Table des matières

Introduction

Nous vivons à une époque ou les appareils mobiles sont très répandus. Ceux-ci sont en effet devenus abordables et accessibles par tout le monde. Ce marché est passé d'un marché de niche à un marché florissant, à tel point que la plupart des grands constructeurs d'électronique possèdent leur propre gamme d'appareils. Ces appareils sont passés de simples téléphones à des PDA qui n'ont rien à envier aux PCs d' il y a quelques années seulement: il n'y a pas si longtemps encore il fallait se tourner vers le marché professionnel pour trouver des PDA qui peinaient à effectuer leur tâches. De plus, la technologie ayant beaucoup évolué, ce ne sont même plus des PDA mais des appareils de haute technologie qui regroupent énormément de fonctionnalités, la plupart des nouveaux téléphones peuvent servir de radio, de GPS, lecteur de musique/vidéo et permettent même de surfer sur le web ...

Les plateformes mobiles les plus répandues à ce jour sont Google et son récent système Android (qui est presque entièrement Open-source à l'exception de certaines librairies et est gratuit), Apple et son iPhone (qui a largement contribué à amener la nouvelle génération de téléphones au grand public), Nokia avec sa plateforme Symbian, sans oublier Black-Berry qui reste très prisé par les professionnels.

Tous les systèmes d'exploitations pour mobiles sont plus ou moins différents les uns des autres, leurs APIs le sont également et sont rarement disponibles pour les autres systèmes: il n'est donc pas aisé de créer une application portable entre ces systèmes. Ce point n'arrange forcément pas les entreprises qui souhaitent se lancer dans le développement mobile.

Il n'est évidemment pas possible d'étudier toutes ces plateformes. Dans ce document, c'est sur Android, le système de Google, que je vais me pencher. C'est un système qui provoque un rare engouement chez les développeurs du au fait que les applications sont écrites en Java ce qui facilite le développement. Il intéresse également la communauté open-source car le code-source d'Android est publiquement disponible et son noyau est un kernel Linux ce qui provoque un sentiment de sécurité.

J'espère que en lisant ce document vous aurez ensuite une meilleure connaissance du système android et/ou des méthodes de développement sur cette plateforme et peut-être même dans le meilleur des cas vous avoir donné envie de commencer à développer des applications tout simplement.

Bonne lecture.

Partie 1

Environnement de développement

1. Environnement de développement

1.1. Choix d'un environnement et justification

La plateforme de développement utilisée dans ces exemples sera Eclipse, la raison étant que le support officiel de Google pour le développement sur Android est cet environnement, il est donc recommande pour une plus grande facilité et également sa stabilité.

Il existe aussi la possibilité d'utiliser Netbeans, qui est un autre environnement, grâce a un plugin mais il n'existe pas de support officiel dans ce cas.

1.2. Qu'est-ce que Eclipse ?

Eclipse est une communauté Open Source, dont les projets ciblent la création d'une plateforme de développement comprenant des frameworks extensibles, outils et environnements d'exécution pour construire, déployer et le management des applications durant toute leur durée de vie.

La Fondation Eclipse est une corporation à but non lucratif, supportée par ses membres, qui regroupe les projets autour d'Eclipse et aide à développer la communauté Open Source ainsi qu'un certain nombre de produits et services complémentaires.

Le Projet Eclipse a originellement été créé par IBM en novembre 2001 et supporté par un consortium de vendeurs de logiciels. La Fondation Eclipse à été créée en janvier 2004 en tant que corporation indépendante à but non lucratif pour agir en tant que administrateur de la Communauté Eclipse. Cette Corporation a été créée pour permettre de fournir une communauté neutre, ouverte et transparente de s'établir autour d'Eclipse. Aujourd'hui la communauté d'Eclipse est composée d'organisations de l'industrie du logiciel et de particuliers.

La Fondation Eclipse est financée par les droits annuels des membres et régie par un conseil d'administration. Les développeurs importants et les consommateurs importants possèdent des sièges sur ce conseil, de même que les représentants élus par les fournisseurs de la communauté Open Source et de add-on (élément qui si il est installé, ajoute ou améliore les fonctionnalités de l'application de base pour laquelle il est destiné).

La Fondation engage du personnel professionnel pour fournir des services à la communauté mais n'emploie pas de développeurs Open Source, appelés les fournisseurs, qui travaillent réellement sur les projets Eclipse. Ces fournisseurs sont typiquement employés par des organisations ou sont des développeurs indépendants qui donnent volontairement de leur temps pour travailler sur un projet Open Source.

La Fondation Eclipse fournit quatre services à la communauté Eclipse

1. des infrastructures IT :

   La fondation Eclipse gère les infrastructures IT de la communauté Open Source, ce qui inclut les dépôts de code CVS/SVN, bases de données Bugzilla (désigne des outils de test et de recherche de bugs), les listes de mail orientées développement, sites de téléchargement et sites web. Cette infrastructure est vouée a fournir un service fiable à la demande pour les fournisseurs qui développent la technologie d'Eclipse ainsi que aux consommateurs qui utilisent cette technologie.

2. la gestion de la propriété intellectuelle :

   Un aspect important d'Eclipse est qu'il se concentre sur le fait d'apporter les technologies Open Source dans les services et les logiciels commerciaux. Ceci est rendu possible par le fait que tout les projets d'Eclipse sont sous la licence EPL (Eclipse Public Licence / voir p.14), une licence pro commerciale approuvée OSI (est une organisation dédiée à la promotion des logiciels open-source).

   La Fondation Eclipse entreprends également un certain nombre d'étapes pour essayer d'assurer l'appartenance de la propriété intellectuelle contenue dans les projets Eclipse. La première étape de ce processus est de s'assurer que toutes les contributions sont faites par le détenteur du copyright et sous la licence EPL (Eclipse Public Licence). Il est demandé à tous les fournisseurs de signer une charte d'agrément de fournisseur qui stipule que toutes leurs contributions sont leur propre travail et est sous licence EPL. Si un fournisseur est sponsorisé par une organisation membre pour travailler sur un projet Eclipse, il est alors demandé à cette organisation de signer une charte de membre fournisseur pour s'assurer que les droits de propriété intellectuelle sont fournis sous licence EPL.

   La seconde étape est que le code source relatif à chaque contribution qui est développée hors du processus de développement d'Eclipse passent par le processus d'approbation de la propriété intellectuelle. Ce procédé inclus l'analyse des contributions de code sélectionnés pour essayer de s'assurer de la provenance du code, et la compatibilité de licence avec EPL. Les contributions qui contiennent du code qui contiennent du code sous des licences non compatibles avec EPL sont sorties du processus d'approbation et donc non ajoutées à Eclipse. Le résultat final est un haut niveau de confiance en ce que les projets Open Source Eclipse utilisent une technologie qui peut être distribuée de manière sûre dans des produits commerciaux.

3. le processus de développement :

   La communauté Eclipse a gagné sa réputation en fournissant des logiciels fiables et de qualité. Ceci grâce à la participation des fournisseurs et des organisations qui contribuent aux projets Open Source. La Fondation Eclipse fournit également des services et du support à ces projets dans le but de leur permettre d'atteindre leur objectif. Les équipes de la fondation aident à implémenter le EDP (Eclipse Development Process).

   C'est un processus qui fournit une assistance pour le démarrage d'un projet et qui s'assure que tout les projets Eclipse se déroulent de manière ouverte et transparente.

