LES GUIDES DE GNU LINUX MAGAZINE / FRANCE

HORS-SÉRIE N°91

France MÉTRO. : 12,90 € — CH : 18,00 CHF — BEL/PORT.CONT : 13,90 € — DOM TOM : 13,90 € — CAN : 18,00 $ CAD

CRÉEZ, PUBLIEZ & MONÉTISEZ VOTRE APPLICATION AVANCÉE ANDROID

INCLUS : Un projet complet de développement d'une application de géocaching !

DÉCOUVREZ les étapes à suivre pour créer un projet structuré

MANIPULEZ les capteurs/périphériques de votre smartphone : accéléromètre, magnétomètre, NFC, GPS, etc.

COMMUNIQUEZ en Bluetooth et par SMS pour partager des caches visibles sur des cartes Google Maps

PUBLIEZ & MONÉTISEZ votre application sur Google Play, rendez-la payante et insérez-y de la publicité

Édité par Les Éditions Diamond

L 15066 - 91 H - F: 12,90 € - RD

www.ed-diamond.com

Retrouvez toutes nos publications

sur **www.ed-diamond.com**

GNU/Linux Magazine Hors-Série
est édité par **Les Éditions Diamond**

10, Place de la Cathédrale - 68000 Colmar - France

Tél. : 03 67 10 00 20 / **Fax :** 03 67 10 00 21

E-mail : cial@ed-diamond.com
lecteurs@gnulinuxmag.com

Service commercial : abo@gnulinuxmag.com

Sites : www.gnulinuxmag.com
www.ed-diamond.com

Directeur de publication : Arnaud Metzler

Chef des rédactions : Denis Bodor

Rédacteur en chef : Tristan Colombo

Responsable service infographie : Kathrin Scali

Réalisation graphique : Thomas Pichon

Remerciements : Frédéric Camps

Responsable publicité : Tél. : 03 67 10 00 27

Service abonnement : Tél. : 03 67 10 00 20

Impression : pva, Druck und Medien-Dienstleistungen GmbH, Landau, Allemagne

Distribution France :
(uniquement pour les dépositaires de presse)

MLP Réassort :
Plate-forme de Saint-Barthélemy-d'Anjou.
Tél. : 02 41 27 53 12
Plate-forme de Saint-Quentin-Fallavier.
Tél. : 04 74 82 63 04

Service des ventes :
Distri-médias : Tél. : 05 34 52 34 01

IMPRIMÉ en Allemagne - PRINTED in Germany

Dépôt légal : A parution

N° ISSN : 0183-0864

Commission Paritaire : K78 976

Périodicité : Bimestrielle

Prix de vente : 12,90 Euros

La rédaction n'est pas responsable des textes, illustrations et photos qui lui sont communiqués par leurs auteurs. La reproduction totale ou partielle des articles publiés dans GNU/Linux Magazine France Hors-série est interdite sans accord écrit de la société Les éditions Diamond. Sauf accord particulier, les manuscrits, photos et dessins adressés à GNU/Linux Magazine France Hors-série, publiés ou non, ne sont ni rendus, ni renvoyés. Les indications de prix et d'adresses figurant dans les pages rédactionnelles sont données à titre d'information, sans aucun but publicitaire. Toutes les marques citées dans ce numéro sont déposées par leur propriétaire respectif. Tous les logos représentés dans le magazine sont la propriété de leur ayant droit respectif.

Les articles non signés contenus dans ce numéro ont été rédigés par les membres de l'équipe rédactionnelle des Éditions Diamond.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

PRÉFACE

La téléphonie mobile connaît un essor absolument incroyable. Qui aurait parié en 2005, au moment du rachat d'une *startup* du nom d'**Android**, que 10 ans après il serait possible d'avoir dans sa poche un smartphone à 6 processeurs ? Les smartphones actuels ont nettement dépassé le concept même de la téléphonie, la partie téléphonie n'est qu'un composant parmi d'autres, l'utilisation de nombreux capteurs a radicalement changé les développements logiciels. Pourquoi ce changement ? Tout d'abord, la simplification des API a permis à de nombreux développeurs d'intégrer des capteurs dans leur application alors qu'il n'y a encore pas si longtemps il fallait avoir des bagages en électronique pour mettre en œuvre de tels logiciels. Sous Android, les couches d'abstraction logicielle ont permis de s'affranchir de nombreuses difficultés de portage, ainsi une application fonctionne aussi bien sur une tablette que sur un smartphone. Avec ce principe, **Google** a fait table rase et a redistribué les cartes. Ce sérieux dépoussiérage a révélé de nouveaux concepts dans le domaine du logiciel embarqué lié à la téléphonie avec l'utilisation de nombreux capteurs : **accéléromètre**, **gyromètre**, **GPS**, **capteur de luminosité**, **de pression**, **de proximité**, **NFC**... D'autres concepts sont alors apparus avec les capteurs composites, mélange d'informatique et d'électronique. Nous sommes dans la virtualisation des capteurs électroniques. Le résultat est époustouflant ! De nouvelles gammes de logiciels sont alors à portée de main, et le développeur peut encore exercer son imagination pour intégrer ces techniques dans des applications d'une diversité incroyable !

Dans ce numéro, qui sera une version « avancée » du hors-série n°82 *« Créez votre première application Android »* (nous vous conseillons de vous référer à ce hors-série si vous n'avez jamais installé le **SDK Android** ou utilisé **Android Studio**), nous allons découvrir avec des exemples concrets et didactiques la mise en œuvre de ces capteurs au travers d'une application de géocaching qui utilise les capteurs suivants : accéléromètre, gyromètre, magnétomètre, NFC, GPS, et capteur composite. Nous aborderons également la **communication Bluetooth** et **SMS** pour réaliser des communications en P2P et longues distances. Dans cette phase de développement, nous passerons en revue les concepts essentiels afin que vous puissiez intégrer facilement les capteurs de la plateforme dans vos propres applications, même celles déjà existantes. L'intérêt des capteurs, c'est qu'ils donnent « vie » aux applications, car ils sont en quelque sorte le liant entre le monde physique et le monde du virtuel et de l'impalpable que sont les logiciels. Les cas d'utilisation sont vastes, le challenge reste alors de trouver l'idée originale qui fera de votre application un véritable best-seller ! Si vous avez une idée sur ce sujet depuis longue date, ce hors-série vous propose maintenant de passer à l'action...

En parlant de best-seller, pourquoi ne pas tester la popularité de votre application tout en rémunérant vos heures de développement ? Dans une seconde étape de ce hors-série, nous allons découvrir les différentes étapes de monétisation de votre application sur le *store* **Google Play**. Les *stores* de téléphonie mobile proposent aujourd'hui aux développeurs des opportunités invraisemblables de diffusion logicielle au niveau international en à peine quelques clics (ou presque...). Bien sûr, cette étape doit faire l'objet d'une attention particulière comme l'internationalisation de l'application, l'optimisation des codes, la protection de votre application, autant de sujets qui vont assurer le succès de la diffusion. Les *stores* ont imposé une grande compétition et, pour s'en apercevoir, il suffit de consulter notamment Google Play pour un type d'application : le choix est pléthorique. N'hésitez pas à devenir un compétiteur international ! C'est parfois juste l'originalité de vos développements qui fera la différence et qui vous rendra riche ! Mais avant de passer par la case « cash », découvrons ensemble les capteurs de votre smartphone...

Pour Adèle.

Frédéric Camps

Il vous manque des bases pour développer votre application Android ? Vous n'avez pas encore installé Android Studio ? Retrouvez toutes les informations qui vous manquent dans notre précédent hors-série « Créez votre première application Android » !

Disponible sur www.ed-diamond.com

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

SOMMAIRE

GNU/Linux Magazine Hors-Série N°91

DÉCOUVREZ

LES ÉTAPES À SUIVRE POUR CRÉER UN PROJET STRUCTURÉ

p.08 Préparez le démarrage de votre projet de géocaching

MANIPULEZ

LES CAPTEURS/PÉRIPHÉRIQUES DE VOTRE SMARTPHONE : ACCÉLÉROMÈTRE, MAGNÉTOMÈTRE, NFC, GPS, ETC.

p.16 Découvrez les capteurs de votre smartphone

p.24 Comment intégrer des capteurs dans une application ?

p.40 Utilisez des tags NFC

p.52 Localisez les caches à l'aide de leurs coordonnées GPS et stockez l'information

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

CRÉEZ, PUBLIEZ & MONÉTISEZ VOTRE APPLICATION AVANCÉE ANDROID !

COMMUNIQUEZ

EN BLUETOOTH ET PAR SMS POUR PARTAGER DES CACHES ET LES AFFICHER SUR DES CARTES GOOGLE MAPS

p.64 Communiquez en Bluetooth

p.80 Envoyez des SMS et utilisez l'API Google Maps pour afficher des cartes

PUBLIEZ & MONÉTISEZ

VOTRE APPLICATION SUR GOOGLE PLAY, RENDEZ-LA PAYANTE ET/OU INSÉREZ-Y DE LA PUBLICITÉ

p.92 Préparez votre application pour le store

p.106 Internationalisez votre application et touchez le plus de monde possible

p.110 Diffusez votre application sur Google Play

p.118 Gagnez de l'argent ! Beaucoup d'argent... ?

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

DÉCOUVREZ

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

1

DÉCOUVREZ

À découvrir dans cette partie...

Préparez le démarrage de votre projet de géocaching

Avant de se lancer dans tout projet, il faut avoir une idée de ce qui va nous attendre. Dans cette partie, nous allons présenter le projet de géocaching qui servira de fil rouge à ce hors-série et réaliser la phase d'analyse en UML/SysML en créant le diagramme des cas d'utilisation. Cet article sera également l'occasion de revoir brièvement l'architecture logicielle d'Android. p. 08

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

PRÉPAREZ LE DÉMARRAGE DE VOTRE PROJET DE GÉOCACHING

Rien de plus motivant que d'apprendre une technique à l'occasion d'un projet. Le but du projet de géocaching est d'aborder pas à pas de nombreuses technologies de la plateforme Android afin de pouvoir les réutiliser dans vos projets. Dans ce but, nous aborderons différents aspects de la plateforme. Ce projet vous donnera l'occasion de découvrir toute la puissance de la plateforme Android : capteur, interface graphique adaptative, monétisation et encore bien d'autres aspects qui permettent de développer des applications vraiment professionnelles et surtout originales.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Le géocaching est un loisir dont le but est de dissimuler et/ou rechercher des « géocaches » où se trouve un trésor, matérialisé par un ou des objets, la plupart du temps sans valeur, plutôt symboliques. Ce trésor est accompagné éventuellement d'un livre d'or qui permet de laisser une trace de son passage aux différents participants et aussi d'y inscrire des indices pour la prochaine géocache. La recherche du trésor se fait à l'aide d'un **GPS** et éventuellement avec un système de cartographie. Dans notre projet de géocaching, nous allons un peu innover en proposant une borne **NFC** dissimulée qui enregistre le passage des participants du jeu qui peuvent lire et inscrire un message, comme par exemple des indications pour une autre borne de géocaching. Ici l'intérêt du NFC c'est qu'aucune source d'énergie n'est nécessaire pour alimenter la borne. Une borne NFC est bon marché, il est alors possible pour un petit budget (de l'ordre d'une vingtaine d'euros !) de réaliser une véritable chasse aux objets avec son smartphone sans pour autant d'avoir besoin d'une connexion internet permanente.

1. AGENDA

Notre projet est organisé en quatre étapes dédiées principalement à des développements logiciels. La quatrième étape révèlera la stratégie de monétisation de l'application, elle est donc moins liée au codage. Chaque étape présente de façon détaillée les objectifs et les moyens de mise en œuvre :

- Étape 1 : Mise en place du projet ;
- Étape 2 : Utilisation des capteurs ;
- Étape 3 : Prise en charge des moyens de communication ;
- Étape 4 : Finalisation de l'application, publication et monétisation.

Voyons maintenant le détail de ces différentes étapes.

1.1 Mise en place du projet

Dans cette étape, nous allons analyser les différents composants logiciels à mettre en œuvre. Cette phase est très importante, car elle donnera le plan du projet que nous suivrons de bout en bout. Si cela vous paraît fastidieux, vous pouvez passer aux étapes suivantes pour revenir éventuellement plus tard à cet article. Néanmoins, par expérience, une brève analyse permet de gagner du temps et surtout de donner une certaine logique aux futurs développements.

Dans cet article, nous rappellerons également certains principes que nous avons évoqués dans le Hors-Série n°82 de *GNU/Linux Magazine* « Créez votre première application Android ». Nous ne pourrons pas revenir en détail sur tous les principes de développement, mais des rappels vous permettront de naviguer tout de même sans problème avec les différents concepts et composants logiciels Android.

1.2 Utilisation des capteurs

Cette étape est dédiée à l'apprentissage des capteurs, éléments essentiels aujourd'hui de toute application embarquée sur un smartphone. Les caractéristiques de l'application de géocaching requièrent une connaissance approfondie de la plateforme et en particulier des capteurs NFC et GPS. Nous aborderons également l'accéléromètre, le gyromètre, composants essentiels dans certaines applications. Nous focaliserons en particulier notre attention sur les aspects NFC de la plateforme.

À l'heure où l'on parle de l'IoT, Android est une plateforme de développement idéale dans ce domaine et à un prix très abordable. Le projet **Android Things [10]** permet d'utiliser une bonne partie de l'API Android pour développer des IoT sur des plateformes dédiées. Vous pourrez alors réutiliser vos connaissances pour des plateformes Android IoT **[10]**.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

1.3 Prise en charge des moyens de communication

Une application doit pouvoir utiliser plusieurs médias de communication ; aujourd'hui ceci est quasiment obligatoire. Ici le but est de pouvoir échanger des données entre différentes personnes qui participent au géocaching. À cette occasion, cette partie présentera une communication point à point courte portée et une communication longue portée par SMS.

1.4 Internationalisation, publication et monétisation

Les applications qui sont destinées à la vente doivent passer par une phase de préparation qui demande une attention non négligeable. Cette phase est souvent méconnue et oubliée par de nombreux développeurs qui tentent alors de publier le plus rapidement leur application. Les phases de publication et de monétisation nécessitent une approche stratégique et la connaissance de l'écosystème de monétisation **Google**.

2. CODE SOURCE DU PROJET

Certains exemples de code, particulièrement longs ou un peu moins intéressants techniquement, n'ont pas été reproduits dans le présent magazine de manière à pouvoir vous donner un maximum d'explications... dans le peu de pages disponibles :-). Dans ces cas, nous vous renverrons aux sources du projet disponibles sur **GitHub** à l'adresse suivante : **https://github.com/frameproject/geocaching**. Vous pouvez réaliser un clone du projet par `git clone https://github.com/frameproject/geocaching.git`. Ou alors vous pouvez télécharger une archive complète à partir de **https://github.com/frameproject/geocaching/archive/master.zip**.

3. MISE PLACE DU PROJET

Nous pouvons définir les blocs fonctionnels selon les exigences fonctionnelles extraites du projet, elles sont les suivantes :

1. IHM : un ensemble d'interfaces pour l'utilisation du système et sa configuration ;
2. Gestion NFC : le gestionnaire NFC doit permettre de lire et d'écrire des messages dans un tag NFC ;
3. Géolocalisation : définition des coordonnées GPS et orientation avec une boussole ;
4. Stockage de données : stockage et restitution des géocaches ;
5. Communication : échange en **Bluetooth** qui permet de transférer des données en point à point, échange **SMS** pour transmettre des informations sur une géocache ;
6. Cartographie : visualisation d'une cartographie avec les géocaches ;
7. Système de monétisation : vente du produit, affichage publicitaire.

Les blocs fonctionnels sont décrits selon le formalisme **SysMl [8]** en figure 1.

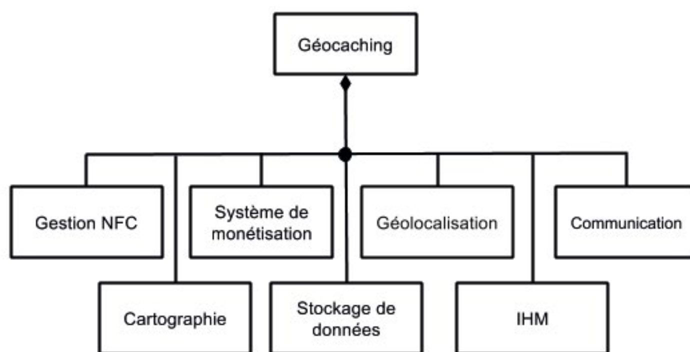


Figure 1 *Schéma des blocs fonctionnels du projet de géocaching.*

Ce document est la propriété exclusive de Johann Locatelli (jacques.thimonier@businessdecision.com)

NOTE

Avant de continuer notre analyse, revenons sur quelques définitions d'UML/SysMl [8] :

- **Acteur : L'acteur est un rôle joué par un utilisateur humain ou un autre système qui interagit avec le système que l'on doit concevoir. L'acteur est lié au système à concevoir par des liens. Dans le cadre du projet de géocaching, nous retrouvons des acteurs humains, des capteurs, des serveurs.**
- **Cas d'utilisation ou « Use Case » : Un cas d'utilisation permet de définir un scénario qui est exécuté par le système à concevoir. Il est représenté par une bulle qui contient un verbe à l'infinitif décrivant l'action réalisée. Un cas d'utilisation est lié à au moins à un acteur qui interagit avec le système. Le projet de géocaching prévoit neuf cas d'utilisation du système.**
- **Cadre du projet : Le cadre du projet (le rectangle du dessin [8]) contient tous les cas d'utilisation par le système. Ce qui se trouve à l'extérieur du cadre ne fait pas l'objet d'un développement.**

Ensuite, les diagrammes de cas d'utilisation permettent de préciser les relations existant entre les acteurs et le système d'un point de vue utilisation du système. Nous retrouverons les blocs fonctionnels dans la description des cas d'utilisation. Il est possible d'identifier les relations avec les systèmes « externes », tels que les capteurs, les serveurs et autres utilisateurs. Le cadre permet de délimiter le périmètre du projet à réaliser. Le travail d'analyse consiste par la suite à rédiger chaque cas d'utilisation afin d'identifier précisément les scénarios. Ce travail de rédaction permet de raffiner les exigences fonctionnelles et les blocs fonctionnels nécessaires aux développements de l'application.

La figure 2 montre un diagramme des cas d'utilisation du système de géocaching.

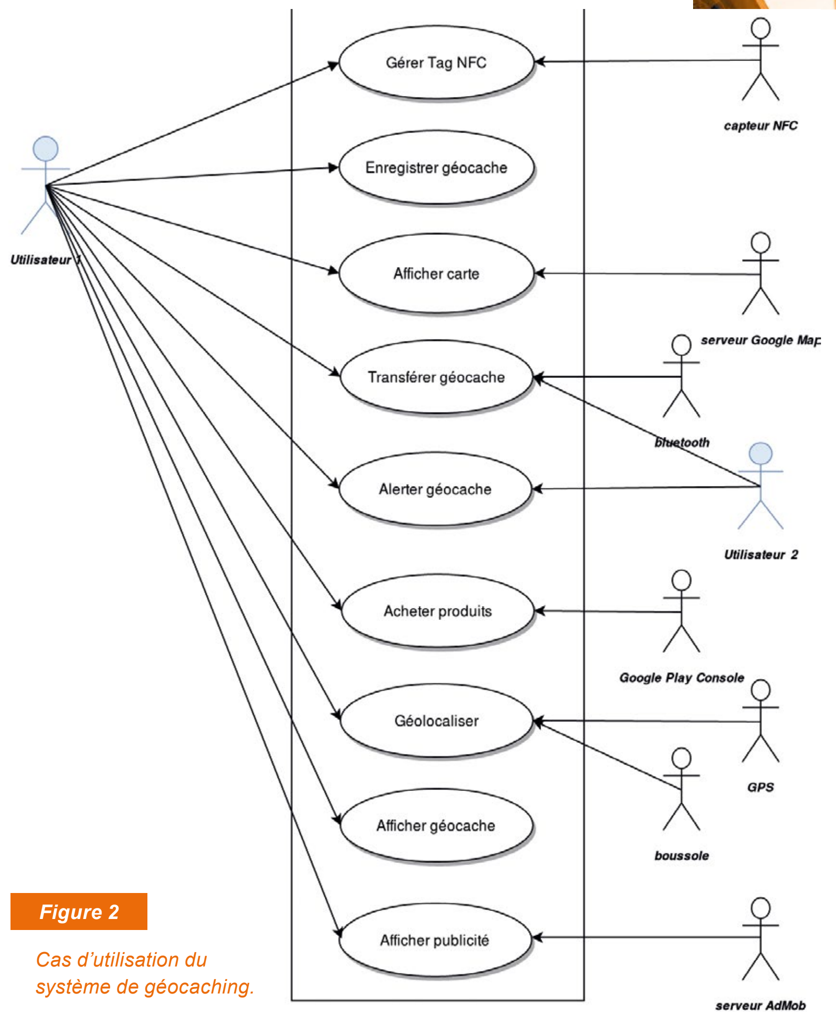


Figure 2

Cas d'utilisation du système de géocaching.

4. COMPOSANTS LOGICIELS ANDROID

Les applications Android sont construites autour de six composants principaux [12]. Cette approche permet d'accélérer considérablement les développements des applications. Les autres composants sont alors utilisés comme des ajustements en fonction de la thématique du projet. Pour le développeur, la phase d'apprentissage est d'autant plus réduite. Ces composants sont les suivants :

- IHM [7] : L'IHM est décrite en **XML**, **Java**.
- Activité [2] : L'activité est la partie logicielle visible par l'utilisateur, car elle est impérativement liée à une IHM.
- Service [3] : Le service est un composant exécuté en arrière-plan, il ne possède pas d'IHM.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

- *Broadcastreceiver* [5] : Ce composant gère les messages qui sont émis en *broadcast* dans le système.
- *Content Provider* [4] : Le *content provider* est responsable de la gestion des données persistantes avec une base de données **SQLite**.
- *Intent* [6] : L'*Intent* n'est pas un composant, mais une fonctionnalité importante d'Android. L'*intent* est un signal asynchrone diffusé par le système ou les applications. Il peut transporter des informations dans une structure de données.

5. ARCHITECTURE LOGICIELLE ANDROID

Avant de se lancer dans les développements, analysons l'architecture logicielle d'Android [9].

5.1 Linux Kernel

Android fonctionne au-dessus du système d'exploitation **Linux**. Ce choix est stratégique, car le noyau Linux est *open source* et dans sa catégorie, il offre un niveau de service inégalé. Comme tout système embarqué, l'OS doit faire l'objet d'une optimisation. Vous ne retrouverez pas ici une distribution standard du type **Ubuntu** ou autres. Le système de fichiers est également largement revisité avec une arborescence spécifique. Les pilotes matériels sont intégrés au noyau, on peut y retrouver par exemple l'USB, ou le Wifi. Il est possible d'y ajouter vos propres pilotes vu qu'Android est open source. Le cœur d'exécution *Android Runtime* (**ART**) se base sur le noyau Linux pour les fonctionnalités sous-jacentes telles que le *multithreading* et la gestion de la mémoire de bas niveau.

5.2 Hardware Abstraction Layer (HAL)

On peut dire que cette couche a largement contribué au succès d'Android, car elle permet d'exécuter une application indépendamment du matériel. Cette couche expose un ensemble d'interfaces standards à la couche supérieure. Elle est composée d'un ensemble de bibliothèques, chacune est spécifique à un matériel tel que le Wifi, le Bluetooth, etc. Lorsqu'une application fait un appel vers un périphérique matériel, le système Android charge en mémoire le module correspondant.

5.3 Android Runtime (ART)

L'ART est une machine virtuelle qui exécute les applications de haut niveau. Ce composant est capable de gérer plusieurs machines virtuelles même sur des équipements à faible capacité mémoire. ART exécute du bytecode au format DEX, conçu spécialement pour Android. Ce format de bytecode propose une empreinte mémoire minimale. À partir d'Android 5 (API 21), chaque application tourne dans son propre processus ART. L'une des originalités d'ART est d'utiliser le concept *Ahead-of-time* (AOT) et *just-in-time* (JIT) en fonction de la version d'Android. AOT est une compilation anticipée qui traduit un langage de haut niveau en langage machine avant l'exécution du programme. Cette opération est réalisée au moment de l'installation de l'application. Le comportement JIT correspond à une compilation à la volée qui permet de traduire du bytecode en code machine natif au moment de l'exécution. JIT est utilisé sur les versions inférieures à Android 5, AOT prend le relais à partir de Android 5.

5.4 Application Native C/C++

La majorité des bibliothèques utilisées dans le code Java sont en fait réellement écrites en **C/C++** pour des raisons de performance. Finalement, les API Java exposent les fonctionnalités des bibliothèques natives via une couche JNI (*Java Native Interface*). Les applications Android peuvent être écrites en Java avec un sup-

Ce document est la propriété exclusive de Johann Locatelli (jacques.thimonier@businessdecision.com)

port C/C++, ou encore complètement en C/C++. Le NDK (*Native Development Kit*) permet d'accéder directement aux bibliothèques natives. À noter que le NDK est livré avec un compilateur croisé *open source*, et que le tout est supporté par l'IDE Android Studio.

5.5 API JAVA Framework

Les applications de haut niveau utilisent cette couche **[1]** comme support de développement qui offre toutes les briques de base pour construire rapidement des applications : IHM, activité, service, etc. C'est le domaine du développeur d'application de haut niveau qui utilise le SDK Java pour accéder aux différentes API.

5.6 System Apps

C'est ici que l'on présente les applications aux utilisateurs. Elles sont liées au *framework* sous-jacent. Les applications proposent des fonctionnalités aux utilisateurs et aux développeurs qui utilisent les applications comme support pour de nouvelles applications.

CONCLUSION

Nous développerons le projet de géocaching avec **Android Studio** mis à disposition gratuitement par Google. Pour l'installer, vous pouvez suivre la procédure décrite dans le Hors-Série de *GNU/Linux Magazine n°82* ou consulter le lien suivant : **https://developer.android.com/studio/install.html**. Il faut compter une dizaine de minutes pour l'installation complète qui ne pose aucun problème particulier sur une machine **Linux**, **Windows** ou encore **Mac**. Sous Windows, il est nécessaire d'installer correctement le driver USB du fabricant de votre matériel afin qu'Android Studio découvre bien votre équipement. Sur Linux, la procédure de configuration USB est assez simple, enfin sur Mac cela est quasiment « automatique ». ■

RÉFÉRENCES

[1] Composants logiciels : **https://developer.android.com/guide/components/fundamentals.html**

[2] Activité : **https://developer.android.com/guide/components/activities/index.html**

[3] Service : **https://developer.android.com/guide/components/services.html**

[4] Content provider :
https://developer.android.com/guide/topics/providers/content-providers.html

[5] Broadcastreceiver : **https://developer.android.com/guide/components/broadcasts.html**

[6] Intent : **https://developer.android.com/guide/components/intents-filters.html**

[7] IHM : **https://developer.android.com/guide/topics/ui/index.html**
et **https://developer.android.com/guide/topics/ui/overview.html**

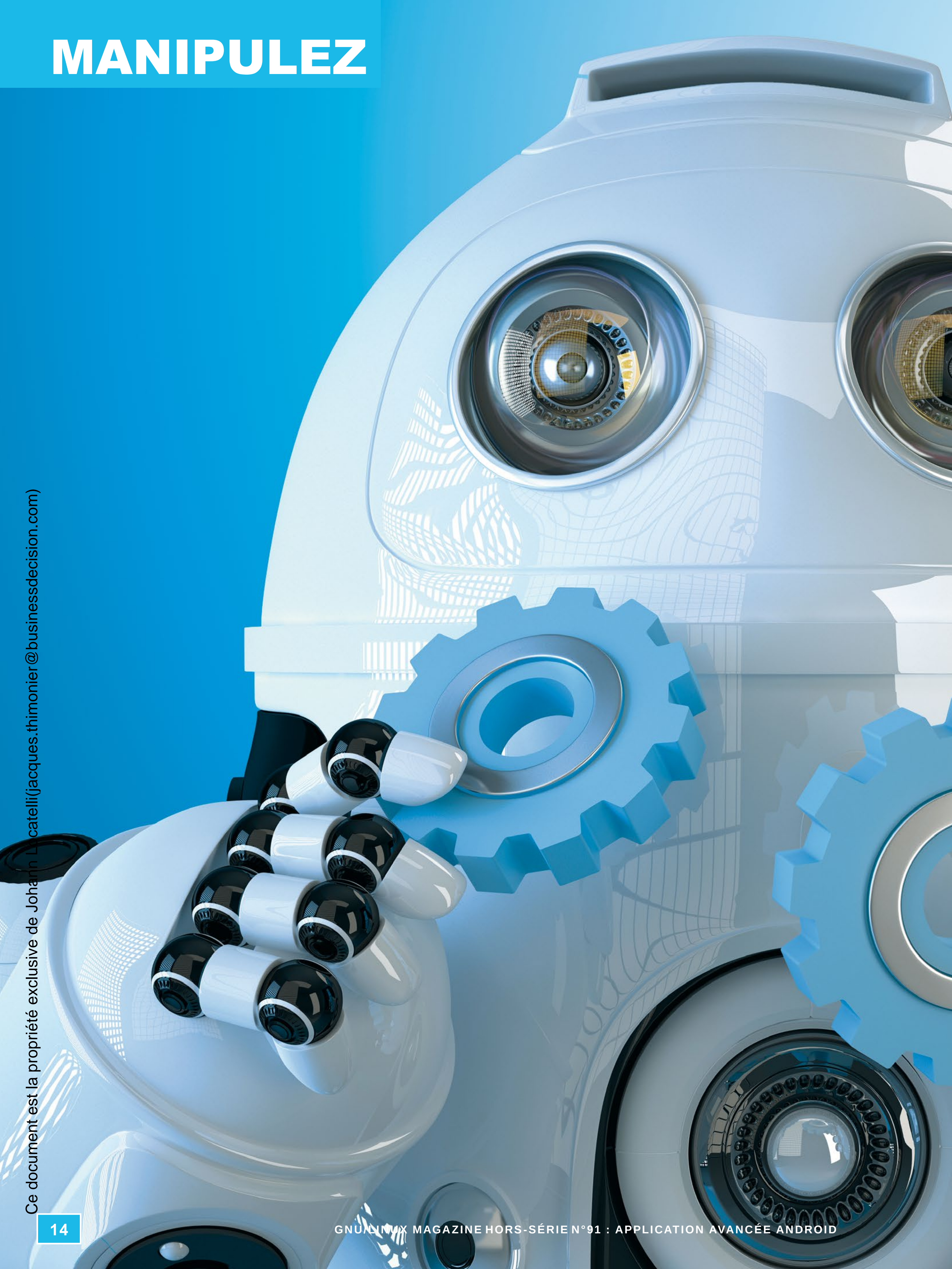
[8] SysMl : **http://www.uml-sysml.org/**

[9] Architecture Android : **https://source.android.com/devices/**

[10] Android Things : **https://developer.android.com/things/hardware/index.html**
et **https://developer.android.com/things/hardware/developer-kits.html**

[12] Première application Android :
https://developer.android.com/training/basics/firstapp/creating-project.html

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)



MANIPULEZ

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

2 MANIPULEZ

À découvrir dans cette partie...

Découvrez les capteurs de votre smartphone

En guise de première approche nous vous proposons de découvrir les différents types de capteurs disponibles sur smatrphone. p. 16

Comment intégrer des capteurs dans une application ?

Nous allons créer ici une boussole électronique de deux manières : à laide des capteurs physiques puis à l'aide du capteur composite « Vector Rotation ». p. 24

Utilisez des tags NFC

Cet article permet d'appréhender l'intégration de la technologie de communication NFC au sein d'une application Android. p. 40

Localisez les caches à l'aide de leurs coordonnées GPS et stockez l'information

Cet article permet d'appréhender l'intégration de la technologie de communication NFC au sein d'une application Android. p.52

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

DÉCOUVREZ LES CAPTEURS DE VOTRE SMARTPHONE

Les smartphones sont bardés de capteurs collectant diverses données. Il est très tentant de les utiliser, mais avant cela il est préférable de mieux les connaître...

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

La première version de l'application de géocaching va nous permettre de découvrir les capteurs sous différents aspects : les capteurs présents, la configuration, le paramétrage de l'acquisition des données. Dans cette partie, l'objectif est de développer les deux premiers blocs fonctionnels visibles sur la figure 1 (en bleu) associés aux cas d'utilisation : « Gérer tag NFC » et « Géolocaliser ».

