

Introduction à l'utilisation du NFC dans une application Android

Par Sylvain Berfini  

Date de publication : 22 mars 2011

Cet article a pour objectif de vous apprendre à manipuler le capteur NFC présent dans certains terminaux Android en vous expliquant comment envoyer et recevoir des données.

Commentez

I - Avant-propos.....	3
II - Prérequis.....	3
III - Recevoir et lire un Tag.....	3
IV - Envoyer des données.....	6
V - Simuler la réception d'un Tag.....	7
VI - Remerciements.....	8

I - Avant-propos

Le NFC (pour Near Field Communication soit communication en champ proche) est une technologie qui n'est pas nouvelle mais qui commence à apparaître dans nos Smartphones, et il y a fort à parier que son apparition va se généraliser dans les années à venir. Ce capteur permet l'envoi et la lecture de données sous forme de "Tags" entre deux terminaux mais également entre un terminal et une puce (que je nommerai dans la suite "tag"), et ce sur de très courtes distances (de l'ordre de quelques centimètres). Il devient alors plus aisé pour les utilisateurs d'échanger des données, simplement en rapprochant leur téléphone d'une borne par exemple. Les applications pour ce capteur sont donc multiples (dont le paiement) et il faut que vous sachiez que beaucoup de chercheurs se penchent dessus depuis quelques années.

Depuis quelques mois le nouveau Androphone de Google, créé en coopération avec Samsung, embarque un capteur de ce type. Le SDK (Software Development Kit) qui doit vous être familier si vous avez déjà développé ne serait-ce qu'un "hello world" s'est vu doter de deux nouvelles mises à jour (API 9 et 10 pour Android 2.3 Gingerbread) entre autres pour permettre de manipuler ce capteur. Bien que le capteur permette (au niveau physique) les trois modes du NFC (à savoir la lecture, l'envoi et l'émulation), actuellement seuls les deux premiers sont supportés par le SDK.

Nous allons donc voir dans ce tutoriel comment se servir de ces deux modes pour envoyer et recevoir des données d'un terminal à un autre. Je ne traiterai pas ici la lecture / écriture depuis / vers un tag physique (les puces), bien que la méthode soit assez proche. Cela fera l'objet d'un prochain tutoriel. Dans la suite de l'article, j'utiliserai le mot Tag pour désigner la structure de données transitant par le capteur NFC, et non la puce physique lisible par le capteur.

II - Prérequis

Pour une bonne compréhension de ce tutoriel vous devrez disposer des connaissances de base en Java et en développement d'applications Android. Les notions d'Activity, Intent, Context et Manifest doivent vous être familières. Je ne me focaliserai donc pas sur la façon dont on code une application, ni sur la structure du projet, mais bien sur le code directement.

Tous les exemples qui suivent ont été codés pour la version 10 de l'API. Si vous ne l'avez pas encore installée, je vous invite à le faire avant de continuer.

III - Recevoir et lire un Tag

Avant de commencer à programmer notre Activity gérer la lecture d'un Tag, il faut que le système sache que notre classe permet de le faire. Pour cela, on va déclarer dans le Manifest de notre application que notre classe (ici RActivity) gère la réception des Tags NFC.