La communauté Eclipse planifie des trains de sorties annuelles qui fournissent une date de sortie coordonnée pour tous les projets d'Eclipse qui souhaitent participer à des mises à jour coordonnées. Ces train de sorties facilite l'adoption des nouvelles versions pour les consommateurs finaux parce que tous les projets sont disponibles selon un même calendrier, un haut niveau de tests d'intégration a lieu avant la sortie finale pour aider à identifier les problèmes inter-projets.

4. un écosystème de développement :

   Un aspect unique de la communauté Eclipse et le rôle de la fondation Eclipse est la promotion active des projets Eclipse et du plus large écosystème Eclipse qui regroupe Eclipse même mais également tout les add-on créés par la communauté et les projets alternatifs. C'est un écosystème en plaine santé qui s'étend au delà de la communauté Open Source d'Eclipse, pour inclure des choses comme des produits commerciaux basés sur Eclipse, d'autres projets Open Source basés sur Eclipse ou qui l'utilisent (tel Carbide qui est utilisé par Nokia pour le développement des applications Qt en C++) , des fournisseurs de formations, des magazines, des portails en ligne, des livres, etc...

   Pour assister au développement de l'écosystème Eclipse, la Fondation Eclipse organise un certain nombre d'activités, ce qui inclus de événements de marketing avec les compagnies membres, des conférences communautaires (EclipseCon et Eclipse Summit Europe), des catalogues de ressources en ligne (EPIC et Eclipse Live), des meetings bisannuels entre membres et d'autres programmes pour promouvoir l'entière communauté Eclipse.

Des équipes à plein temps sont associées avec chacun de ces services et travaillent avec la communauté d'Eclipse pour subvenir aux besoins des dépositaires.

2. La licence EPL

La licence EPL est une licence open source utilisée par la fondation Eclipse pour ses applications. Elle remplace la licence CPL (Common Public Licence) et supprime certains termes relatifs aux litiges liés aux brevets. La licence EPL est conçue pour être une licence de logiciel libre favorable aux entreprises et comporte des privilèges plus faibles contre la copie que des licences contemporaines telles la GNU General Public Licence.

Le récipient d'une applications sous licence EPL peut utiliser, modifier, copier et distribuer le travail et des versions modifiées, dans certains cas avec l'obligation de fournir leurs propres changements.

La licence EPL est approuvée par la Open Source Initiative et listée en tant que "licence de logiciel libre" par la Free Software Foundation.

3. SDK (Software Development Kit)

Le SDK d'Android inclus un groupe d'outils de développement intuitifs. Ceux-ci incluent un débugger, des librairies, un émulateur de téléphone (basé sur QEMU), de la documentation, des exemples de code et des tutoriels. Les plateformes de développement sont les architectures x86/64 qui utilisent Linux (distribution moderne), Mac OS X 10.4.8 ou plus, Windows XP ou Vista.

Sont aussi requis le Java Development Kit, Apache Ant et Python 2.2 ou plus. L'environnement de développement officiellement supporté est Eclipse (3.2 ou plus) en utilisant le plugin Android Development Tool, on peut également faire tout à la main en utilisant les CLI (Command Line Interface) des outils du SDK mais c'est tout de suite plus difficile.

3.3.1. Historique des versions du SDK

Une version de test du SDK d'Android à été mise à disposition du public le 12 Novembre 2007.

Le 18 aout 2008 la version 0.9 Beta du SDK sortait, cette version propose une version mise à jour et étendue de l'API, des outils de développement améliorés et une refonte du design de l'écran d'accueil.

Le 23 septembre 2008 la version 1.0 du SDK fait son apparition. Selon les notes de distribution, elle contiens principalement des correctifs, néanmoins quelques additions furent présentes. C'est à partir de cette API que vont disparaitre les fameuses classes DrivingDirections (voir partie problèmes et solutions) qui servent a récupérer une route d'un point à l'autre et de dessiner le chemin lui même.

Le 9 mars 2009, Google a distribué la version 1.1 pour le téléphone de développement Android. Alors qu'elle contiens quelques mises à jour esthétiques, quelques ajouts importants sont effectués tels la recherche vocale, applications payantes, corrections dans l'horloge de l'alarme, correction d'un crash à l'envoi de mails via Gmail, correctif dans la notification des mails et dans le temps de rafraichissement ainsi que l'ajout des commentaires sur les commerces. Une autre mise a jour importante est que les téléphones de développement peuvent maintenant avoir accès aux applications payantes de l'Android Market.

À la mi mai 2009, Google a distribué la version 1.5 (Cupcake) du système Android et du SDK. Cette mise à jour inclus de nombreuses nouvelles options telles l'enregistrement vidéo, support pour le Bluetooth stéréo, un clavier virtuel éditable et la reconnaissance vocale. Cette version mets aussi à la disposition des développeurs le AppWidget Framework qui leur permets de créer leur propres applications pour l'écran d'accueil.

En septembre 2009, la version 1.6 (Donut) à été distribuée, elle contiens des améliorions au niveaux de la recherche, de l'utilisation de la batterie, applet de contrôle des VPN, de nouvelles technologies dont un moteur Text to Speech (synthèse vocale) gestionnaire de mouvements.

4. Eclipse

4.1. Présentation de la plateforme

Eclipse est un environnement de développement logiciel multi langages. L'interface d'Eclipse est fort similaire à tout les environnements de développement actuels (VisualStudio, Netbeans, ...), mais elle a la particularité, de par la partie Open Source du projet, d'être extrêmement modulaire, et donc de pouvoir directement inclure des plugins quelconques.

Il est amusant de constater que le nom utilisé : Eclipse est en totale opposition avec le nom de la société fondatrice Java c'est a dire Sun.

Eclipse est, à la base, écrit en Java et peut être utilisé pour développer des applications en Java et, grâce à de nombreux différents plugins, dans d'autres langages également, tels que C, C++, COBOL, Python, Perl, PHP, ainsi que beaucoup d'autres. L'IDE est souvent appelé Eclipse ADT pour Ada, Eclipse CDT pour le C, Éclipse JDT pour le Java et Eclipse PDT pour le PHP.

Android offre un plugin spécifique pour Eclipse, appelé ADT (Android Développement Tools),

qui est destiné à offrir un puissant environnement de développement intégré dans lequel créer des applications pour Android.

ADT étends les capacités d'Eclipse pour créer rapidement un nouveau projet pour Android, une interface utilisateur pour les applications, ajouter des composants basés sur le Framework Android, débugger les applications grâce au SDK ainsi que d'exporter les APK (voir 3.6) signés ou non dans le but de distribuer les applications.

En général, utiliser Eclipse avec ADT est une approche fortement recommandée pour le développement sur la plateforme Android ainsi que la manière la plus rapide dans son apprentissage.

4.2. Architecture

Eclipse utilise des plugins afin de fournir toutes ses fonctionnalités de même que l'environnement d'exécution, en contraste avec d'autres applications où les fonctionnalités sont typiquement directement codées a l'intérieur de celle-ci.
L'environnement d'exécution d'Eclipse est basé sur Equinox (système de plugins désigné pour Eclipse qui permets aux développeurs d'implémenter une application en tant que paquets), une implémentation standard de gestion de plugins compatible OSGI (plateforme de services et de modules systèmes pour le langage JAVA permettant de les installer, désinstaller, démarrer, arrêter, ... sans devoir requérir a un redémarrage des applications).

Ce mécanisme de plugins est un framework léger de composantes logiciels. En plus d'offrir la possibilité à Eclipse de recevoir des extensions en utilisant d'autres langages de programmation tels que du C ou du Python, cette gestion des plugins permets à Eclipse de travailler avec des langages typés comme LaTeX, des applications réseaux telles que telnet, et des systèmes de gestions de base de données. Cette architecture supporte la création de n'importe quelle extension à l'environnement, tels des gestionnaires de configuration. À l'exception d'un léger kernel d'exécution, tout dans Eclipse est donc plugins.