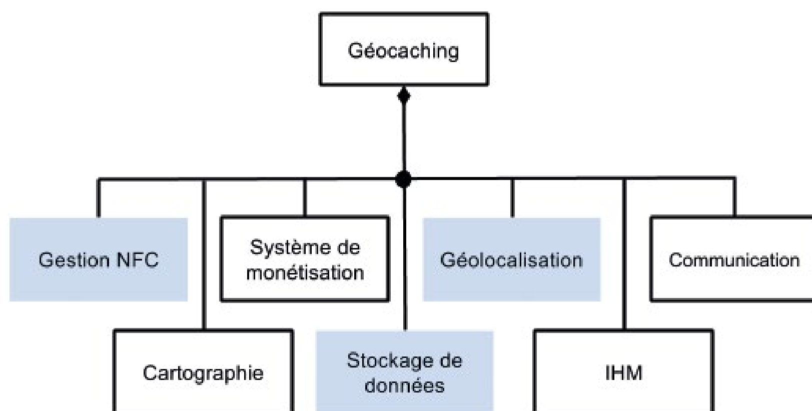
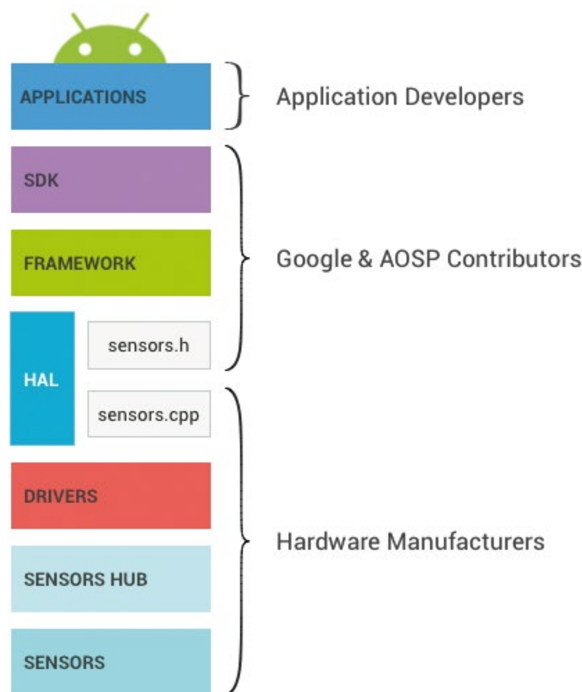


Schéma des blocs fonctionnels du projet de géocaching. **Figure 1**

La plateforme Android possède une surprenante collection de capteurs électroniques qui autorise le développement d'applications d'une très grande variété et surtout d'une grande originalité. L'atout de cette plateforme est surtout la simplicité déconcertante d'utilisation des capteurs avec des interfaces **Java** qui pilotent les couches basses programmées en **C/C++** via l'interface Jni. Les capteurs fournissent des données brutes qu'il convient ensuite d'interpréter afin de les rendre utiles à vos applications. Les capteurs ou périphériques les plus connus sont le GPS, l'accéléromètre, mais bien d'autres capteurs sont aussi disponibles.

1. ARCHITECTURE LOGICIELLE DES CAPTEURS

L'architecture logicielle des capteurs (voir figure 2) est relativement simple, elle est constituée de la couche Android AOSP et celle fournie par les constructeurs d'équipement **[1]**. Les drivers fournis par les fabricants ne sont pas *open source* (en principe) ce qui peut poser des problèmes pour les développements des couches basses.



Architecture logicielle des capteurs. **Figure 2**

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

1.1 SDK

Le SDK en Java permet de traverser les couches logicielles en toute transparence pour le développeur d'applications de haut niveau. On peut demander par exemple l'état d'un capteur à une certaine fréquence d'échantillonnage. Il est également possible d'utiliser plusieurs capteurs dans la même application.

1.2 Framework

La couche *framework* réalise le lien avec la couche HAL [2] avec un comportement de multiplexage :

- Lorsqu'une application demande un accès à un capteur, la couche *framework* envoie une demande vers la couche HAL pour activer le capteur ;
- Si une nouvelle application demande un accès à un même capteur, les nouveaux paramètres sont envoyés à la couche HAL. Pour toutes les applications, le capteur fonctionnera alors à la plus haute fréquence d'acquisition demandée ;
- La désactivation du capteur est réalisée par la dernière application qui l'utilise. Ce dernier point est important, car il faut considérer que les systèmes mobiles ne sont pas toujours alimentés, cette économie d'énergie non négligeable permet d'augmenter l'autonomie de la batterie.

1.3 HAL

Comment mon application peut fonctionner d'un équipement à un autre si les composants électroniques sont différents ? Bien sûr le problème se pose, car chaque constructeur de smartphone ou tablette utilise son propre lot de composants électroniques, la configuration peut aussi changer pour une même série d'équipements. Le problème est résolu par une couche d'abstraction logicielle (HAL) [2] qui impose les mêmes interfaces à tous les constructeurs, si bien que votre application ne verra aucune différence d'un système à un autre, car dans chaque cas un « driver » spécifique du capteur devra être fourni par le constructeur. Google met à disposition le document « Android Compatibility Definition Document » [3] qui définit le comportement des API et des composants électroniques. Par exemple, dans le CDDv7 l'accéléromètre doit proposer un taux de rafraîchissement d'au moins 200 Hz. Le fournisseur doit développer le driver Linux, mais aussi les API de plus haut niveau en Jni C /C++.

Chaque constructeur doit ensuite s'assurer de l'entière compatibilité de l'ensemble avec un jeu de test fourni par Google, il s'agit du « Compatibility Test Suite » (CTS) [4]. Si les tests sont passés, alors la plateforme est considérée comme « compatible Android ». Cela permet alors d'avoir accès au « Licensing Google Mobile Services » (GMS), une suite logicielle Google (**Google Play**, **YouTube**, **Google Maps**, **Gmail**...) qui ne fait pas partie de la partie *open source* du projet Android. Cette suite logicielle non gratuite est d'une grande importance, car elle permet entre autres d'accéder au store officiel et à la remise à jour logicielle. Vous trouverez des équipements Android dont le constructeur n'a pas opté pour cette suite logicielle pour des raisons financières. L'équipement est moins onéreux à l'achat, mais très rapidement inutilisable, car limité dans les fonctionnalités essentielles.

1.4 Sensor Hub

Cette zone permet de réaliser des calculs un peu lourds pour la partie logicielle, comme la fusion de données,ou encore la détection de pas de marche. L'architecture logicielle n'est pas strictement définie par Google, il s'agit plutôt de coprocesseurs ou processeurs embarqués dans les capteurs.

Ce document est la propriété exclusive de Johann Locatelli (jacques.thimonier@businessdecision.com)

1.5 Capteur MEMs

Les équipements Android utilisent des capteurs de type MEMs (*Microelectromechanical systems*). Un composant MEMs peut embarquer plusieurs fonctionnalités comme un accéléromètre et un gyromètre. Une petite unité de calcul est parfois associée au composant pour réaliser des filtres numériques. Il est préférable de dédier les calculs lourds à ce niveau plutôt que les couches logicielles, le système d'exploitation sera d'autant plus disponible pour réaliser d'autres tâches.

2. PRÉSENTATION DES CAPTEURS

Les capteurs de la plateforme Android sont regroupés selon plusieurs catégories **[5]** :

- capteurs de mouvement **[6]** ;
- capteurs environnementaux **[7]** ;
- capteurs de position **[8]**.

2.1 Les capteurs de mouvement

Les capteurs de mouvement ou « Motion Sensors » sont principalement l'accéléromètre et le gyromètre. Si vous n'êtes pas très habitué à ces capteurs, voici quelques explications.

L'accéléromètre **[5, 6]** est un capteur qui ne mesure que des variations de vitesse. Si vous posez le capteur au sol et que vous enregistrez les données brutes, alors vous ne verrez pas grand-chose mis à part du bruit qui se caractérise par de petites variations quasi aléatoires de faible amplitude. À chaque changement de position de l'équipement, l'accéléromètre détecte donc une variation de la vitesse. L'accéléromètre possède trois axes orthogonaux qui représentent un repère en trois dimensions x,y,z, c'est pour cela que l'on parle d'accéléromètre 3D. Pour chaque axe, le composant électronique délivre une valeur numérique disponible via l'API Java.

Le gyromètre, lui, enregistre les variations de rotations autour d'un axe. Il possède lui aussi trois axes x,y,z.

2.2 Les capteurs environnementaux

Ces capteurs **[7]** mesurent différents paramètres environnementaux tels que la température et la pression atmosphérique, l'éclairage, l'humidité.

2.3 Capteurs de position

Les capteurs de position **[8]** mesurent la position physique d'un dispositif. Cette catégorie comprend entre autres l'accéléromètre, le magnétomètre, le détecteur de proximité.

2.4 Capteurs composites

Les capteurs composites **[9]** sont un assemblage de capteurs physiques et de logiciels qui simulent le fonctionnement d'un capteur. Par exemple, la plateforme fournit un capteur « Accélération linéaire ». Celui-ci n'existe pas physiquement, mais un logiciel observe les comportements de l'accéléromètre, du gyromètre, du magnétomètre pour calculer l'accélération linéaire.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

2.5 Système de coordonnées de l'équipement

La plupart des capteurs utilisent un repère à trois dimensions [5] (voir figure 3) afin de représenter les données qu'ils produisent. Ce repère orthonormé représente votre équipement lorsqu'il est utilisé dans son orientation par défaut, c'est-à-dire en mode portrait pour les smartphones, en mode paysage pour les tablettes. Quelques rares équipements ne suivent pas cette directive et ne facilitent pas la gestion des capteurs. L'axe des X est horizontal et pointe vers la droite, l'axe Y est vertical et pointe du bas vers le haut. L'axe Z pointe vers vous lorsque votre téléphone est vertical et face à vous. l'intersection de ces repères représente le point `O(0,0,0)`.

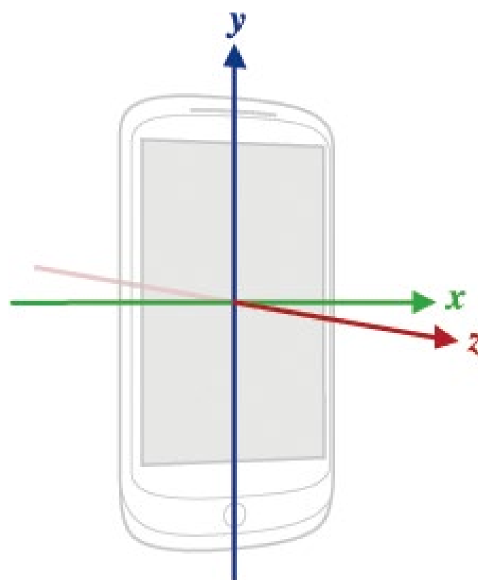


Figure 3 Système de coordonnées.

Ce système de représentation est utilisé pour les capteurs suivants : capteur d'accélération, détecteur de gravité, gyroscope, capteur d'accélération linéaire, ou encore capteur de champ géomagnétique.

Ce système reste fixe par rapport à l'équipement, il n'est jamais modifié. Les valeurs renvoyées par les capteurs sont toujours des valeurs dans ce repère :

- axe x : l'axe longitudinal représente l'axe de roulis ou *roll* (en vert).
- axe y : l'axe transversal représente l'axe de lacet ou *yaw* (en bleu).
- axe z : l'axe de tangage ou *pitch* (en rouge).

2.6 Représentation dans le repère

Le repère à trois dimensions représente l'environnement naturel. Les axes du capteur sont alignés sur les axes physiques du smartphone.

Prenons le cas de l'accéléromètre qui mesure l'accélération selon trois axes `[x,y,z]`, le capteur mesure les projections orthogonales `[Ax, Ay, Az]` du vecteur `Acc` sur les axes du repère (voir figure 4) :

- l'axe des X mesure l'accélération latérale ;
- l'axe des Y mesure l'accélération longitudinale ;
- l'axe des Z mesure l'accélération verticale.

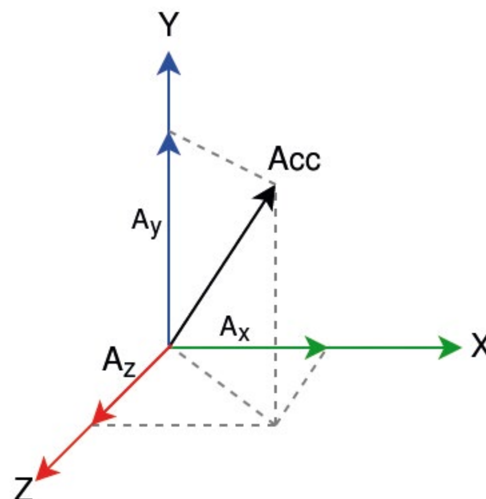


Figure 4 Représentation d'un capteur 3D.

Ce document est la propriété exclusive de Johann Locatelli (jacques.thimonier@businessdecision.com)

C'est grâce à ces projections que l'on peut par exemple détecter une accélération sur un axe en particulier. La plupart du temps, on veut définir une accélération maximale, sans prendre en considération un axe en particulier. Dans ce cas, il faut mesurer la distance en le point `[0,0,0]` qui représente le centre du repère et la pointe du vecteur `Acc`. Cette distance se calcule avec la formule suivante :

$$||Acc|| = \sqrt{Ax^2 + Ay^2 + Az^2}$$

L'accéléromètre indique au repos une accélération verticale sur l'axe Z de `||Acc||=+9.81 m/s²`, soit `1g`, cette valeur représente l'accélération normale que l'on subit sur la Terre en étant au repos. Le capteur ne fait pas la différence entre l'accélération de l'appareil causée par un mouvement et celle due à la gravité.

2.7 Calibration des capteurs

Les capteurs ne sont pas parfaits, ils doivent subir une calibration qui permet de corriger certains défauts. Cette étape est importante, sous peine d'avoir des mesures erronées.

Reprenons le cas de l'accéléromètre, lorsque l'appareil est posé à plat avec l'écran pointant vers le ciel (axe Z pointé vers le ciel) sur un plan parfaitement horizontal (perpendiculaire à la terre), il mesure environ `9.81 m/s²` sur l'axe Z. Si ce n'est pas le cas, alors le capteur a un défaut de mesure. Le défaut peut provenir :

- d'un gain : la valeur de la projection est multipliée par un gain ;
- d'un biais : le système de mesures est décalé ;
- d'une erreur résiduelle due au bruit ;
- d'une erreur d'orthogonalité des axes du capteur.

Au final, la mesure de l'accélération exprimée par le capteur suit la relation suivante : `Val = c * Acc + b + ε` avec `Val` la valeur mesurée finale avec erreur, `c` le gain, `Acc` la mesure réelle sans erreur, `b` le biais et `ε` le bruit blanc de moyenne zéro.

Le vecteur étant en trois dimensions, nous pouvons exprimer le système sous forme matricielle :

$$\begin{pmatrix} Val_x \\ Val_y \\ Val_z \end{pmatrix} = \begin{pmatrix} c_x & 0 & 0 \\ 0 & c_y & 0 \\ 0 & 0 & c_z \end{pmatrix} \begin{pmatrix} Acc_x \\ Acc_y \\ Acc_z \end{pmatrix} + \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix} + \begin{pmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_z \end{pmatrix}$$

Ici, le système de matrice ne prend pas en compte l'erreur d'orthogonalité.

2.7.1 Dérive du capteur

La mesure d'un capteur peut aussi se dégrader dans le temps, on appelle ceci la dérive. Cela peut être dû par exemple à la température, l'humidité, la pression, le champ magnétique. Dans de tels cas, un petit décalage de mesure s'accumule dans chaque itération jusqu'à produire des valeurs complètement erronées. La dérive peut-être effective après quelques minutes voire quelques heures de fonctionnement, d'où la nécessité de tests longs dans un environnement réel.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

2.7.2 Calibration automatique

Les capteurs intelligents intègrent parfois aussi un système de calibration qui prend en compte plusieurs valeurs environnementales pour calibrer le capteur selon un modèle. Cette approche permet d'éviter des calculs trop lourds dans le logiciel embarqué.

2.7.3 Calibration « manuelle »

Il est possible dans une application de réduire l'erreur de la mesure en observant le comportement et en appliquant un facteur de correction directement. Cette phase demande une bonne connaissance du comportement du capteur. Certains constructeurs équipent leur smartphone/tablette d'un logiciel de calibration, cependant ceci reste à leur discrétion et n'est pas une obligation.

2.7.4 Utilisation d'un filtre

Un filtre (passe-haut, passe-bas , passe-bande) n'efface pas les défauts de mesures, mais peut rejeter des données à une fréquence donnée, réduire le bruit.

NOTE

Certaines applications préfèrent travailler avec les données brutes non calibrées. En effet, les données brutes ne subissent pas de modification et peuvent être exploitées pour la fusion de données. L'utilisation de données calibrées peut fausser complètement les résultats, car les données ont déjà subi des traitements. Android propose des données brutes et calibrées pour le magnétomètre et le gyromètre.

2.8 Performances

Comme nous l'avons vu dans l'article précédent, le CDD **[3][10]** impose de nombreuses caractéristiques techniques. Dans le CDD v7, au chapitre capteurs, vous trouverez par exemple les points suivants : fréquence d'échantillonnage, précision (résolution en bits), écart type de la mesure, dérive, etc.

Certaines caractéristiques décrites dans le CDD sont obligatoires, d'autres ne sont que des suggestions. Si vous désirez connaître précisément votre système, ce document est une ressource indispensable. À noter que les performances peuvent bien sûr aller au-delà de ce que préconise Google. L'utilisation peut être aussi limitée par l'interface elle-même qui impose une configuration et une exploitation standards du composant électronique. Par exemple, vous n'aurez sûrement pas accès à toutes les possibilités du capteur GPS via les primitives de l'API. Libre à vous de développer alors le driver Linux et tout le *middleware* associé.

Les données du capteur sont produites à une fréquence **[11]** qui représente le délai entre deux événements **[12][5]** (l'acquisition de données est gérée sous forme d'événement). Cette fréquence est fixée par la variable `int delay` de la primitive `SensorManager.registerListener(Activity, Sensor, int delay)`. Vous pouvez spécifier d'autres fréquences prédéfinies, telles que : `SENSOR_DELAY_NORMAL` (200 000 microsecondes), `SENSOR_DELAY_GAME` (20 000 microsecondes), `SENSOR_DELAY_UI` (60 000 microsecondes), et `SENSOR_DELAY_FASTEST` (0 microsecondes).

La fréquence entre deux événements ne représente pas la fréquence d'acquisition du capteur. La fréquence spécifiée entre deux événements n'est qu'un délai suggéré. Le système Android et d'autres applications peuvent altérer ce délai. Néanmoins, il est possible de vérifier cette fréquence,

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

car chaque événement produit est associé à un *timestamp* du système. L'API fournit également une primitive `getMinDelay()` [11] qui permet de connaître, pour un capteur, l'intervalle de temps minimum (en microsecondes) entre deux acquisitions de données. ■

NOTE

À partir d'Android 3.0 (API Level 11), vous pouvez également spécifier le délai comme valeur absolue (en microsecondes).

Synthèse

Nous avons pu voir dans cet article les différents types de capteurs : les capteurs de mouvement, les capteurs environnementaux et les capteurs de position. La plupart d'entre eux utilisent un système de repère à trois dimensions pour représenter les données qu'ils produisent.

Il ne faut pas oublier que les capteurs ne sont pas parfaits, et il faut les calibrer de manière à éviter le plus possible l'obtention de mesures erronées.

RÉFÉRENCES

[1] Sensor stack : **https://source.android.com/devices/sensors/sensor-stack.html**

[2] HAL : **https://source.android.com/devices/** et **https://source.android.com/devices/sensors/hal-interface.html**

[3] CDD : **https://source.android.com/compatibility/cdd.html**, **https://source.android.com/compatibility/7.1/android-7.1-cdd.html**, et **http://source.android.com/compatibility/7.1/android-7.1-cdd.html#7_3_sensors**

[4] Compatibility Test Suite : **http://source.android.com/compatibility/cts/index.html**

[5] Capteurs Android : **https://developer.android.com/guide/topics/sensors/sensors_overview.html**

[6] Capteur de mouvement : **https://developer.android.com/guide/topics/sensors/sensors_motion.html**

[7] Capteur d'environnement : **https://developer.android.com/guide/topics/sensors/sensors_environment.html**

[8] Capteur de position : **https://developer.android.com/guide/topics/sensors/sensors_position.html**

[9] Capteur composite : **https://source.android.com/devices/sensors/sensor-types**

[10] Qualité des capteurs : **http://sensmark.info/sensor-benchmarks**

[11] Vitesse acquisition : **https://developer.android.com/reference/android/hardware/SensorManager.html** et **https://developer.android.com/guide/topics/sensors/sensors_overview.html#sensors-monitor**

[12] Calcul de position : **https://developer.android.com/guide/topics/sensors/sensors_position.html**

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

COMMENT INTÉGRER DES CAPTEURS DANS UNE APPLICATION ?

Nous allons créer une première version de notre projet en y intégrant des capteurs. L'objectif est ici de réaliser une boussole pour la première version de notre projet.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

L'intégration d'un capteur dans une application est grandement facilitée par les API sous-jacentes. Ici, nous allons passer en revue les différentes étapes pour arriver à l'élaboration de la première version du projet de géocaching.

1. GESTION DU MANIFEST

NOTE

Le `manifest` est le fichier de configuration de l'application Android qui permet de définir de nombreux paramètres, nous y trouverons la description de l'application, mais aussi tous les droits de sécurité.

L'accès aux capteurs requiert parfois une permission comme le capteur `HEART_RATE`.

Lorsqu'on utilise des capteurs dans une application, il est préférable de le déclarer dans le `manifest`. Cela permet de réaliser un filtre sur le « store » pour les systèmes qui ne possèdent pas un capteur. Ici un exemple avec l'accéléromètre et le gyromètre :

Fichier

```
<uses-feature android:name="android.hardware.sensor.accelerometer"
android:required="true" />
<uses-feature android:name="android.hardware.sensor.compass"
android:required="false" />
```

2. APIS DE PROGRAMMATION

Les APIs des capteurs offrent une grande facilité d'utilisation, car celle-ci a été simplifiée afin de pouvoir à moindre coût intégrer les capteurs dans n'importe quelle application. Le *package* `android.hardware` gère l'accès et l'utilisation des capteurs :

- SensorManager : L'API des capteurs se base sur un service appelé « sensor service ». Celui-ci fournit toutes les méthodes pour définir les capteurs présents sur l'équipement, s'enregistrer auprès du service de gestion du capteur. Ce service permet aussi de connaître la précision des mesures des capteurs, de préciser la vitesse d'acquisition.
- Sensor : Une instance de cette classe permet de créer une instance d'un capteur qui fournit alors toutes les méthodes pour son utilisation.
- SensorEvent : L'acquisition des données d'un capteur se fait par événements. À chaque événement, une donnée du capteur est produite avec : le type de capteur qui produit les données, les valeurs physiques, la précision de la mesure, et la date de l'événement (ou *timestamp*).

Cette classe possède un tableau de données `public final float[ ] value`. Ce tableau contient les données brutes envoyées par le capteur. La taille du tableau et son contenu dépendent de chaque capteur. Les événements sont également horodatés par le système avec `public long timestamp`.

- SensorEventListener : Cette interface peut être utilisée pour créer une méthode de *callback* qui gère alors les événements des capteurs (données de capteur ou modification de précision du capteur).

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

- **Template d'application** : Google propose un *template* pour l'utilisation des capteurs dans les applications Android. Cette proposition de structuration entre dans le cadre des bonnes pratiques pour le développement. La phase d'identification des capteurs et de détermination de leur capacité permet de définir la présence réelle du capteur sur la plateforme, et ensuite de réaliser un programme de vérification. Cette première étape permet d'analyser la performance des composants (même si le CDD permet de définir les capacités max) et de configurer l'application en conséquence. Il faut ensuite surveiller les événements du capteur : il s'agit ici de réaliser un programme d'acquisition des données brutes. Un événement de capteur vous fournit trois informations : le nom du capteur qui a déclenché l'événement, l'horodatage de l'événement, et les données brutes du capteur.

Les API ne supportent pas systématiquement tous les types de capteurs. En fonction de la version d'Android, vous trouverez quelques différences, ce qui signifie que pendant la phase de développement il faudra bien cibler jusqu'à quelle version de l'OS vous assurez la compatibilité.

La disponibilité des capteurs en fonction des versions d'Android est consultable sur Internet.

3. PREMIÈRE VERSION DE NOTRE APPLICATION

Revenons à la structure de notre application décrite dans l'article introductif. Nous allons commencer la réalisation du projet de géocaching en appliquant le *pattern* de développement proposé par Google, c'est-à-dire faire un état des lieux des capteurs présents et créer un programme de test.

Nous allons utiliser le *Wizard* d'**Android Studio** pour créer notre application. Utilisez comme vous en avez l'habitude le menu ***File > New Project*** pour créer un nouveau projet. Le choix de l'API est une décision importante, car cela vous permet de cibler vos futurs équipements qui seront compatibles avec votre application. Ici en choisissant l'API 5, l'application sera compatible avec 24% des terminaux en circulation. Le nom du *package* est de votre choix, utilisez tout de même une logique pour la structuration future. Nous utiliserons une application « Empty Activity » afin d'avoir un squelette logiciel le plus simple possible dans un premier temps.

3.1 Détection des capteurs présents

Nous allons utiliser le *package* `SensorManager` qui donne accès aux couches sous-jacentes et autorise le pilotage des capteurs de la plateforme.

Modifions l'activité Android vide créée précédemment avec le code suivant :

Fichier

```
package mag.linux.android_sensors;

import android.content.Context;
import android.hardware.Sensor;
import android.hardware.SensorManager;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;

import java.util.List;
```

Ce document est la propriété exclusive de Johann Locatelli (jacques.thimonier@businessdecision.com)

```
public class MainActivity extends AppCompatActivity {

   final String TAG="sensor";

   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main);

       sensorDetection();
   }


 void sensorDetection()
{
  SensorManager mSensorManager;

  mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);

  List<Sensor> deviceSensors = mSensorManager.getSensorList(Sensor.TYPE_ALL);

   if(deviceSensors!=null && !deviceSensors.isEmpty()) {

       for (Sensor mySensor : deviceSensors) {
           Log.v(TAG, "info: " + mySensor.toString());
       }

       if (mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER) != null){
           Log.v(TAG, "info: Accelerometer found !");
       }
       else {
           Log.v(TAG, "info: Accelerometer not found !");
       }
   }
}
}
```

La détection des capteurs est réalisée par la méthode `sensorDetection()` qui s'appuie comme prévu sur `SensorManager` qui renvoie une liste de capteurs. Les capteurs sont ensuite affichés via la méthode `toString()` qui factorise toutes les méthodes de `Sensor` renvoyant `String`. Les logs sont enregistrés dans le `LogCat`, que l'on peut visualiser dans la fenêtre *Android Monitor*. Les logs nous montrent les capteurs physiques et les capteurs composites. La détection des capteurs permet de configurer dynamiquement les applications. Si vous utilisez l'accéléromètre pour réaliser un podomètre alors que le capteur `STRING_TYPE_STEP_COUNTER` est présent sur la plateforme, alors il est préférable d'utiliser celui-ci plutôt que de réaliser des calculs intensifs inutiles. À noter que le podomètre est disponible à partir de l'API Kikat sous condition d'avoir aussi le capteur, c'est par exemple le cas du NEXUS 5 et version supérieure.

RAPPEL

Pour résoudre les imports de *package*, cliquez sur le mot-clé en rouge, puis appuyez sur <Alt> + <Enter> : une liste apparaît et cliquez alors sur Import.

Voici un exemple de log de détection des capteurs :

Fichier

```
info: {Sensor name="LGE Accelerometer Sensor", vendor="InvenSense", version=1,
type=1, maxRange=39.226593, resolution=0.0011901855, power=0.5, minDelay=5000}
...
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Bien sûr les logs vont dépendre de votre équipement. Dans la plupart des configurations matérielles, nous retrouverons : l'accéléromètre, le gyromètre, le détecteur de proximité, et le capteur de lumière. D'autres capteurs peuvent être présents, mais sont plus rares, comme les capteurs de température ou d'humidité.

Les logs nous informent sur les caractéristiques des capteurs tels que le nom du capteur (`Sensor name`), la mesure maximale du capteur dans l'unité du capteur (`maxRange`), la consommation du capteur en mA lorsqu'il est utilisé (`power`), etc.

Examinons par exemple l'accéléromètre :

Fichier

```
Sensor name="LGE Accelerometer Sensor",
vendor="InvenSense",
version=1,
type=1,
maxRange=39.226593 m/s2
resolution=0.0011901855 m/s2
power=0.5 mA
minDelay=5000 µs
```

3.2 Manifest

Le fichier `manifest` déclare l'utilisation de l'accéléromètre :

Fichier

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="mag.linux.android_sensors">

    <uses-feature android:name="android.hardware.sensor.accelerometer"
        android:required="true" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

3.3 Acquisition d'un capteur

Une fois la détection des capteurs réalisée, il est temps de coder l'acquisition des données brutes via l'implémentation de l'interface `SensorEventListener`. Pour cette interface Java, il faut implémenter les méthodes suivantes : `onAccuracyChanged()` et `onSensorChanged()`.

Changement de précision d'un capteur avec `onAccuracyChanged()` : pendant la phase d'acquisition, un capteur peut changer de précision et à ce moment la méthode `onAccuracyChanged(Sensor sensor, int accuracy)` est appelée avec en

Ce document est la propriété exclusive de Johann Locatelli (jacques.thimonier@businessdecision.com)

paramètre la nouvelle précision (`accuracy`) du capteur selon quatre constantes possibles : `SENSOR_STATUS_ACCURACY_LOW`, `SENSOR_STATUS_ACCURACY_MEDIUM`, `SENSOR_STATUS_ACCURACY_HIGH`, ou `SENSOR_STATUS_UNRELIABLE` si la précision est indéterminée. À chaque appel, il convient aussi de vérifier le capteur qui annonce cette nouvelle précision. Le changement de précision peut être lié à une calibration nécessaire du capteur ou à une mesure improbable : le capteur présente un dysfonctionnement.

Le capteur annonce une nouvelle donnée avec `onSensorChanged()` : à chaque nouvel événement du capteur, le système appelle la méthode `onSensorChanged(SensorEvent event)` avec les nouvelles données en paramètre. Les données sont dans l'objet de type `SensorEvent` ainsi que le *timestamp* du système à cet instant.

Dans la plupart des applications, le délai de rafraîchissement par défaut de 200 ms est bien adapté. Pour les jeux, il est préférable d'augmenter la fréquence de rafraîchissement avec `SENSOR_DELAY_GAME` (20 ms), pour d'autres utilisations, `SENSOR_DELAY_UI` propose une fréquence de 60 ms, et `SENSOR_DELAY_FASTEST` (0 ms) donne la vitesse maximale de rafraîchissement.

Dans cette nouvelle version, nous allons à nouveau utiliser `SensorManager` pour accéder à l'accéléromètre : `mAccelerometer = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)` ; deux méthodes de l'interface `SensorEventListener` ont été implémentées, et ces méthodes sont appelées par le système (*callBack*). Vous remarquerez également que `OnResume()` et `Onpause()` ont été surchargées, car il est nécessaire de désenregistrer le gestionnaire de capteur afin de réduire la consommation d'énergie lorsque l'application n'est pas en mesure de suivre les données produites par le capteur, ici l'accéléromètre. Analysons maintenant justement les données produites : la méthode `void onSensorChanged(SensorEvent event)` reçoit un objet de type `SensorEvent` qui contient un tableau de trois `float` représentant les trois axes `(x,y,z)` de l'accéléromètre. Ce tableau peut varier en fonction du capteur qui appelle cette méthode. L'objet `event` contient aussi le temps avec `event.timestamp` qui exprime le temps système en nanosecondes.

Fichier

```
package mag.linux.android_sensors;

...

public class MainActivity extends AppCompatActivity implements
SensorEventListener {
   final String TAG="sensor";
   SensorManager mSensorManager;
   private Sensor mAccelerometer;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        sensorDetection();
        if(mSensorManager==null) {
            mSensorManager = (SensorManager) getSystemService(Context.
SENSOR_SERVICE);
        }
        mAccelerometer = mSensorManager.getDefaultSensor(Sensor.TYPE_
ACCELEROMETER);
   }

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

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

```
        if(mSensorManager!=null) {
            mSensorManager.registerListener(this, mAccelerometer,
SensorManager.SENSOR_DELAY_NORMAL);
        }
    }
    @Override
    protected void onPause() {
        super.onPause();
        if(mSensorManager!=null) {
            mSensorManager.unregisterListener(this);
        }
    }

    @Override
    public final void onAccuracyChanged(Sensor sensor, int accuracy) {
        // Do something here if sensor accuracy changes.
    }

    @Override
    public final void onSensorChanged(SensorEvent event) {
        // Many sensors return 3 values, one for each axis.
        float Ax = event.values[0];
        float Ay = event.values[1];
        float Az = event.values[2];
        // Do something with this sensor value.
          Log.v(TAG, "TimeAcc = " + event.timestamp + " Ax = " + Ax + " " +
"Ay = " + Ay + " " + "Az = " + Az);
    }
...
}
```

NOTE

En observant les logs de l'accéléromètre, on s'aperçoit que si l'équipement au repos est tenu à plat, l'écran vers le ciel, on observe une valeur d'environ `+9.8 m/s2` soit `1g` sur l'axe Y. Au repos (sans bouger votre équipement), le capteur lit bien l'accélération verticale opposée au vecteur représentant le poids.

