About This Article

This article explains about Near Field Communications (NFC) in android, which is a standards-based, short range wireless connectivity technology that enables simple and safe two way interactions between electronic devices primarily aimed at usage in mobile devices. It enables the exchange of data between devices over about a 10 centimeter (around 4 inches) distance.

Scope:

This article guides on NFC in android mobiles. It assumes that the user has already installed Android and necessary tools to develop application. It also assumes that the user to be familiar with basic knowledge of Android and Java.

Introduction

Near-field Communication or NFC is a standard defined by the NFC Forum, a global consortium of hardware, software/application, credit card companies, banking, network-providers, and others who are interested in the advancement and standardization of this promising technology.
It is a set of short-range wireless technologies, typically requiring a distance of 4cm or less. NFC operates at 13.56mhz, and at rates ranging from 106 kbit/s to 848 kbit/s. NFC communication always involves an initiator and a target. The initiator actively generates an RF field that can power a passive target. This enables NFC targets to take very simple form factors such as tags, stickers or cards that do not require power. NFC peer-to-peer communication is also possible, where both devices are powered. Compared to other wireless technologies such as Bluetooth or WiFi, NFC provides much lower bandwidth and range, but enables low-cost, un- powered targets and does not require discovery or pairing. Interactions can be initiated with just a tap.

NFC Specifications

Below are the some of the Network Related package in Android SDK

- **NFC Data Exchange Format (NDEF)** Common data format for devices and tags
- **NFC Record Type Definition (RTD)** Standard record types used in messages between devices/tags
- **Smart Poster RTD** For posters with tags with text, audio or other data
- **java.nio**Contains classes that represent buffers of specific data types. Handy for network communications between two Java language-based end points.
- **Text RTD** For records containing plain text
- **Uniform Resource Identifier (URI) RTD** For records that refer to an Internet resource
- **Generic Control RTD** Way to request an action
- **NFC Tag Types 1-4 Operation** Defines R/W operation for NFC tags