Manifest.xml

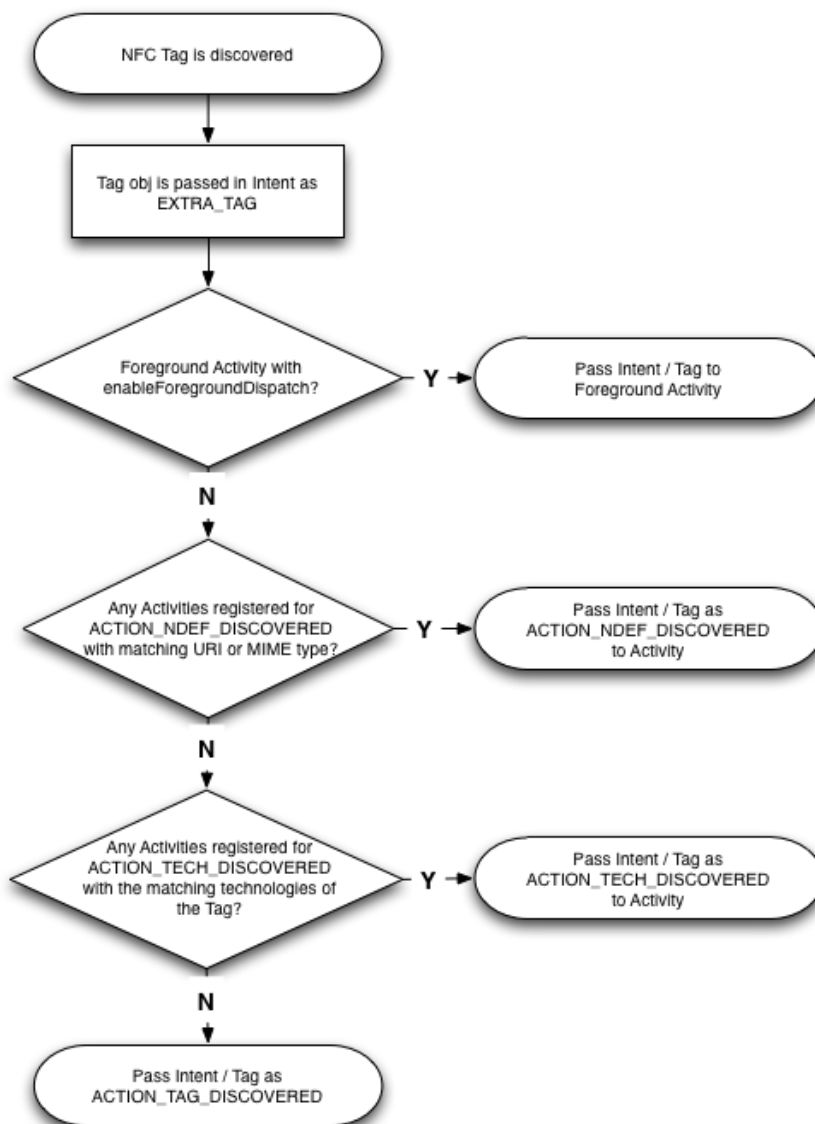
```
<activity android:name=".RActivity" android:label="RActivity">
    <intent-filter>
        <action android:name="android.nfc.action.TAG_DISCOVERED"/>
        <category android:name="android.intent.category.DEFAULT"/>
    </intent-filter>
</activity>
```

Vous remarquerez les deux lignes atypiques dans la balise intent-filter qui servent à "inscrire" auprès du système notre Activity pour que lorsqu'un TAG arrive, le système propose (possiblement entre autres) la nôtre pour gérer l'évènement.



Si votre application est la seule à être inscrite, le système la lancera automatiquement.

Notez qu'il est possible de s'inscrire à différents niveaux d'abstraction. Observez le schéma suivant qui montre comment le système détermine quelle Activity va être utilisée :



Celui-ci explique que lors de la découverte d'un TAG, le système va regarder si l'application au premier plan peut gérer l'événement. Sinon, il regarde s'il existe une (ou plusieurs) application(s) sur le système s'étant enregistrée(s) pour gérer les événements de type "NDEF_DISCOVERED".

Pour que l'application au premier plan puisse gérer directement la découverte d'un TAG, il est nécessaire de modifier quelque peu la méthode onResume() de l'Activity.

OnResume()

```

PendingIntent pendingIntent =
    PendingIntent.getActivity(this, 0, new Intent(this, this.getClass()).addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP),
        mAdapter.enableForegroundDispatch(this, pendingIntent, null, null);
  
```

Remarquez ici que je passe deux fois null en paramètre pour intercepter tous les TAGs, mais vous pouvez filtrer de la même manière que dans le Manifest les TAGs à gérer. Le NDEF étant le nom donné au message contenu dans les TAGs, il est donc à un niveau d'abstraction plus bas. Si tel est le cas, alors il lance l'application ou demande à l'utilisateur si plusieurs demandent la main. Sinon c'est TECH (pour technologie) découvert qui prendra la main pour le choix de l'Activity à lancer, sur le même principe que pour NDEF. Pour finir, le système en cherche une proposant ses services lorsqu'un TAG est découvert. Là encore, même chose en cas d'application(s) trouvée(s).

i D'une manière générale, préférez l'utilisation du TAG_DISCOVERED au NDEF_DISCOVERED, la majorité des applications du Market utilisant TAG_DISCOVERED (à

moins que vous ne teniez absolument à avoir la priorité). Notez également qu'il est possible d'affecter un filtre à ces précédentes règles pour affiner le type de TECH, TAG ou de NDEF qui a été découvert. Ici nous nous contenterons de DEFAULT, qui regroupe tous les TAGs.

i L'utilisation de TECH_DISCOVERED est très utile si vous créez une application compatible uniquement avec un sous-ensemble des technologies NFC (comme les chips MiFare Classic).

Une fois ceci fait, attaquons-nous à la partie Java du code. Créez dans votre classe une méthode (que je nommerai `resoudreIntent`) prenant en paramètre un Intent. Si vous êtes habitué au développement Android cela ne devrait pas vous choquer. Que va faire cette méthode ? Tout simplement vérifier la présence d'un TAG dans l'Intent courant et le cas échéant le lire.

i Le système se charge de lui-même de passer à l'Intent le TAG.