Voici le log de l'accéléromètre dans logcat :

Fichier

```
Time = 1488810980149390 Ax = 0.094680 Ay = -0.149749 Az = 9.482895
Time = 1488810980215943 Ax = 0.100631 Ay = -0.140228 Az = 9.507889
...
```

NOTE

L'accéléromètre permet de visualiser le phénomène d'apesanteur. Il suffit de laisser tomber avec précaution votre équipement d'une hauteur raisonnable : pendant la chute, l'accéléromètre n'enregistre pas d'accélération.

3.4 Acquisition multi-capteurs

Une application peut utiliser plusieurs capteurs et la méthode `event.sensor.getType()` permet alors d'identifier le type de capteur lors de l'appel à `onSensorChanged(SensorEvent event)`. Ajoutons le gyromètre à notre

Ce document est la propriété exclusive de Johann Locatelli (jacques.thimonier@businessdecision.com)

application : le capteur doit être enregistré auprès du `SensorManager`, la méthode `OnResume()` enregistre le *listener* et `void onSensorChanged(SensorEvent event)` fait la différence entre les appels de l'accéléromètre et gyromètre.

Fichier

```
package mag.linux.android_sensors;
...
public class MainActivity extends AppCompatActivity implements
SensorEventListener {
   final String TAG="sensor";
   SensorManager mSensorManager;
   private Sensor mAccelerometer;
   private Sensor mGyroscope;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
       ...
       mGyroscope = mSensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE);
   }

   @Override
   protected void onResume() {
       super.onResume();
       if(mSensorManager!=null) {
           mSensorManager.registerListener(this, mAccelerometer,
SensorManager.SENSOR_DELAY_NORMAL);
           mSensorManager.registerListener(this, mGyroscope, SensorManager.
SENSOR_DELAY_NORMAL);
       }
   }
...
   @Override
   public final void onSensorChanged(SensorEvent event) {
       if(event.sensor.getType()== Sensor.TYPE_ACCELEROMETER) {
         // TYPE_ACCELEROMETER -> 3 values, one for each axis
           float Ax = event.values[0];
           float Ay = event.values[1];
           float Az = event.values[2];
           // Do something with this sensor value.
           Log.v(TAG, "TimeAcc = " + event.timestamp + " Ax = " + Ax + " " +
"Ay = " + Ay + " " + "Az = " + Az);

       }
       if(event.sensor.getType()== Sensor.TYPE_GYROSCOPE)
       {
           // TYPE_GYROSCOPE -> 3 values, one for each axis
           float Gx = event.values[0];
           float Gy = event.values[1];
           float Gz = event.values[2];
           // Do something with this sensor value.
           Log.v(TAG, "TimeGyro = " + event.timestamp + " Gx = " + Gx + " " +
"Gy = " + Gy + " " + "Gz = " + Gz);
       }
   }
   ...
}
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Modification du `manifest` pour ajouter le gyromètre :

Fichier

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

Voici le log de l'accéléromètre et du gyromètre :

Fichier

```
TimeGyro = 1488811983068660436 Gx = -0.0072784424 Gy = 0.0044403076 Gz =
-9.613037E-4
TimeAcc = 1488811983159114537 Ax = 0.07563782 Ay = -0.16998291 Az = 9.528122
...
```

3.5 Capteur de position

L'utilisation du magnétomètre et de l'accéléromètre permet de déterminer la position de votre équipement par rapport au pôle nord magnétique. Ces deux capteurs sont quasiment toujours présents sur les équipements. Le magnétomètre et l'accéléromètre font partie des capteurs physiques qui fournissent des données brutes.

3.6 Capteur magnétomètre

Le magnétomètre est utilisé comme boussole, il indique le pôle nord magnétique. À chaque mesure, ce capteur renvoie un tableau de `float` avec les valeurs X, Y, Z du champ magnétique terrestre observé. La mesure du champ magnétique terrestre est délicate, car relativement faible, les mesures du capteur sont d'autant plus facilement perturbées par l'environnement électromagnétique, par la présence de matériaux conducteurs. Ce capteur demande souvent une phase de calibration. La plupart des procédures de calibration demandent de réaliser un mouvement en forme de 8. Cependant, si votre environnement électromagnétique est fortement perturbé, le résultat de la calibration restera hasardeux.

Le magnétomètre est maintenant ajouté au niveau de la méthode `onCreate()` et `onResume()` :

Fichier

```
private Sensor mMagneto;
...
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //Gui
        textDirection = (TextView) findViewById(R.id.textViewLocation);

        if(mSensorManager==null) {
            mSensorManager = (SensorManager) getSystemService(Context.SENSOR_
SERVICE);
        }
        mAccelerometer = mSensorManager.getDefaultSensor(Sensor.TYPE_
ACCELEROMETER);
        mGyroscope = mSensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE);
```

Ce document est la propriété exclusive de Johann Locatelli (jacques.thimonier@businessdecision.com)

```
        mMagneto = mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);
    }
...
@Override
protected void onResume() {
    super.onResume();
    if(mSensorManager!=null) {
        mSensorManager.registerListener(this, mAccelerometer, SensorManager.
SENSOR_DELAY_UI);
        mSensorManager.registerListener(this, mGyroscope, SensorManager.SENSOR_
DELAY_UI);
        mSensorManager.registerListener(this, mMagneto, SensorManager.SENSOR_
DELAY_UI);
    }
}
...
if (event.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD) {
        // Valeur du vecteur du champ magnétique (x,y,z)
      magetoSensor[0] = event.values[0];
      magetoSensor[1] = event.values[1];
      magetoSensor[2] = event.values[2];
      Log.v(TAG, "Magneto = " + event.timestamp + " Mx = " + magetoSensor[0] +
" " + "My = " + magetoSensor[1] + " " + "Mz = " + magetoSensor[2]);
}
...
```

Ajout du magnétomètre au `manifest` :

Fichier

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

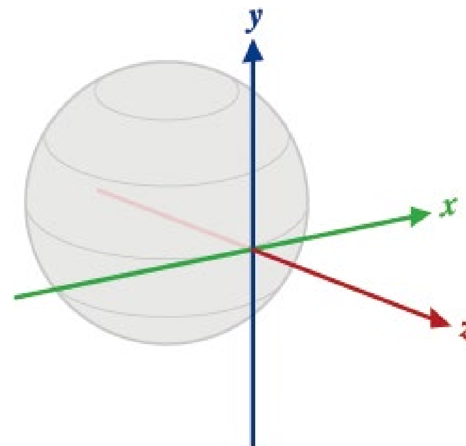
Le magnétomètre retourne trois valeur X, Y, Z qui représentent la mesure du champ magnétique ambiant en µT (micro Tesla) dans le repère du système, dans un objet `event.values[]`. On retrouve `event.values[0]`, `event.values[1]` et `event.values[2]` les projections respectives sur les axes X, Y et Z.

3.7 Calcul de l'orientation - magnétomètre accéléromètre

Nous allons à présent intégrer un système d'orientation basé sur le magnétomètre qui joue le rôle de boussole.

Le système de coordonnées Est-Nord (le nord vers le haut) est défini comme une base orthonormale directe où X pointe vers l'est et Y vers le nord (ils sont tangents au sol) et Z pointe vers le ciel (il est perpendiculaire au sol).

L'équipement se trouve au centre de ce repère. Pour définir sa position par rapport à ce repère, il faut trouver une matrice de rotation qui consiste à aligner l'appareil dans le repère Est-Nord. À partir d'Android API 3, deux primitives réalisent ces calculs : `getRotationMatrix()` et `getOrientation()`.



Coordonnées Est-Nord.

Figure 1

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

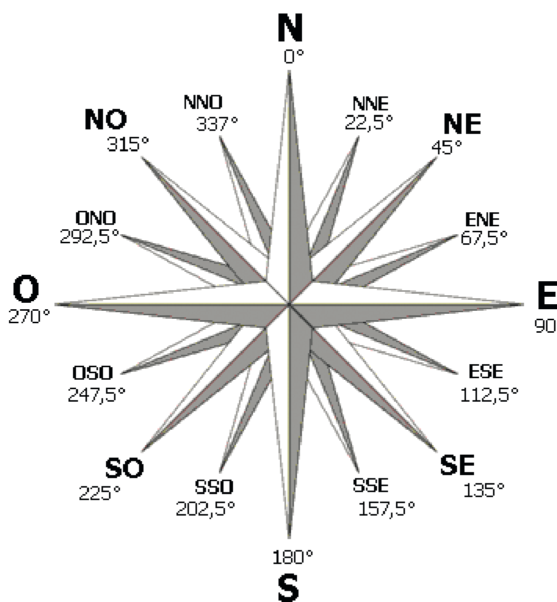
Le calcul se fait en trois étapes :

1 Obtenir une matrice d'inclination `I` et une matrice de rotation `R` représentant l'équipement dans le repère Est-Nord (voir figure 1). Les calculs sont réalisés sur la base des données de l'accéléromètre (argument `gravity`) et du magnétomètre (argument `geomagnetic`) exprimés dans le repère de l'équipement. Ces calculs sont réalisés par `getRotationMatrix()` : `boolean getRotationMatrix (float[] R, float[] I, float[] gravity, float[] geomagnetic)`.

2 Calculer l'orientation de l'équipement dans le repère Est-Nord en fonction de la matrice de rotation `R`. À cette étape, il est possible d'obtenir le nord magnétique. Le résultat est le tableau `values[ ]` dont l'unité est en radian. Calcul de l'orientation : `float[ ] getOrientation (float[ ] R, float[ ] values)`. Le tableau `float[] values` renvoie les informations suivantes :

- `values[0]`: Azimuth (angle de rotation autour de l'axe z, `event.values[0]` - de -180° à 180°). Cette valeur représente l'angle entre l'axe Y de l'appareil et le pôle nord magnétique. En tournant vers le nord, cet angle est de 0°, lorsque l'appareil est orienté vers le sud, cet angle est de 180°. De même, face à l'est, cet angle est de 90°, et lorsqu'il est orienté vers l'ouest, cet angle est de -90°.
- `values[1]`: Tangage (angle de rotation autour de l'axe x, `event.values[1]` - de -180° à 180°). Angle entre un plan parallèle à l'écran de l'appareil et un plan parallèle au sol. Si vous maintenez l'appareil parallèlement au sol avec le bord inférieur le plus proche de vous et inclinez le bord supérieur de l'appareil vers le sol, l'angle de tangage devient positif. En inclinant l'appareil dans la direction opposée, l'angle de tangage devient négatif.
- `values[2]`: Roulis (degrés de rotation autour de l'axe des y, `event.values[2]` - de -90° à 90°). C'est l'angle entre un plan perpendiculaire à l'écran de l'appareil et un plan perpendiculaire au sol. Si vous maintenez l'appareil parallèlement au sol avec le bord inférieur le plus proche de vous et inclinez le bord gauche de l'appareil vers le sol, l'angle de roulis devient positif. En inclinant l'appareil dans la direction opposée, en déplaçant le bord droit de l'appareil vers le sol, l'angle de roulis devient négatif.

3 L'azimut calculé en radian se trouve dans `values[0]`, il faut convertir cette valeur en degré et ramener le résultat dans une échelle lisible dans le repère d'une boussole.



Rose des vents.

Figure 3

Une boussole peut être représentée en 16 intervalles de 22,5°, soit 360°. Chaque intervalle correspond à une direction par rapport au pôle nord géographique (principe de la rose des vents).

L'accéléromètre et le magnétomètre sont enregistrés auprès du `SensorManager` avec la méthode `mSensorManager.getDefaultSensor`. Le délai de rafraîchissement de l'IHM est fixé avec `SENSOR_DELAY_UI`. Lorsque l'application n'est plus active, la méthode `onePause()` utilise `mSensorManager.unregisterListener(this)` pour libérer le *listener* de `mSensorManager`.

L'affichage se fait par la méthode `updateTextDirection(double bearing)` qui détermine alors le cadran de la rose des vents :

Ce document est la propriété exclusive de Johann Locatelli (jacques.thimonier@businessdecision.com)

Fichier

```
package mag.linux.android_sensors;
...
public class MainActivity extends AppCompatActivity implements SensorEventListener
{
  final String TAG="sensor";
  SensorManager mSensorManager;
  private Sensor mAccelerometer;
  // variable gestion magnetometre
  private Sensor mMagneto;
  private float[] mAcceleroTab = new float[3];
  private float azimuth = 0f;
  float RotationMat[] = new float[9];
  float I[] = new float[9];
  float orientation[] = new float[3];
  float magetoSensor[] = new float[3];
  // filtre passe bas
  static final float ALPHA = 0.25f; // si ALPHA = 1 ou 0, le filtre n'est plus actif
  //Gui
  private TextView textDirection;
...
  @Override
  public final void onSensorChanged(SensorEvent event) {
      synchronized (this) {
          if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
              // TYPE_ACCELEROMETER -> 3 values, one for each axis.
              mAcceleroTab[0] = event.values[0];
              mAcceleroTab[1] = event.values[1];
              mAcceleroTab[2] = event.values[2];
              mAcceleroTab = lowPass(event.values.clone(), mAcceleroTab);
          }
          if (event.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD) {
              // Valeur du vecteur du champ magnetique (x,y,z)
              magetoSensor[0] = event.values[0];
              magetoSensor[1] = event.values[1];
              magetoSensor[2] = event.values[2];
              magetoSensor = lowPass(event.values.clone(), magetoSensor);
          }
          RotationMat[0]=0.0f; RotationMat[1]=0.0f; RotationMat[2]=0.0f;
          I[0]=0.0f; I[1]=0.0f; I[2]=0.0f;
          if (SensorManager.getRotationMatrix(RotationMat, I, mAcceleroTab,
magetoSensor)) {
              orientation[0]=0.0f; orientation[1]=0.0f; orientation[2]=0.0f;
              SensorManager.getOrientation(RotationMat, orientation);
              azimuth = (float) Math.toDegrees(orientation[0]); // orientation
              azimuth = (azimuth + 360) % 360;
              Log.d(TAG, "azimuth (deg): " + azimuth);
              updateTextDirection(azimuth);
          }
      }
  }

  private void updateTextDirection(double bearing) {
      int range = (int) (bearing / (360f / 16f));
      String directionTxt = "";
      // Directions
      if (range == 15 || range == 0)
          directionTxt = "N";
      ...
      textDirection.setText("" +((int) bearing)+((char) 176)+" " + directionTxt );
//char 176=degres
  }
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

```
protected float[] lowPass( float[] input, float[] output ) {
    if ( output == null ) return input;
    for ( int i=0; i<input.length; i++ ) {
        output[i] = output[i] + ALPHA * (input[i] - output[i]);
    }
    return output;
  }
}
```

Il est nécessaire de ramener les valeurs calculées dans le repère géographique avec la formule suivante : `azimuth = (azimuth + 360) % 360`.

Les capteurs du type MEMS sont très sensibles et entraînent du bruit dans les mesures. Le bruit est perturbant pour les calculs et le comportement des IHM qui sont rafraîchies au rythme des mesures. Même si la fréquence des mesures est atténuée par le choix du délai du type `SENSOR_DELAY_UI` avec `SensorManager`, le bruit peut être atténué par l'utilisation d'un filtre numérique passe-bas basique de la forme :

```
for i from 1 to n
    y[i] := y[i-1] + α * (x[i] - y[i-1])
```

Le paramètre alpha permet d'atténuer la prise en compte de la nouvelle mesure, ce qui explique le comportement « passe-bas ». Le choix de alpha va dépendre du niveau de « coupure » que l'on veut choisir pour le filtrage. Le filtre est utilisé à chaque mesure de l'accéléromètre ou du magnétomètre :

Fichier

```
...
 mAcceleroTab = lowPass(event.values.clone(), mAcceleroTab);
...
 magetoSensor = lowPass(event.values.clone(), magetoSensor);
```

La méthode pour le filtre est la suivante :

Fichier

```
protected float[] lowPass( float[] input, float[] output ) {
   if ( output == null ) return input;
   for ( int i=0; i<input.length; i++ ) {
      output[i] = output[i] + ALPHA * (input[i] - output[i]);
   }
   return output;
}
```

Modifions maintenant l'IHM. L'IHM doit prendre les différents cas d'utilisation de notre logiciel de géocaching. Nous intégrons dès maintenant les différents *widgets* nécessaires à la géolocalisation et la gestion NFC (voir code sur **https://github.com/frameproject/geocaching**). La figure 2 montre cette IHM.

3.8 Calcul de l'Orientation : fusion de données

Jusqu'à présent, nous avons utilisé des capteurs « réels » de la plateforme : l'accéléromètre, le magnétomètre. Android propose également des capteurs composites qui réalisent de la fusion de données, c'est-à-dire que l'on va utiliser plusieurs capteurs réels de la plateforme pour extraire une information. Cette fusion de données est repré-

Ce document est la propriété exclusive de Johann Locatelli (jacques.thimonier@businessdecision.com)

sentée par un capteur virtuel, c'est-à-dire non physique. Il est nécessaire de prendre quelques précautions lors du choix d'un capteur composite, car celui-ci requiert plusieurs capteurs qui ne sont pas toujours disponibles dans l'équipement. La phase de découverte des capteurs physiques présents doit déterminer le comportement de l'application.

Dans cette section, nous allons découvrir comment utiliser le capteur « Vecteur Rotation » (*"Rotation vector"*) qui utilise l'accéléromètre, le magnétomètre et le gyromètre si celui-ci est présent dans l'équipement. Son principe est de définir la position de l'équipement dans les coordonnées du repère Est-Nord. C'est le capteur le plus précis pour la gestion de l'attitude du système. Il est donc inutile de conserver le code précédent, seul le capteur « vecteur rotation » nous permettra de réaliser la boussole.

Le capteur est enregistré auprès du `sensorManager` de la même façon qu'un capteur « réel ». Les calculs d'orientation sont exécutés à chaque appel de la méthode `onSensorChanged(SensorEvent event)`. Le délai de rafraîchissement de type `SENSOR_DELAY_UI` est suffisant pour ce type d'utilisation.

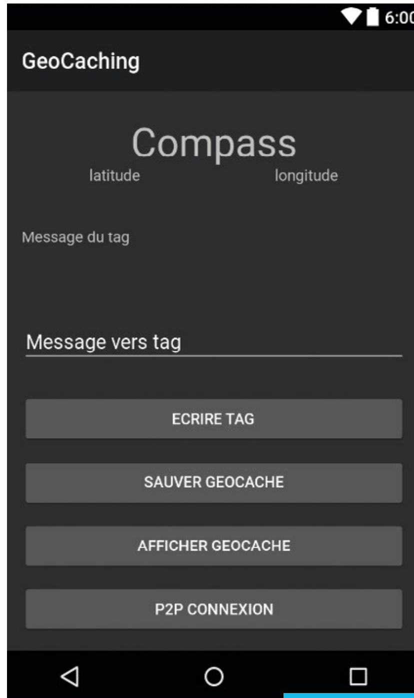


Figure 2
IHM de l'activité principale.

Fichier

```
package mag.linux.android_sensors;

...

public class MainActivity extends AppCompatActivity implements
SensorEventListener {
 final String TAG="sensor";
 SensorManager mSensorManager;
 private Sensor mRotationVector;
 // gestion vecteur rotation
 float[] orientation = new float[3];
 float[] rotateMat = new float[9];

 //Gui
 private TextView textDirection;

 @Override
 protected void onCreate(Bundle savedInstanceState) {
         ...
        mRotationVector = mSensorManager.getDefaultSensor(Sensor.TYPE_
ROTATION_VECTOR);
 }
...

 @Override
 public final void onSensorChanged(SensorEvent event) {
     synchronized (this) {
         if( event.sensor.getType() == Sensor.TYPE_ROTATION_VECTOR ){
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

```
                // calcul matrice de rotation
                SensorManager.getRotationMatrixFromVector( rotateMat, event.
values );
                // calcul azimuth en degres
                updateTextDirection(( Math.toDegrees( SensorManager.
getOrientation( rotateMat, orientation )[0] ) +360 ) % 360);
          }
      }
}
...
}
```

L'utilisation des capteurs est signalée par la directive `uses-feature` dans le `manifest` :

Fichier

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="mag.linux.android_sensors">
   <uses-feature android:name="android.hardware.sensor.accelerometer"
       android:required="true" />
   <uses-feature android:name="android.hardware.sensor.compass"
       android:required="true" />
   <uses-feature android:name="android.hardware.sensor.gyroscope"
       android:required="true" />
   <application
       android:allowBackup="true"
       android:icon="@mipmap/ic_launcher"
       android:label="@string/app_name"
       android:supportsRtl="true"
       android:theme="@style/AppTheme">
       <activity android:name=".MainActivity">
           <intent-filter>
               <action android:name="android.intent.action.MAIN" />
               <category android:name="android.intent.category.LAUNCHER" />
           </intent-filter>
       </activity>
   </application>
</manifest>
```

À ce stade du développement, l'application est capable d'afficher une boussole électronique selon le repère de la rose des vents. Nous ajouterons par la suite les coordonnées GPS. ■

Synthèse

Pour commencer cet article, nous avons abordé les capteurs physiques de la plateforme et réalisé une première version de notre boussole électronique. Nous avons vu la simplicité d'utilisation de l'API qui permet en un temps record l'utilisation d'un accéléromètre et d'un magnétomètre. Ensuite, à peine plus complexe, l'utilisation des capteurs composites est vraiment simplifiée par l'API. Nous avons d'ailleurs réalisé notre boussole grâce au capteur « Vector Rotation » avec moins de lignes de code que la version précédente basée sur les capteurs réels. L'application est très stable et indique bien la direction selon les points cardinaux.

Ce document est la propriété exclusive de Johann Locatelli (jacques.thimonier@businessdecision.com)

ACTUELLEMENT DISPONIBLE
GNU/LINUX MAGAZINE N°206

FAITES DU JEU
DUKE NUKEM 3D
UN OUTIL SYSADMIN !

NE LE MANQUEZ PAS
CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :
http://www.ed-diamond.com

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

UTILISEZ DES TAGS NFC

Il est possible d'effectuer une communication sur courte distance grâce à la technologie radio NFC. Nous allons découvrir dans cet article comment l'employer sous Android.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Android 2.3 (API niveau 9) a introduit les API NFC (*Near Field Communication*). Cette technologie radio permet de transmettre de petites quantités de données sur de courtes distances de l'ordre de 4 à 5 cm en fonction de la sensibilité des systèmes. Cette technique permet une communication entre deux systèmes équipés de système NFC ou encore entre un système NFC et un tag NFC. Un tag est un équipement passif qui autorise une lecture et une écriture d'une mémoire non volatile. L'utilisation de systèmes NFC est assez commune aujourd'hui, micropaiement, domotique, capture de données, on pourrait citer de nombreux cas d'utilisation.

1. BORNE NFC M24LR-DISCOVERY

Dans notre système de géocaching, nous allons utiliser un tag NFC et le lecteur NFC d'un smartphone pour inscrire un message à l'attention des participants. Le tag NFC retenu est le **M24LR-Discovery** , un kit de découverte NFC qui comprend un lecteur et un tag. Vous pouvez vous procurer ce kit pour un prix très raisonnable de l'ordre de 20 euros. Le M24LR-Discovery utilise la technologie NfcV - ISO 15693, celle-ci est compatible avec les équipements Android. Le kit contient deux éléments : une carte de transceiver RF et le M24LR-Discovery.

STMicroelectronics met à disposition une application gratuite pour smartphone, celle-ci est disponible sur **Google Play** (mots clés : « nfc reader »). L'application vous permet de découvrir l'utilisation du kit M24LR-Discovery en lecture et en écriture selon plusieurs formats. Le tag M24LR04E possède aussi des fonctionnalités non standards que l'API Android ne couvre pas. Cependant le transceiver NFC du smartphone Android peut envoyer des suites de commandes spécifiques afin d'exploiter l'ensemble des fonctionnalités du tag.

2. ANDROID ET NFC

Les équipements Android supportent de nombreuses technologies de tag NFC. Il en est de même pour le SDK qui offre de nombreuses possibilités :

- mode lecture/écriture : lecture et écriture d'un tag NFC passif dans différentes technologies. Sachant qu'il existe des technologies plus ou moins complexes en fonction notamment des critères de capacité, sécurité, type de codage ;
- mode P2P : échange de données entre deux équipements non passifs, connu sous le nom d'Android Beam ;
- mode émulation de tag : cette configuration moins connue permet à un équipement non passif de se transformer en tag « passif » et d'autoriser une lecture par un lecteur externe.

NFC utilise le couplage inductif entre les appareils NFC et fonctionne avec un champ électromagnétique à 13,56 MHz, cette fréquence du spectre HF radioélectrique est sans licence. Il existe plusieurs technologies de tag NFC. Dans le cadre du projet de géocaching, nous utiliserons celle du tag M24LR04E de classe NfcV ISO 15693.

Android propose une API spécifique pour chaque type de technologie. L'API de type NfcV ISO 15693 se trouve à cette adresse : **https://developer.android.com/reference/android/nfc/tech/NfcV.html**.

Le type 5 (NFC-V) a été adopté récemment par NFC Forum. Ce type se base sur la norme ISO/IEC 15693. Le tag peut contenir plus de 64 Kbytes de mémoire, et supporte une vitesse de 26.48 kbit/s avec un système d'anti-collision.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

NORME NDEF

NFC Forum NDEF (*NFC Data Exchange Format*) définit un format de données normalisé pour le stockage des données dans un tag NFC. Ce format est utilisable sur toutes les technologies de tag (de type 1 à type 5). L'utilisation d'un standard de données facilite les échanges de données et le développement des applications. Il n'est cependant pas obligatoire, il est possible de développer un format de données propriétaire. Dans ce dernier cas, les API standards ne seront pas utilisables.

Le contenu d'un message NDEF peut être le suivant : enregistrement de texte simple, URI, affiche intelligente, signature, VCard (un format de carte de commerce électronique standard), association Bluetooth ou Wifi.

La mémoire M24LR utilise le format TLV dont la structure générique permet de stocker les données d'un enregistrement. Le format TLV est composé de trois champs : *Type field* (T), *Length field* (L), *Value field* (V), qui structurent un message NDEF. Enfin, NDEF peut être de type `short`, c'est-à-dire contenir un seul enregistrement ou encore un type `long` avec plusieurs enregistrements.

Lorsque le capteur électronique NFC d'un smartphone Android détecte un tag, un *intent* est diffusé dans le système. Ces *intents* sont de trois types :

- `ACTION_NDEF_DISCOVERED` : un tag contenant un message NDEF a été détecté, un *intent* lance une *Activity* afin de réaliser un traitement. Cela signifie aussi qu'au moins une *Activity* utilise un *intent-filter* du type émis afin de réaliser le traitement. Cet *intent* est prioritaire, son comportement est assez générique et indépendant des technologies de tag NFC.
- `ACTION_TECH_DISCOVERED` : le précédent *intent* n'a pas pu lancer une activité alors le système diffuse un nouvel *intent*. Cet *intent* peut également être directement lancé si le tag qui contient un message NDEF ne possède pas un type MIME reconnu ou encore une URI. Cet *intent* est également diffusé si le tag ne possède pas de message NDEF, la technologie utilisée doit être connue.
- `ACTION_TAG_DISCOVERED` : les deux précédentes tentatives sont infructueuses alors le système lance un *intent* de ce type.

Le traitement des *Intents* de type NFC dépend aussi d'autres facteurs (mauvaise lecture, erreur dans le tag, etc.) qu'il est nécessaire de prendre en compte pour une application robuste à la lecture des tags NFC.

RAPPEL

***Intent* (diminutif de *Intentions*) : L'*intent* est un message asynchrone qui circule à l'intérieur du dispositif Android, pour avertir les applications de divers événements, par exemple : une carte SD a été insérée, l'état de la batterie, un SMS est arrivé. Il est également possible de répondre à ces messages et de diffuser votre propre message. Enfin, les *intents* permettent également le lancement d'autres applications en recherchant l'application la plus adaptée à la demande.**

3. TYPE DE FORMAT TNF

Il existe plusieurs formats de tag NFC décrits par la valeur de TNF (3 octets). Chaque format décrit ensuite plus précisément son contenu selon la valeur du champ `TYPE`. Par exemple, si nous choisissons un format du type `TNF_WELL_KNOWN` (type MIME ou URI en fonction de la définition de type d'enregistrement) alors le contenu sera du type RTD.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Les messages possèdent un formatage bien précis, si l'on veut placer un message de type texte dans le tag NFC, il faut utiliser :

- TNF TYPE : `TNF_WELL_KNOWN` ;
- RTD (*Record Type Definition*) pour `TNF_WELL_KNOWN` : `RTD_TEXT` (type MIME pour le texte).

Ce format sera utilisé pour définir nos propres tags au moment de l'enregistrement d'une géocache.

4. CONFIGURATION DU MANIFEST

Autorisation pour le capteur NFC :

Fichier

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

Version minimum du SDK (la version 9 possède un support NFC assez limité) :

Fichier

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

Filtrage pour les applications sur Google Play :

Fichier

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

5. FILTRAGE DES INTENTS

Le filtrage des *intents* est un point important pour le processus de découverte d'un tag NFC. Nous avons vu précédemment que le système produit un intent de type `ACTION_NDEF_DISCOVERED`, ou `ACTION_TECH_DISCOVERED` et en dernier lieu `ACTION_TAG_DISCOVERED`. Le premier *intent* semblerait suffisant sauf que certains systèmes Android ne sont pas tout à fait compatibles avec le technologie NFC-V ISO 15693, de ce fait le système peut envoyer un *intent* de type `ACTION_TECH_DISCOVERED` ou encore `ACTION_TAG_DISCOVERED`, ce dernier peut être diffusé en cas de mauvaise lecture du tag : distance trop importante, rupture de la communication. Nous allons configurer notre application pour répondre à ces comportements :

Fichier

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="mag.linux.android_sensors">

   <uses-feature android:name="android.hardware.sensor.accelerometer"
       android:required="true" />
   <uses-feature android:name="android.hardware.sensor.compass"
       android:required="true" />
   <uses-feature android:name="android.hardware.sensor.gyroscope"
       android:required="true" />
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

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

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

        <activity android:name=".MainActivity">
            <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="text/plain" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.nfc.action.TECH_DISCOVERED"/>
                <category android:name="android.intent.category.DEFAULT"/>
            </intent-filter>

            <meta-data android:name="android.nfc.action.TECH_DISCOVERED"
android:resource="@xml/nfc_tech_filter"/>
        </activity>
    </application>
</manifest>
```

Dans le répertoire `res/`, créez un répertoire `xml` et placez-y le fichier `nfc_tech_filter.xml`. Ce fichier contient les technologies de tag pour lesquelles l'application peut recevoir un *intent* :

Fichier

```
<resources xmlns:xliff="urn:oasis:names:tc:xliff:document:1.2">
   <tech-list>
       <tech>android.nfc.tech.NfcV</tech>
       <tech>android.nfc.tech.Ndef</tech>
   </tech-list>
</resources>
```

6. INTÉGRATION LECTEUR NFC

L'intégration du capteur NFC dans l'application passe par l'intégration de l'API NFC.

Le processus standard de résolution d'*intent* permet de définir l'application qui doit être utilisée. Si plusieurs applications sont déclarées pour recevoir un type d'*intent* alors le système proposera une liste d'applications susceptibles de réaliser le traitement. Android propose ce comportement par défaut si aucune application n'est lancée, ou si l'applica-

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

tion qui est lancée n'est pas prévue pour recevoir ce type d'*intent*. Cependant, lorsqu'une application est déjà lancée, le système peut à nouveau demander à l'utilisateur le choix d'une application même si l'application lancée permet de réaliser le traitement ! Pour pallier à ce comportement, il faut définir l'activité en cours comme étant prioritaire, ce qui lui permet d'être le receveur par défaut de l'*intent* en question :

Fichier

```
package mag.linux.android_sensors;
...
public class MainActivity extends AppCompatActivity implements
SensorEventListener {
   ...
   // NFC
   private PendingIntent mNfcPendingIntent;
   private NfcAdapter mNfcAdapter;
   NfcTagManager mNfcIntentManager;
  ...
   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main);
...
       //NFC
       mNfcAdapter = NfcAdapter.getDefaultAdapter(this);
       mNfcIntentManager = new NfcTagManager(this.getApplicationContext());
   }

   @Override
   protected void onResume() {
       super.onResume();
...
       // NFC
       mNfcPendingIntent = PendingIntent.getActivity(this, 0,new
Intent(this, getClass()).addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP), 0);
       mNfcAdapter.enableForegroundDispatch(this, mNfcPendingIntent, null,
null);
   }

   @Override
   protected void onPause() {
      ...
       // NFC
       mNfcAdapter.disableForegroundDispatch(this);
   }
...
   private void updateTextDirection(double bearing) {
       ...
   }

   @Override
   public void onNewIntent(Intent intent) {
       String action = intent.getAction();
       if (NfcAdapter.ACTION_TAG_DISCOVERED.equals(action) ||
               NfcAdapter.ACTION_TECH_DISCOVERED.equals(action) ||
               NfcAdapter.ACTION_NDEF_DISCOVERED.equals(action)) {
           Log.v(TAG, "***** " + intent.getAction());
           mNfcIntentManager.computeNfcIntent(intent);

       }
   }
  ...
}
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

7. CAPTURE DES INTENT NFC

La capture des *Intent* se fait avec la méthode *onNewIntent(Intent intent)* qui reçoit les *intent* filtrés à la demande du `manifest` et le transmet à un objet de type `NfcTagManager`.