NFC Architecture on mobile

NFC Forum defines three communication modes, as shown in figure 1

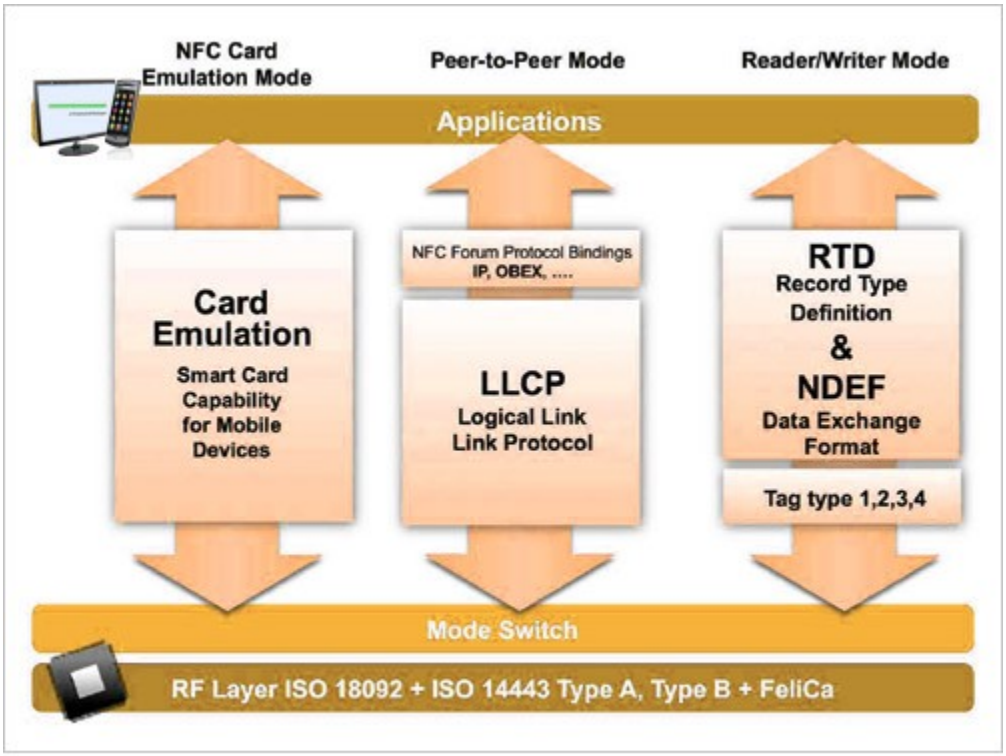

Figure 1: NFC Communication Modes

The three communication modes are defined as below

- **Peer-to-Peer** This mode is defined for device-to-device link-level communication.
- **Read/Write** This mode allows applications for the transmission of NFC Forum-defined messages. Note that this mode is not secure.
- **NFC Card Emulation** This mode allows the NFC-handset behave as a standard Smartcard. This mode is secure.

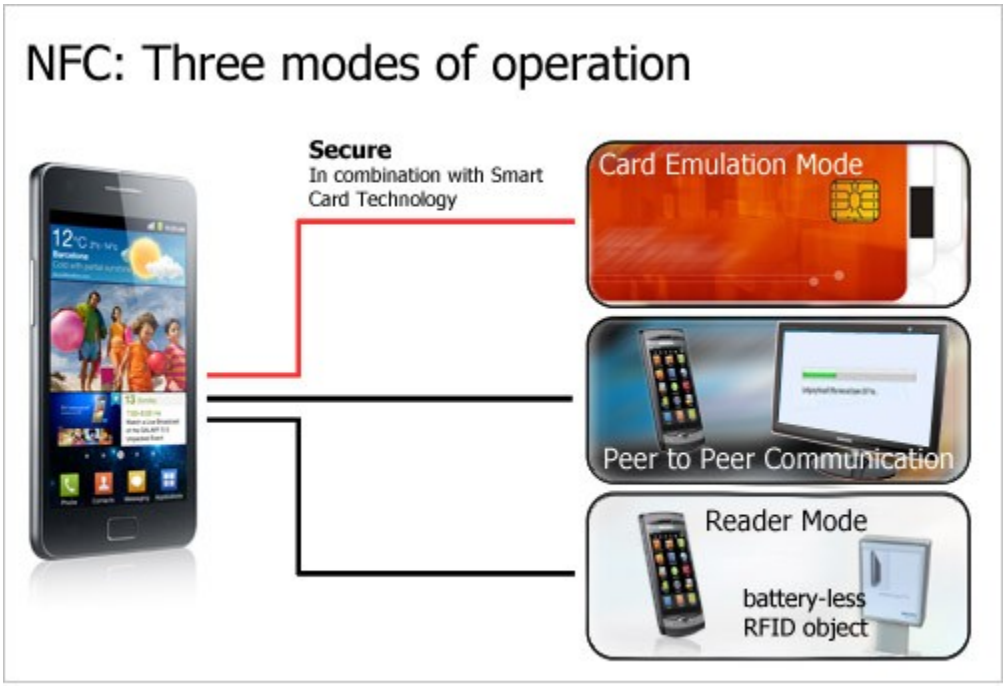

Figure 2: Three modes of operation

NFC Terminology

- **NDEF** NFC Data Exchange Format - standard exchange formats for URI, Smart Posters, other.
- **RTD** Record Type Definition - An NFC-specific record type and type name which may be carried in an NDEF record.
- **NDEF message** Basic message construct defined by this specification. An NDEF message contains one or more NDEF records.
- **NDEF record** Contains a payload described by a type, a length, and an optional identifier.
- **NDEF payload** The application data carried within an NDEF record.

NFC Use-cases

NFC has multiple use cases

- Get information by touching smart posters
- Use your NFC phone as an event ticket
- Set up your wireless home office with a touch
- Print your camera by holding it close to printer
- Share business card with a touch
- Pay for goods with a tap of your NFC phone
- Get on the bus by waving your NFC phone