! Si votre Activity peut être déjà lancée par votre programme, vous devrez surcharger la méthode `OnNewIntent(Intent intent)` et y effectuer de la même manière un appel à votre `resoudreIntent`.

Vérifions maintenant dans `resoudreIntent` la présence d'un Tag dans l'Intent passé en paramètre :

Vérification de la présence d'un Tag

```
String action = intent.getAction();
if (NfcAdapter.ACTION_TAG_DISCOVERED.equals(action)) {
```

! Si vous utilisez dans le Manifest `NDEF_DISCOVERED` ou `TECH_DISCOVERED`, vous devrez utiliser respectivement `ACTION_NDEF_DISCOVERED` ou `ACTION_TECH_DISCOVERED`.

Si votre condition est fausse, à vous de faire ce que vous désirez (ce ne sera le cas que si l'utilisateur peut lancer l'Activity « à la main »). Nous allons nous intéresser à la lecture de ce TAG s'il est présent. Nous allons récupérer trois objets : un de type `Tag` qui contiendra l'entête du TAG, un de type `byte[]` qui sera l'id du tag et un de type `NdefMessage[]` qui contiendra le message en lui-même. Je vous rappelle que nous ne nous intéressons pas ici à la lecture d'une puce Tag, mais juste à un message transitant par le NFC. C'est pourquoi nous n'allons récupérer que le message lui-même, et pas l'entête Tag.

Récupération du contenu du Tag

```
Parcelable[] rawMsgs = intent.getParcelableArrayExtra(NfcAdapter.EXTRA_NDEF_MESSAGES);
NdefMessage[] messages;
if (rawMsgs != null) {
    msgs = new NdefMessage[rawMsgs.length];
    for (int i = 0; i < rawMsgs.length; i++) {
        messages[i] = (NdefMessage) rawMsgs[i];
    }
}
```


! Il faut savoir qu'un tag peut contenir plusieurs messages, donc pensez bien à vérifier le contenu de chacun.


i Un `NdefMessage` est un ensemble de `NdefRecord`, et qu'un `NdefRecord` est caractérisé par quatre valeurs : son TNF (Type Name Format), son type, son id et ses données (appelées Payload).


Voici donc comment récupérer depuis un `NdefMessage` tous ces `NdefRecord`, puis pour chacun ses caractéristiques.

Extraction des données d'un record

```
NdefRecord record = messages[i].getRecords()[i];
byte[] id = record.getId();
short tnf = record.getTnf();
byte[] type = record.getType();
String message = getTextData(record.getPayload());
```

 Les TNFs et les Types disponibles sont énumérés dans la classe NdefRecord.

 Les TNFs commencent tous par TNF_.

 Les Types commencent tous par RTD_.

La fonction getTextData est générique pour les TAGs de type RTD_TEXT (texte brut). Elle est définie comme suit.

Décodage du payload de type Texte

```
String getTextData(byte[] payload) {
    String texteCode = ((payload[0] & 0xFF) == 0) ? "UTF-8" : "UTF-16";
    int languageCodeTaille = payload[0] & 0xFF;
    return new String(payload, languageCodeTaille + 1, payload.length - languageCodeTaille - 1, texteCode);
}
```

 Si vous lisez des TAGs de type URI ou SMART_POSTER, la méthode de récupération des données diffère quelque peu. Vous pouvez visiter cette page qui donne des exemples de décodage en fonction du type (en anglais) : <http://programming-android.labs.oreilly.com/ch14.html>

Et voilà ! Maintenant vous êtes en mesure de récupérer sous forme d'une chaîne de caractères le contenu d'un message arrivé par le capteur NFC de votre terminal.

IV - Envoyer des données

Voyons maintenant comment utiliser le capteur NFC pour envoyer des données à un autre terminal.

Contrairement à la réception où nous récupérons directement les données depuis l'Intent, pour envoyer un message nous allons avoir besoin de nous servir directement du capteur NFC. Et comme pour toute ressource sur Android, il faut avoir la permission appropriée. Rajoutons donc à notre Manifest la demande de permission.

Manifest.xml

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

Pour être certain que le téléphone sur lequel votre application va être exécutée dispose d'un capteur NFC, vous pouvez rajouter cette ligne au Manifest pour interdire l'installation de votre application sur un téléphone non compatible.

Manifest.xml

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

Maintenant que nous avons le droit de manipuler le capteur NFC, récupérons l'objet le représentant. Ce code devant être placé dans votre Activity, this fait ici référence au Context.