La classe `NfcTagManager` permet de réaliser l'ensemble des traitements :

- `protected void computeNfcIntent(Intent intent)` : traitement des trois types d'*Intent* ;
- `boolean NdefComputeData (Parcelable[] rawMsgs)` : recherche des messages et enregistrement NDEF ;
- `private String computeNdefRecord(NdefRecord record)` : permet de récupérer les informations d'encodage, de langue et extraction des données utiles ;
- `private String nfcInfo(NdefRecord record)` : affichage des informations du tag.

Voici le code de la classe `NfcTagManager` :

Fichier

```
package mag.linux.android_sensors;

import android.nfc.FormatException;
import android.nfc.NdefMessage;
import android.nfc.NdefRecord;
import android.nfc.NfcAdapter;
import android.nfc.Tag;
import android.nfc.tech.Ndef;
...

public class NfcTagManager  {
   final String TAG_NFC="sensor";
   Context mContext;

     NfcTagManager(Context context) {
       mContext = context;
   }


   protected String computeNfcIntent (Intent intent) {
       Parcelable[] rawMsgs = null;
       String action = intent.getAction();

       Log.v(TAG_NFC, "computeNfcIntent = " + action);

       if (NfcAdapter.ACTION_NDEF_DISCOVERED.equals(action)) {
           rawMsgs = intent.getParcelableArrayExtra(NfcAdapter.EXTRA_NDEF_
MESSAGES);

       } else if (NfcAdapter.ACTION_TECH_DISCOVERED.equals(action)) {
           rawMsgs = intent.getParcelableArrayExtra(NfcAdapter.EXTRA_TAG);
       }
       else if (NfcAdapter.ACTION_TAG_DISCOVERED.equals(action)) {
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

```
            Tag tagFromIntent = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);

            for (String tech : tagFromIntent.getTechList()) {
                Log.v(TAG_NFC, "computeNfcIntent = " + tech);
            }
        }

        if(rawMsgs != null && rawMsgs.length > 0) {
           return NdefComputeData(rawMsgs);
        }
        else {
            Toast.makeText(mContext, "This NFC tag has no NDEF data.", Toast.
LENGTH_LONG).show();
        }
        return null;
    }

 String NdefComputeData (Parcelable[] rawMsgs) {
        NdefMessage msgs[] = new NdefMessage[rawMsgs.length];

        for (int i = 0; i < rawMsgs.length; i++) {
            msgs[i] = (NdefMessage) rawMsgs[i];
            NdefRecord[] records = msgs[i].getRecords();

            for (NdefRecord ndefRecord : records) {
                if (ndefRecord.getTnf() == NdefRecord.TNF_WELL_KNOWN && Arrays.
equals(ndefRecord.getType(), NdefRecord.RTD_TEXT)) {
                    try {

                        nfcInfo(ndefRecord);

                        return computeNdefRecord(ndefRecord);
                    } catch (UnsupportedEncodingException e) {
                        e.printStackTrace();
                    }
                }
            }
        }
        return null;
    }

private String computeNdefRecord(NdefRecord record) throws
UnsupportedEncodingException {
        byte[] payload = record.getPayload();

        // bit_7 defines encoding - mask 1000 0000
        String textEncoding = ((payload[0] & 128) == 0) ? "UTF-8" : "UTF-16";

        // bit_5..0 length of IANA language code - mask 0011 1111
        int languageCodeLength = payload[0] & 0063;

        if (payload == null) return ;

        int languageCodeLength = payload[0] & 0063;
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

```
        byte[] tagInfo = record.getType();

        // Most classical Type. @see NdefRcord class for more type
        if(tagInfo[0] == NdefRecord.RTD_TEXT[0]) {
            Log.v(TAG_NFC, "TYPE = RTD_TEXT ");
        }
        else if(tagInfo[0] == NdefRecord.RTD_URI[0]) {
            Log.v(TAG_NFC, "TYPE = RTD_URI ");
        }
        else if(tagInfo[0] == NdefRecord.RTD_SMART_POSTER[0] && tagInfo[1]
== NdefRecord.RTD_SMART_POSTER[1]) {
            Log.v(TAG_NFC, "TYPE = RTD_SMART_POSTER ");
        }

        Log.v(TAG_NFC,  "MimeType=" +  record.toMimeType());

        byte []id = record.getId();

        if(id.length>0) {
            for (byte b : id) {
                Log.i(TAG_NFC, "ID = " + String.format("0x%20x", b));
            }
        }
        else {
            Log.v(TAG_NFC,  "ID= Not ID defined" );
        }

        // TNF
        Log.v(TAG_NFC,  "TNF=" +  record.getTnf());

        switch (record.getTnf()) {
            // Debug
            case NdefRecord.TNF_EMPTY:
                Log.v(TAG_NFC, "TNF = TNF_EMPTY ");
                break;
            ...
            default: Log.v(TAG_NFC, "TNF = not detected ");
                break;
        }

        int len = payload.length - languageCodeLength - 1;

        Log.v(TAG_NFC,  "TEXT LENGHT=" +  len );
        Log.v(TAG_NFC, new String(payload, 1, languageCodeLength, "US-
ASCII"));
    }

    private static StringBuilder bytesToString(byte[] bs) {
        StringBuilder s = new StringBuilder();
        for (byte b : bs) {
            s.append(String.format("%02X", b));
        }
        return s;
    }
}
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Lors des tests, en fonction de l'état du tag voici les logs de l'application (dans le Logcat) :

➪ Tag vide ou non formaté :

Fichier
```
android.nfc.action.TAG_DISCOVERED
computeNfcIntent = android.nfc.action.TAG_DISCOVERED
computeNfcIntent = android.nfc.tech.NfcV
computeNfcIntent = android.nfc.tech.Ndef
```

➪ Tag avec message NFC :

Fichier
```
android.nfc.action.NDEF_DISCOVERED
computeNfcIntent = android.nfc.action.NDEF_DISCOVERED
Record =NdefRecord tnf=1 type=54 payload=02667268656C6C6F206E6663
TYPE = RTD_TEXT
MimeType=text/plain
ID= Not ID defined
TNF=1
TNF = TNF_WELL_KNOWN
TEXT ENCODING = UTF-8
TEXT LENGHT=9
fr
Payload = hello nfc
```

8. ÉCRITURE D'UN TAG NFC

Nous avons fait connaissance avec les tags NDEF, reste maintenant l'écriture d'un tag. Pour écrire un tag, il faut obtenir un objet de type `Tag` à partir d'un *Intent*. Lorsqu'un tag est détecté dans `public void onNewIntent(Intent intent)`, l'objet de type `Tag` est extrait à partir de l'*Intent*. `writeTag(...)` est appelé à la demande de l'utilisateur.

Dans la classe `NfcTagManager` se trouve la méthode `public NdefRecord createTextRecord(String payload, Locale locale, boolean encodeInUtf8)` qui permet de créer un enregistrement `RTD_TEXT` conforme à NDEF. Cette méthode utilise un encodage UTF-8 (`boolean encodeInUtf8 = true`), ou UTF-16.

Fichier
```
...
public NdefRecord createTextRecord(String payload, Locale locale, boolean
encodeInUtf8) {
byte[] langBytes = locale.getLanguage().getBytes(Charset.forName("US-
ASCII"));//US_ASCII - Seven-bit ASCII
    Charset utfEncoding = encodeInUtf8 ? Charset.forName("UTF-8") :
Charset.forName("UTF-16");
    byte[] textBytes = payload.getBytes(utfEncoding);
    int utfBit = encodeInUtf8 ? 0 : (1 << 7);
    char status = (char) (utfBit + langBytes.length);
    byte[] data = new byte[1 + langBytes.length + textBytes.length];
    data[0] = (byte) status;
    System.arraycopy(langBytes, 0, data, 1, langBytes.length);
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

```
    System.arraycopy(textBytes, 0, data, 1 + langBytes.length, textBytes.
length);
    NdefRecord record = new NdefRecord(NdefRecord.TNF_WELL_KNOWN,
            NdefRecord.RTD_TEXT, new byte[0], data);
    return record;
}
...
boolean writeTag(Tag tag, String data) throws IOException, FormatException,
java.lang.IllegalStateException {
    if(tag==null || data==null || data.trim().length()==0) return false;

    NdefRecord relayRecord = createTextRecord(data, Locale.FRENCH, true);
    NdefMessage message = new NdefMessage(new NdefRecord[] {relayRecord});
    Ndef ndef = Ndef.get(tag);

    if(ndef != null) {
        try {
            if(!ndef.isConnected()) {
                ndef.connect();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

        // Make sure the tag is writable
        if(!ndef.isWritable()) {
            Toast.makeText(mContext, "NFC NOT WRITTABLE", Toast.LENGTH_
LONG).show();
            return false;
        }

        // Check if there's enough space on the tag for the message
        int size = message.toByteArray().length;
        if(ndef.getMaxSize() < size) {
            Toast.makeText(mContext, "NO SPACE, AVAILABLE MEMORY : "+ ndef.
getMaxSize() + " MESSAGE SIZE : " + size, Toast.LENGTH_LONG).show();
            return false;
        }

        // Write the data to the tag
        try {
            if(ndef.isConnected()) {
                ndef.writeNdefMessage(message);
                ndef.close();
                Toast.makeText(mContext, "NFC TAG WRITE OK", Toast.LENGTH_
LONG).show();

                Log.v(TAG_NFC,  "WRITEN PAYLOAD SIZE = " +data.length());
                Log.v(TAG_NFC,  "TOTAL MSG SIZE = " +message.
getByteArrayLength());
            }
            else {
                Toast.makeText(mContext, "NFC NOT CONNECTED", Toast.LENGTH_
LONG).show();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        else {
         // try to format the Tag in NDEF
         NdefFormatable nForm = NdefFormatable.get(tag);
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

```
            if (nForm != null) {
              nForm.connect();
              nForm.format(message);
              nForm.close();
            }
          }
          return true;
        }
        return false;
    }
```

Un tag NFC n'est pas forcément formaté au format NDEF, c'est le cas du kit M24LR-Discovery. Lors de la première utilisation, un formatage est réalisé avec un objet de type `NdefFormatable`, puis à partir de la deuxième écriture, un message NDEF pourra s'inscrire dans la mémoire.

Le test en écriture permet d'inscrire dans la mémoire un message `RTD_TEXT` conforme aux spécifications NDEF **[5-7]**. La relecture par un autre logiciel permet de vérifier le contenu. Vous pouvez revérifier éventuellement avec l'application de chez STMicroelectronics en mode NDEF.

Écriture d'un tag NFC avec NDEF :

Fichier

```
Payload = hello Linux mag
WRITEN PAYLOAD SIZE = 15
TOTAL MSG SIZE = 22
```

Lecture d'un tag NFC avec NDEF :

Fichier

```
android.nfc.action.NDEF_DISCOVERED
Record =NdefRecord tnf=1 type=54 TYPE = RTD_TEXT
MimeType=text/plain
ID= Not ID defined
TNF=1
TNF = TNF_WELL_KNOWN
TEXT ENCODING = UTF-8
TEXT LENGHT=15
fr
Payload = hello Linux mag
```

Synthèse

Le capteur radio NFC est un peu particulier, mais sa mise en œuvre est d'une grande facilité. Le point le plus difficile se situe au niveau de la norme NFC dont la lecture peut être un peu fastidieuse, mais se révèle indispensable pour comprendre le fonctionnement. Ici nous avons abordé uniquement les tags de type « text », mais il existe bien d'autres formats que vous pouvez découvrir suite à cet exemple.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

LOCALISEZ LES CACHES À L'AIDE DE LEURS COORDONNÉES GPS ET STOCKEZ L'INFORMATION

Pour déterminer la position d'une géocache, il faut lui associer des coordonnées GPS. Nous allons donc améliorer notre application en utilisant le GPS et en enregistrant les géocaches au format XML.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

La géolocalisation GPS vient en complément du premier système de géolocalisation basé sur la boussole. L'objectif est d'associer une latitude et une longitude à une Géocache.

1. INTÉGRATION GPS

L'utilisation du GPS requiert des permissions spécifiques :

Fichier

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

La classe `GPSManager` implémente l'interface `LocationListener`. La méthode `public void onLocationChanged(Location location)` est appelée à chaque nouvelle position GPS. D'autres méthodes de l'interface `LocationListener` sont également implémentées pour la gestion des comportements du capteur GPS. Il est possible de les exploiter pour garantir un fonctionnement plus stable de l'application.

Fichier

```
package mag.linux.android_sensors;

import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
...

public class GPSManager implements LocationListener {
 public static final String TAG="TAG_GPS";
 private LocationManager locationManager;
 private Location currentLocation;
 private double bearing = 0;
 private static final int LOCATION_MIN_TIME = 30 * 1000;
 // location min distance
 private static final int LOCATION_MIN_DISTANCE = 15;
 public static final String FIXED = "FIXED";
 MainActivity main;
 double lat;
 double lgt;
 Context mContext;

 GPSManager(MainActivity myMain, Context myContext) {
     mContext = myContext;
     main = myMain;
     // location manager from system service
     locationManager = (LocationManager) mContext.getSystemService(Context.
LOCATION_SERVICE);
     try {
         // request location data
         locationManager.requestLocationUpdates(LocationManager.GPS_
PROVIDER, LOCATION_MIN_TIME, LOCATION_MIN_DISTANCE, this);

         // get last known position
         Location gpsLocation = locationManager.getLastKnownLocation(Locati
onManager.GPS_PROVIDER);
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

```
        if (gpsLocation != null) {
                currentLocation = gpsLocation;
        } else {
                // try with network provider
                Location networkLocation = locationManager.getLastKnownLo
cation(LocationManager.NETWORK_PROVIDER);

            if (networkLocation != null) {
                currentLocation = networkLocation;
            } else {
                // Fix a position
                currentLocation = new Location(FIXED);
                currentLocation.setAltitude(1);
                currentLocation.setLatitude(43.77409084);
                currentLocation.setLongitude(1.43368426);
            }
            // set current location
            onLocationChanged(currentLocation);
        }
    } catch (SecurityException e) {
        e.printStackTrace();
        }
 }
 public double getLatitude() {
        return lat;
 }
 public double getLongitude() {
        return lgt;
 }
 @Override
 public void onLocationChanged(Location location) {
        currentLocation = location;

          lat = location.getLatitude();
         lgt = location.getLongitude();
        main.displayGeo(String.valueOf(lat), String.valueOf(lgt));
        Log.v(TAG, "lat = " + String.valueOf(lat) + "   long = " +
String.valueOf(lgt));
 }
...
 public boolean isGPSEnabled(Context mContext) {
        LocationManager locationManager = (LocationManager) mContext.
getSystemService(Context.LOCATION_SERVICE);
        return locationManager.isProviderEnabled(LocationManager.GPS_
PROVIDER);
 }
}
```

La classe principale est légèrement impactée avec une méthode qui affiche les coordonnées GPS dans le *layout* `activity_main` :

Fichier

```
public class MainActivity extends Activity implements SensorEventListener {
...
static private TextView textLat;
static private TextView textLong;

@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

```
...
textLat = (TextView) findViewById(R.id.textViewLat);
textLong = (TextView) findViewById(R.id.textViewLong);
...
}
public static void displayGeo(String lat, String lgt) {
   textLat.setText(lat);
   textLong.setText(lgt);
}
...
}
```

Avant de lancer l'application, vérifiez dans les paramètres systèmes si le GPS est bien lancé : ***Paramètres > Localisation > Autorisé***. Si la première position n'est pas connue alors une position par défaut peut être utilisée.

2. SAUVEGARDE DES GÉOCACHES

La sauvegarde est réalisée dans un format XML simple afin de faciliter l'échange en P2P avec un support Bluetooth. Le format XML nous permet de réaliser un bon compromis entre le poids et la simplicité de codage.

Le format du fichier xml permet de définir quatre champs pour une géocache. La classe `StoreManager` propose les opérations standards : création du fichier, ajout, suppression d'un point de géolocalisation. La classe `GeoPoint` représente une structure de données de type `<geo>...</geo>`.

Voici le format des points de géolocalisation :

Fichier

```
<data>
    <geo>
         <nom>geopoint1</nom>
         <lat>1.36</lat>
         <long>4.32</long>
         <info>mes infos</info>
   </geo>
...
</data>
```

La classe `GeoPoint` ne présentant que peu d'intérêt par rapport à la programmation Android, vous la trouverez directement sur **https://github.com/frameproject/geocaching**.

Par défaut, les fichiers de stockage sont créés dans le répertoire `files` à partir du *package* de l'application. Cet emplacement correspond à la mémoire interne du smartphone ou de la tablette. Il est également possible de l'enregistrer sur un support externe comme la carte SD (à condition d'avoir un système qui propose une carte SD). Vu que le volume des fichiers est assez faible, la mémoire interne peut sans aucun problème assurer ce stockage.

La classe `DataManager` est responsable du stockage des géocaches et de la restitution des données. Le constructeur reçoit le contexte de l'activité afin de créer un fichier `Data.xml` dans le répertoire de l'application. Le fichier `Other.xml` servira à stocker les géocaches échangées en P2P.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Voici le constructeur **DataManager** :

Fichier

```
package mag.linux.android_sensors;
import java.io.File;
...
public class DataManager {
   static final String TAG="XML";
   static final String NODE_GEO = "GEO";
   static final String xmlFile = "Data.xml";
   static final String xmlFileOther = "Other.xml";
   Context mContext;
   Document doc;

   DataManager(Context context) {
       mContext = context;
       String filePath = mContext.getFilesDir().getPath().toString() + "/" +
xmlFile;
       File file = new File(filePath);
       if(!file.exists()){
           init();
       }

       try {
           prettyPrint(getDocument(xmlFile));
           prettyPrint(getDocument(xmlFileOther));
       } catch (Exception e) {
           e.printStackTrace();
       }
   }
...
```

La méthode **void prettyPrint(Document xml)** permet simplement d'afficher le contenu d'un fichier XML pendant la phase de débug.

Le fichier **Data.xml** est formaté par la méthode **init()** qui insère le premier niveau de la structure de données XML :

Fichier

```
...
void init()
{
   DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
   DocumentBuilder dBuilder;
   try {
       dBuilder = dbFactory.newDocumentBuilder();
       Document doc = dBuilder.newDocument();
       //add elements to Document
       Element rootElement = doc.createElement("data");
       //append root element to document
       doc.appendChild(rootElement);
       //for output to file, console
       TransformerFactory transformerFactory = TransformerFactory.newInstance();
       Transformer transformer = transformerFactory.newTransformer();
       //for pretty print
       transformer.setOutputProperty(OutputKeys.INDENT, "yes");
       DOMSource source = new DOMSource(doc);
       //write to file
       String filePath = mContext.getFilesDir().getPath().toString() + "/" +
xmlFile;
       StreamResult file = new StreamResult(new File(filePath));
       //write data
       transformer.transform(source, file);
   } catch (Exception e) {
       e.printStackTrace();
   }
}
...
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@pressdevision.com)

`getContent()` renvoie le contenu du fichier `Data.xml` sous forme d'un tableau de `GeoPoint` :

Fichier

```
...
public GeoPoint [] getContent(String myXmlFile) {
   AssetManager manager = mContext.getAssets();
   InputStream stream;
   GeoPoint [] tabGeo=null;

   try {
       stream = mContext.openFileInput(myXmlFile);
       doc =getDocument(stream);
       // Get elements
       NodeList nodeList = doc.getElementsByTagName(NODE_GEO);
       if(nodeList.getLength() >0) {
           tabGeo = new GeoPoint[nodeList.getLength()];
           Log.v(TAG, "taille de tab = " + nodeList.getLength());
           for (int i = 0; i < nodeList.getLength(); i++) {
               Element e = (Element) nodeList.item(i);
               tabGeo[i] = new GeoPoint(getValue(e, "name"), Double.
parseDouble(getValue(e, "lat")), Double.parseDouble(getValue(e, "long")),
getValue(e, "info"));
               Log.v(TAG, getValue(e, "name"));
               ...
           }
       }
   } catch (IOException e1) {
       e1.printStackTrace();
   }
   return tabGeo;
}
...
```

Le document XML est renvoyé par la méthode `getDocument()` :

Fichier

```
public Document getDocument(InputStream inputStream) {
   Document document = null;
   DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
   DocumentBuilder db = null;
   try {
       db = factory.newDocumentBuilder();
   } catch (ParserConfigurationException e) {
       e.printStackTrace();
   }
   InputSource inputSource = new InputSource(inputStream);
   try {
       document = db.parse(inputSource);
   } catch (SAXException e) {
       e.printStackTrace();
   } catch (IOException e) {
       e.printStackTrace();
   }
   return document;
}
...
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

`addGeoPoint()` sauvegarde une géocache selon la structure de données XML puis formate le fichier dans un format lisible avec un objet de type `TransformerFactory` :

Fichier

```
public void addGeoPoint(String name, String lat, String lgt, String info)
{
   Document doc = null;
   DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
   InputStream stream = null;
   DocumentBuilder db = null;

   Log.v(TAG, "addGeoPoint");
   try {
       try {
           stream = mContext.openFileInput(xmlFile);
       } catch (FileNotFoundException e) {
           e.printStackTrace();
       }
       db = factory.newDocumentBuilder();
   } catch (ParserConfigurationException e) {
       e.printStackTrace();
   }
   InputSource inputSource = new InputSource(stream);
   try {
       doc = db.parse(inputSource);
   } catch (SAXException e) {
       e.printStackTrace();
   } catch (IOException e) {
       e.printStackTrace();
   }
   NodeList root = doc.getElementsByTagName("data");
   Node elem = root.item(0);
   elem.appendChild(getGeoPoint(doc, name, lat, lgt, info));
   TransformerFactory transformerFactory = TransformerFactory.
newInstance();
   Transformer transformer = null;
   try {
       transformer = transformerFactory.newTransformer();
   } catch (TransformerConfigurationException e) {
       e.printStackTrace();
   }
   //for pretty print
   transformer.setOutputProperty(OutputKeys.INDENT, "yes");
   DOMSource source = new DOMSource(doc);
   //write to file
   String filePath = mContext.getFilesDir().getPath().toString() + "/" +
xmlFile;
   StreamResult file = new StreamResult(new File(filePath));
   //write data
   try {
       transformer.transform(source, file);
   } catch (TransformerException e) {
       e.printStackTrace();
   }
}
...
```

La suppression d'une Géocache est assurée par la méthode `delGeoPoint()`. Ici aussi le fichier est formaté avec un objet de type `TransformerFactory` :

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@pressdecision.com)

Fichier

```
public void delGeoPoint(String geoName, String mXMLFile)
{
   Document doc = null;
   DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
   InputStream stream = null;
   DocumentBuilder db = null;

   String mFilePath = mContext.getFilesDir().getPath().toString() + "/" +
mXMLFile;
   File mFile = new File(mFilePath);
   if(!mFile.exists()) {
       return;
   }
   try {
       try {
           stream = mContext.openFileInput(mXMLFile);
       } catch (FileNotFoundException e) {
           e.printStackTrace();
       }
       db = factory.newDocumentBuilder();
   } catch (ParserConfigurationException e) {
       e.printStackTrace();
   }
   InputSource inputSource = new InputSource(stream);
   try {
       doc = db.parse(inputSource);
   } catch (SAXException e) {
       e.printStackTrace();
   } catch (IOException e) {
       e.printStackTrace();
   }
   NodeList nodes = doc.getElementsByTagName(NODE_GEO);
   for (int i = 0; i < nodes.getLength(); i++) {
       Element person = (Element) nodes.item(i);
       Element name = (Element) person.getElementsByTagName("name").
item(0);
       String pName = name.getTextContent();

       if (pName.equals(geoName)) {
           person.getParentNode().removeChild(person);
           Log.v(TAG, "XML : NODE DELETE " + pName);
       }
   }
   //for output to file
   TransformerFactory transformerFactory = TransformerFactory.
newInstance();
   Transformer transformer = null;
   try {
       transformer = transformerFactory.newTransformer();
   } catch (TransformerConfigurationException e) {
       e.printStackTrace();
   }
   //for pretty print
   transformer.setOutputProperty(OutputKeys.INDENT, "yes");
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

```
    DOMSource source = new DOMSource(doc);
    //write to file
    String filePath = mContext.getFilesDir().getPath().toString() + "/" +
mXMLFile;
    StreamResult file = new StreamResult(new File(filePath));
    //write data
    try {
        transformer.transform(source, file);
    } catch (TransformerException e) {
        e.printStackTrace();
    }
    System.out.println("DONE");
    try {
        stream.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
...
```

La méthode `getDocument(String xmlFileName)` est une alternative à `getDocument()` qui renvoie également le document XML :

Fichier

```
public Document getDocument(String xmlFileName){
    Document doc = null;
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    InputStream stream = null;
    DocumentBuilder db = null;
    try {
        try {
            stream = mContext.openFileInput(xmlFileName);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        db = factory.newDocumentBuilder();
    } catch (ParserConfigurationException e) {
        e.printStackTrace();
    }
    InputSource inputSource = new InputSource(stream);
    try {
        doc = db.parse(inputSource);
    } catch (SAXException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return doc;
}
...
```

Lors de la création d'une nouvelle Géocache, les deux méthodes `Node getGeoPoint(...)` créent un nouvel objet `Node` qui représente les données de la Géocache. Cette structure est ensuite ajoutée à la structure `<data> ... </data>` :

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonie... ion.com)

Fichier

```
...
private static Node getGeoPoint(Document doc, String name, String lat,
String longitude, String info) {
   Element geo = doc.createElement(NODE_GEO);
   // même chose pour lat, longitude et info
   geo.appendChild(getGeoPoint(doc, "lat", lat));
   //create name element
   geo.appendChild(getGeoPoint(doc, "name", name));
   ...
   return geo;
}

private static Node getGeoPoint(Document doc, String name, String value) {
   Element node = doc.createElement(name);
   node.appendChild(doc.createTextNode(value));
   return node;
}
...
```

Chaque attribut de la structure de données est associé à une valeur ; `getValue()` renvoie une valeur en fonction du nom de l'attribut `item` :

Fichier

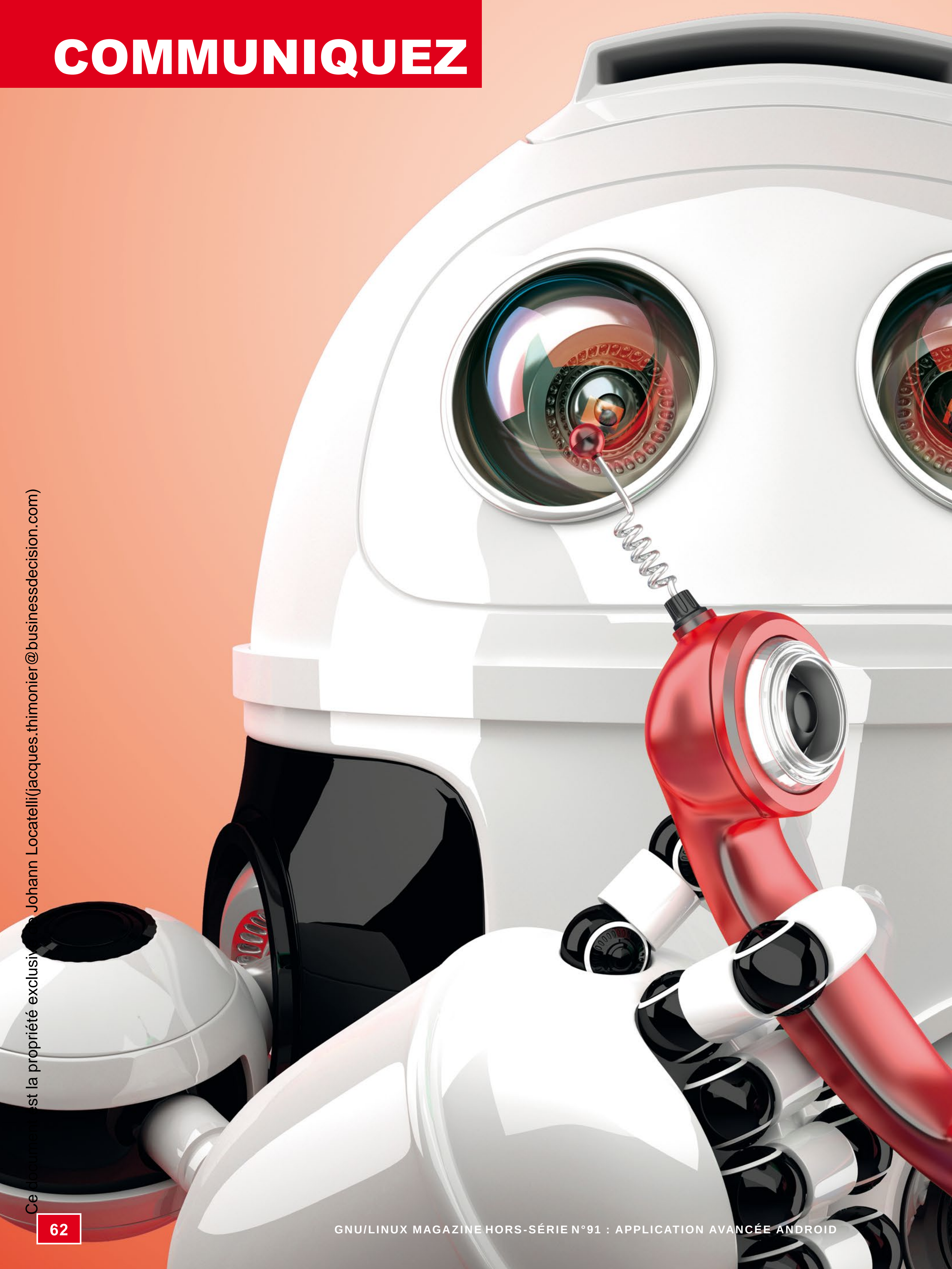
```
...
public String getValue(Element item, String name) {
   NodeList nodes = item.getElementsByTagName(name);
   return this.getTextNodeValue(nodes.item(0));
}

private final String getTextNodeValue(Node node) {
   Node child;
   if (node != null) {
      if (node.hasChildNodes()) {
         child = node.getFirstChild();
         while(child != null) {
            if (child.getNodeType() == Node.TEXT_NODE) {
               return child.getNodeValue();
            }
            child = child.getNextSibling();
         }
      }
   }
   return "";
}
} //  fin de classe
```

Synthèse

Le capteur GPS fournit les coordonnées d'un point géographique. Le système GPS est une technologie complexe, les API simplifient également son utilisation, quelques lignes de code ont permis d'obtenir un point de géolocalisation. L'étape de sauvegarde des points de géolocalisation est essentielle et permettra dans la suite d'échanger les géocaches entre les participants.