Figure 3 below shows different use-cases of NFC

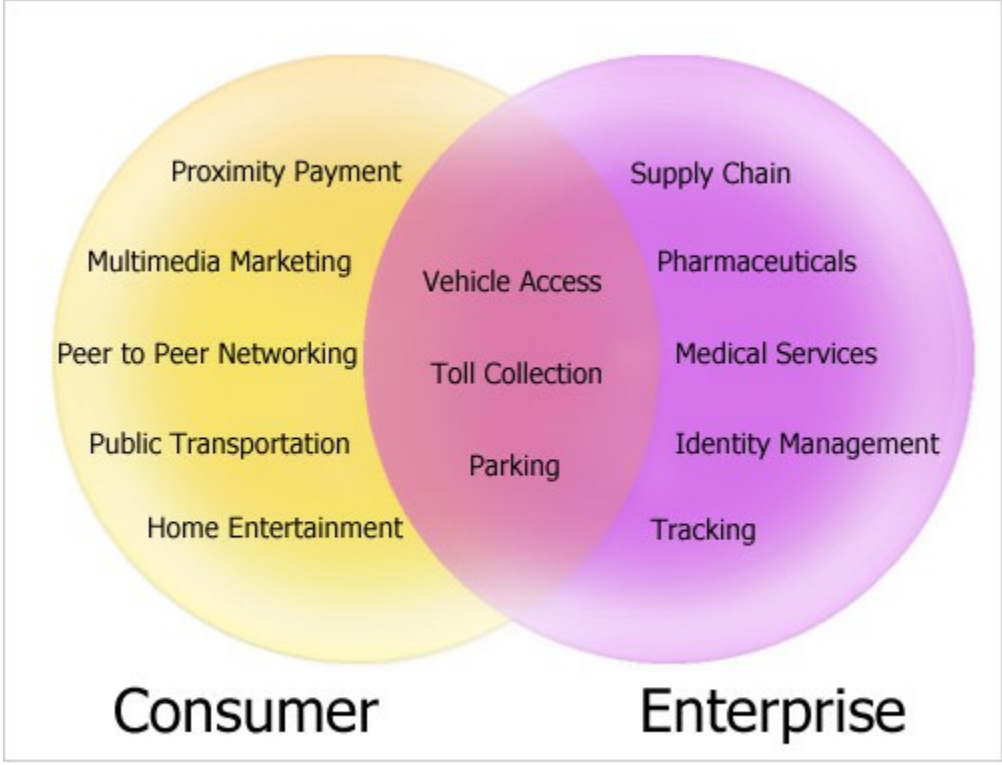

Figure 3: Use-Cases of NFC

NFC API Overview

The **android.nfc** package contains the high-level classes to interact with the local device's NFC adapter, to represent discovered tags, and to use the NDEF data format.
Table 1: android.nfc package

Class	Description
NfcManager	A high-level manager class that enumerates the NFC adapters on the Android device. Since most Android devices only have one NFC adapter, you can just use the static helper getDefaultAdapter(Context) for most situations.
NfcAdapter	Represents the local NFC adapter. Defines the intent's used to request tag dispatch to your activity, and provides methods to register for foreground tag dispatch and foreground NDEF push. Foreground NDEF push is the only peer-to- peer support that is currently provided in Android.
NdefMessage and NdefRecord	NDEF is an NFC Forum defined data structure, designed to efficiently store data on NFC tags, such as text, URL's, and other MIME types. A NdefMessage acts as a container for the data that you want to transmit or read. One NdefMessage object contains zero or more NdefRecords. Each NDEF record has a type such as text, URL, smart poster, or any MIME data. The type of the first NDEF record in the NDEF message is used to dispatch a tag to an activity on Android.
Tag	Represents a passive NFC target. These can come in many form factors such as a tag, card, key fob, or even a phone doing card emulation. When a tag is discovered, a Tag object is created and wrapped inside an Intent. The NFC dispatch system sends the intent to a compatible activity using startActivity(). You can use the getTechList() method to determine the technologies supported by this tag and create the corresponding TagTechnology object with one of classes provided by android.nfc.tech

The android.nfc.tech package contains classes to query properties and perform I/O operations on a tag. The classes are divided to represent different NFC technologies that can be available on a Tag:

Table 2: android.nfc.tech

Class	Description
TagTechnology	The interface that all tag technology classes must implement.
NfcA	Provides access to NFC-A (ISO 14443-3A) properties and I/O operations.
NfcB	Provides access to NFC-B (ISO 14443-3B) properties and I/O operations.
NfcF	Provides access to NFC-F (JIS 6319-4) properties and I/O operations.
NfcV	Provides access to NFC-V (ISO 15693) properties and I/O operations.
IsoDep	Provides access to ISO-DEP (ISO 14443-4) properties and I/O operations.
Ndef	Provides access to NDEF data and operations on NFC tags that have been formatted as NDEF.
NdefFormatable	Provides format operations for tags that may be NDEF formattable.
MifareClassic	Provides access to MIFARE Classic properties and I/O operations, if this Android device supports MIFARE
MifareUltralight	Provides access to MIFARE Ultralight properties and I/O operations, if this Android device supports MIFARE

NFC Application Design

Android Manifest File

Before you can access a device's NFC hardware and properly handle NFC intents, declare these items in your AndroidManifest.xml file.

- The NFC <uses-permission> element to access the NFC hardware:

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

- The uses-feature element so that your application can show up in the Android Market for devices that have NFC hardware:

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

- The NFC intent filter to tell the Android system your Activity can handle NFC data. Specify one or more of these three intent filters:

```
<intent-filter>
 <action android:name="android.nfc.action.NDEF_DISCOVERED"/>
 <data android:mimeType="mime/type" />
</intent-filter>

<intent-filter>
  <action android:name="android.nfc.action.TECH_DISCOVERED"/>
  <meta-data android:name="android.nfc.action.TECH_DISCOVERED"
                   android:resource="@xml/nfc_tech_filter.xml" />
</intent-filter>

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

Reading an NFC Tag

When a device comes in proximity to an NFC tag, the appropriate intent is started on the device, notifying interested applications that a NFC tag was scanned. By previously declaring the appropriate intent filter in your AndroidManifest.xml file or using foreground dispatching, your application can request to handle the intent.

```
NdefMessage[] getNdefMessages(Intent intent) {
```

```
  // Parse the intent
  NdefMessage[] msgs = null;
  String action = intent.getAction();
  if (NfcAdapter.ACTION_TAG_DISCOVERED.equals(action)) {
     Parcelable[] rawMsgs =
                    intent.getParcelableArrayExtra(NfcAdapter.EXTRA_NDEF_MESSAGES);
            if (rawMsgs != null) {
               msgs = new NdefMessage[rawMsgs.length];
                    for (int i = 0; i < rawMsgs.length; i++) {
                      msgs[i] = (NdefMessage) rawMsgs[i];
                    }
        }else {
          // Unknown tag type
          byte[] empty = new byte[] {};
                  NdefRecord record = new NdefRecord(NdefRecord.TNF_UNKNOWN,
                                empty, empty, empty);
                  NdefMessage msg = new NdefMessage(new NdefRecord[] {record});
                  msgs = new NdefMessage[] {msg};
                }
        }else {
             Log.e(TAG, "Unknown intent " + intent);
                 finish();
        }
        return msgs;
}
```

Writing to an NDEF Tag

Writing to an NFC tag involves constructing your NDEF message in bytes and using the appropriate tag technology for the tag that you are writing to. The following code sample shows you how to write a simple text message to a NdefFormatable tag:

```
 NdefFormatable tag = NdefFormatable.get(t);
 Locale locale = Locale.US;
 final byte[] langBytes = locale.getLanguage().getBytes(Charsets.US_ASCII);
 String text = "Tag, you're it!";
 final byte[] textBytes = text.getBytes(Charsets.UTF_8);
 final int utfBit = 0;
 final char status = (char) (utfBit + langBytes.length);
 final byte[] data = Bytes.concat(new byte[] {(byte) status}, langBytes, textBytes);
 NdefRecord record = NdefRecord(NdefRecord.TNF_WELL_KNOWN, NdefRecord.RTD_TEXT, new byte[0],
data);
 try {
  NdefRecord[] records = {text};
```

```
 NdefMessage message = new NdefMessage(records);
 tag.connect();
 tag.format(message);
}catch (Exception e){
   //do error handling
}
```

ref:http://developer.samsung.com/android/technical-docs/Near-Field-Communications-in-Android