4.3. Versions

Voici les différentes versions d'Eclipse à ce jour :

Nom	Date	Version
Eclipse 3.0	*28 Juin 2004*	*v3.0*
Eclipse 3.1	*28 Juin 2005*	*v3.1*
Callisto	*30 Juin 2006*	*v3.2*
Europa	*29 Juin 2007*	*v3.3*
Ganymede	*25 Juin 2008*	*v3.4*
Galileo	*24 Juin 2009*	*v3.5*
Helios	*23 Juin 2010*	*v3.6*

4.4. Installation Eclipse

4.4.1. Quel package choisir ?

Comme expliqué plus haut, Eclipse change littéralement de fonctionnalités en rapport avec le module de développement qu'il utilise. La plupart des modules complémentaires sont disponibles au téléchargement directement en bundle avec Eclipse. Les versions disponibles de cette manière sont :

- Eclipse IDE for Java EE Developers : Outils pour les développeurs Java EE et applications Web, inclus les outils pour : Java EE, JPA, JSF, Mylyn (un sous système d'Eclipse qui se charge de la gestion des tâches) et autres.
- Eclipse IDE for Java Developers : Les outils essentiels pour tout développeur Java, contiens en plus de l'IDE un client CVS, un éditeur XML et Mylyn. (C'est cette version qui est conseillée et que nous allons utiliser pour nos développements sur la plateforme Android).
- Eclipse for PHP Developers : Outils Permettant aux développeurs PHP de créer des applications Web, inclus PDT (PHP Development Tools), Web Tools Platform, Mylyn et autres.
- Eclipse IDE for C/C++ Developers : Un IDE pour les développeurs C/C++ avec l'intégration de Mylyn.
- Eclipse for RCP/Plug-in Developers : un set d'outils complet pour les développeurs souhaitant créer des plugins pour Eclipse ou des RCA (Rich Client Applications).
  Il inclus un SDK complet en plus de Mylyn, un éditeur XML et le ECF (Eclipse Communication Framework).
- Eclipse Modeling Tools : se package de modélisation contiens des composants tels que EMF, GMF, MDT XSD/OCL/UML2, M2M, M2T, et EMFT. Il contiens également un SDK complet.
- Eclipse IDE for Java and Report Developers : contiens JEE (Java Enterprise Edition) et l'outil de rapport BIRT pour les développeurs Java afin de créer des applications Web et JEE qui ont également le besoin de faire des rapports.
- Pulsar for Mobile Java Developers : Pulsar est une plateforme d'outils pour les développeurs mobiles Java. Il comprends Java Development Tools (JDT), Mobile Tools for Java (MTJ), Mylyn et Plugin Development Environment (PDE). Pulsar facilite le téléchargement des SDK des différents constructeurs d'appareils mobiles.
- Eclipse SOA Platform for Java and SOA Developers : Plateforme d'environnements et d'outils d'intégration pour les développeurs SOA. Inclus un IDE Java, un PDE (Plugin Development Environment), un éditeur XML et WSDL.

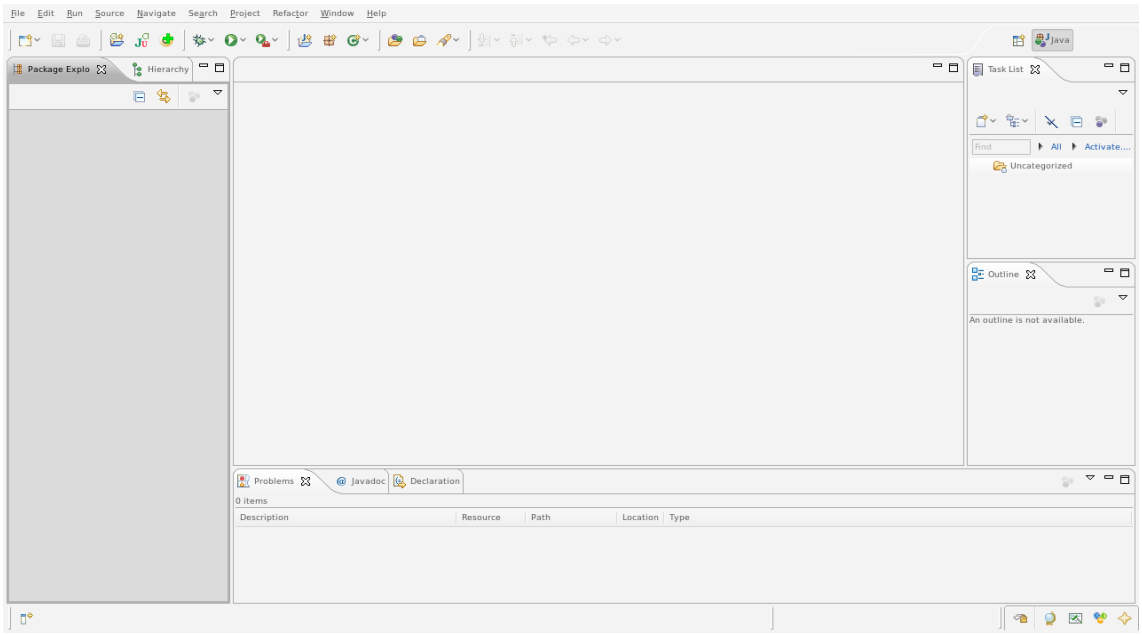
4.4.2. Installation sous Linux

Eclipse ne possède pas de programme d'installation, il est fournit tel quel. Néanmoins malgré le fait que Eclipse soit écrit en Java, il existe des versions séparées pour les différents systèmes d'exploitation, ceci est du au fait que son noyau est compilé et optimisé pour chacun de ceux-ci. Je vais donc expliquer la marche a suivre pour le système d'exploitation que j'utilise personnellement (Linux en anglais), mais la marche a suivre est semblable sous Windows.

- pour télécharger Eclipse nous irons donc sur le site officiel : http://www.eclipse.org/downloads/ il est fortement conseillé de choisir Eclipse IDE for Java Developers.
- il suffit ensuite d'extraire le dossier de l'archive avec : "tar -xvf eclipse-Java-galileo-SR1-linux-gtk-x86_64.tar.gz"
- à partir de maintenant nous allons pouvoir déjà exécuter Eclipse a partir du répertoire eclipse récupéré. Pour faire cela proprement, nous allons néanmoins le mettre dans un autre répertoire : "mv eclipse /opt/" (nécessite l'utilisation d'un compte avec droits)
- ainsi que lui créer un lien symbolique dans le répertoire /usr/bin pour que tout le monde sache l'exécuter. "ln -s /usr/bin/eclipse /opt/eclipse/eclipse"

4.4.3. Interface

Voici à quoi ressemble l'interface utilisateur d'éclipse :



4.4.4. Débuter dans le développement Java sous Linux

Pour les débutants en Linux, en développement Java ou tout simplement ceux qui ont des problèmes pour installer Java sur leur machine de développement ces liens peuvent s'avérer utiles :

- https://help.ubuntu.com/community/Java
- https://help.ubuntu.com/community/JavaInstallation

Voici les étapes à effectuer pour installer Java avant d'installer le SDK d'Android et le plugin ADT (ici en utilisant une distribution basée sur Debian) :

- si une distribution 64 bits est utilisée sur la machine de développement, il faut alors installer les librairies 32bit:

  "aptitude install ia32-libs";

- il faut maintenant installer Java:

  "aptitude install sun-Java6-bin";

- pour ce qui est de l'installation d'Eclipse, les dépôts ne contiennent pas une version à jour, il est donc conseillé de se reporter a la marche à suivre expliquée précédemment. En se qui concerne le SDK la marche a suivre se trouve un peut plus loin.