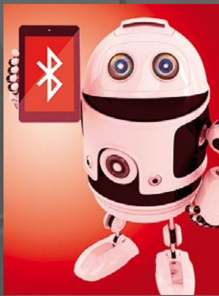
Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)



Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

3 COMMUNIQUEZ

À découvrir dans cette partie...



Communiquez en Bluetooth

La communication Bluetooth permet l'échange d'informations sur de courtes distances. Ce sera un des moyens que nous allons déployer pour pouvoir partager des géocaches entre plusieurs systèmes.
p. 64

Envoyez des SMS et utilisez l'API Google Maps pour afficher des cartes

Le partage de cache sur de courtes distances a ses limites (la distance justement !). Nous allons donc implémenter un mécanisme permettant de transmettre des géocaches via SMS et les caches seront affichées dans l'application sur des cartes Google Maps pour un meilleur confort d'utilisation.
p. 80

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

COMMUNIQUEZ EN BLUETOOTH

Notre application doit pouvoir communiquer des géocaches à un autre smartphone simplement. Pour cela, nous allons mettre en place une communication courte distance en Bluetooth.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Nous allons entamer avec cet article le développement du système de communication et de cartographie (voir figure 1 en bleu). Ces deux blocs fonctionnels sont liés aux cas d'utilisation suivants : Transférer Géocache, Alerter Géocache, et Afficher carte.

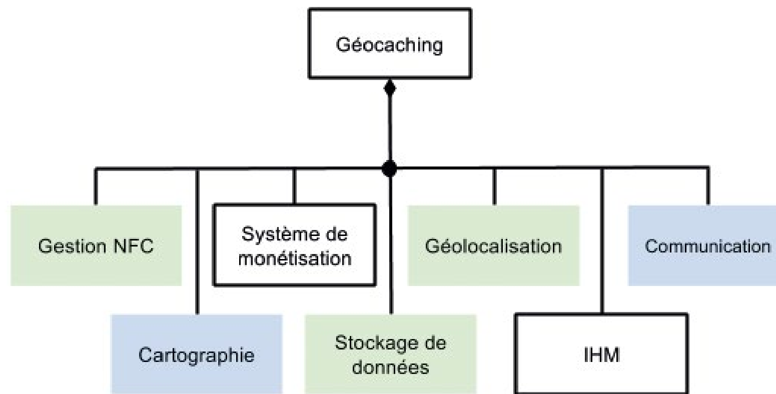


Schéma des blocs fonctionnels Géocaching.

Figure 1

Notre application prévoit un système de communication de type P2P (point à point) qui permet l'échange de géocaches. Ce système permet de s'affranchir d'un site web dédié et lourd à maintenir. Pour réaliser cette fonctionnalité, nous allons utiliser la technologie radio Bluetooth (BT) qui autorise une communication radio bidirectionnelle sans licence sur la bande des 2,4 GHz. La portée est d'environ une dizaine de mètres en champ libre. Ainsi, un fichier de géocaches est transmis d'un système à un autre avec une grande facilité. Bien sûr, le code reste toutefois assez générique ; à vous de le détourner complètement pour réaliser ensuite toute sorte d'applications d'échanges P2P utilisant cette technologie.

1. INTÉGRATION DU BLUETOOTH

Cette section présente pas à pas l'intégration du module de communication P2P. Au final, nous allons mettre en œuvre une structure client/serveur qui autorise les transferts de fichiers de géocaches d'un système à un autre.

Nous commençons par inscrire les permissions Bluetooth dans le **manifest** du projet :

Fichier

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

Il est préférable ensuite de vérifier la présence du *package* Bluetooth avant tout. Certains équipements bas de gamme peuvent ne pas être équipés :

Fichier

```
mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

if(mBluetoothAdapter==null) {
   Toast.makeText(this, "Device does not support Bluetooth", Toast.LENGTH_
LONG).show();
}
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Pour pouvoir communiquer, il est indispensable de mettre en marche le système Bluetooth via le menu ***Bluetooth*** ou programmatiquement (voir code suivant) via l'API qui propose alors de rendre l'équipement visible pendant un laps de temps à définir (voir section 2.4).

Fichier

```
private static final int ENABLE_BLUETOOTH = 1;

private void initBluetooth() {
   if (!mBluetoothAdapter.isEnabled()) {
       Intent intent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
       startActivityForResult(intent, ENABLE_BLUETOOTH);
   }
}

protected void onActivityResult(int requestCode, int resultCode, Intent
data) {
   if (requestCode == ENABLE_BLUETOOTH)
       if (resultCode == RESULT_OK) {
           Log.v(TAG, "BT = " + ENABLE_BLUETOOTH);
       }

   if (requestCode == DISCOVERY_REQUEST) {
       if (resultCode == RESULT_CANCELED) {
           Log.d(TAG, "Discovery cancelled by user");
       } else {
           Log.v(TAG, "Discovery allowed");
       }
   }
}
```

La méthode **initBluetooth()** permet de lancer un *Intent* avec un retour, c'est-à-dire que l'on va pouvoir capturer ce qui s'est passé dans l'*Intent* grâce à la surcharge de la méthode **protected void onActivityResult(int requestCode,int resultCode, Intent data)** qui est appelée comme *callback* à la fin de l'exécution de l'*Intent*. Nous pourrons capturer si l'utilisateur a appuyé sur ***Refuser*** ou ***Accepter***. À cet instant, il est possible de scanner les différentes fréquences afin de réaliser la découverte des équipements Bluetooth à proximité.

La mise en marche du système radio BT ne suffit pas, il est encore nécessaire de le rendre visible, en d'autres termes de l'autoriser à répondre aux autres équipements présents. Il est possible de configurer une période pour rendre le système Bluetooth visible. Ici nous avons choisi une durée d'environ cinq minutes, ce qui est amplement suffisant :

Fichier

```
private void makeDiscoverable() {
        Intent discoverableIntent = new Intent(BluetoothAdapter.ACTION_
REQUEST_DISCOVERABLE);
        //discoverable for 5 minutes (~300 seconds)
        discoverableIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_
DURATION, 300);
        startActivityForResult(discoverableIntent, DISCOVERY_REQUEST);
        Log.i("Log", "Discoverable ");
    }
```

L'étape de découverte permet d'identifier les équipements qui sont à portée radio et susceptibles de communiquer. Les équipements Bluetooth possèdent une adresse MAC et un nom. D'ailleurs sur les systèmes Android, il est possible de déterminer ce nom qui permettra de donner une identification plus facile à retenir qu'une adresse sur 48

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

bits. Par exemple, sur Nexus 4 : ***Menu Paramètres > Bluetooth > Menu en haut à droite > Renommer cet appareil***.

Le canal de communication radio Bluetooth est découpé en 79 canaux, chaque équipement utilise l'un de ces canaux. Parmi les 79 canaux, 32 canaux sont utilisés pour réaliser la découverte des équipements voisins et établir une communication. Un équipement envoie des messages de découverte sur l'un de ces canaux et un autre équipement écoute périodiquement et répond à ces messages. En cas de non-réponse sur un canal, le message de découverte est renvoyé 256 fois. Une transaction « message découverte + réponse » prend environ 625µs. Par défaut, la découverte sur l'ensemble des canaux prend environ 10s, ce qui est loin d'être négligeable. Évidemment, le scan de toutes les fréquences est consommateur d'énergie, mais il n'existe pas d'autre solution pour découvrir de nouveaux équipements. La découverte permet de récupérer l'adresse MAC et le nom de l'équipement pour ensuite établir la communication.

La méthode `startDiscovery()` du `BluetoothAdapter` permet de lancer le scan sur tout le spectre radio Bluetooth.

Fichier

```
...
mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

if(mBluetoothAdapter==null) {
   Toast.makeText(this, "Device does not support Bluetooth", Toast.LENGTH_
LONG).show();
}

//start discovery
mBluetoothAdapter.startDiscovery();
...
```

Pour récupérer les équipements découverts par le scan, il faut implémenter un `broadcastReceiver` qui reçoit les *Intent* du système. Le `broadcastReceiver` reçoit également les événements sur l'état des connexions BT :

- `ACTION_DISCOVERY_STARTED` : début de la recherche des équipements ;
- `ACTION_DISCOVERY_FINISHED` : fin de la recherche ;
- `ACTION_BOND_STATE_CHANGED` : changement de la connexion ;
- `BOND_BONDED` : connexion établie ;
- `BOND_NONE` : connexion non établie.

Fichier

```
...
// Register for broadcasts when a device is discovered.
IntentFilter filter = new IntentFilter();

...
filter.addAction(BluetoothAdapter.ACTION_STATE_CHANGED);
filter.addAction(BluetoothDevice.ACTION_FOUND);
filter.addAction(BluetoothAdapter.ACTION_DISCOVERY_STARTED);
filter.addAction(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
filter.addAction(BluetoothDevice.ACTION_BOND_STATE_CHANGED);
registerReceiver(mReceiver, filter);
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

```
private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
   public void onReceive(Context context, Intent intent) {
       String action = intent.getAction();

       if (BluetoothAdapter.ACTION_DISCOVERY_STARTED.equals(action)) {
           mesgText.setText("Discovery started");
           Log.v(TAG, "ACTION_DISCOVERY_STARTED");
           //discovery starts, we can show progress dialog or perform
other tasks
       } else if (BluetoothAdapter.ACTION_DISCOVERY_FINISHED.
equals(action)) {
           //discovery finishes, dismis progress dialog
           mesgText.setText("Discovery finished");
           Log.v(TAG, "ACTION_DISCOVERY_FINISHED");
       } else if (BluetoothDevice.ACTION_FOUND.equals(action)) {
           //bluetooth device found
           BluetoothDevice device = (BluetoothDevice) intent.getParcelable
Extra(BluetoothDevice.EXTRA_DEVICE);

           if(!deviceListName.isEmpty() && firstElement) {
               firstElement = false;
               deviceListName.clear();
           }

           deviceBT.add(device);
           deviceListName.add(device.getName());
           arrayAdapter.notifyDataSetChanged();
           Log.v(TAG, "Found device " + device.getName());

       } else if (BluetoothDevice.ACTION_BOND_STATE_CHANGED.
equals(action)) {
           final int state = intent.getIntExtra(BluetoothDevice.EXTRA_
BOND_STATE, BluetoothDevice.ERROR);
           final int prevState = intent.getIntExtra(BluetoothDevice.EXTRA_
PREVIOUS_BOND_STATE, BluetoothDevice.ERROR);
           if (state == BluetoothDevice.BOND_BONDED) {
               Log.v(TAG, "Paired");
           } else if (state == BluetoothDevice.BOND_NONE && prevState ==
BluetoothDevice.BOND_BONDED) {
               Log.v(TAG, "Unpaired");
           }
       }
   }
};
...
```

Des équipements sont associés s'ils ont réalisé la phase de découverte mutuelle et ont effectué une demande d'association, alors ils sont enregistrés par le système comme équipements associés. Pendant l'association un code secret est aussi échangé afin de vérifier l'équipement qui demande l'association. Ces équipements pourront se reconnecter automatiquement par la suite sous condition que le Bluetooth soit activé. Ils n'auront pas besoin d'effectuer une nouvelle phase de découverte ni d'association (sauf si l'association est supprimée). Ce comportement par défaut peut poser d'ailleurs des problèmes de sécurité.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Fichier

```
private Set<BluetoothDevice> getPairedDevices(BluetoothAdapter
mBluetoothAdapter)
{
   Set<android.bluetooth.BluetoothDevice> pairedDevices =
mBluetoothAdapter.getBondedDevices();

   if (pairedDevices.size() > 0) {
       // There are paired devices. Get the name and address of each
paired device.
       for (android.bluetooth.BluetoothDevice device : pairedDevices) {
           Log.v(TAG, "paired = " + device.getName());
           Log.v(TAG, "paired MAC = " + device.getAddress());
           deviceBT.add(device);
           deviceListName.add(device.getName() + "  paired");
           arrayAdapter.notifyDataSetChanged();
       }
   }
   return pairedDevices;
}
```

Les équipements ont été préalablement associés puis dissociés. Pour pouvoir communiquer, ils doivent réaliser la phase de découverte puis à nouveau s'associer.

Association BT :

Fichier

```
private void pairDevice(BluetoothDevice device) {
   try {
       Method method = device.getClass().getMethod("createBond", (Class[])
null);
       method.invoke(device, (Object[]) null);
   } catch (Exception e) {
       e.printStackTrace();
   }
}
```

Suppression de l'association BT :

Fichier

```
private void unpairDevice(BluetoothDevice device) {
   try {
       Method method = device.getClass().getMethod("removeBond", (Class[])
null);
       method.invoke(device, (Object[]) null);
   } catch (Exception e) {
       e.printStackTrace();
   }
}
```

2. STRUCTURE CLIENT/SERVEUR

Dans notre projet, chaque smartphone est client/serveur afin de pouvoir récupérer les géocaches d'un autre joueur. Le serveur doit être démarré pour assurer les communications entrantes, de l'autre côté le client se connecte au serveur. Dans cette hypothèse de

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

structure, nous n'utiliserons qu'une connexion entrante. Le code est tout à fait modifiable pour obtenir des communications à *n* clients. La structure client/serveur étant symétrique, chaque système peut donc démarrer un serveur et recevoir un client. L'échange de données peut se faire en boucle, le serveur accepte les connexions dans une boucle sans fin (voir figure 2).

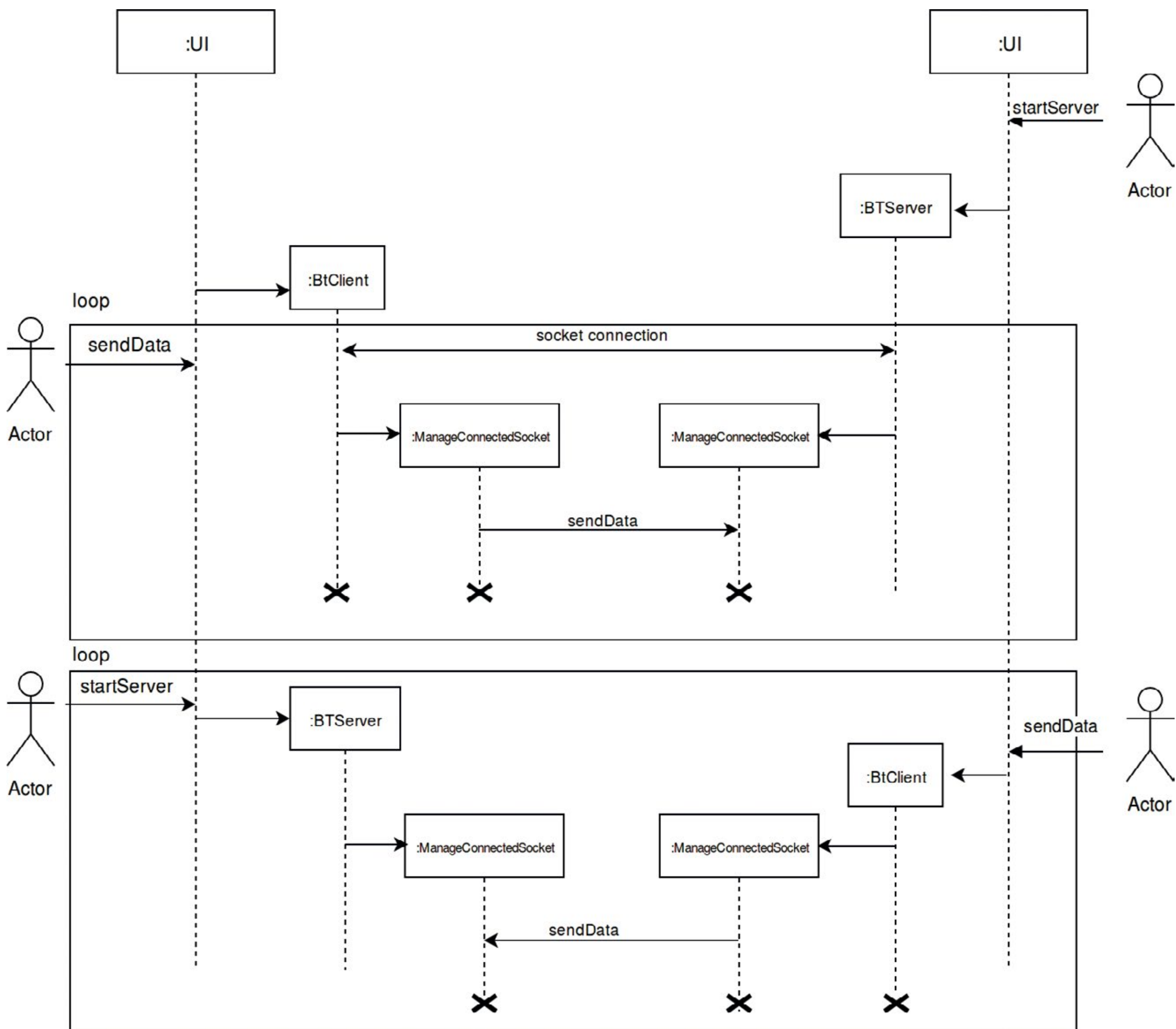


Figure 2 *Diagramme de séquence client/serveur.*

Le serveur est arrêté à la demande de l'utilisateur. À chaque connexion, le même objet `ManageConnectedSocket` est utilisé (c'est ici qu'il est possible de définir une structure à *n* clients). L'appel à `connect()` est bloquant jusqu'à la connexion d'un nouveau client. À chaque demande de transfert, un objet de type `BtClient` est créé. Celui-ci établit une connexion Bluetooth avec le serveur qui utilise un UUID, une clé d'identification de service entre le client et le serveur. Si la connexion est bien établie alors la gestion de la transaction est confiée à un objet de type `ManageConnectedSocket`.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

NOTE

Le service Bluetooth SPP utilise une clé UUID pour identifier le service proposé, aussi il est possible d'utiliser votre propre identifiant UUID. L'UUID de votre application peut être généré de façon aléatoire avec la commande `uuidgen`. Cette commande fait partie du *package* `util-linux` :

Terminal

```
$ uuidgen
d0b18f04-c7fc-454c-9f61-330e62d999ba
```

2.1 Bluetooth - Structure serveur

Le serveur BT est exécuté dans un *thread* et s'arrête lorsque la transaction P2P est terminée, autrement dit lorsque le fichier de géocache a été transmis. L'appel `mmServerSocket.accept()` est bloquant, lorsqu'une connexion BT arrive, le serveur crée un objet de type `ManageConnectedSocket` pour la gestion de la transaction. Le serveur peut gérer plusieurs clients, il utilise toujours le même UUID.

Fichier

```
private class BtServer extends Thread {

   BluetoothServerSocket mmServerSocket = null;


   public BtServer(BluetoothAdapter mBTdapter, UUID mUUID) {

       UUID MY_UUID = mUUID;
       BluetoothAdapter mBluetoothAdapter = mBTdapter;

       try {
           Log.v(TAG, "mBluetoothAdapter ...");
           mmServerSocket = mBluetoothAdapter.listenUsingRfcommWithService
Record("server", MY_UUID);
           Log.v(TAG, "mBluetoothAdapter ok");


       } catch (IOException e) {
           Log.e(TAG, "Socket's listen() method failed", e);
       }
   }

   public void run() {
       BluetoothSocket socket = null;
       // Keep listening until exception occurs or a socket is returned.
       while (true) {
           try {
               Log.v(TAG, "wait cx");
               socket = mmServerSocket.accept();
               Log.v(TAG, "SocketAccepted");
               Log.v(TAG, "Manage cx");
               mManageConnectedSocket = new
ManageConnectedSocket(socket);
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

```
                mManageConnectedSocket.start();
            } catch (IOException e) {
                //  mesgText.setText(«BT socket closed or timeout»);
                Log.e(TAG, "Socket's accept() method failed", e);
                break;
            }
        }
    }
    // Closes the connect socket and causes the thread to finish.
    public void cancel() {
        try {
            mmServerSocket.close();
        } catch (IOException e) {
            Log.e(TAG, "Could not close the connect socket", e);
        }
    }
}
```

2.2 Bluetooth - Structure client

La classe **BtClient** est également exécutée dans un *thread*. La connexion est établie selon **BluetoothDevice** qui représente le serveur BT. L'ouverture de la session RFCOMM est réalisée par la primitive **device.createRfcommSocketToServiceRecord()**. Si la connexion est bien établie, alors un objet de type **ManageConnectedSocket** réalise la transaction.

Fichier

```
private class BtClient extends Thread {
   private BluetoothSocket mmSocket;
   private final BluetoothDevice mmDevice;
   BluetoothAdapter mBluetoothAdapter;
   UUID mUUID;

   public BtClient(BluetoothDevice device, BluetoothAdapter
myBluetoothAdapter, UUID MY_UUID) {
       mmDevice = device;
       mUUID = MY_UUID;
       mBluetoothAdapter = myBluetoothAdapter;

       try {
           // Get a BluetoothSocket to connect with the given
BluetoothDevice.
           // MY_UUID is the app's UUID string, also used in the server
code.
           mmSocket = device.createRfcommSocketToServiceRecord(mUUID);
           Log.v(TAG, "Client get BT remote server : " + mmSocket.
getRemoteDevice().getName());
       } catch (IOException e) {
           Log.e(TAG, "Socket's create() method failed", e);
       }
   }

   public void run() {
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

```
            // Cancel discovery because it otherwise slows down the connection.
            mBluetoothAdapter.cancelDiscovery();
            try {
                // Connect to the remote device through the socket. This call blocks
                // until it succeeds or throws an exception.
                Log.v(TAG, "Client try to connected to server");
                mmSocket.connect();
                Log.v(TAG, "Client socket connected");
                // The connection attempt succeeded. Perform work associated with
                // the connection in a separate thread.
                mySocketManager = new ManageConnectedSocket(mmSocket);
                mySocketManager.start();

            } catch (IOException connectException) {
                // Unable to connect; close the socket and return.
                try {
                    mmSocket.close();
                } catch (IOException closeException) {
                    Log.e(TAG, "Could not close the client socket", closeException);
                }
                return;
            }
        }

        // Closes the client socket and causes the thread to finish.
        public void cancel() {
            try {
                mmSocket.close();
            } catch (IOException e) {
                Log.e(TAG, "Could not close the client socket", e);
            }
        }
    }
```

2.3 Transfert des données

Le transfert de données est réalisé par la classe `ManageConnectedSocket` qui écrit dans la *socket* Bluetooth avec la méthode `write(String msg)` et lit avec `getInputStream()`. Lorsque la transaction est finie, le descripteur de *socket* est fermé. Le fichier reçu est enregistré dans le répertoire `files` du *package* de l'application. Nous utilisons ici la mémoire interne du système, car la taille du fichier est assez modeste. Si vous prévoyez un volume (plusieurs mégas !) conséquent, il est préférable d'utiliser le stockage externe sur un support carte SD.

Le code source de la classe `ManagedConnectedSocket` vous est proposé sur : **https://github.com/frameproject/geocaching**.

2.4 Intégration - Classe de pilotage

Viens la phase d'assemblage des différents composants en commençant par la classe `CommManager` qui sera responsable de la gestion Bluetooth.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

L'activité principale (classe **MainActivity**) a été légèrement modifiée afin de pouvoir appeler la gestion Bluetooth.

Fichier

```
...
// BT P2P connection
buttonP2P.setOnClickListener(new View.OnClickListener() {
   public void onClick(View v) {
       Intent p2pIntent = new Intent(MainActivity.this.
getApplicationContext(), mag.linux.android_sensors.CommManager.class);
       startActivity(p2pIntent);
   }
});
...
```

La classe **CommManager** intègre la gestion de l'IHM et d'autres méthodes nécessaires pour la gestion de la communication Bluetooth que nous avons vue précédemment.

Fichier

```
package mag.linux.android_sensors;
...
public class CommManager extends Activity {
   protected static final String TAG = "BLUETOOTH";
   protected static final int DISCOVERY_REQUEST = 1;
   BluetoothAdapter mBluetoothAdapter;
   BluetoothDevice btDeviceSelected;
   UUID MY_UUID = UUID.fromString("a60f35f0-b93a-11de-8a39-
08002009c666");
   private ArrayList<String> deviceListName = new ArrayList<String>();
   private ArrayList<BluetoothDevice> deviceBT = new
ArrayList<BluetoothDevice>();

   BtServer mBtServer;
   BtClient mBtClient;
   ManageConnectedSocket mManageConnectedSocket;
   ManageConnectedSocket mySocketManager;
   boolean transaction = false;
   // UI
   private ListView lv;
   ArrayAdapter<String> arrayAdapter = null;
   TextView mesgText;
   boolean firstElement = false;
   EditText msg;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_p2p);
       //UI
       Button btnStartServer = (Button) findViewById(R.
id.buttonStartServer);
       Button btnScan = (Button) findViewById(R.id.buttonScan);
       Button btnDiscover = (Button) findViewById(R.
id.buttonDiscoverable);
       Button btnSendData = (Button) findViewById(R.id.buttonSendData);
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

```
        Button btnStopServer = (Button) findViewById(R.id.buttonStopServer);
        mesgText = (TextView) findViewById(R.id.textViewMsg);
        lv = (ListView) findViewById(R.id.listViewDevice);
        // data devices
        deviceListName.add(" ");
        arrayAdapter = new ArrayAdapter<>(
                this,
                android.R.layout.simple_list_item_1,
                deviceListName);
        lv.setAdapter(arrayAdapter);
        mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
        if (mBluetoothAdapter == null) {
            Toast.makeText(this, "Device does not support Bluetooth", Toast.
LENGTH_LONG).show();
        }
        btnStartServer.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                mesgText.setText("Start BT server");
                if (mBtServer == null && mBluetoothAdapter != null) {
                    mBtServer = new BtServer(mBluetoothAdapter, MY_UUID);
                    mBtServer.start();
                }
            }
        });

        btnDiscover.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                if (mBluetoothAdapter.isEnabled()) {
                    makeDiscoverable();
                }
            }
        });

        btnScan.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                if (!mBluetoothAdapter.isDiscovering() && mBluetoothAdapter.
isEnabled()) {
                    // reset list
                    deviceListName.clear();
                    deviceBT.clear();
                    getPairedDevices(mBluetoothAdapter);
                    if (deviceListName.isEmpty()) {
                        deviceListName.add(" ");
                        firstElement = true;
                    }
                    arrayAdapter.notifyDataSetChanged();
                    //start discovery
                    mBluetoothAdapter.startDiscovery();
                    mesgText.setText("Scan in progress ... wait");
                }
            }
        });

        btnSendData.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                if (!mBluetoothAdapter.isDiscovering()) {
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

```
                    if (btDeviceSelected == null) {
                        refreshUI("Select a device before ...");
                    } else if (!transaction) {
                        refreshUI("Send file ...");
                        sendFile();
                    }
                }
            }
        });

        lv.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int
position, long arg3) {
                view.setSelected(true);
                btDeviceSelected = deviceBT.get(position);
                if (btDeviceSelected.getBondState() == BluetoothDevice.BOND_
NONE) {
                    mesgText.setText("Pairing... ... wait");
                    pairDevice(btDeviceSelected);
                }
            }
        });

        btnStopServer.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                if (mBtServer != null) {
                    mBtServer.cancel();
                    mBtServer = null;
                    mesgText.setText("Server stopped");
                }
                if (mManageConnectedSocket != null) {
                    mManageConnectedSocket.cancel();
                }
            }
        });

        // algo
        initBluetooth();
        makeDiscoverable();
        // Register for broadcasts when a device is discovered.
       ...
        // Get paired devices
        deviceBT.clear();
        getPairedDevices(mBluetoothAdapter);
    }
...
    private static final int ENABLE_BLUETOOTH = 1;
...
    @Override
    protected void onDestroy() {
        super.onDestroy();
        if (mBtServer != null)
            mBtServer.cancel();

        if (mBtClient != null)
            mBtClient.cancel();
        unregisterReceiver(mReceiver);
    }
}
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

2.5 IHM

Le *layout* présente les différentes fonctions que nous venons de décrire. Les intitulés des boutons sont écrits dans `res/strings.xml` sur **https://github.com/frameproject/geocaching** et la figure 3 montre l'IHM de l'application.

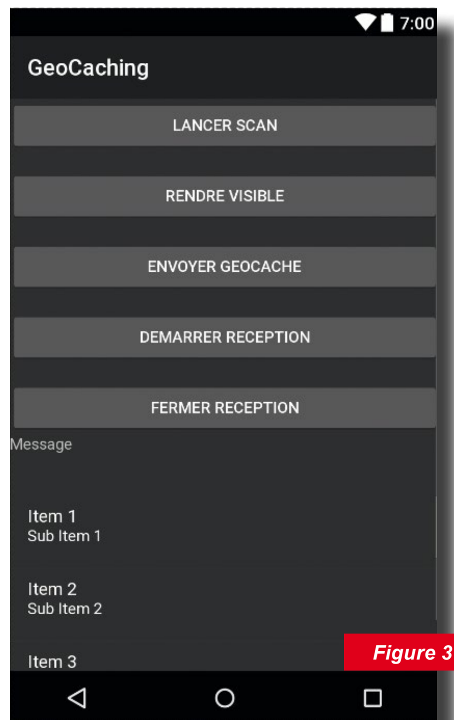


Figure 3

IHM de l'application.

2.6 Logs et tests

Les tests demandent deux équipements Android : l'un sera le client, l'autre le serveur. Dans un premier temps, le transfert de fichier peut être simulé par l'envoi d'une chaîne de caractères.

Les tests consistent à réaliser les séquences suivantes sur deux systèmes différents (voir le tableau ci-dessous).

La réception des données se fait avec la primitive `read()` jusqu'à la réception de la commande « stop » qui permet de définir la fin de lecture. À ce moment, côté client, le *socket* est fermé. Côté serveur, le contenu est enregistré dans le fichier `Other.xml`.

Séquences	Système 1	Système 2
1	Rendre visible le Bluetooth en appuyant sur ***make discoverable***.	Rendre visible le Bluetooth en appuyant sur ***make discoverable***.
2	Attendre de voir le système 2 puis le sélectionner.	
Sinon choisir le système 2 déjà associé - retourner à 1 si invisible.	Attendre de voir le système 1 puis le sélectionner.	
Sinon choisir le système 1 déjà associé - retourner à 1 si invisible.		
3	Attendre que le serveur système 1 soit démarré.	Démarrer le serveur.
4	Appuyer sur ***send data***.	Visualiser la réception des données.
5	Le fichier est envoyé.	-
6	Inverser les séquences de 3 à 5.	-
7	Terminer avec ***stop server***.	Terminer avec ***stop server***.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Voici un exemple de Logcat côté serveur lors de la réception des données :

Fichier

```
04-04 16:06:01.970: V/BLUETOOTH(21181): Read socket =<?xml version="1.0"
encoding="UTF-8"?><data>***
04-04 16:06:01.970: V/BLUETOOTH(21181): Read socket wait data
04-04 16:06:01.994: V/BLUETOOTH(21181): Read socket =<GEO>***
...
```

L'écriture est réalisée par la primitive **write()**. Lorsque le *socket* est fermé, une exception est déclenchée, car la transaction Bluetooth est détruite à cet instant.

Logcat côté client pour l'envoi du fichier :

Fichier

```
04-04 16:06:01.753: V/BLUETOOTH(11753): Write socket : <?xml version="1.0"
encoding="UTF-8"?><data>
04-04 16:06:01.757: V/BLUETOOTH(11753): ************ <GEO>
04-04 16:06:01.769: V/BLUETOOTH(11753): Write socket : <GEO>
04-04 16:06:01.769: V/BLUETOOTH(11753): ************ <name>aaaa</name>
04-04 16:06:01.780: V/BLUETOOTH(11753): Write socket : <name>aaaa</name>
04-04 16:06:01.780: V/BLUETOOTH(11753): ************ <lat>1.2</lat>
04-04 16:06:01.792: V/BLUETOOTH(11753): Write socket : <lat>1.2</lat>
...
```

2.7 Debug Bluetooth

La phase de débogage Bluetooth peut être complexe en cas de problème de communication. Avec un PC portable sous Linux, quelques outils open source du *package* **bluez** permettent de contrôler les échanges.

La commande **hciconfig** permet de vérifier le matériel Bluetooth de votre PC et connaître le nom Bluetooth utilisé :

Terminal

```
# hciconfig <device> <command> <arguments...>
```

Vérifiez votre configuration Bluetooth :

Terminal

```
$ hciconfig
hci0: Type: BR/EDR  Bus: USB
      BD Address: D8:FC:93:21:6D:16  ACL MTU: 1021:5  SCO MTU: 96:5
      UP RUNNING PSCAN
      RX bytes:1249 acl:0 sco:0 events:142 errors:0
      TX bytes:22965 acl:0 sco:0 commands:141 errors:0
```

Le nom de l'adaptateur Bluetooth peut être connu avec la commande suivante :

Terminal

```
$ sudo hciconfig hci0 down
$ hciconfig
hci0: Type: BR/EDR  Bus: USB
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

```
        BD Address: D8:FC:93:21:6D:16  ACL MTU: 1021:5  SCO MTU: 96:5
        DOWN
        RX bytes:1249 acl:0 sco:0 events:142 errors:0
        TX bytes:22965 acl:0 sco:0 commands:141 errors:0

$ hciconfig hci0 name
hci0: Type: BR/EDR  Bus: USB
        BD Address: D8:FC:93:21:6D:16  ACL MTU: 1021:5  SCO MTU: 96:5
        Name: 'endeavour-0'
```

Lancer un scan Bluetooth pour vérifier la présence des équipements :

Terminal

```
$ hcitool scan
Scanning ...
  98:D6:F7:6B:B1:77  Nexus 4

$ sdptool browse 98:D6:F7:6B:B1:77
Browsing 98:D6:F7:6B:B1:77 ...
...
```

Il est possible de faire un « dump » des paquets Bluetooth avec la commande **hcidump** :

Terminal

```
$ sudo apt-get install bluez-hcidump

$ hcidump -X
HCI sniffer - Bluetooth packet analyzer ver 2.5
...
```

Demande de ping d'un système BT avec **l2ping** (« L » minuscule au début) :

Terminal

```
$ sudo l2ping -c 5 98:D6:F7:6B:B1:77
Ping: 98:D6:F7:6B:B1:77 from D8:FC:93:21:6D:16 (data size 44) ...
0 bytes from 98:D6:F7:6B:B1:77 id 0 time 7.02ms
...
```

Synthèse

L'intégration du module de communication BT ne représente pas de difficulté particulière. La structure client/serveur peut être facilement étendue pour assurer des communications multiples. Enfin, le débogage radio peut être difficile lorsqu'apparaît un dysfonctionnement quelconque. Vous pouvez utiliser le *package* **Linux/Bluez** qui permet de vérifier l'état d'un équipement Bluetooth.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

ENVOYEZ DES SMS ET UTILISEZ L'API GOOGLE MAPS POUR AFFICHER DES CARTES

Pour pouvoir déterminer où se trouve une géocache, il va falloir l'afficher sur une carte. Cela sera fait à l'aide de l'API Google Maps. Nous ajouterons également le partage de géocache via SMS de manière à ne plus être restreint à une communication courte distance.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Il est temps d'afficher les géocaches sur une carte. Pour cela, nous allons utiliser une activité spécialisée de cartographie. Celle-ci est créée directement avec l'aide d'**Android Studio** en suivant les séquences suivantes : ***File > New > Activity > Gallery > Google Maps Activity***. Une activité est alors ajoutée dans le *package* ainsi qu'un *layout* de cartographie. Cette activité est nommée `GeoPointMap`.