Récupération du capteur NFC

```
NfcAdapter nfcAdapter = NfcAdapter.getDefaultAdapter(this);
```


Comme nous l'avons vu précédemment, la partie « données » d'un TAG est un NdefMessage, qui est constitué d'un ou plusieurs NdefRecord. Les données transitant sous forme de bytes, il va falloir encoder notre texte, tout en encodant le Charset pour que l'application de lecture puisse savoir comment décoder les données. Voici les fonctions qui vont encoder un message (un String) en un NdefRecord, puis encapsuler le NdefRecord dans un NdefMessage.

Création d'un record à partir d'un String


```
NdefRecord creerRecord(String message)
{
    byte[] langBytes = Locale.ENGLISH.getLanguage().getBytes(Charset.forName("US-ASCII"));
    byte[] textBytes = message.getBytes(Charset.forName("UTF-8"));
    char status = (char) (langBytes.length);
    byte[] data = new byte[1 + langBytes.length + textBytes.length];
    data[0] = (byte) status;
    System.arraycopy(langBytes, 0, data, 1, langBytes.length);
    System.arraycopy(textBytes, 0, data, 1 + langBytes.length, textBytes.length);
    return new NdefRecord(NdefRecord.TNF_WELL_KNOWN, NdefRecord.RTD_TEXT, new byte[0], data);
}
```


Création d'un message à partir d'un record

```
NdefMessage creerMessage(NdefRecord record)
{
    NdefRecord[] records = new NdefRecord[1];
    records[0] = record;
    NdefMessage message = new NdefMessage(records);
    return message;
}
```

 *Puisqu'ici nous encodons du texte simple, j'utilise comme TNF la constante TNF_WELL_KNOWN puisqu'on connaît le type de données que l'on fait passer, et RTD_TEXT pour le type. Ici je me moque de l'id, mais comme la spécification interdit de passer null comme id, je crée un new byte[0]. Pour finir je passe mes données encodées en bytes.*

Maintenant que nous pouvons créer nos messages à faire passer, occupons-nous de les envoyer. Nous allons pour cela faire appel à la méthode `enableForegroundNdefPush(Context, NdefMessage)` de la classe `NfcAdapter`.

 *Il est obligatoire d'appeler cette méthode dans le `OnResume()` de votre Activity.*

 *Il faut penser également à stopper la propagation du message lorsque votre Activity n'est plus au premier plan.*

Activer et stopper l'envoi du message

```
protected void onPause()
{
    super.onPause();
    nfcAdapter.disableForegroundNdefPush(this);
}


protected void onResume()
{
    super.onResume();
    NdefRecord record = creerRecord("Exemple de message à envoyer");
    NdefMessage message = creerMessage(record);
    nfcAdapter.enableForegroundNdefPush(this, message);
}
```

Et voilà. Tant que votre Activity sera au premier plan, elle diffusera en boucle son message via le capteur NFC. Dès lors que vous changerez d'Activity, la diffusion du message s'arrêtera.

V - Simuler la réception d'un Tag


Maintenant que l'on peut envoyer et recevoir des données par NFC, il peut être intéressant de simuler la réception d'un Tag, que ce soit à des fins de débogage (il n'est pas toujours possible d'avoir sous la main deux terminaux équipés), ou parce que cela présente un avantage quelconque pour votre application.

Lors de la réception d'un Tag, on a vu que l'on récupérerait les informations à partir de l'Intent. Et bien pour simuler la réception d'un Tag, nous allons tout simplement créer un Intent qui contiendra notre "faux" Tag et l'envoyer au système, qui se chargera de le renvoyer à la bonne application. Pour cela nous allons réutiliser les deux fonctions précédentes, "creerRecord" et "creerMessage" afin de créer le message qui sera contenu dans le Tag faussement détecté.

 *Rappelez-vous que lorsque l'on lit un Tag, celui-ci est constitué (potentiellement) de plusieurs NdefMessage. Il faut donc penser ici à faire de même.*

Simulation de la réception d'un Tag

```
final Intent intent = new Intent(NfcAdapter.ACTION_TAG_DISCOVERED);
NdefMessage[] messages = new NdefMessage[1];
messages[0] = creerMessage(creerRecord("Simulation d'une réception de Tag"));
intent.putExtra(NfcAdapter.EXTRA_NDEF_MESSAGES, messages);
startActivity(intent);
```

 *Il ne faut pas que votre Activity qui simule la réception d'un Tag soit configurée pour gérer l'arrivée d'un Tag lorsqu'elle est au premier plan (cf. premier tutoriel), sinon rien ne se passera. En effet elle propagera et interceptera en même temps le Tag généré.*

C'est fini. Oui c'est exactement comme appeler une Activity classique. La seule différence réside dans le passage de NfcAdapter.ACTION_TAG_DISCOVERED à l'Intent au lieu de passer la classe de votre projet.

VI - Remerciements

Je voudrais remercier **Claude Leloup** pour sa relecture attentive.