4.4.5. Bogue de l'interface et solution

Dans sa version actuelle 3.5 Eclipse à un léger problème de boutons non cliquables sous Linux, la solution à se problème est de forcer éclipse à utiliser GDK (une librairie graphique des systèmes basés sur UNIX permettant la gestion de fenêtres) en placent cette ligne à la fin du fichier .bashrc dans le répertoire utilisateur : "export GDK_NATIVE_WINDOWS=true"

4.4.6. Problème de Heap et solution

Lorsque Eclipse contiens beaucoup de projets ouverts, il se peut qu'il soit lent, voire très lent et, dans le pire des cas, qu'il nous gratifie d'un message d'erreur du style *"memory stack overhead"*. Dans ce cas la, on a deux solutions: soit on subit, et plus les projets deviennent complexes plus on souffre, soit on augmente la taille qu'Eclipse peut utiliser au niveau de la mémoire centrale.

Pour ce faire, il suffit d'aller dans le fichier de configuration d'Eclipse: *"eclipse.ini"* qui se trouve à la racine du répertoire d'Eclipse: *"/opt/ecplipse/"* (si l'installation a été effectuée de la manière précédemment expliquée), et d'y modifier les valeurs associées aux champs suivants: -XX:MaxPermSize, –launcher.XXMaxPermSize et -Xmx.

Au final nous obtiendrons un fichier de configuration qui ressemble à ceci (il est également possible de personnaliser ses valeurs mais attention a ne pas mettre trop sinon Eclipse va vite saturer la mémoire centrale) :

```
-startup
plugins/org.eclipse.equinox.launcher_1.0.201.R35x_v20090715.jar
--launcher.library
plugins/org.eclipse.equinox.launcher.gtk.linux.x86_64_1.0.200.v20090519
-product
org.eclipse.epp.package.Java.product
-showsplash
org.eclipse.platform
--launcher.XXMaxPermSize512m
-vmargs
-Dosgi.requiredJavaVersion=1.5
-XX:MaxPermSize=512m
-Xms80m
-Xmx512m
```

4.5. Installation du SDK

- Pour télécharger le SDK nous irons sur le site officiel : http://developer.android.com/sdk/index.html
- il faut alors sélectionner la version adéquate pour le système d'exploitation utilisé;
- ensuite, le dossier sera extrait de l'archive avec : "tar -xvf android-sdk_r04-linux_86.tgz"
- et de le placer dans un endroit approprié : "mv android-sdk-linux_86 /opt/".

4.6. Installation modules ADT (Android Developer Tools)

- Tout d'abord: démarrer Eclipse, sélectionner ensuite **Help** > **Install New Software**.
- dans la boite de dialogue **Available Software**, cliquer sur **Add...**.
- dans la boite de dialogue **Add Site** apparue, il faut entrer un nom pour ce site distant (par exemple, "Android Plugin") dans le champs **Name**.
- dans le champs "Location", il faut entrez cette URL:

  https://dl-ssl.google.com/android/eclipse/

  Note: Si il y a un problème pour acquérir le plugin, il est possible d'essayer d'utiliser **http** dans l'URL, à la place de **https** (https est préféré pour des raisons de sécurité).

- Cliquez **OK**.

- de retour dans la boite de dialogue **Available Software**, **Developer Tools** devrait maintenant avoir été ajouté à la liste et est visible. Il faut cocher la case à coté de celui-ci ce qui va automatiquement sélectionner les outils DDMS (serveur permettant le debugging d'une application Android) et ADT (Android Developer Tools), faites **Next**;
- dans la boite de dialogue **Install Details**, les différents éléments à être installés sont listés. Cliquer sur **Next** permets de lire la licence et ensuite d'installer les dépendances, il ne reste plus alors qu'a cliquer sur **Finish;**
- finalement il est recommandé de redémarrer Eclipse.

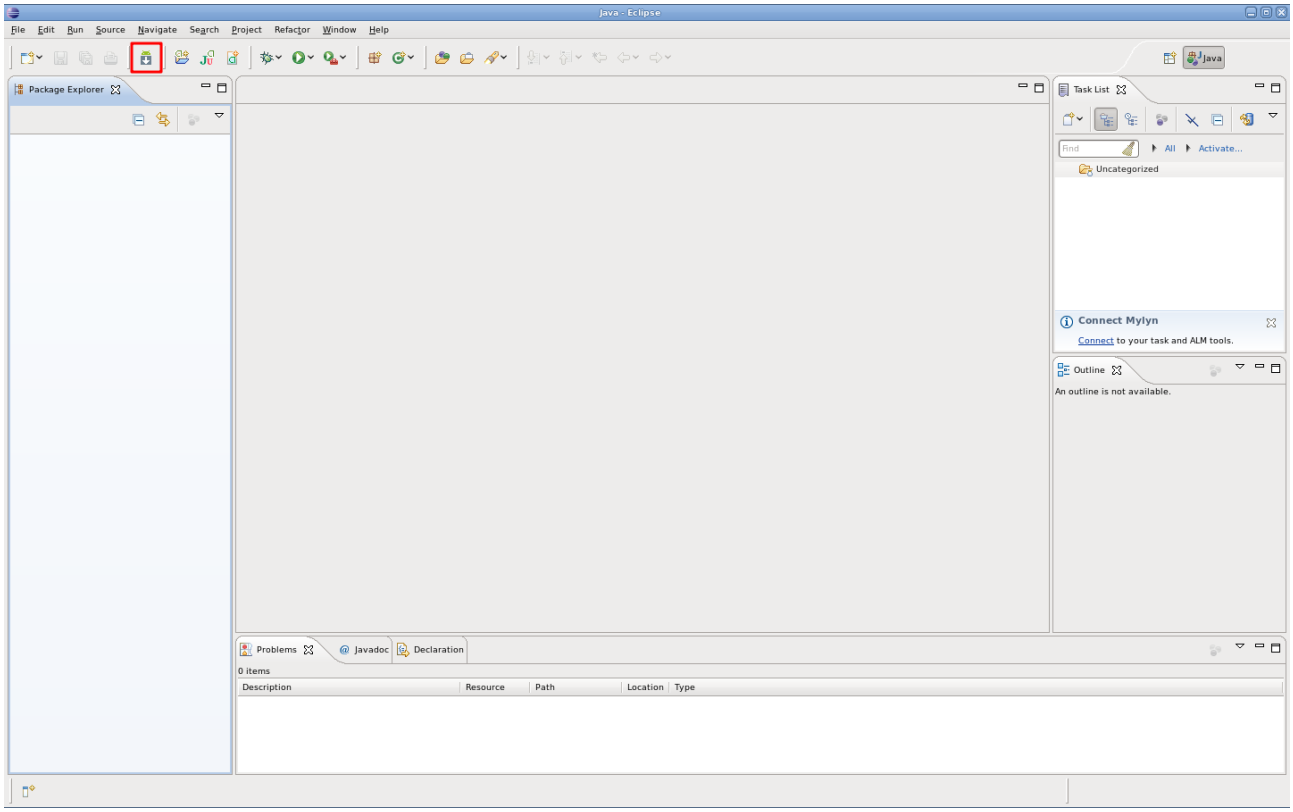
Maintenant il faut modifier les préférences dans Eclipse pour pointer le répertoire du SDK Android :

- il faut sélectionner **Window** > **Preferences...** pour ouvrir la fenêtre des préférences;
- ensuite sélectionner **Android** dans le panneau de gauche;
- pour choisir le **SDK,** dans le panneau principal il faut cliquer sur **Browse...** et sélectionner le répertoire ou le **SDK** a été téléchargé;
- finalement il faut cliquer sur **Apply**, ensuite sur **OK**.