1. IHM

L'interface graphique est composée principalement d'un `LinearLayout`, et d'un fragment **[1]** d'affichage dans lequel est insérée la carte avec la géolocalisation des géocaches (voir figure 1). Une `ListView` permet d'afficher l'ensemble des géocaches. La barre de menu au-dessus du fragment propose deux actions : ***share*** et ***delete***.

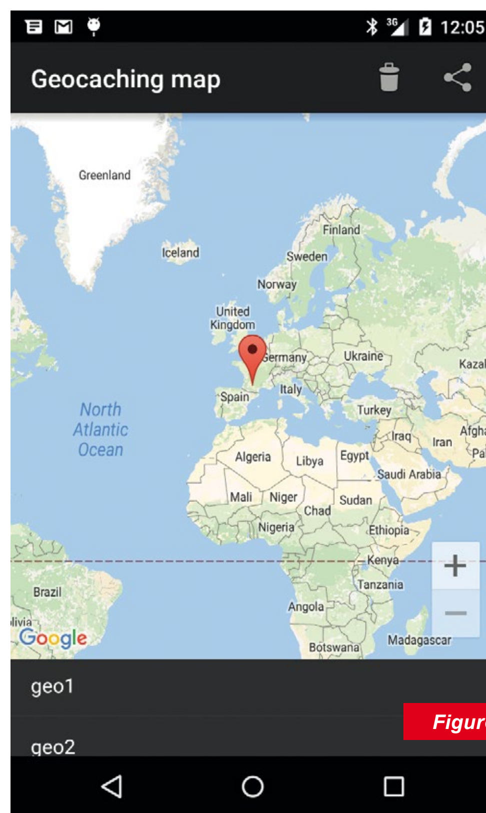


Figure 1

La notion de fragment d'affichage est apparue dès l'API V11, et permet de définir des zones d'affichage qui permettent une grande souplesse pour la mise en page. Contrairement au *layout* standard, un fragment possède un cycle sous forme de machine à état de la même façon qu'une activité ou encore un service Android. Les principales fonctionnalités sont les suivantes :

- Un fragment est quasi monolithique, donc très facile à réutiliser dans diverses applications.
- Un fragment d'affichage peut être lié directement à une classe java avec `extends Fragment` qui permet alors d'implémenter des méthodes pour la gestion de l'affichage, ce comportement n'est pas possible avec l'utilisation de *layout* standard.
- La possibilité de communication entre les événements du fragment et le code Java pour les classes qui n'héritent pas directement via `extends Fragment` : par exemple, un clic souris peut être intercepté dans le code java si la classe implémente l'interface `MainFragment.MainFragmentCallback` avec la surcharge de la méthode `public void onTitleClicked()`.

Nous allons apporter quelques modifications au *layout* avec :

- L'ajout d'un menu pour la suppression d'une géocache et le partage d'une géocache par SMS.
- Une liste qui affiche vos géocaches et celles reçues en P2P par Bluetooth. Le détail d'une géocache (nom, latitude, longitude, infos) est obtenu en appuyant longuement sur un élément de la liste. Cet événement est capturé par un *listener* de type `setOnItemLongClickListener(new AdapterView.OnItemLongClickListener()`.

Layout et rendu de l'IHM ainsi que la barre de menu de l'IHM de géolocalisation `res/layout/menu.xml` sont disponibles sur **https://github.com/frameproject/geocaching**.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

2. GOOGLE MAPS API KEY

L'utilisation de la cartographie requiert l'activation d'une clé API Google Maps [2]. En effet, les outils de cartographie sont des services Google configurables via la console des développeurs : **https://console.developers.google.com**. Pour y accéder, il est nécessaire de créer un compte Google.

Le *wizard* de création d'activité simplifie la mise en œuvre en préparant directement un lien http qui comprend les informations nécessaires à l'activation d'une clé Google Maps : type de client + signature SHA du certificat par défaut + nom du *package*. Ce lien se trouve dans le fichier `res/google_maps_api.xml`. Attention, la clé ne sera délivrée que pour un *package* de l'application, il est donc préférable de concentrer toutes les activités de cartographie dans un seul *package* pour des raisons évidentes de gestion de clé.

Le fichier `res/google_maps_api.xml` se trouve sur **https://github.com/frameproject/geocaching**.

2.1 Signature de certificat

RAPPEL

Lors de l'installation d'Android Studio, un certificat par défaut est créé dans la répertoire de login, en principe il se trouve dans `~/.android/debug.keystore`. Vous pouvez obtenir la signature de votre certificat avec la commande suivante :

Terminal

```
$ keytool -list -v -keystore ~/.android/debug.keystore -alias
androiddebugkey -storepass android -keypass android
Alias name: androiddebugkey
...
Valid from: Wed Feb 15 16:30:31 CET 2012 until: Fri Feb 07 16:30:31 CET
2042
Certificate fingerprints:
  MD5:  82:94:BA:0D:E6:58:E7:2D:01:50:90:2D:FD:6E:79:CD
  SHA1: A2:31:10:69:AF:94:3E:7C:80:2B:2A:E1:BE:FA:E1:EC:4A:F3:2B:58
  SHA256: 00:D2:CB:45:0A:20:20:03:7B:24:EC:E1:20:44:B5:53:A0:D2:54:06:11:3
7:82:3E:BF:E3:CD:CB:85:7B:C4:41
  Signature algorithm name: SHA256withRSA
  Version: 3
...
```

La console **Google developer** permet de délivrer une clé d'activation. Auparavant, vous devez créer un projet, les clés d'activation Google Maps API seront ensuite associées à ce projet et un *package* en particulier.

2.2 Sécurité

Selon la version d'Android, l'utilisateur accorde l'autorisation soit lors de l'installation de l'application (sur Android 5.1 et plus bas), soit lors de l'exécution de l'application (sur Android 6.0 et versions ultérieures). Ces aspects de sécurité seront abordés lors du déploiement.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

2.3 Manifest

L'intégration de la cartographie implique une modification du **manifest** pour l'accès au GPS et l'activité de cartographie :

Fichier

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="mag.linux.android_sensors">
 ...
   <uses-permission android:name="android.permission.ACCESS_COARSE_
LOCATION" />
   <uses-permission android:name="android.permission.ACCESS_FINE_
LOCATION" />
   <uses-permission android:name="android.permission.SEND_SMS" />

   <application
      ...
       <meta-data
           android:name="com.google.android.geo.API_KEY"
           android:value="@string/google_maps_key" />

          <activity
           android:name=".GeoPointMap"
           android:label="@string/title_activity_geo_point_map">
       </activity>
   </application>
</manifest>
```

3. INTÉGRATION - CLASSE DE PILOTAGE

La classe d'intégration **GeoPointMap** réalise tous les traitements de cartographie : affiche les informations d'une géocache, positionne une géocache sur la carte avec un marqueur [3], fusionne les géocaches, et supprime une géocache. La gestion d'envoi de SMS sera également intégrée à ce niveau.

Voici la classe principale :

Fichier

```
package mag.linux.android_sensors;
...
public class GeoPointMap extends AppCompatActivity implements
OnMapReadyCallback {

   final String TAG="GEOMAP";
   // Google map
   private GoogleMap mMap;
   // List of geocache
   private ArrayList<String> tagListName = new ArrayList<String>();
   ArrayAdapter<String> arrayAdapter = null;
   // UI
   ListView lv;
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

```
    // Geocache points
    int geoSelected;
    GeoPoint [] geoPointTab=null;
    GeoPoint [] geoPointTabOther=null;
    GeoPoint [] tabGeo=null;
    // sectioned geocache
    LatLng lastGeo;
    String geoTitle;
    String geoName;
    String geoInfo;
    DataManager storageGeo;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        computeGeoPoint();
        setContentView(R.layout.activity_geo_point_map);
        // Obtain the SupportMapFragment and get notified when the map is
ready to be used.
        SupportMapFragment mapFragment = (SupportMapFragment)
getSupportFragmentManager()
                .findFragmentById(R.id.map);
        arrayAdapter = new ArrayAdapter<>(
                this, android.R.layout.simple_list_item_1, tagListName);
        lv = (ListView) findViewById(R.id.listViewTagList);
        lv.setAdapter(arrayAdapter);
        arrayAdapter.notifyDataSetChanged();
        // select a geocache point
        lv.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int
position, long id) {
                view.setSelected(true);
                geoSelected=position;
                String name = tagListName.get(geoSelected);

                if(!name.trim().isEmpty())
                addGeoMarker(tabGeo[position]);
            }
        });
        // listView long click show geocache info
        lv.setOnItemLongClickListener(new AdapterView.
OnItemLongClickListener() {
            ...
        });
        mapFragment.getMapAsync(this);
    }
...
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        //ajoute les entrées de menu_test à l'ActionBar
        getMenuInflater().inflate(R.menu.menu, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()){
            case R.id.action_delete:
                removeElement();
                return true;
            case R.id.action_share:
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thiemer@businessdecision.com)

```
                sendSMS();
                return true;
        }
        return super.onOptionsItemSelected(item);
    }
...
   @Override
   public void onMapReady(GoogleMap googleMap) {
       mMap = googleMap;
       mMap.getUiSettings().setZoomControlsEnabled(true);

       // Add a marker and move the camera
      mMap.addMarker(new MarkerOptions().position(lastGeo).
title(geoTitle));
      mMap.moveCamera(CameraUpdateFactory.newLatLng(lastGeo));
   }
   ...
}
```

3.1 Afficher les informations d'une géocache

L'écran peut être de taille limitée. Nous avons choisi ici d'utiliser une boîte de dialogue pour afficher les détails d'une géocache. Un appui long sur une géocache de la liste permet d'afficher les détails d'une géocache :

Fichier

```
// listView long click show geocache info
lv.setOnItemLongClickListener(new AdapterView.OnItemLongClickListener() {
   @Override
   public boolean onItemLongClick(AdapterView<?> arg0, View view, int
position, long id) {
        view.setSelected(true);
        geoSelected=position;
        String name = tagListName.get(geoSelected);
        if(!name.trim().isEmpty()) {
        Dialog myTextDialog= myTextDialog(tabGeo[position]);
        myTextDialog.show();
        }
        return true;
   }
});
```

Les détails de la Géocache sont affichés par **Dialog myTextDialog()** :

Fichier

```
private Dialog myTextDialog(GeoPoint mGeoPoint) {
   final View layout = View.inflate(this, R.layout.dialog_geo_info, null);
   TextView textViewName = (TextView) layout.findViewById(R.
id.textDiagViewName);
   TextView textViewLat = (TextView) layout.findViewById(R.
id.textDiagViewLat);
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

```
    TextView textViewLng = (TextView) layout.findViewById(R.
id.textDiagViewLng);
    TextView textViewInfo = (TextView) layout.findViewById(R.
id.textDiagViewInfo);
    textViewName.setText(mGeoPoint.getName());
    textViewLat.setText(String.valueOf(mGeoPoint.getLatitude()));
    textViewLng.setText(String.valueOf(mGeoPoint.getLongitude()));
    textViewInfo.setText(mGeoPoint.getInfo());
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    builder.setIcon(0);
    builder.setPositiveButton("OK", new Dialog.OnClickListener() {
        public void onClick(DialogInterface dialog, int which) {
        }
    });
    builder.setView(layout);
   return  builder.create();
}
```

3.2 Position d'une géocache sur la carte

À chaque appui sur une géocache de la liste, une Géocache est imprimée sur la carte avec la méthode **addGeoMarker()**. Cela implique aussi la suppression sur la carte de la précédente géocache :

Fichier

```
void addGeoMarker(GeoPoint mGeoCache)
{
    LatLng geoCache = new LatLng(mGeoCache.getLatitude(), mGeoCache.
getLongitude());
    mMap.clear();
    mMap.addMarker(new MarkerOptions().position(geoCache).title(mGeoCache.
getName()));
    mMap.moveCamera(CameraUpdateFactory.newLatLng(geoCache));
}
```

3.3 Fusion des géocaches

Les géocaches peuvent provenir de vos recherches, mais aussi du fichier **Other.xml** récupéré en P2P en Bluetooth. Les fichiers **Other.xml** et **Data.xml** sont fusionnés dans une même liste puis affichés. La suppression d'une géocache de cette liste entraîne aussi sa suppression dans le fichier de stockage correspondant. La méthode **computeGeoPoint()** réalise l'ensemble des opérations :

Fichier

```
void computeGeoPoint()
{
    int sizeGeoPointList=0;
    int ind=0;
    storageGeo = MainActivity.getDataManager();
    // my geocache
    geoPointTab = storageGeo.getContent(DataManager.xmlFile);
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

```
    // other geocache
    geoPointTabOther = storageGeo.getContent(DataManager.xmlFileOther);

    if(geoPointTab!=null && geoPointTab.length>0) {
        sizeGeoPointList += geoPointTab.length;
    }
    if(geoPointTabOther!=null && geoPointTabOther.length>0) {
        sizeGeoPointList += geoPointTabOther.length;
    }
    if(sizeGeoPointList >0)   {
        tabGeo =  new GeoPoint[sizeGeoPointList];
        if(geoPointTab!=null && geoPointTab.length>0) {
            for(int i=0; i<geoPointTab.length;i++) {
                ind++;
                tabGeo[i]=geoPointTab[i];
            }
        }
        if(geoPointTabOther!=null && geoPointTabOther.length>0) {
            for(int i=0; i<geoPointTabOther.length;i++) {
                tabGeo[ind]=geoPointTabOther[i];
                ind++;
            }
        }
    }

    if(tabGeo != null && tabGeo.length>0)   {
        for(int i=0; i<tabGeo.length; i++) {
            ind++;
            tagListName.add(tabGeo[i].toString());
        }
        lastGeo = new LatLng(tabGeo[0].getLatitude(), tabGeo[0].
getLongitude());
        geoInfo = tabGeo[0].getInfo();
        geoName = tabGeo[0].getName();
    }else{ // default point
        tagListName.add(" ");
        lastGeo = new LatLng(48.866667, 2.333333);
        geoTitle = "Paris";
    }
}
```

3.4 Alerte par SMS

Le P2P Bluetooth ne permet pas des échanges à longue portée. Pour échanger des géocaches, nous pouvons également utiliser les SMS. Le menu ***share*** de l'action barre permet de déclencher un SMS avec une saisie automatique des informations de géolocalisation :

Fichier

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()){
        case R.id.action_delete:
            removeElement();
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

```
            return true;
        case R.id.action_share:
            sendSMS();
            return true;
    }
    return super.onOptionsItemSelected(item);
}

private void sendSMS()
{
  GeoPoint geoPoint = tabGeo[geoSelected];
    final int PERMISSION_REQUEST_CODE = 1;
    if (android.os.Build.VERSION.SDK_INT >= android.os.Build.VERSION_
CODES.M) {
        if (checkSelfPermission(android.Manifest.permission.SEND_SMS) ==
PackageManager.PERMISSION_DENIED) {
            Log.d("permission", "permission denied to SEND_SMS - requesting
it");
            String[] permissions = {android.Manifest.permission.SEND_SMS};
            requestPermissions(permissions, PERMISSION_REQUEST_CODE);
        }
    }
    String msg = "geocache: " + geoPoint.getName() + " " + String.
valueOf(geoPoint.getLatitude()) + " " +    String.valueOf(geoPoint.
getLongitude());
    Intent smsIntent = new Intent(Intent.ACTION_SENDTO);
    smsIntent.addCategory(Intent.CATEGORY_DEFAULT);
    smsIntent.setType("vnd.android-dir/mms-sms");
    smsIntent.putExtra("sms_body",msg);
    smsIntent.setData(Uri.parse("sms:"));
    startActivity(smsIntent);
}
```

3.5 Tests

Voici les étapes permettant de tester l'application :

1 Sauvegarder une géocache (voir figure 2): cliquez sur ***save location***.

Extrait du log :

Fichier

```
...
25914-25914/mag.linux.android_sensors V/XML: addGeoPoint
```

2 Appuyez sur ***Afficher Géocache***, la carte doit apparaître avec la Géocache (voir figure 3).

Extrait du log :

Fichier

```
25914-25914/mag.linux.android_sensors V/XML: Geo 2
25914-25914/mag.linux.android_sensors V/XML: 43.77409084
25914-25914/mag.linux.android_sensors V/XML: 1.43368426
25914-25914/mag.linux.android_sensors V/XML: info 2
...
25914-25914/mag.linux.android_sensors V/XML: XML : NODE DELETE Geo 2
25914-25914/mag.linux.android_sensors I/System.out: DONE
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.lhermier@businessdecision.com)

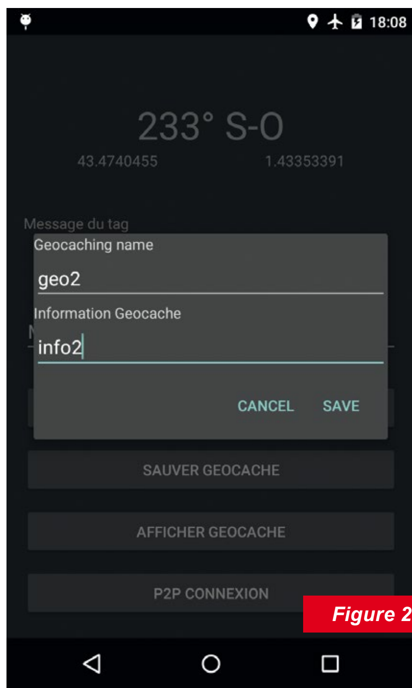


Figure 2

Sauvegarde d 'une géocache.

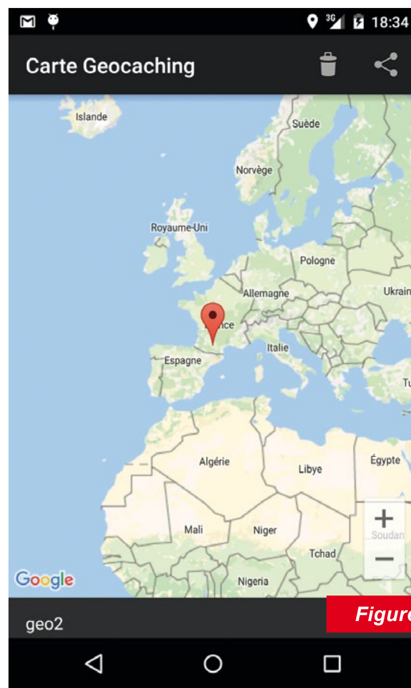


Figure 3

Affichage d 'une géocache.

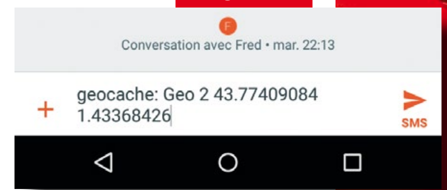


Figure 4

Envoi d 'une géocache par SMS.

3 Sélectionnez la Géocache et appuyez sur le symbole ***Share*** de la barre de menu. Un sms (voir figure 4) est configuré avec les détails de la Géocache.

4 Sélectionnez la Géocache de la liste et appuyez sur le symbole ***delete*** de la barre de menu, la Géocache est supprimée de la liste. ■

Synthèse

L'application peut maintenant annoncer une géocache par SMS avec le support du *framework* d'application. Le code est d'une grande simplicité, c'est l'avantage des *Intents*.

Le *wizard* d'Android Studio facilite grandement l'intégration de la cartographie. Lors du déploiement, n'oubliez pas d'utiliser une clé d'activation Google Maps API avec un certificat de *release*, nous aborderons ce point dans la suite. Avec l'intégration de la cartographie, nous avons découvert l'utilisation des fragments d'affichage qui se généralise de plus en plus dans les applications Android.

RÉFÉRENCES

[1] Fragment d'affichage : **https://developer.android.com/guide/components/fragments.html**, **https://developer.android.com/training/basics/fragments/index.html**, et **https://developer.android.com/training/basics/fragments/creating.html**

[2] Clé API : **https://developers.google.com/maps/documentation/android-api/signup**

[3] Ajouter un marqueur : **https://developers.google.com/maps/documentation/android-api/map-with-marker**

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

PUBLIEZ & MONÉTISEZ

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

4 PUBLIEZ & MONÉTISEZ

À découvrir dans cette partie...

Préparez votre application pour le store

Le projet n'est pas terminé ! Il faut tester correctement l'application à l'aide de tests unitaires, générer un nouveau certificat, signer l'application et l'optimiser. p. 92

Touchez le plus de monde possible grâce à l'internationalisation

Si vous voulez faire en sorte que votre application soit diffusée le plus largement possible, il va falloir passer par une étape de traduction... p. 106

Diffusez votre application sur Google Play

Pour que d'autres utilisateurs puissent bénéficier de votre merveilleuse application, il faut la partager et pour cela la diffuser (gratuitement ou pas), soit par mail soit sur le Google Play. p. 110

Gagnez de l'argent ! Beaucoup d'argent... ?

Une fois votre application achevée, vous pourrez choisir différentes stratégies pour tenter de gagner quelque argent (avec une bannière de publicités par exemple). p.118

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

PRÉPAREZ VOTRE APPLICATION POUR LE STORE

Avant de diffuser votre application, il va encore falloir la tester correctement, générer un certificat, signer l'application, et l'optimiser.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

À cette étape du projet, il nous reste deux blocs fonctionnels à réaliser correspondant aux cas d'utilisation « Acheter Produit » et « Afficher Publicité ». Les IHM ont été développées au fil de l'eau, mais nous allons les améliorer avec le concept d'interface adaptative.

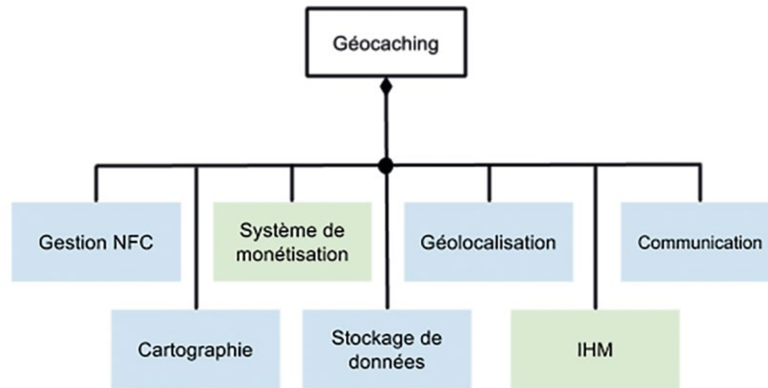


Schéma des blocs fonctionnels géocaching.

Figure 1

La première version de votre application est quasiment prête ! Il est temps de la préparer avant de la diffuser via les différents « store ». Cette phase est importante et nécessaire, car les différents « store » ne peuvent accepter votre application en l'état. L'application doit être optimisée afin de réduire son poids et signée par un certificat numérique de *release*. Google propose un canevas de publication qui présente les étapes essentielles [1][2].

1. IDENTIFICATION DE L'APPLICATION

Les informations d'identification [3] se trouvent dans le fichier **`build.gradle`**, elles permettent d'identifier et versionner votre application :

Fichier

```
android {
   signingConfigs {
       release {
           keyAlias 'keyRelease'
           storeFile file('/home/android/.android/releasekey.jks')
       }
   }
   compileSdkVersion 23
   buildToolsVersion '25.0.0'
   defaultConfig {
       applicationId "linuxmag.app.hs.store"
       minSdkVersion 16
       targetSdkVersion 23
       versionCode 1
       versionName "1.0"
       signingConfig signingConfigs.release
   }
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Google Play Console utilisera également ces informations lors du déploiement de vos applications. Voici la signification de certains des champs :

- `applicationId` : permet de différencier l'application des autres sur le store de Google. Cet identifiant doit toujours être le même et doit être signé avec le même certificat, si l'application ID ou le certificat sont différents alors l'application est considérée comme une nouvelle application.

NOTE

`applicationId` ne doit pas être confondu avec le nom de *package* de l'application : ils sont indépendants, même si Android Studio reporte le nom de *package* dans cette variable au moment de la création du projet. Vous pouvez modifier le nom de *package* de votre application, sans que cela n'affecte cette variable.

- `versionCode` : représente le numéro de version interne (entier entre 1 et 2 100 000 000). Information utilisée comme référence uniquement pour le développement de l'application.
- `versionName` : c'est la version officielle qui est présentée aux utilisateurs. En principe, la syntaxe est la suivante `<major>.<minor>`. Elle peut être synchronisée avec `versionCode`, mais ce n'est pas une obligation.

Les deux directives `versionCode` et `versionName` sont par la suite recopiées dans le fichier `manifest` de l'archive APK. Il en est de même pour `minSdkVersion` et `targetSdkVersion`.

2. APPLIQUEZ VOS DERNIERS TESTS

La phase de test **[4-7]** permet de réduire considérablement les dysfonctionnements de votre application. En principe, les tests sont réalisés au fil des développements et sont codés en fonction des spécifications du projet. L'idéal serait d'avoir un jeu de tests fourni par un autre développeur afin d'assurer l'indépendance entre le code réalisé et les exigences du projet. Android studio permet de réaliser des tests unitaires et des tests d'intégration avec le support « Android Testing Support Library ».

2.1 Tests en environnement réel

Les tests unitaires et d'intégration ne couvrent pas l'utilisation de l'application dans son environnement réel. Les scénarios de test permettent de valider les fonctionnalités, découvrir des bugs logiciels, mais ne permettent pas ou difficilement de tester tous les cas de figure surtout avec des utilisations de plusieurs capteurs. À la suite des tests unitaires et d'intégration, une suite de scenarii doit être élaborée afin de démontrer le bon fonctionnement de l'application en conditions réelles.

2.2 Tests unitaires locaux

Le test unitaire ne possède en principe aucune dépendance avec le code Android. Nous pouvons tester ici des classes métiers plutôt que des comportements liés au système Android. Les tests sont exécutés dans la JVM de la machine de développement, ce qui permet d'avoir une interaction rapide. Les tests se trouvent dans le répertoire `module-name/src/test/java/` qui est créé pour chaque nouveau projet. Il est nécessaire de configurer votre fichier `build.gradle` pour intégrer **Junit4** :

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Fichier

```
dependencies {
...
// test unitaire avec JUNIT
testCompile 'junit:junit:4.12'
...
}
```

Pour créer un nouveau test unitaire, utilisons par exemple la classe `GeoPoint`. Il faut ouvrir le fichier `GeoPoint.java`, puis après appui sur <Control> + <Shift> + <t>, choisir par exemple la méthode `getName()` et cliquer sur ***OK***. Choisir ensuite le type de test (ici on choisira un test unitaire).

Un test vide est alors créé au niveau du répertoire `test` et Il faut le compléter en utilisant la bibliothèque JUnit :

Fichier

```
package mag.linux.android_sensors;

import org.junit.Test;

import static org.junit.Assert.*;

public class GeoPointTestName {
   @Test
   public void getName() throws Exception {
       assertTrue(new GeoPoint("name", 0, 0, "info").getLatitude() == 0);
   }
}
```

Pour exécuter le test unitaire, positionnez le curseur sur le test, puis effectuez un clic droit et ***Run "Test ..."***. Une fois le jeu de test exécuté, il est possible d'obtenir un rapport dans différents formats (html, xml, personnalisé). Dans la fenêtre d'exécution du test, cliquez sur l'icône rectangulaire contenant une flèche verte pointant vers le haut pour exporter l'ensemble des résultats.

2.3 Tests unitaires instrumentés

Les tests instrumentés sont exécutés dans une machine Android réelle ou un émulateur. Ils sont stockés dans le répertoire `module-name/src/androidTest/java/`. Pour créer un test instrumenté, la démarche reste la même que pour le test unitaire, mais ici vous choisirez ***AndroidTest***. Votre fichier `build.gradle` doit définir une nouvelle dépendance avec `AndroidJUnitRunner` qui est embarqué sur la cible pour exécuter le test :

Fichier

