private String readText(NdefRecord record) throws UnsupportedEncodingException {
36         /*
37         * See NFC forum specification for "Text Record Type Definition" at 3.2.1
38         *
39         * http://www.nfc-forum.org/specs/
40         *
41         * bit_7 defines encoding
42         * bit_6 reserved for future use, must be 0
43         * bit_5..0 length of IANA language code
44         */
45
46         byte[] payload = record.getPayload();
47
48         // Get the Text Encoding
49         String textEncoding = ((payload[0] & 128) == 0) ? "UTF-8" : "UTF-16";
50
51         // Get the Language Code
52         int languageCodeLength = payload[0] & 0063;
53
54         // String languageCode = new String(payload, 1, languageCodeLength, textEncoding);
55         // e.g. "en"
56
57         // Get the Text
58         return new String(payload, languageCodeLength + 1, payload.length - languageCodeLength - 1, textEncoding);
59     }
60
61     @Override
62     protected void onPostExecute(String result) {
63         if (result != null) {
64             mTextView.setText("Read content: " + result);
65         }
66     }
67 }

```

The app now successfully reads the content.



Useful Apps

To check whether data is read and written properly, I personally like to use following apps:

- [NFC TagInfo](#) by NFC Research Lab for reading data
- [TagInfo](#) by NXP SEMICONDUCTORS for reading data
- [TagWriter](#) by NXP SEMICONDUCTORS for writing data

Conclusion

In this tutorial I have shown you how the data from a NDEF tag can be extracted. You could expand the example to other mime types and chip technologies; a feature to write data would be useful as well. The first step to work with NFC was made. However, the Android SDK offers much more possibilities, such as an easy exchange of data (called Android Beam).

If you want to take your Android development further, check out the huge range of useful [Android app templates](#) on Envato Market.

About the Author

Ralf Wondratschek is a computer science student from Germany. In addition to his studies, Ralf works as a freelancer in the field of mobile computing. In the last few years he has worked with Java, XML, HTML, JSP, JSF, Eclipse, Google App Engine, and of course Android. He has published two Android apps to date [which can be found here](#).

You can find out more about the author's work on his homepage [vrallev.net](#).

Sources

<http://www.nfc-forum.org/home/n-mark.jpg>

<http://commons.wikimedia.org/wiki/File%3A%C3%9Cberlagert.jpg>

http://developer.android.com/images/nfc_tag_dispatch.png

Android SDK

Mobile Development

Did you find this post useful?



Want a weekly email summary?

Subscribe below and we'll send you a weekly email summary of all new Code tutorials. Never miss out on learning about the next big thing.

Sign up



Ralf Wondratschek

Ralf Wondratschek is a computer science student from Germany. Next to his study, Ralf works for a German company as a software developer. In the last few years, he got in touch with Java, XML, HTML, JSP, JSF, Eclipse, Google App Engine, and the Android SDK. In his free time, he has [written and published two Android apps](#). You can find out more about the author's work on [his homepage](#).

Download Attachment



QUICK LINKS - Explore popular categories

ENVATO TUTORIALS+



HELP



 **tuts+** 
31,140 Tutorials 1,281 Courses 47,342 Translations
Certified Corporation

[Envato](#) [Envato Elements](#) [Envato Market](#) [Placeit by Envato](#) [All products](#) [Careers](#) [Sitemap](#)

© 2023 Envato Pty Ltd. Trademarks and brands are the property of their respective owners.