4.7. Eclipse après installation de ADT

4.7.1. Interface principale

Voici à quoi va ressembler l'interface d'Eclipse après l'installation du SDK et des modules ADT :



Comme on peut le voir le changement n'est certes pas radical mais l'on peut observer l'apparition d'une icône supplémentaire (entourée en rouge) dans la barre d'outils en plus du menu supplémentaire dans les propriétés d'Eclipse dans lequel le chemin du SDK a été configuré.

Cette icône permets de lancer la boite de dialogue "Android SDK and AVD Manager" dans la quelle les différentes machines virtuelles pourront être configurées pour tester les applications ainsi que de mettre à jour le SDK. N.B. Cette fenêtre peut être lancée en lançant l'exécutable "android" qui se trouve dans le répertoire tools du SDK précédemment installé. Le chemin devrait être celui-ci :

- "/opt/android-sdk-linux_86/tools"

C'est dans se même répertoire que se trouvent tout les exécutables qui permettent de configurer l'environnement de développement associé à Android, de la configuration d'une machine virtuelle à son exécution en passant par le debugging.

4.7.2. Les différents outils fournis avec le SDK

Le SDK d'Android inclus une variété d'outils qui sont fait pour aider a développer des applications mobiles sur la plate-forme Android. Le plus important de ceux ci est l'émulateur Android et les plugins ADT pour Eclipse, mais le SDK comporte également une variété d'autres outils pour le debugging, l'empaquetage et l'installation d'applications sur l'emulateur.

En voici une liste non exhaustive :

- plugin ADT pour Eclipse :
  C'est lui qui permets d'inclure les outils du SDK dans Eclipse et de permettre un développement et un debugging des applications plus facile et plus rapide.
- émulateur Android :
  Émulateur de matériel basé sur QEMU utilisable pour designer, débugger, et tester vos applications dans un véritable environnement.
- AVD (Android Virtual Devices) :
  Ce sont des configurations d'appareils virtuels que l'on crée pour avoir un émulateur "sur mesure". Chaque AVD fonctionne comme un appareil indépendant avec son propre stockage pour données utilisateur, carte SD, ...
- Hierarchy Viewer :
  C'est un outil qui permets de débugger et d'optimiser vos interfaces utilisateur. Il fournit une représentation visuelle de la hiérarchie de vos différents écrans.
- layoutopt:
  Cet outil permets d'analyser rapidement les différents écrans pour en améliorer l'efficacité.
- Draw 9-patch
  Cet outil permets de créer des images "compatibles" avec la classe NinePatch, c'est à dire une image PNG dans laquelle on peut définir des sections qui seront "étirables" et que Android redimensionnera pour adapter l'objet au moment de l'exécution par rapport à son contenu, par exemple du texte.
- ddms (Dalvik Debug Monitor Service) :
  Intégré à Dalvik, la machine virtuelle spécifique à Android, cet outil permet de gérer les processus sur l'émulateur ou sur un appareil physique connecté, et facilite le debugging. Il permet de tuer un processus spécifique, générer une trace, voir la pile et les threads, prendre une capture d'écran, ...
- adb (Android Debug Bridge) :
  Cet outil permets d'installer les ".apk" de vos applications sur un émulateur ou sur un appareil physique connecté. Il permets également d'accéder a un débugger (permets de voir les différents prints).
- aapt (Android Asset Packaging Tool) :
  Cet outil permets de créer des fichiers ".apk" qui contiennent les ressources et fichiers binaires de vos applications.

- aid (Android Interface Description Language) :
  Permets de générer du code pour une interface inter processus, tel ce que un service pourrait utiliser.
- sqlite3 :
  Inclus pour plus de facilité, cet outil permets d'accéder aux fichiers de données créées par SQLite et utilisés dans les applications Android.
- Traceview :
  Cet outil produits des analyses graphiques des données de log qu'il est possible de générer a partir des applications.
- mksdcard :
  Aide a créer une image disque que l'on peut utiliser avec l'émulateur, pour simuler la présence d'une carte de stockage externe (tel une carte SD).
- dx :
  Cet outil réécris les fichiers de bytecode .class en fichiers de bytecode android qui sont des .dex.
- Monkey
  Monkey est un programme qui s'exécute sur un émulateur ou sur un appareil physique connecté et génère des flux d'actions utilisateur pseudo aléatoires tels des cliques, des touchers, ou des gestes, ainsi que un certain nombres d'événements systèmes. Monkey est utilisable pour effectuer des stress-test des applications en développement.
- android :
  C'est un script qui permets de gérer les AVD et de générer des fichiers de compilation Ant qui permettent de générer vos applications.
- zipalign :
  C'est un outil important d'optimisation des fichiers ".apk" . Cet outil s'assure que toutes les données non compressées démarrent avec un alignement particulier relatif au début du fichier. Il devrait tout le temps être utilisé pour aligner les fichiers ".apk" après qu'ils aient été signés.

4.7.3. Configuration d'une machine virtuelle

Les configurations de machines virtuelles (AVD) sont des options de configuration d'émulateur qui permettent de modeler un appareil pouvant exister.

Chaque AVD est constitué de :

- un profil matériel dans lequel il est possible de configurer les options pour définir les capacités de l'appareil virtuel. Par exemple: si l'appareil a un appareil photo, un clavier physique, ...
- un système de mapping d'images qui permets de choisir la version du système Android que l'on souhaite utiliser sur l'émulateur ;
- d'autres options telles l'apparence de l'émulateur, les dimensions de l'écran, etc.
  Il est aussi possible de définir une carte SD (carte mémoire) émulée;
- Un espace de stockage spécifique pour les données utilisateur dans lequel sont installées les applications utilisateur et les préférences.

Il est possible de créer autant d'AVD que l'on souhaite, basé sur le type d'appareil que l'on désire modeler. De se fait, lorsque l'on crée un AVD, on a le choix d'utiliser plusieurs options. La table ci dessous reprends les options disponibles de même que les valeurs par défaut ainsi que le nom de la propriété du fichier config.ini à laquelle l'option est rattachée :

Charactéristique	Description	Propriété
Device Ram Size	*La quantité de RAM physique sur l'appareil, en megabytes, la valeur par défaut est "96"*	*hw.ramSize*
Touch-screen Support	*Si il y a un écran tactile ou non sur l'appareil, la valeur par défaut est "oui"*	*hw.touchScreen*
Trackball Support	*Si il y a un trackball ou non sur l'appareil, la valeur par défaut est "oui"*	*hw.trackBall*
Keyboard support	*Si il y a un clavier QWERTY ou non sur l'appareil, la valeur par défaut est "oui"*	*hw.keyboard*
DPad support	*Si il y a un pavé dirrectionel ou non sur l'appareil, la valeur par défaut est "oui"*	*hw.dPad*
GSM modem support	*Si il y a un modem GSM ou non dans l'appareil, la valeur par défaut est "oui"*	*hw.gsmModem*
Camera support	*Si il y a un appareil photo ou non dans l'appareil, la valeur par défaut est "non"*	*hw.camera*
Maximum horizontal camera pixels	*Valeur par défaut 640*	*hw.camera.maxHorizontalPixels*
Maximum vertical camera pixels	*Valeur par défaut 480*	*hw.camera.maxVerticalPixels*
GPS support	*Si il y a un GPS ou non dans l'appareil, la valeur par défaut est "oui"*	*hw.gps*