```
defaultConfig {
   ...
   testInstrumentationRunner «android.support.test.runner.
AndroidJUnitRunner"
}
```

`AndroidJUnitRunner` est une classe fournie par Android Studio qui permet d'exécuter les tests de type JUnit-3, JUnits-4, **Espresso** et **UI Automator** dans une machine Android. Cette classe charge, exécute les tests et envoie un rapport. Les tests instrumen-

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

tés fonctionnent par défaut uniquement en mode debug, pour configurer ce mode : menu ***Build > Select Build Variant > Debug***. Pour l'exécution, vous pouvez utiliser le menu contextuel : clic droit sur le test puis ***run***, le résultat s'affiche dans la fenêtre d'exécution.

Pour exécuter un test dans la console :

Terminal
```
$ adb shell am instrument -w -r   -e debug false -e class mag.linux.
android_sensors.ExampleInstrumentedTest linuxmag.app.hs.store.debug.test/
android.support.test.runner.AndroidJUnitRunner
Client not ready yet..
Started running tests
Tests ran to completion.
```

3. TYPE DE BUILD DEBUG/RELEASE

La construction ou « build » de l'application **[8]** est différente pendant la phase de développement et la phase de déploiement. Dans la phase de développement, le mode debug permet de réaliser des points d'arrêt avec le debugger, de visualiser les exceptions. Dans cette configuration, l'ensemble du code et les ressources ne sont pas optimisés. Enfin, lorsqu'arrive le déploiement sur un smartphone ou une tablette, le poids final du binaire devient un souci, il faut le réduire autant que possible. Dans cette phase, l'optimisation du code et des ressources permet de réduire parfois considérablement la taille de l'application, mais également d'augmenter ses performances.

Android Studio automatise les tâches de compilation du projet :

- Chaque application est gérée sous forme de module qui comprend l'ensemble des codes sources, les fichiers ressources (images, xml...) et les fichiers AIDL (*Android Interface Definition Language*) qui contiennent des interfaces pour la communication entre les applications.
- Chaque application embarque des bibliothèques sous forme de fichier jar. Certains modules peuvent être aussi des bibliothèques. Les fichiers AAR (*Android Archive*) sont des bibliothèques qui contrairement aux archives JAR contiennent des fichiers ressources et un `manifest`.

Le compilateur transforme les sources en fichier dex lisible par la machine virtuelle Android, les ressources subissent également un processus de compilation. Le tout est ensuite embarqué dans un fichier archive avec une extension `.apk` (*Android Package Kit*) . Un certificat de debug ou *release* est alors utilisé pour signer cette archive. L'archive APK est ensuite installée sur un smartphone ou une tablette en version debug pour le développement ou *release* pour la version livrable aux utilisateurs.

Par défaut, Android Studio crée un projet avec les modes release et debug. Le mode de compilation est défini par la section `buildTypes` du fichier `build.gradle` (`Module:app`) qui se trouve sur la racine du projet. Le mode debug n'apparaît pas directement dans la section `buidTypes`, Android Studio configure le mode debug avec la directive `debuggable true`. La configuration est également visible via le menu ***File > Project Structure*** qui permet de configurer ces deux modes.

Pour personnaliser le mode debug, ajoutons la directive `debug` dans la section `buildTypes`. Nous analyserons plus loin le contenu.

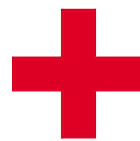
Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

CONNECT ÉVOLUE !

LISEZ CE NUMÉRO ET PLUS DE 230 AUTRES EN LIGNE !

ACTUELLEMENT SUR CONNECT :

CE NUMÉRO
et + de 160 autres numéros de GNU/Linux Magazine



75 numéros Hors-Séries de GNU/Linux Magazine

TOUT CELA À PARTIR DE 199 € TTC*/AN !

* Tarif France Métropolitaine

OFFRE DÉCOUVERTE CONNECT

1 MOIS GRATUIT, RÉSERVÉE AUX PROFESSIONNELS

Appelez le 03 67 10 00 28 et donnez le code « GLMF206 » pour découvrir Connect gratuitement pendant 1 mois !

Pour tous renseignements complémentaires, contactez-nous via notre site internet : **www.ed-diamond.com**, par téléphone : **03 67 10 00 28** ou envoyez-nous un mail à **connect@ed-diamond.com** !

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Fichier

```
...
buildTypes {
   debug {
       minifyEnabled true
       useProguard false
       proguardFiles getDefaultProguardFile('proguard-android.txt'),
                'proguard-rules.pro'
   }
   release {
       minifyEnabled true
       proguardFiles getDefaultProguardFile('proguard-android.txt'),
                'proguard-rules.pro'
       }
}
...
```

NOTE

À chaque modification du fichier `buid.gradle`, une synchronisation est alors nécessaire afin qu'Android Studio prenne en compte la nouvelle configuration. Vous trouverez l'icône de synchronisation dans la barre d'outils.

3.1 Gérer les clés Google Maps API en mode debug/release

Précédemment, nous avons créé un projet de cartographie qui par défaut génère deux fichiers `google_maps_api.xml` pour le stockage des clés d'activation Google Maps Api, respectivement pour les modes debug et *release*: `src/debug/res/values/google_maps_api.xml` et `src/release/res/values/google_maps_api.xml`. Les clés d'activation Google Maps peuvent être différentes en mode debug/release, car le certificat de debug/release peut également être différent. En fonction de votre configuration, vous devez alors placer les clés dans ces deux fichiers.

NOTE

D'une façon plus générale, il est possible d'ajouter diverses ressources appelées *sourceSet* qui seront utilisées selon le mode de production debug ou release. Ceci permet de différencier par exemple dans notre cas, les certificats de debug et release, les clés d'activation des API Google, ou d'autres ressources qui peuvent être différenciées selon ces deux modes.

3.2 Créer de nouveaux répertoires de ressources

Il est possible également de différencier des ressources en créant de nouveaux répertoires de stockage. Par exemple pour un *sourceSet* de type Java : effectuez un clic droit sur le répertoire ***main > New > Folder > Java Folder***, puis saisissez le nom du répertoire. Il en va de même pour la création d'un répertoire ressources.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

3.3 Gestion des sourceSets par gradle

La section *sourceSets* du fichier **build.gradle** permet de prendre en compte les différentes ressources du projet en fonction du mode debug/release. Cette section est normalement mise à jour dès que vous avez créé un *sourceSet* via le menu ***New > Folder***.

Fichier

```
...
android {
...
sourceSets {
   main {
       java.srcDirs = ['src/main/java', 'src/main/java/optionA']
       res.srcDirs = ['src/main/res', 'src/main/res-optionA']
   }
...
}
```

L'archive APK de debug/release est générée dans le répertoire **project_name/module_name/build/outputs/apk/**.

4. OPTIMISATION

Par défaut, le fichier **~/Android/Sdk/tools/proguard/proguard-android.txt** permet de définir les règles d'optimisation de **proGuard**. Si les règles d'optimisation par défaut ne sont pas suffisantes, le fichier **proguard-rules.pro** sur la racine du projet permet de personnaliser la compilation avec vos jeux d'instructions additionnelles.

ATTENTION !

L'optimisation peut parfois entraîner des dysfonctionnements dans l'application pendant son exécution, ceci est dû à la suppression accidentelle de certaines classes, méthodes, variables. Le logcat signale alors des erreurs de type java.lang.ClassNotFoundException. De plus, le résultat de l'optimisation est différent en fonction de la version du SDK *tools*, ou du *plugin* Gradle. Pour ces raisons, le passage à la version release nécessite encore des tests. Pour éviter la suppression de classe, méthode, proguard-rules.pro peut alors être configuré, voici un exemple :

Fichier

```
keep class com.google.android.gms.googlecertificates
-keep class com.google.android.gms.dynamite.DynamiteModule$DynamiteLoaderClassLoader
...
```

4.1 Zipalign

Zipalign [9] est un outil d'alignement d'archives qui optimise de façon importante les archives APK. Les contenus de l'archive APK y compris les images sont alignés sur 4 octets. L'alignement sur 4 octets permet d'accélérer les accès mémoires et de gagner de l'espace mémoire pendant l'exécution de l'application.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Zipalign est utilisé différemment en fonction du contexte de signature du fichier APK :

- signature de type V1 / `jarsigner` : zipalign ne doit être utilisé que lorsque le fichier APK a été signé ;
- signature de type V2 / `apksigner` : zipalign doit être effectué uniquement avant la signature du fichier APK. Si vous signez votre APK en utilisant `apksigner`, toutes modifications ultérieures de l'APK invalident sa signature.

Le processus de build d'Android Studio utilise zipalign pour réaliser ces deux types de signatures. Le principe des signatures est présenté plus loin dans la section « Signez votre application ».

4.2 Optimiser les ressources

Cette phase d'optimisation doit être réalisée après avoir testé votre application avec l'optimisation `minifyEnabled true` et Proguard, car le risque est d'avoir trop optimisé le code final et d'avoir une application qui ne s'exécute plus du tout ! L'optimisation des ressources concerne uniquement le répertoire `res/` dont certains contenus non référencés dans le code sont tout simplement supprimés. La directive `shrinkResources true` permet, en association avec `minifyEnabled true`, de réaliser cette dernière optimisation. Ici aussi, il convient d'effectuer des tests pour vérifier le bon fonctionnement de l'application.

5. COMPILATION EN MODE RELEASE/ DEBUG ET TAILLE DU FICHIER APK

Pour passer d'un type de construction à un autre, il suffit d'utiliser le menu ***Build > Select build variant***. Gradle prend ensuite en considération la section debug ou release. En phase *release*, la génération de code peut jusqu'à doubler son temps d'exécution.

Nous pouvons maintenant apprécier la taille des différents composants du fichier archive apk. Le poids du binaire est réduit de plus de moitié, ce qui est vraiment appréciable pour certains types de déploiement que nous aborderons plus loin (mode release 1,8MB et mode debug 3,7MB !).

6. CERTIFICAT

Un certificat permet en principe de garantir la provenance d'un logiciel. Le certificat est composé d'une clé publique qui est diffusée et d'une clé privée qui est secrète et protégée par un mot de passe. Dans le cas d'Android, le **Package Manager** contrôle que l'archive APK est bien signée avec la clé publique qui est embarquée dans l'archive APK. La clé publique est intégrée à l'application au moment de la génération de l'archive APK.

Jusqu'à présent, nous avons utilisé un certificat par défaut généré par Android Studio lors de son installation, celui-ci se trouve dans `~/.android/debug.key`. En mode *release*, le certificat doit être généré avec de vrais identifiants (par défaut l'émetteur est Android avec un mot de passe `android`). La signature SHA1 de ce nouveau certificat permettra d'obtenir de nouvelles clés d'activation pour l'API Google Maps. Nous utiliserons `keytool`, un outil standard de Java SE, pour générer un nouveau certificat :

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Terminal

```
$ keytool -genkey -v -keystore release.keystore -alias releasekey -keyalg
RSA -keysize 2048 -validity 10000
```

Il est également possible de créer un certificat via Android Studio : menu ***Build > Generate Signed APK > Create New***, puis saisir le formulaire de la fenêtre ***New Key Store***.

L'empreinte du certificat peut être également être obtenue avec la commande `keytool` :

Terminal

```
$ keytool -list -v -keystore ~/.android/releasekey.jks -alias keyRelease
-storepass "mypass" -keypass "mypass"
...
Empreintes du certificat :
  MD5:  FA:BE:4A:C1:C1:57:AF:CE:8B:C0:DA:AB:E7:A5:BD:EB
  SHA1 : D2:AD:39:E1:14:93:02:AD:66:1C:DA:B4:C1:45:32:CD:1A:93:07:7C
...
```

Le fichier de stockage généré contient une clé publique/privée. Celles-ci ne sont pas directement accessibles dans un format standard. Si vous voulez utiliser ce certificat pour d'autres raisons, il est nécessaire de connaître son contenu et surtout d'en extraire : la clé publique, la clé privée, et les informations du certificat.

La première opération consiste à exporter le contenu du fichier « jks » dans le format normalisé **PKCS#12** :

Terminal

```
$ keytool -importkeystore -srckeystore releasekey.jks -destkeystore
keystore.p12 -deststoretype PKCS12 -srcalias keyrelease
```

La clé privée au format **pem** est ensuite extraite à partir du format PKCS#12. Attention ! La clé privée doit être stockée dans un support absolument sécurisé :

Terminal

```
$ openssl pkcs12 -in keystore.p12  -nodes -nocerts -out key.pem
```

Le certificat au format pem peut être extrait également à partir du format PKCS#12. Nous y trouvons les informations de l'émetteur :

Terminal

```
$ openssl pkcs12 -in keystore.p12  -nokeys -out cert.pem
```

La clé publique au format pem est extraite du certificat au format pem :

Terminal

```
$ openssl x509 -pubkey -noout -in cert.pem  > pubkey.pem
```

7. SIGNER VOTRE APPLICATION

7.1 Signer manuellement votre application

Le certificat de *release* permet maintenant de signer votre application à partir du menu ***Build > Generate signed apk***. Par défaut, Android studio 2.2 et Gradle 2.2 utilisent deux types de signature. Il est recommandé d'utiliser la signature V2 **[10]**, sauf si le build

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

ne se passe pas correctement, utilisez alors uniquement la signature V1 (jar signature). Les dysfonctionnements qui peuvent apparaître sont principalement dus à la configuration de l'environnement de développement. L'archive APK est ensuite générée dans le répertoire de destination de votre choix :

- **Signature type V1 - Signature JAR** : Cette signature est utilisée depuis les débuts d'Android. Cependant, cette signature V1 ne protège pas certaines parties de l'APK, telles que les métadonnées ZIP. Ce qui entraîne certaines failles de sécurité. De plus, le processus de vérification est plus long, car il doit décompresser les entrées compressées. Ce type de signature est voué à disparaître à partir d'Android 7 qui utilise le schéma de compression V2.
- **Signature type V2 / Signature APK** : Cette signature est plus fiable que la version V1. Le contenu est haché puis signé, le résultat est ensuite inséré dans l'archive. Toute modification d'un contenu de l'archive invalide la signature contrairement à la signature de type V1.

NOTE

Le schéma de signature APK V2 est possible à partir de Android 7.0 (Nougat). Pour utiliser ce type de signature avec Android 6.0 (Marshmallow) et les versions précédentes, l'archive APK doit tout d'abord être signée avec la version V1 puis la version V2.

7.2 Automatiser la signature de l'application

Android Studio permet de gérer un ensemble de signatures en fonction du mode de production et de signer l'archive apk générée. L'interface de gestion se trouve dans ***File > Project Structure***. Il est possible de déclarer plusieurs types de signatures dans l'onglet ***Signing***, chaque signature fait référence à un fichier qui contient un certificat. Pour créer un type de signature, cliquez sur **+** et saisissez les données du formulaire. Ensuite, dans l'onglet ***Build Types***, il faut associer cette signature à un type de build.

Le fichier `build.gradle` est impacté. Une nouvelle section `signingConfigs` apparaît avec le détail de la configuration :

Fichier

```
android {
   signingConfigs {
       release {
           keyAlias 'relaseKey'
           storeFile file('~/.android/keystore.jks')
           keyPassword 'myPass'
           storePassword ''myPass''
       }
   }
   ...
```

ATTENTION !

Par défaut, les mots de passe sont recopiés dans le fichier `build.gradle`, si vous n'y prêtez pas garde vous les retrouverez dans votre dépôt de code.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

LINUX MAGAZINE / FRANCE

M'abonner ?

Me réabonner ?

Compléter ma collection en papier ou en PDF ?

Pouvoir lire en ligne mon magazine préféré ?

C'est simple... c'est possible sur :

http://www.ed-diamond.com

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Pour éviter le stockage des mots de passe [11], il est préférable de les saisir au moment de la signature de l'application. Pour cela, nous allons modifier le fichier **build.gradle** sur la racine du projet. Au moment du build, ce code permet d'afficher une fenêtre de saisie du mot de passe. La saisie se fera uniquement au moment de la génération de l'archive APK, les builds intermédiaires ne le nécessitent pas :

Fichier

```
apply plugin: 'com.android.application'

import groovy.swing.SwingBuilder

gradle.taskGraph.whenReady { taskGraph ->
   if (taskGraph.hasTask(':app:assembleRelease')) {
       def storePassword = ''
       def keyPassword = ''
       if (System.console() == null) {
           new SwingBuilder().edt {
               dialog(modal: true, title: 'Enter password', alwaysOnTop:
true, resizable: false, locationRelativeTo: null, pack: true, show: true) {
                   vbox {
                       label(text: «Please enter store passphrase:")
                       def input1 = passwordField()
                       label(text: «Please enter key passphrase:")
                       def input2 = passwordField()
                       button(defaultButton: true, text: 'OK',
actionPerformed: {
                           storePassword = input1.password;
                           keyPassword = input2.password;
                           dispose();
                       })
                   }
               }
           }
       } else {
           storePassword = System.console().readPassword(«\nPlease enter
store passphrase: «)
           keyPassword = System.console().readPassword(«\nPlease enter key
passphrase: «)
       }
       if(storePassword.size() <= 0 || keyPassword.size() <= 0) {
           throw new InvalidUserDataException(«You must enter the passwords
to proceed.")
       }
   storePassword = new String(storePassword)
       keyPassword = new String(keyPassword)
       android.signingConfigs.release.storePassword = storePassword
       android.signingConfigs.release.keyPassword = keyPassword
   }
}
android {
...
```

Pendant la phase d'installation sur la cible, le processus d'installation vérifie le type de signature V1 ou V2 puis calcule les empreintes avec le certificat présent dans l'archive. Si une signature n'est pas valide alors l'application n'est pas installée.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

8. ANALYSER LE CONTENU DE L'ARCHIVE APK

L'analyse de l'archive apk permet une dernière vérification du contenu de notre application : `manifest`, signature, version, ressources embarquées. L'archive apk est générée via le menu : ***Build > Build APK***. Vous pouvez décompacter l'archive de la même façon qu'un zip, et consulter le contenu. Ici, le fichier `manifest` est binaire, il n'est pas possible de le consulter.

Android Studio possède un analyseur d'archive qui permet de consulter tout le contenu d'une archive APK. L'analyseur est disponible à partir du menu ***Build > Analyse APK***. Il est possible de consulter l'ensemble des signatures, les ressources embarquées, le certificat qui a signé l'archive, le volume de chaque objet dans l'archive. Le fichier dex permet de vérifier le nombre de méthodes appelées dans votre programme.

Une partie de la configuration `build.gradle` est recopiée dans le `manifest` de l'archive, ainsi on y retrouve les directives suivantes : `versionCode`, `versionName`, `platformBuildVersionCode` et `platformBuildVersionName` qui correspond à la version cible du SDK Android. ■

Synthèse

L'étape de préparation est essentielle à la réussite du projet, certaines étapes, comme la génération d'un nouveau certificat et la signature de l'application sont des points de passage obligatoires. La phase d'optimisation est particulièrement importante si vous décidez de diffuser votre application par des moyens plus légers qu'un store classique.

RÉFÉRENCES

[1] **https://developer.android.com/studio/publish/preparing.html**

[2] **https://developer.android.com/distribute/best-practices/launch/launch-checklist.html**

[3] **https://developer.android.com/studio/build/application-id.html**

[4] **https://developer.android.com/studio/test/index.html**

[5] **https://developer.android.com/training/testing/start/index.html**

[6] **https://developer.android.com/training/testing/index.html**

[7] **https://developer.android.com/topic/libraries/testing-support-library/index.html**

[8] **https://developer.android.com/studio/build/build-variants.html**

[9] **https://developer.android.com/studio/command-line/zipalign.html**

[10] **https://source.android.com/security/apksigning/v2**

[11] **https://www.timroes.de/2014/01/19/using-password-prompts-with-gradle-build-files/**, **http://stackoverflow.com/questions/19487576/gradle-build-null-console-object**, et **https://coderwall.com/p/zrdsmq/signing-configs-with-gradle-android**

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

INTERNATIONALISEZ VOTRE APPLICATION ET TOUCHEZ LE PLUS DE MONDE POSSIBLE

Un facteur limitant à l'utilisation d'une application peut être la barrière de la langue. En internationalisant votre application, vous lui garantissez une meilleure visibilité.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Vous pouvez choisir une diffusion de l'application au niveau international [1-3] afin de toucher un maximum de mobinautes. Dans cette approche, votre application doit être lisible dans plusieurs langues. Dans cet article, nous allons préparer notre application pour une diffusion internationale avec une version en français par défaut et une version en langue anglaise.

Les ressources du projet sont situées dans `res/layout` pour la description IHM. Cette structuration est créée automatiquement par Android Studio. Le répertoire `res/` contient les ressources de l'application, il est ensuite embarqué dans l'archive APK comme nous l'avons précédemment vu. Les autres répertoires ne sont pas embarqués. Il est impératif de respecter cette structuration, car les API fonctionnent selon ce principe :

Terminal

```
MyProject/
    src/
        MyActivity.java
    res/
        drawable/
            graphic.png
        layout/
            main.xml
            info.xml
        mipmap/
            icon.png
        values/
            strings.xml
```

1. LES TYPES DE RESSOURCES

Le répertoire `res/` contient des ressources de différents types que nous utiliserons pour internationaliser l'application :

Répertoire	Type de ressource
`mipmap/`	Ces répertoires contiennent les icônes de l'application pour différentes densités d'écran (hdpi, mdpi...).
`layout/`	Définition des IHM de l'application en format xml.
`menu/`	Définition des menus de l'application en format xml.
`values/`	Ensemble de fichiers au format xml pour décrire par exemple les chaînes de caractères, les thèmes graphiques de l'application, couleurs, etc. Pour faciliter la compréhension des ressources, le nom des fichiers suit certaines conventions. Par exemple, nous pouvons retrouver les fichiers suivants : ⇨ `arrays.xml` : ressources sous forme de tableau, ⇨ `colors.xml` : pour les couleurs, ⇨ `dimens.xml` : pour les dimensions, ⇨ `strings.xml` : pour les chaînes de caractères, ⇨ `styles.xml` : pour les styles d'affichage.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

2. TRADUCTION

Le fichier res/strings.xml contient les textes qui sont affichés dans votre application. Pendant la phase de développement, il faut reporter les textes des *layout* dans ce fichier et éviter autant que possible le codage en dur » de l'affichage dans le code source Java. Ce fichier doit être ensuite traduit et placé dans un répertoire spécifique. À ce stade, le fichier res/strings.xml contient les informations suivantes :

Fichier

```
<resources>
   <string name="app_name">GeoCaching</string>
   <string name="title_activity_geo_point_map">Geocaching map</string>
   <string name="action_delete">Delete</string>
   <string name="action_share">Share</string>
</resources>
```

Commençons par le contenu des IHM, et regardons plus précisément le contenu de res/layout/activity_main.xml. Certains textes sont écrits dans le layout. Ces textes doivent être retirés de ce fichier puis écrits dans le fichier res/values/strings.xml. Le fichier res/layout/activity_main.xml fait maintenant référence à res/values/strings.xml. Par exemple, le bouton buttonWriteNFC utilise la chaîne de caractères android:text=»@string/btnWriteMsg», soit <string name=»btnWriteMsg»>Ecrire tag</string>.

Le fichier res/values/strings.xml contient maintenant tous les textes de l'application en français. Il reste encore deux étapes, traduire tous les textes en conservant les mêmes noms pour les variables, puis déclarer une nouvelle ressource dans le projet en conservant le même nom de fichier, c'est-à-dire strings.xml. Android Studio va nous permettre de définir cette nouvelle ressource : à partir du répertoire values en effectuant un clic droit puis ***New > Values resource file***, dans la fenêtre qui apparaît saisir le nom du fichier string.xml, choisir le *qualifier Locale*, puis cliquer sur **>>**. Choisir ensuite le langage (ici ***en:English***) puis cliquer sur ***OK***. (voir figure 1).

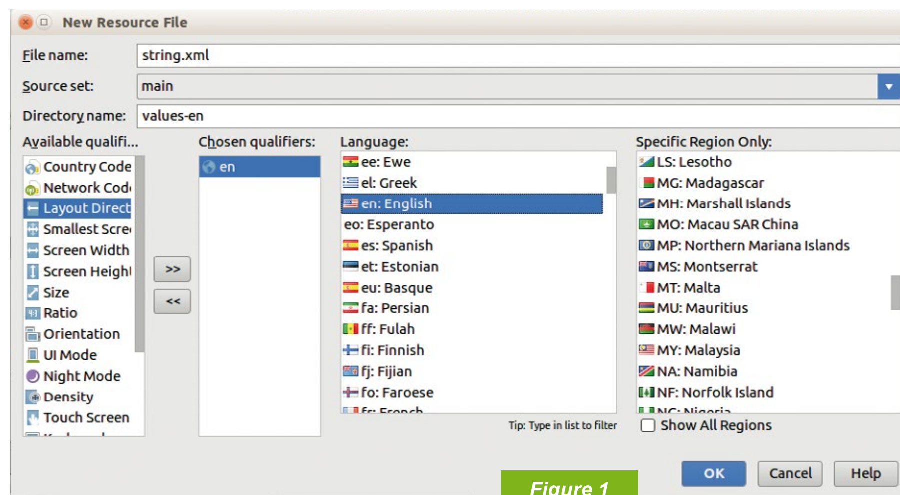


Figure 1

Spécifier le langage de la ressource.

Android Studio crée un nouveau répertoire values-en dans lequel il place un fichier string.xml. L'extension « en » provient de la norme ISO 639-1 qui décrit chaque langage par un code à deux lettres. Ce fichier doit maintenant contenir tous les textes traduits, ici en anglais :

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Fichier

```
<resources>
   <!-- App info -->
   <string name="app_name">GeoCaching</string>
   <string name="title_activity_geo_point_map">Geocaching map</string>

   <!-- App menu -->
   <string name="action_delete">Delete</string>
   ...
</resources>
```

Pour tester le fonctionnement de la traduction, il suffit de changer le langage de votre équipement dans ***Paramètres > Langue***, choisir ***English USA***, et redémarrer votre application : votre application est internationalisée ! Vous pouvez ajouter autant de langages que vous voulez selon ce même principe.

Au moment de lancer l'activité, le système fait un « routage » vers les fichiers ressources de l'application qui sont les plus proches de la configuration du système, ici il s'agit des ressources de type texte. Ce principe est également identique pour l'affichage des écrans où le système détecte le mode portrait ou paysage, la densité de l'écran, etc. Ainsi vous pouvez définir des applications particulièrement bien adaptées au contexte d'utilisation. N'oubliez pas que chaque ressource doit conserver le même nom de fichier et le même nom pour les variables, celles-ci seront utilisées par le système avec des valeurs différentes en fonction du contexte.

3. TRADUCTION ACCÉLÉRÉE

Android Studio peut vous faciliter la tâche de traduction avec un éditeur intégré. Pour y accéder, effectuez un clic droit sur le fichier `res/values/strings.xml` puis ***Open translations Editor***. L'éditeur permet de visualiser les deux fichiers qui contiennent les textes de l'application. Il est également possible d'ajouter un langage supplémentaire en cliquant sur l'icône représentant la Terre. Enfin, Google vous propose un service payant de traduction en ligne, pour cela cliquez sur ***Order a translation*** en haut à droite de cette même fenêtre. Le coût de traduction pour notre configuration est de l'ordre de six euros. ∎

Synthèse

Nous avons vu comment traduire les différents messages présents au sein d'une application de manière à l'internationaliser. Cette étape est très importante pour garantir le plus grand rayonnement possible à votre application. Si vous n'êtes pas à même de traduire vous-même certains textes, Google propose un service (payant) de traduction.

RÉFÉRENCES

[1] **https://developer.android.com/training/basics/supporting-devices/languages.html**

[2] **https://developer.android.com/distribute/best-practices/launch/localization-checklist.html**

[3] **https://cloud.google.com/translate/**

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

DIFFUSEZ VOTRE APPLICATION SUR GOOGLE PLAY

Il est temps de penser aux autres utilisateurs et de diffuser enfin notre application. Libre à vous de choisir si celle-ci doit être gratuite ou payante.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Avant de passer à l'ultime étape de notre projet et déployer notre application, nous allons réaliser quelques opérations de « nettoyage ».

1. SUPPRIMER LES LOGS ET ANDROID:DEBUGGABLE

Les logs vers le Logcat doivent être supprimés ou suspendus, le mode *release* ne doit pas générer des logs inutiles.

Le logcat peut être exécuté de façon conditionnelle en fonction du mode de build. La variable `static BuildConfig.DEBUG` permet de déterminer le mode de build courant. Il est alors possible de conditionner l'utilisation du logcat :

Fichier

```
...
if(BuildConfig.DEBUG) {
   Log.v(TAG, intent.getAction());
}
...
```

Le mode de build est choisi via le menu ***Build > Select Build Variant***, sélectionner ***release*** ou ***debug***. Dans le répertoire `app/build/generated/source/debug` et `app/build/generated/source/release`, sous le nom de *package*, se trouve le fichier `BuildConfig.java` qui contient des variables. Ces deux fichiers sont utilisés selon le mode de build :

Fichier

```
package mag.linux.android_sensors;

public final class BuildConfig {
 public static final boolean DEBUG = false;
 public static final String APPLICATION_ID = "linuxmag.app.hs.store.release";
 public static final String BUILD_TYPE = "release";
 public static final String FLAVOR = "";
 public static final int VERSION_CODE = 3;
 public static final String VERSION_NAME = "1.2";
}
```

Un nettoyage plus radical consiste à rechercher tout simplement le mot-clé `Log` dans tout le code pour le supprimer ou le commenter.

Dans le fichier `manifest` embarqué dans l'archive APK, la directive `android:debuggable` doit être enlevée, il est possible de le vérifier via l'analyseur APK : menu ***Build > Analyse APK***, puis rechercher la version de *release* (en principe dans `build/outputs/apk`).

2. CHOISIR L'ICÔNE

L'icône illustre le cœur de votre application, c'est en quelque sorte la vitrine extérieure. Son choix est stratégique, car une fois déployé, il sera difficile de la changer, car elle deviendra vite un repère visuel. Google vous propose un *template* d'icône au format

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Photoshop qui vous permettra d'affiner votre choix **[1]**. L'icône doit être générée dans différents formats correspondant à la densité d'affichage des différents équipements : `ldpi` (*low*), `mdpi` (*medium*), `hdpi` (*high*), `xhdpi` (*extra-high*), `xxhdpi` (`extra-extra-high`), et `xxxhdpi` (`extra-extra-extra-high`).

3. LE MODÈLE DE PERMISSION

Une application Android requiert des permissions **[2-5]** afin d'accéder à certaines fonctionnalités du système. Le modèle de sécurité Android est basé en partie sur ce principe. Celui-ci a connu quelques modifications à partir d'Android 6 (API 23). Ce changement implique une gestion de la sécurité différente qu'il faut prendre en compte lors de la phase de déploiement. Le modèle de permission se base sur deux types de permission :

- **Permission normale** : Android définit une permission normale comme étant sans risque pour l'application, les données de l'utilisateur, le système. Celles-ci sont inscrites dans le `manifest` et sont accordées sans interaction avec l'utilisateur.
- **Permission dangereuse** : Une permission est considérée comme dangereuse si par exemple elle permet l'accès aux contacts, ou au système de géolocalisation.

Si le `manifest` contient des permissions normales, alors ces permissions seront accordées par le système. Si les permissions sont dangereuses alors le système va réagir différemment en fonction de la version d'Android :

- Android 6 (API 23) et supérieure : les permissions sont demandées à l'utilisateur pendant l'exécution de l'application, l'utilisateur peut refuser une ou plusieurs permissions. Si les permissions sont accordées par l'utilisateur alors le système ne les redemandera plus. Les permissions refusées seront demandées à nouveau à chaque utilisation de l'application.
- Android 5.1 (API 22) et inférieure : le système demande à l'utilisateur d'accorder les permissions lors de l'installation de l'application.

Le modèle de permission est pris en compte dans la classe `MainActivity`. Sous Android 6 et supérieur, la méthode `checkAndRequestPermissions()` demande les permissions à l'utilisateur :

Fichier

```
private void checkAndRequestPermissions() {
   String [] permissions=new String[]{
           android.Manifest.permission.SEND_SMS,
           android.Manifest.permission.ACCESS_COARSE_LOCATION,
           android.Manifest.permission.ACCESS_FINE_LOCATION
   };

   int i=1;
   List<String> listPermissionsNeeded = new ArrayList<>();
   for (String permission:permissions) {
       PERMISSION_REQUEST_CODE=i++;
       if (ContextCompat.checkSelfPermission(this,permission )!=
PackageManager.PERMISSION_GRANTED){
           listPermissionsNeeded.add(permission);
       }
   }
   if (!listPermissionsNeeded.isEmpty()) {
       ActivityCompat.requestPermissions(this, listPermissionsNeeded.
toArray(new String[listPermissionsNeeded.size()]), 1);
   }
}
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

La méthode de *callback* **onRequestPermissionsResult()** capture la réponse de l'utilisateur :

Fichier
```
...
static boolean PERM_ACCESS_COARSE_LOCATION=false;
static boolean PERM_ACCESS_FINE_LOCATION=false;
static boolean PERM_SMS=false;
...
public void onRequestPermissionsResult(int requestCode, String[] permissions,
int[] grantResults) {
  for(int i=0;i<permissions.length;i++) {
      if (permissions[i].equals("android.permission.SEND_SMS")) {
          if(grantResults[i]==PackageManager.PERMISSION_GRANTED){
              PERM_SMS=true;
          }
      }
      if (permissions[i].equals("android.permission.ACCESS_COARSE_LOCATION")) {
          if(grantResults[i]==PackageManager.PERMISSION_GRANTED){
              PERM_ACCESS_COARSE_LOCATION=true;
          }
      }

      if (permissions[i].equals("android.permission.ACCESS_FINE_LOCATION")) {
          if(grantResults[i]==PackageManager.PERMISSION_GRANTED){
              PERM_ACCESS_FINE_LOCATION=true;
          }
      }
      Log.d(TAG, "mypermission = " + permissions[i]);
      Log.d(TAG, "myGrant = " + grantResults[i]);
  }
}
```

Les logs permettent de vérifier le contenu des tableaux **String[] permissions**, **int[] grantResults** de la méthode **onRequestPermissionsResult()** :

Fichier
```
05-21 16:17:18.466 3831-3831/? D/sensor: mypermission = android.permission.SEND_SMS
05-21 16:17:18.466 3831-3831/? D/sensor: myGrant = 0
05-21 16:17:18.466 3831-3831/? D/sensor: mypermission = android.permission.ACCESS_
COARSE_LOCATION
...
```

4. DIFFUSION DE VOTRE APPLICATION

Après la phase de préparation vient maintenant la diffusion de votre application. Nous allons donc aborder précisément les modalités de diffusion d'une application Android.

4.1 Diffusion directe

Vous pouvez diffuser votre application par vos propres moyens sans passer par un site marchand. Votre application doit être signée, c'est la seule obligation pour ce type de diffusion. Dans ce processus de diffusion, vous devez vous-même gérer la monétisation de votre application, cela demande bien sûr une logistique spécifique, mais vous n'aurez pas d'intermédiaire commercial.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Par défaut, Android fait confiance uniquement aux applications provenant de Google Play, le store de Google. Pour la diffusion par vos propres moyens, la configuration des machines cibles doit être modifiée pour accepter les applications hors Google Play. Il faut autoriser les « sources inconnues » via le menu ***Paramètres > Sécurité > Unknown sources***. Ce type de diffusion a ses limites, car il faut faire confiance à une application inconnue. Pour éviter ceci, il est préférable de passer par un store « officiel ».

4.2 Diffusion sur un store

Pour optimiser vos chances de succès, il est possible de diffuser votre application sur un site marchand spécialisé que l'on appelle plus communément « store ». En quelques années, le nombre de *stores* a explosé. Une concurrence féroce s'est installée entre les différents *leaders* de ce marché. Au début Google Play était le store de référence pour les applications Android, et très vite d'autres enseignes ont créé de nouveaux points de diffusion. Vient ensuite la question de diffuser une application gratuite ou payante. Il tient au développeur de définir sa stratégie.

Initialement lancé en octobre 2008 sous le nom Android Market, Google Play est le site officiel de Google pour la diffusion des applications Android. Les contenus sont très divers : applications, musiques, magazines, livres, cinéma et la télévision. En mai 2016, 65 milliards d'applications ont été téléchargées depuis Google Play. En mars 2017, le nombre d'applications disponibles dans Google Play Store a atteint 2,8 millions avec en moyenne 5000 nouvelles applications par mois. Environ 2,6 millions d'applications gratuites sont avec ou sans publicités, alors que 230.000 applications monétisent leurs contenus. Ce modèle économique a facilité la diffusion d'un grand nombre d'applications, permettant aussi d'amortir les frais de développement.

La plupart des pays peuvent accéder à Google Play avec un affichage dans leur propre devise ; lors d'un achat, la transaction est effectuée conformément au prix et à la devise par défaut du développeur. Google Play impose également des gammes de prix pour certains pays.

Lorsque vous proposez des applications et des produits intégrés à l'application sur Google Play, vous recevez 70 % du montant versé. Google prélève 30 % de vos gains pour les frais de gestion.

Règles de transition pour les applications gratuites et payantes :

- Une application gratuite peut devenir payante.
- Une application payante peut devenir gratuite, mais attention vous ne pourrez plus sélectionner l'option « Payante ». Si vous souhaitez facturer l'application, vous devez créer une autre application avec un nouveau nom de *package* et fixer un prix pour celle-ci.

La diffusion sur Google Play n'est pas exclusive, vous pouvez publier votre application sur d'autres stores.

Google Play Console est le site de référence pour la diffusion de vos applications. Pour y accéder, vous devez au préalable créer un compte Google à cette adresse **https://accounts.google.com/SignUp**.

Avec votre compte Google, créez ensuite un compte développeur à cette adresse **https://play.google.com/apps/publish**, c'est à partir de ce compte que vous pourrez démarrer le processus de diffusion. La création du compte développeur se déroule en quatre étapes et dure environ une dizaine de minutes :

1. Connexion avec votre compte Google,
2. Acceptez le contrat proposé par Google,
3. Payez les frais d'inscription de 25$,
4. Renseignez votre compte.

IMPORTANT !

Vous ne payerez qu'une seule fois les frais d'inscription de 25$. Vous pouvez ensuite publier autant d'applications que vous le souhaitez !

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Faisons maintenant connaissance avec Google Play Console. La barre de gauche vous propose le menu ***Toutes les applications***, ce menu vous donne accès à l'ensemble de vos applications déjà publiées et permet aussi de créer des modèles de tarification que nous utiliserons.

4.3 Publiez votre première application en quelques clics !

Pour chaque application que vous désirez diffuser, vous devez créer une nouvelle application dans Google Play Console. Chaque application est diffusée indépendamment des autres, car les modalités de diffusion peuvent justement être très différentes d'une application à une autre.

La publication suit les étapes suivantes : Version de l'application, Fiche Play Store, Classification du contenu et enfin Tarifs et disponibilité. Cliquez tout d'abord sur ***Publier une application Android sur Google Play***. Précisez la langue par défaut puis, saisissez le nom de l'application (au maximum de 30 caractères), ensuite cliquez sur ***Créer***. Ces paramètres seront modifiables par la suite. Chaque étape est ensuite validée par un « voyant vert » dans le menu de gauche.

Chaque application possède une « Fiche Play Store » obligatoire qui permet de caractériser précisément votre application. Dans un premier temps, la fiche est enregistrée dans une version « brouillon » puis dans une version « publiée » lorsque tout le processus de publication est achevé.

La fiche de l'application peut être créée dans différentes langues, c'est d'ailleurs quasiment une obligation d'avoir une version en anglais vu que le produit est diffusé à l'international. Sur le menu ***Gérer les traductions***, cliquez sur ***Ajouter vos propres traductions*** : votre fiche est maintenant présentée avec une langue par défaut et en anglais. Saisir ensuite la fiche en français et en anglais ; tous les champs marqués d'un astérisque sont obligatoires.

Dans la rubrique « Classification » de la fiche, un questionnaire est proposé afin de définir précisément les contenus. Avant de remplir ce formulaire, il faut envoyer l'archive APK via le menu ***Version de l'application***. Vous pouvez décider si votre première publication est une version de production, bêta, ou alpha :

- La version de production représente la version stable de votre application.
- Les versions alpha et bêta vous permettent de réaliser des tests en donnant accès à l'archive APK via le store, dans notre cas : **https://play.google.com/apps/testing/linuxmag.app.hs.store.release**. Les tests sur votre application peuvent être fermés, et ouverts à tout moment. Vous pouvez déposer une version de votre APK pour le déploiement production, bêta ou encore alpha.

La fiche est maintenant validée. Allons maintenant à la rubrique « Classification » via le menu de gauche ***Classification du contenu***. Le formulaire de classification est conséquent, tous les items de saisie sont obligatoires.

4.4 Modèles de tarification

Vient le moment du choix du type de diffusion gratuit/payant... mais avant nous allons utiliser le principe du modèle de tarification via le menu ***Paramètres > Modèles de Tarification***. Le modèle de tarification permet de définir le prix de votre application et des taxes associées. Vous pouvez créer plusieurs modèles de tarification pour ensuite les appliquer à vos applications au moment du déploiement.

Revenons dans le menu ***Tarifs et disponibilité*** de l'affiche de l'application. Nous pouvons appliquer maintenant un modèle de tarif et choisir la liste des pays pour la diffusion. Par défaut, l'application sera diffusée dans 136 pays.

Le compte marchand est nécessaire dès lors que vos applications sont payantes, Google utilisera les informations de ce compte pour vous payer les applications téléchargées à partir de Google Play. Pour confi-

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

gurer ce compte : allez dans le menu ***Paramètres > Détails du compte***. Dans ***Compte marchand***, cliquez sur ***Configurer votre compte marchand***.

RAPPEL

Si vous choisissez une diffusion « payante » vous pouvez par la suite la rendre gratuite, mais si vous voulez plus tard passer d'une diffusion gratuite à payante vous devez créer une autre application avec un nouveau nom de package et fixer un prix pour celle-ci.

Lorsque les étapes sont validées, vous pouvez demander la publication de votre application, car jusqu'à présent le contenu a été sauvegardé en mode « brouillon ». À partir de la page de garde, sélectionnez votre application puis ***Gestion de la publication > Version de l'application***, cliquez sur ***Lancer le déploiement*** ; le déploiement correspond au type de version : production, bêta, alpha. Le déploiement effectif sur Google Play peut prendre plusieurs heures, en cas de problème vous serez notifié. À tout moment, Google Play Console permet d'annuler le déploiement et supprimer une application. La figure 1 montre notre application publiée.

4.5 Autres Stores

Le store Google Play est prédominant et spécifiquement dédié au système Android. Vous pouvez élargir la diffusion de votre application sur des stores moins spécialisés comme **Amazon AppStore** qui possède une clientèle plus diverse avec des comptes déjà configurés avec un process d'achat très simplifié. D'autres stores, comme **mobogenie.com** sont encore plus ouverts avec une offre pour les logiciels PC et Android. Le choix final se fera en fonction de nombreux critères :

- store spécialisé ou généraliste,
- compte gratuit ou payant,
- somme reversée au développeur (en principe 70% de la transaction),
- possibilité d'insérer des publicités de monétisation.

RAPPEL

Pour les applications provenant d'un store « non officiel », n'oubliez pas d'autoriser les applications de sources inconnues : menu Paramètres > Sécurité puis cochez Sources inconnues.

4.6 Art Generator

La plupart des stores vous demanderont des images de votre application. Google propose un générateur d'images **[1]** qui permet de fondre une capture d'écran dans une image d'un équipement « réel ».

Synthèse

Dans cet article, nous avons détaillé toutes les étapes permettant de publier une application propre, de sélectionner son type de diffusion (gratuit ou payant) jusqu'à aboutir à la phase ultime : une application disponible et exploitable par d'autres utilisateurs.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

TARIFS ÉLÉMENTS ASSOCIÉS

Nom EUR 1 - Logiciel Geocaching 19/30

Prix

Votre prix est utilisé pour générer des prix locaux dans d'autres pays. Ce tarif sera également utilisé dans les pays où les paiements en devise locale ne sont pas acceptés.

Prix * EUR 1

Taxes

La taxation relative à chaque pays est disponible dans le compte Merchant Center du propriétaire de votre compte.

- Le prix comprend les taxes applicables.
- Ajouter les taxes applicables au prix

Prix dans la devise locale

Les prix locaux utilisent des schémas de tarification pertinents au niveau local et le taux de change en vigueur pour la date à laquelle vous définissez le prix. Vous pouvez actualiser le prix manuellement afin de vous assurer que le prix local reflète le dernier taux de change.

La liste comprend les pays où les utilisateurs effectuent des paiements dans la devise locale. Dans les 67 autres pays où vous distribuez votre application, le prix en EUR sera utilisé.

Pays	Prix		Taxes
Afrique du Sud	ZAR	13,99	
Allemagne	EUR	1,19	19 % (EUR 0,19)
Arabie saoudite	SAR	4,09	
Australie	AUD	1,49	

CRÉER UN MODÈLE ANNULER

Figure 1

Fiche de l'application Geocaching sur Google Play via un smartphone.

RÉFÉRENCES

[1] **https://developer.android.com/guide/practices/ui_guidelines/icon_design.html** et **https://developer.android.com/distribute/marketing-tools/device-art-generator.html**

[2] **https://developer.android.com/guide/topics/permissions/index.html**

[3] **https://developer.android.com/guide/topics/permissions/requesting.html**

[4] **https://developer.android.com/training/permissions/requesting.html**

[5] **https://developer.android.com/training/permissions/usage-notes.html**

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

GAGNEZ DE L'ARGENT... BEAUCOUP D'ARGENT ?

Maintenant que votre application est finie, vous pouvez songer éventuellement à essayer d'obtenir quelques retombées financières venant saluer votre travail.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Le smartphone est peut-être l'objet personnel par excellence, il vous accompagne partout ou presque, dans vos activités privées et personnelles. Au final nous consultons son écran plus qu'une télévision. Les applications mobiles n'ont jamais été d'une aussi grande richesse : réseaux sociaux, achats de toutes sortes, actualité, etc. Vu ce contexte, les possibilités de monétiser une application n'ont jamais été aussi importantes. Nous avons vu précédemment une première solution de monétisation directe avec la diffusion d'une application payante. D'autres solutions sont également possibles, il est même question d'un choix stratégique si vous désirez optimiser vos revenus dans les différents stores.

1. STRATÉGIE DE MONÉTISATION

Tous les efforts de monétisation seront vains si vous ne positionnez pas votre application parmi les millions d'applications déjà existantes. Voici quelques pistes qui vous permettront d'établir votre stratégie :

- **Description de votre application** :

  Votre application sera recherchée par mot-clé et il est donc très important de bien rédiger la fiche de votre application. Il faut mettre en avant les principales caractéristiques, les nouveautés, etc.

- **Icône de l'application** :

  L'icône est en quelque sorte « la page de garde » de l'application, il faut attirer l'attention avec un design original reflétant le contenu de l'application. Les captures d'écran sont également importantes et montrent les fonctionnalités, en mode portrait et paysage.

- **Commentaires de l'application** :

  Des commentaires positifs peuvent vous donner un avantage, aussi vous pouvez demander à vos bêtas testeurs un avis sur vos applications.

- **Catégorie de l'application** :

  Il faut définir à quelle catégorie appartient votre application : jeux, information, etc. Votre application doit faire preuve d'originalité par rapport aux applications déjà existantes dans la même catégorie. Par exemple : proposer de nouvelles fonctionnalités, une ergonomie plus adaptée, des IHM attrayantes.

- **Observer les prix et le modèle de monétisation** :

  Le prix fait parfois la différence, alors une petite enquête de prix sur vos concurrents peut éventuellement créer une opportunité de positionnement. Le modèle de monétisation des applications concurrentes doit également faire l'objet d'une attention particulière, car le modèle adopté par vos concurrents doit vous donner des informations précieuses pour votre choix de monétisation : gratuit au téléchargement avec contenus payants, etc. Enfin, le prix doit être adapté en fonction des pays, si vous utilisez Google Play Console, vous pouvez par exemple utiliser un modèle de prix spécifique à certains pays.

- **Proposer des contenus** :

  Une application sans contenu peut être vouée à disparaître. Si votre application consomme des contenus, alors prévoyez une alimentation continue avec par exemple un système d'abonnement. Vous pouvez proposer, par exemple, une application gratuite au téléchargement avec une période d'essai. Les mobinautes seront sensibles aussi à une application qui évolue régulièrement avec de nouvelles versions, cela permet de démontrer une certaine pérennité et donc de déclencher un achat.

- **Promouvoir votre application** :

  Google Play Console permet de déclencher une promotion de votre application, ceci peut être aussi un bon support pour faire connaître votre application. Pour créer le buzz, vous pouvez également utiliser

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

votre réseau : réseau professionnel, base d'e-mails, etc. Une page web peut également avoir un impact important, vous pouvez y ajouter librement des images et des vidéos, et éventuellement un lien de téléchargement.

⇨ **Comportement de l'application** :

Pendant la phase de développement, les développeurs considèrent souvent que le système mobile est toujours connecté : Wifi, 4G, etc. Il faut cependant prendre en compte certains fonctionnements dégradés avec l'absence du réseau ou une qualité de service très faible.

2. OPTIMISER VOTRE MONÉTISATION

La monétisation d'une application peut être envisagée selon plusieurs stratégies plus ou moins complexes à mettre en œuvre pour les développeurs. Ici sont exposées les grandes tendances des modèles de monétisation, sachant qu'il est possible de les mixer pour obtenir une optimisation relative à votre application :

⇨ **Application payante** :

Le prix de l'application est fixé au moment du téléchargement. Les mises à jour peuvent être gratuites ou payantes.

⇨ **Achat de contenu** :

L'application est gratuite au téléchargement, elle propose des fonctionnalités et des contenus, éventuellement des bandeaux publicitaires sont insérés. Les achats sont motivés pour obtenir de nouvelles fonctionnalités, une version « pro », des contenus, la suppression de la publicité. Ici l'achat n'est pas récurrent. Cette stratégie est apparentée à une stratégie de type « Freemium » **[1]**, qui permet l'accès gratuit au logiciel, puis une version « Premium » payante, mais plus haut de gamme.

⇨ **Abonnement** :

L'application est gratuite au moment du téléchargement, elle propose un accès continu payant à des contenus, des services, des mises à jour. L'achat est récurrent.

⇨ **Publicité** :

L'application est gratuite au moment du téléchargement ainsi que pour les mises à jour. Les revenus ne sont basés sur aucun achat. Seule la publicité permet de monétiser l'application. L'utilisation intensive de la publicité peut devenir aussi un frein sachant que la taille moyenne des écrans utilisés est de 3,5 pouces. La majorité des utilisateurs pensent que la publicité sur les applications est particulièrement intrusive. Il est préférable de limiter les insertions de publicité de tous types et éventuellement se fixer un seuil à ne pas dépasser, par exemple utiliser une seule bannière de publicité en haut ou en bas du *layout*.

⇨ **Vente de produit** :

Ici la monétisation est basée sur les achats de produit et/ou services réalisés via une application mobile. C'est le principe retenu par les entreprises d'e-commerce.

3. SYNTHÈSE DES OFFRES DE MONÉTISATION GOOGLE

L'offre de monétisation est vaste, Google a mis œuvre un véritable écosystème pour répondre aux différentes stratégies. Dans chaque cas (hormis *Google Licensing*), nous retrouvons des services dans le « cloud » associés à des API à embarquer dans votre code. La mise en œuvre de la monétisation est vrai-

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecis

ment accessible et vous permet de monétiser rapidement voire très rapidement une application dans le cas de l'insertion d'une publicité. Les autres stores ne vous proposeront pas de système de monétisation aussi intégré, c'est vraiment la force de Google Play.

Voici une synthèse des offres de monétisation de Google :

Type de monétisation	Offre Google
Application payante	Avec **Google Play Console**, vous pouvez définir votre application comme payante lors du déploiement.
Achat de contenu	**Google Play Billing** propose avec la notion de produits intégrés à l'application : « produit géré ». Le contenu est uniquement numérique.
Abonnement	**Google Play Billing** propose également la possibilité d'abonnement avec encore la notion de produits intégrés à l'application. Le contenu est uniquement numérique.
Application sous licence	Avec **Google Play Licensing**, une application peut être utilisée pendant une période de temps puis demander un achat au-delà de cette période, ou encore restreindre des fonctionnalités. À noter : ce service Google est principalement destiné aux applications payantes qui souhaitent vérifier que l'utilisateur a effectivement payé pour l'application qu'il utilise.
Publicité	**AdMob** permet l'insertion de publicité sous différents formats.
Vente de produit	Pour le e-commerce non numérique Google propose l'environnement **Android Pay.**

4. INSÉRER DE LA PUBLICITÉ

L'utilisation de la publicité est un moyen de monétiser votre application. AdMob [2-6] est un écosystème qui vous permet d'insérer de la publicité dans vos applications, et de suivre l'évolution de vos gains via un véritable tableau de bord dédié. AdMob réalise le lien entre vos applications et les annonceurs.

Un système de filtre permet tout de même de limiter certains thèmes des publicités dans vos applications. Par défaut, aucun filtre n'est positionné. Pour accéder au filtrage publicitaire, cliquez sur votre insertion de publicité puis ***Autorisation/Blocage d'annonces***.

Les annonceurs rémunèrent Google AdMob à chaque clic sur une publicité, un clic sur une publicité est aussi appelé une « impression ». AdMob considère certaines activités comme anormales, car elles faussent le niveau de rémunération escompté par les annonceurs. Voici une liste non exhaustive d'activités anormales :

- clics générés par les éditeurs cliquant sur leurs propres annonces en ligne ;
- clics sur les annonces répété(e)s généré(e)s par un ou plusieurs utilisateurs ;
- éditeurs qui encouragent les clics sur leurs annonces (quelques exemples : toute formulation encourageant les utilisateurs à cliquer sur les annonces, mises en œuvre d'annonces susceptibles de générer un volume élevé de clics accidentels, etc.) ;
- outils de clics automatiques ou sources de trafic, robots, ou tout autre logiciel de détournement.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Avant de réaliser l'insertion d'une publicité dans votre application, vous devez créer un compte AdMob sur le site **https://apps.admob.com** ; ce compte peut être ouvert avec le compte Google que vous avez précédemment créé pour les étapes de déploiement.

AdMob permet de rechercher ensuite votre application sur Google Play et de l'associer au système AdMob. Un identifiant d'application est attribué à cet instant. À ce stade vous pouvez choisir le type de publicité :

- Bannière : une barre graphique doit être placée dans votre layout ;
- Interstitiel : une publicité qui recouvre tout l'écran ;
- Annonce vidéo avec récompense ;
- Natif : les annonces sont adaptées à l'application dans laquelle elles sont diffusées (choix de taille, couleur, etc.).

AdMob attribue un identifiant unique à chaque bloc d'annonce. Celui-ci permet de référencer les zones d'affichage publicitaire dans votre application. Une application peut intégrer plusieurs types d'annonces.

Créons par exemple une bannière pour obtenir un identifiant. Il existe différents formats de bannière et il faut en choisir un :

Taille (Largeur x Hauteur)	Description	Disponibilité	Paramètre AdMob
320x50	Bannière standard	Smartphone et tablette	BANNER
320x100	Bannière large	Smartphone et tablette	LARGE_BANNER
300x250	IAB Medium Rectangle	Smartphone et tablette	MEDIUM_RECTANGLE
468x60	IAB Full-Size Bannière	Tablette	FULL_BANNER
728x90	IAB Leaderboard	Tablette	LEADERBOARD
Largeur écran 32|50|90	Bannière intelligente	Smartphone et tablette	SMART_BANNER

AdMob utilise une bibliothèque spécifique, que nous devons ajouter au fichier `build.gradle` dans le bloc des dépendances. Il est également possible d'ajouter automatiquement cette bibliothèque avec le menu ***File > Project Structure > Ads*** puis cocher ***AdMob > Ok***. Les bibliothèques `ads` et `play-services` doivent être de la même version sous peine d'avoir des plantages de l'application pendant son exécution (Gradle affiche un message d'avertissement le cas échéant).

Fichier

```
...
dependencies {
...
compile 'com.google.android.gms:play-services:10.2.4'
compile 'com.google.android.gms:play-services-ads:10.2.1'
...
  }
...
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecis

L'application doit pouvoir contacter les serveurs AdMob, nous ajoutons les droits d'accès à Internet :

Fichier

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="mag.linux.android_sensors">
...
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_
STATE"/>
...
</manifest>
```

La bannière de publicité possède un ID que nous avons obtenu précédemment, il est utilisé par le serveur AdMob. L'ID est stocké dans le fichier `res/strings.xml`. Attention, dans notre projet, nous avons deux fichiers `strings.xml` : celui de debug et celui de release ; il convient donc de stocker l'ID dans ces deux fichiers ou encore de définir une nouvelle ressource.

Fichier

```
<resources>
...
<string name="banner_ad_unit_id">ca-app-pub-597785469991376 70/1731475544
</string>
...
</resources>
```

NOTE

AdMob propose un Id de test pour insérer une publicité sans la création d'un compte. Vous trouverez cet Id à cette adresse : https://firebase.google.com/docs/admob/android/quick-start.

Regardons maintenant en détail les modifications à réaliser dans le code.

NOTE

Pendant la phase de test, il est important de ne pas cliquer sur les publicités affichées, car cela entraîne des traitements associés. Votre compte AdMob peut être suspendu en cas d'abus.

La position de la publicité dans le *layout* relève du bon sens :

- placer la publicité dans le *layout* principal de l'application, éviter les boîtes de dialogue ;
- éviter de placer la publicité à côté d'une zone d'interaction avec l'utilisateur comme un bouton ;
- ne pas couvrir une zone d'affichage utile aux utilisateurs.

Le *layout* doit maintenant intégrer la bannière. Cette opération doit être réalisée avec soin, car une bannière de publicité positionnée au centre de l'application peut rendre l'utilisateur assez irritable. Le haut ou le bas de l'écran permet de limiter l'impact visuel.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Ici la bannière est placée sur le haut du *layout*. Vous pouvez par la suite ajuster la taille de la bannière en modifiant `ads:adSize` avec les valeurs présentées dans le tableau des formats de bannières. Voici la modification du *layout* `activity_main.xml` :

Fichier

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
  ...
   xmlns:ads="http://schemas.android.com/apk/res-auto"
   ...">
       <LinearLayout
           ...
           >
           <com.google.android.gms.ads.AdView
               android:id="@+id/adView"
               android:layout_width="wrap_content"
               android:layout_height="wrap_content"
               android:layout_centerHorizontal="true"
               android:layout_alignParentBottom="true"
               ads:adSize="BANNER"
               ads:adUnitId="@string/banner_ad_unit_id">
           </com.google.android.gms.ads.AdView>
...
```

Vous pouvez placer le code dans l'activité principale par exemple ou encore définir une méthode d'une classe qui gère les aspects publicités. La classe `AdView` suit le cycle de vie de l'activité via les méthodes `onResume()`, `onPause()`, `onDestroy()` :

Fichier

```
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
...

// AdMob
MobileAds.initialize(getApplicationContext(), "ca-app-
pub-5973967299137670/1731475544");
AdView mAdView = (AdView) findViewById(R.id.adView);
AdRequest adRequest = new AdRequest.Builder().build();
mAdView.loadAd(adRequest);
...
}

@Override
public void onResume() {
super.onResume();
// Resume the AdView.
...
mAdView.resume();
}

@Override
public void onPause() {
// Pause the AdView.
mAdView.pause();
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecis

Fichier

```
...
super.onPause();
}

@Override
public void onDestroy() {
// Destroy the AdView.
mAdView.destroy();
super.onDestroy();
}
...
```

Résultat : la bannière s'affiche en haut du *layout*, par défaut le taux de rafraîchissement de l'affichage est raisonnable (voir figure 1).

Android Studio permet également d'automatiser l'insertion de publicité avec la création d'une activité dédiée. Une activité peut intégrer une bannière ou un interstitiel : ***New > Google > Google AdMob Ads Activity***.

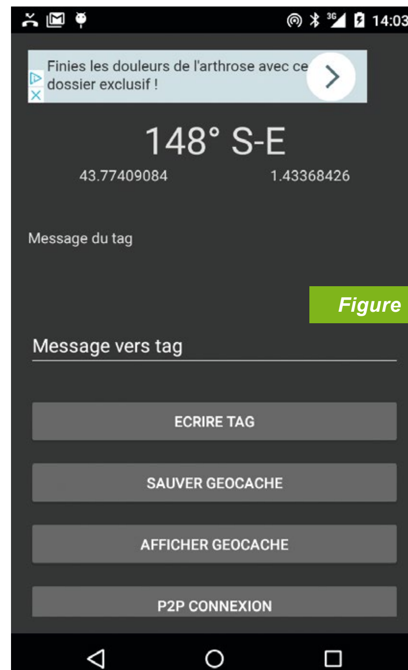


Figure 1

Affichage de la bannière de publicité avec la taille BANNER.

5. CIBLER VOS PUBLICITÉS

Il est possible de cibler les publicités en spécifiant plusieurs caractéristiques [5]: homme/femme, date d'anniversaire, destiné à la famille, ou enfant :

Fichier

```
mAdView = (AdView) findViewById(R.id.adView);
AdRequest adRequest = new AdRequest.Builder()
.setGender(AdRequest.GENDER_FEMALE)
       .setBirthday(new GregorianCalendar(1987, 5, 3).getTime())
       .tagForChildDirectedTreatment(true)
       .build();

mAdView.loadAd(adRequest);
```

Pour observer la bannière, il est possible d'associer des méthodes de *callback* :

- `void onAdLoaded()` : chargement de la publicité ;
- `void onAdFailedToLoad(int errorCode)` : retourne une erreur au chargement de la publicité ;
- `void OnAdOpened ()` : une annonce ouvre une superposition qui couvre l'écran ;

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

- `void onAdClosed()` : l'utilisateur revient vers l'application après avoir cliqué sur une publicité ;
- `void onAdLeftApplication()` : une publicité fait sortir de l'application (par exemple, pour lancer un navigateur).

Synthèse

Nous avons abordé ici la monétisation de l'application. Cette étape n'est pas toujours habituelle pour un développeur et demande la maîtrise de techniques particulières liées aux systèmes de monétisation. Nous avons décrit le fonctionnement du système de monétisation de Google avec Google Play Console qui présente un fort niveau d'intégration. Bien sûr d'autres possibilités peuvent s'offrir à vous sur d'autres stores plus indépendants. La monétisation de votre application peut être réalisée selon différentes stratégies : la publicité, les produits intégrés, la vente directe et aussi une astucieuse combinaison de ces derniers.

CONCLUSION

L'opportunité de monétiser des applications n'a jamais été aussi grande, car le smartphone est devenu en quelques années une fenêtre ouverte sur le monde du numérique, certains foyers qui n'ont même pas de PC, peuvent posséder jusqu'à plusieurs smartphones. Le smartphone a su trouver très rapidement sa place grâce à son offre de plusieurs millions de logiciels répondant à tous les secteurs d'activité. Les mobinautes possèdent un comportement quasi fusionnel avec leur smartphone, car il est devenu un outil indispensable pour l'accès à ce monde numérique sans limites. Cet engouement ouvre un champ de développement considérable.

Côté développeur, les moyens de diffusion et de monétisation sont exceptionnels, un développeur dans « son garage » peut, en quelques clics, proposer une application à des millions de mobinautes dans des centaines de pays. Le tout développé avec des outils open source, c'est vraiment génial !

Nous avons abordé dans ce numéro hors-série les points essentiels pour vous amener rapidement à la monétisation d'une application Android qui attend peut-être depuis longtemps dans votre dépôt git ou autre. Le choix de diffusion étant vraiment pléthorique, vos chances de monétisation en seront décuplées, la différence se fera par votre stratégie de positionnement, l'originalité de l'application, votre prix de vente. N'hésitez pas à utiliser toute la puissance de la plateforme Android : capteurs, systèmes de communication, qui rendent vos applications plus vivantes, plus intuitives, plus monétisables... ■

RÉFÉRENCES

[1] https://fr.wikipedia.org/wiki/Freemium

[2] https://apps.admob.com et https://support.google.com/admob/answer/3342054?hl=fr

[3] https://firebase.google.com/docs/admob/android/quick-start?hl=fr

[4] https://firebase.google.com/docs/admob/android/banner

[5] https://firebase.google.com/docs/admob/android/targeting

[6] https://support.google.com/admob/answer/3342054

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecis

VISITEZ NOTRE NOUVELLE BOUTIQUE ET DÉCOUVREZ NOS GUIDES !

ET VOUS ?
COMMENT LISEZ-VOUS VOS MAGAZINES PRÉFÉRÉS ?

« Moi, je les lis en version PAPIER ! »

« Moi, je les lis en version PDF ! »

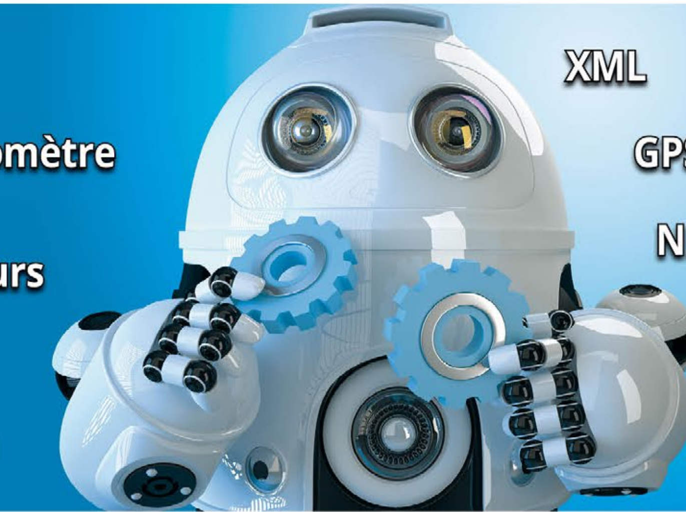
« Moi, je consulte la BASE DOCUMENTAIRE ! »

RENDEZ-VOUS SUR **www.ed-diamond.com**

POUR DÉCOUVRIR TOUTES LES MANIÈRES DE LIRE VOS MAGAZINES PRÉFÉRÉS !

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

DÉCOUVREZ les étapes à suivre pour créer un projet structuré



MANIPULEZ les capteurs/périphériques de votre smartphone : accéléromètre, magnétomètre, NFC, GPS, etc.

COMMUNIQUEZ en Bluetooth et par SMS pour partager des caches visibles sur des cartes Google Maps

PUBLIEZ & MONÉTISEZ votre application sur Google Play, rendez-la payante et insérez-y de la publicité

Retrouvez toutes nos publications
LES ÉDITIONS DIAMOND
sur www.ed-diamond.com

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)