Charactéristique	Description	Propriété
Battery support	*Si l'appareil peut utiliser une batterie, la valeur par défaut est "oui"*	*hw.battery*
Accelerometer	*Si il y a un accéléromètre ou non dans l'appareil, la valeur par défaut est "oui"*	*hw.accelerometer*
Audio recording support	*Si l'appareil peut enregistrer du son, la valeur par défaut est "oui"*	*hw.audioInput*
Audio playback support	*Si l'appareil peut jouer du son, la valeur par défaut est "oui"*	*hw.audioOutput*
SD Card support	*Si l'appareil supporte l'insertion ou le retirement de cartes SD, la valeur par défaut est "oui"*	*hw.sdCard*
Cache partition support	*Si il y a une partition /cache sur l'appareil, la valeur par défaut est "oui"*	*disk.cachePartition*
Cache partition size	*Valeur par défaut 66*	*disk.cachePartition.size*
Abstracted LCD density	*Configure les characteristiques de densité généralisée utilisée par un écran d'AVD, la valeur par défaut est 160.*	*hw.lcd.density*

4.7.3.1 Utilisation du GUI

Voici la fenêtre principale du manager :

Pour la faire apparaitre, il existe deux moyens :

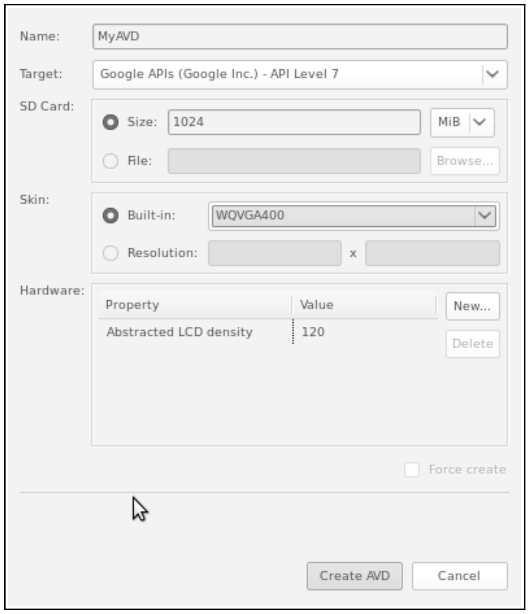
1. lancer l'exécutable android qui se trouve dans le répertoire tools du SDK à partir du terminal:

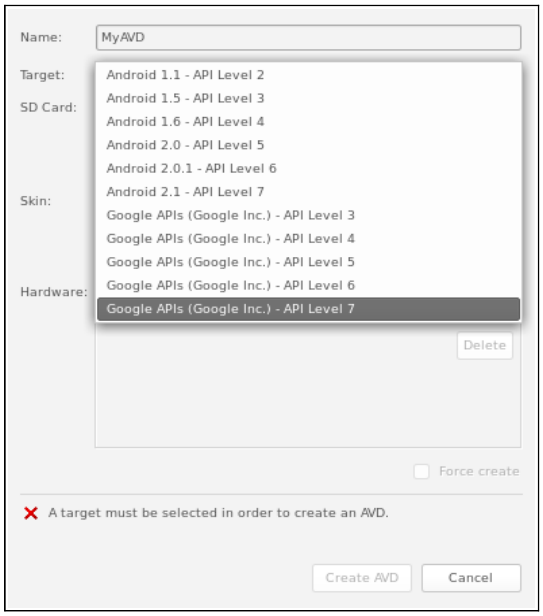
   "/opt/android-sdk-linux_86/tools/android"

2. ou utiliser Eclipse en cliquant simplement sur le nouveau bouton apparu dans la barre d'outils : 

On peut voir ici la liste des différentes machine virtuelles déjà créées, en créer de nouvelles et en supprimer. Pour en créer une nouvelle il suffit de cliquer sur **NEW**.

Apparait alors la fenêtre contenant les options de la machine virtuelle, on peut ici choisir ses caractéristiques.





4.7.3.2 Par ligne de commande

Pour créer un AVD, il faut utiliser l'outil android, un utilitaire par ligne de commande qui se trouve dans le répertoire "/opt/android-sdk-linux_86/tools/". La gestion des AVD est une des deux fonctions principales de cet outil (l'autre étant de créer et de mettre a jour des projets Android).
Il faut ouvrir un terminal et se déplacer dans son répertoire :

- "cd /opt/android-sdk-linux_86/tools/"

Pour créer un AVD, il faut effectuer la commande *"android create avd"*, avec les options pour spécifier un nom pour le nouvel AVD ainsi que l'image système que l'on désire utiliser sur l'émulateur lorsque l'AVD est invoqué.

Voici la syntaxe type de la ligne de commande pour créer un AVD :

- "android create avd -n <name> -t <targetID> [-<option> <value>]"

Pour générer une liste d'images systèmes utilisables il suffit d'utiliser cette commande :

- "android list targets"

L'outil va donc scanner les répertoires "/opt/android-sdk-linux_86/platforms" et "/opt/android-sdk-linux_86/add-ons/" à la recherche d'images système valides et ensuite générer une liste ce cibles.
Voici un exemple de résultat :

```
Available Android targets:
id:1
    Name: Android 1.1
    Type: platform
    API level: 2
    Skins: HVGA (default), HVGA-L, HVGA-P, QVGA-L, QVGA-P
id:2
    Name: Android 1.5
    Type: platform
    API level: 3
    Skins: HVGA (default), HVGA-L, HVGA-P, QVGA-L, QVGA-P
id:3
    Name: Google APIs
    Type: add-on
    Vendor: Google Inc.
    Description: Android + Google APIs
    Based on Android 1.5 (API level 3)
    Libraries:
    * com.google.android.maps (maps.jar)
        API for Google Maps
    Skins: HVGA (default), HVGA-L, QVGA-P, HVGA-P, QVGA-L
```

En ce qui concerne la cible, elle désigne la version du système android qu'utilisera l'émulateur, il faut savoir qu'il peut exister de plus ou moins grosses différences d'une version à l'autre. Il faut également noter que si l'application que l'on désire créer utilise des librairies Google comme par exemple pour la cartographie (ce qui est le cas ici) il faut utiliser une cible désignée en tant que Google API, celles ci existent également pour chaque version d'Android de manière indépendante.

Cela peut paraitre étrange mais il faut savoir une chose cruciale avant de continuer: autant le système d'exploitation Android est Open Source, autant les librairies de Google sont propriétaires c-a-d que le code source est fermé et non disponible et donc susceptible de changer et d'apporter des problèmes si une régressions à lieu (j'expliquerai plus loin que ce n'est pas juste une vue de l'esprit malheureusement).

4.7.4. Émulateur Android

Ceci est la fenêtre de l'émulateur Android: ce n'est pas simplement un petit environnement de test mais bel et bien l'environnement complet, ou en tout cas Android dans sa vraie forme tel qu'il a été développé par Google: sans les modifications visuelles que les différents constructeurs ajoutent au dessus.

Il possède toutes les caractéristiques d'un des téléphones de développement que Google vend pour le développement, c'est à dire avec des options permettant le debug directement sur l'appareil tel l'attachement automatique du débugger sur une application donnée, le choix d'activer ou non l'affichage des taches courantes, des services, de l'utilisation du processeur, utilisation de la RAM, affichage du nombre d'images par seconde, activation ou désactivation des effets visuels, ...

Pour le faire apparaitre, il existe deux moyens :

1. lancer l'exécutable android qui se trouve dans le répertoire tools du SDK à partir du terminal suivi du nom de l'avd que l'on souhaite lancer:

   "emulator -avd lenomdelavd"

2. la deuxième manière est de démarrer un projet Android, l'émulateur démarrera alors automatiquement (on reviendra plus tard sur la façon de démarrer un projet).

Partie 3

Étude du système d'exploitation Android

1. Vue d'ensemble

Android est un système d'exploitation mobile utilisant une version modifiée du kernel Linux. Il a originellement été développé par Android Inc., une firme qui fut plus tard rachetée par Google, et dernièrement par la OHA (Open Handset Alliance).

Il permets aux développeurs d'écrire du code managé en Java, contrôlant les appareils via des librairies Java développées par Google.

Le dévoilement de la distribution du système Android le 5 novembre 2007 fut annoncé avec la fondation de la Open Handset Alliance, un consortium de 47 compagnies dans les domaines du matériel, du logiciel et des télécommunications engagées à la promotion de standards ouverts pour les appareils mobiles. Google a distribué la plupart du code source d'Android sous la licence Apache, une licence Open Source pour les logiciels gratuits.
Le 16 Février 2010, Google a annoncé que 60.000 téléphones Android se vendait par jour.

2. Histoire

En juillet 2005, Google acquière Android Inc., une petite startup basée à Palo Alto. A ce moment là, on en savait peu sur les activités d'Android Inc, si ce n'est qu'ils concevaient des logiciels pour téléphones mobiles. Ceci mis en avant des rumeurs selon lesquelles Google planifiait d'entrer dans le marché du téléphone mobile.

Chez Google, l'équipe dirigée par Rubin (CO-fondateur d'Android Inc.) a développé une plateforme mobile utilisant le kernel Linux qu'ils ont proposé aux concepteurs de téléphones mobiles et aux fournisseurs de réseaux téléphoniques en mettant en avant un système flexible et extensible.

Google s'était déjà associé avec une série de constructeurs de composant matériel et de développeurs de logiciels et avait signalé aux fournisseurs de réseaux téléphoniques qu'il était ouvert à divers niveaux de coopération de leur part.

Des spéculations celons lesquelles Google entrerait dans le marché du mobile allaient bon train en décembre 2006. Des rapports de la BBC et du Wall Street Journal montraient que Google voulait apporter son moteur de recherche et ses applications sur les téléphones mobiles et qu'ils y travaillaient. D'autres spéculations laissaient entendre que Google sortirait son propre téléphone.

Google révéla finalement son propre téléphone mobile, le Nexus One qui utilise le système Android. L'appareil est construit par le Taïwanais HTC, et est devenu disponible le 5 Janvier 2010.

2.1. Open Handset Alliance

Le 5 novembre 2007, la Open Handset Alliance, un consortium de plusieurs compagnies incluant Texas Instrument, Broadcom, Google, HTC, Intel, LG, Marvell, Motorola, Nvidia, Qualcomm, Samsung, Sprint et T-Mobile à été officialisé avec le but de développer des standards ouverts pour appareils mobiles. En même temps que la formation de la OHA, un premier produit à été révélé utilisant Android construit autours de la version 2.6 du kernel Linux.

Le 9 décembre 2008, fut annoncé que 14 nouveaux membres allaient rejoindre le projet Android, ceux-ci sont ARM Holdings, Atheros Communications, Asustek Computer Inc, Garmin Ltd, Softbank, Sony Ericsson, Toshiba Corp et Vodafone Group Plc.

2.2. Licence Utilisée

À l'exception de brèves périodes de mise à jour, les sources d'Android sont disponibles depuis le 21 Octobre 2008. Google a ouvert tout le code source (pile téléphonique et réseau inclues) sous la licence Apache. Avec la licence Apache, les vendeurs peuvent ajouter des extensions propriétaires sans devoir les soumettre a la communauté Open Source.

2.3. Historique des Versions

Version :	*Améliorations :*
1.5 (Cupcake) *Basée sur Kernel Linux 2.6.27*	*Le 30 Avril 2009, la mise à jour 1.5 (Cupcake) fut dévoilée. Il existe plusieurs nouveautés dans les options et dans l'UI.* • *Possibilité d'enregistrer et de regarder des vidéo en mode photo.* • *Upload de vidéos vers Youtube et de photos sur Picasa.* • *Nouveau clavier logiciel avec autocompletion.* • *Support Bluetooth A2DP.* • *Possibilité de connecter automatiquement un casque Bluetooth.* • *Nouveaux widgets et dossiers a utiliser sur le bureau.* • *Animations entre écrans.* • *Copier / Coller étendu.*

Version :	*Améliorations :*
1.6 (Donut) *Basée sur Kernel* *Linux 2.6.27*	*Le 15 Septembre 2009, la mise à jour 1.6 (Donut) fut dévoilée. Les innovations inclues sont :* • *Expérience Android Market améliorée.* • *Un appareil photo, enregistreur vidéo et galeries intégrées.* • *Galerie permets dorénavant a l'utilisateur de sélectionner plusieurs photos pour la suppression.* • *Recherche vocale mise à jour avec une réponse plus rapide et une intégration plus avancée avec les applications natives.* • *Possibilité d'effectuer des recherches sur les bookmarks, l'historique, les contacts ...* • *Support de CDMA/EVDO, 802.1x, VPN et des actions gestuelles.* • *Amélioration de l'accès a l'appareil photo.*
2.0/2.1 (Eclair) *Basée sur Kernel* *Linux 2.6.29*	*Le 26 Octobre 2009, la mise à jour 2.0 (Eclair) fut dévoilée. Les innovations inclues sont :* • *Optimisation de la vitesse.* • *Support pour plus de résolutions et de tailles d'écrans.* • *UI retravaillée.* • *Nouvelle interface navigateur web et support HTML5.* • *Nouvelle liste de contacts.* • *Meilleur ratio de couleurs fond d'écrans.* • *Amélioration de Google Maps* • *Support de Microsoft Exchange* • *Support flash de l'appareil photo.* • *Zoom digital.* • *Amélioration du clavier virtuel* • *Bluetooth 2.1* • *Fonds d'écran mouvants.* *Le 3 Décembre 2009 sortit le SDK 2.0.1* *Le 12 Janvier 2010 sortit le SDK 2.1.*

2.4. Premier téléphone Android sur le marché

Le premier téléphone du marché a utiliser le système d'exploitation Android était le HTC Dream, il sortit le 22 Octobre 2008.

À la fin de l'année 2009, il y avais au moins 18 modèles de téléphones utilisant Android celons Google. En plus des appareils vendus avec Android, certains utilisateurs ont réussi (de manière limitée) à l'installer sur des téléphones utilisant à la base d'autres systèmes d'exploitation.

3. Développement Logiciel

3.1. Débuts difficiles

Au début, le retour sur le développement d'applications pour la plateforme Android était mitigé. Les problèmes cités contenaient des bugs, manque de documentation, infrastructure de réponse aux questions inadaptée, ainsi que système de rapports non publique. Malgré cela, la semaine après l'annonce de la plateforme des applications ont commencé à apparaitre tels le jeu du serpent.

3.2. Dev Phone

Le téléphone de développement Android est un appareil déverrouillé au niveau de la carte SIM et du matériel qui est désigné pour les développeurs avancés. Pendant que les développeurs normaux peuvent utiliser un appareil du marché pour tester et utiliser leurs applications.

3.2.1. Configuration matérielle

La configuration matérielle d'un téléphone de développement Android est : Un écran tactile, Trackball, Appareil photo 3.2 MegePixel, Wifi, Gps, Bluetooth, 3G, Quadri Bande, Clavier coulissant QWERTY, carte SD 1GB.

Il s'agit en fait au niveau matériel du HTC Dream.

3.3. Problèmes du framework

3.3.1. Version de Java

Android n'utilise pas les standards établis par Java, tels Java SE et ME. Ceci empêche la compatibilité à travers les applications écrites pour ces plateformes et celles écrites pour la plateforme Android, ces dernières utilisent juste la syntaxe du langage Java, mais le fournissent pas les librairies de classes complètes et les API fournies avec Java SE ou ME.

3.3.2. Versions d'Android

Plusieurs développeurs on rapporté qu'il est difficile de maintenir des applications fonctionnant sur différentes versions d'Android, du fait des nombreux problèmes de compatibilité entre les différentes versions. Et également à cause des différents ratios de résolution d'écran des différents téléphones Android.

3.3.3. Garbage Collector

Le collecteur va, de manière plus ou moins drastique, ralentir les programmes qui font trop d'allocations mémoire, de façon à se que Dalvik puisse garder un certain espace de mémoire libre. Ceci affecte énormément la fluidité des applications.

3.4. NDK (Native Development Kit)

Des librairies écrites en C et en d'autres langages peuvent être compilées en du code ARM natif et installées en utilisant ne NDK. Les classes natives peuvent être appelées à partir de code Java étant exécuté dans la machine virtuelle Dalvik en utilisant l'appel à ***System.loadLibrary*** , lequel fait partie des classes Android standard.

Des applications complètes peuvent être compilées et installées en utilisant les outils de développement traditionnels. Le débugger ADB offre un terminal root dans l'émulateur Android qui autorise du code ARM natif à être envoyé et exécuté.

Le code ARM peut être compilé en utilisant GCC sur un pc standard. Faire tourner du code natif n'est pas simple à faire du fait que Android utilise une librairie C non standard (appelée Bionic). La sous-jacente couche d'affichage est accessible en tant qu'un framebuffer à l'emplacement ***/dev/graphics/fb0***. La librairie graphique utilisée par Android pour gérer et contrôler l'accès à cet élément est appelé Skia Graphics Library (SGL). SGL à un support pour win32 et Cairo, permettant le développement d'applications multiplateformes, il est également le moteur graphique se trouvant derrière le navigateur web Google Chrome.

3.5. Déploiement et applications

Les applications Android sont empaquetées au format .apk et stockées dans le répertoire /data/app sur la partition du Système d'exploitation.. L'utilisateur peut lancer la commande adb root pour accéder à se répertoire car seul root à les droits d'accéder à se répertoire.

4. Kernel

Android utilise Linux comme kernel (néanmoins modifié par Google pour l'ajuster à ses besoins et séparé de branches officielles du kernel Linux officiel), mais se n'est pas une distribution Linux conventionnelle; elle ne possède pas de Système X Window (gestionnaire d'interface graphique sous Linux) et elle ne supporte pas non plus le set complet des librairies standard GNU tel la librairie système GNU C. Ce qui rends difficile de réutiliser des applications Linux existantes ou même des librairies sur Android.

Google ne maintiens plus le code qu'ils ont fournis précédemment au kernel Linux, ils ont en effet récupéré leur branche pour la placer dans leur propre arbre, séparant donc leur code de Linux. Le code qui n'était donc plus maintenu à été effacé en Janvier 2010 de la base de codes de Linux.

5. Machine Virtuelle Dalvik

Dalvik est le nom de la machine virtuelle qui fait tourner la plateforme Java sur les appareils mobiles Android. Elle exécute les applications qui ont été converties aux format compacte .dex (Dalvik Executable), qui est le plus approprié pour les machines qui ont de fortes contraintes au niveau de la vitesse du processeur et de la quantité de RAM.

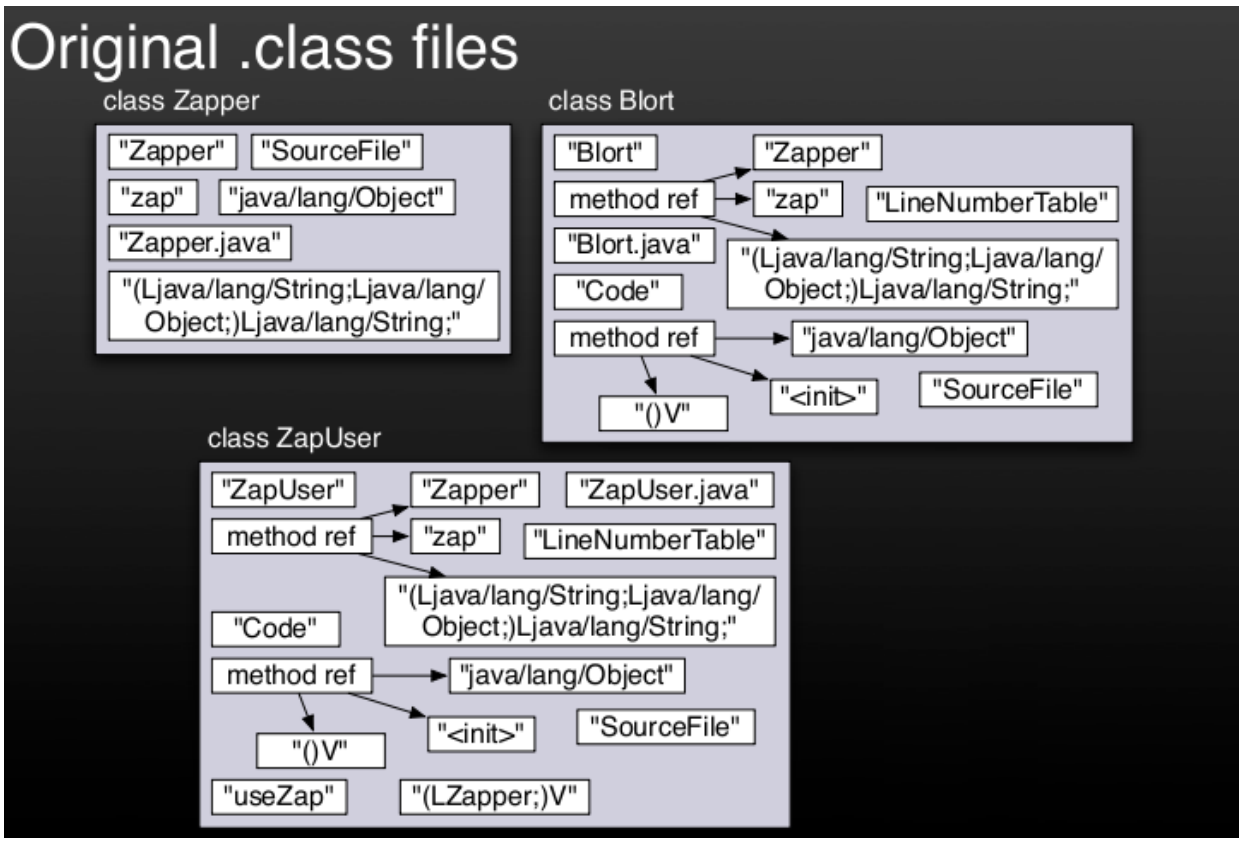
La VM Dalvik à été écrite par Dan Bornstein, qui lui a donné le nom du village Islandais dans lequel vivaient ses ancêtres.

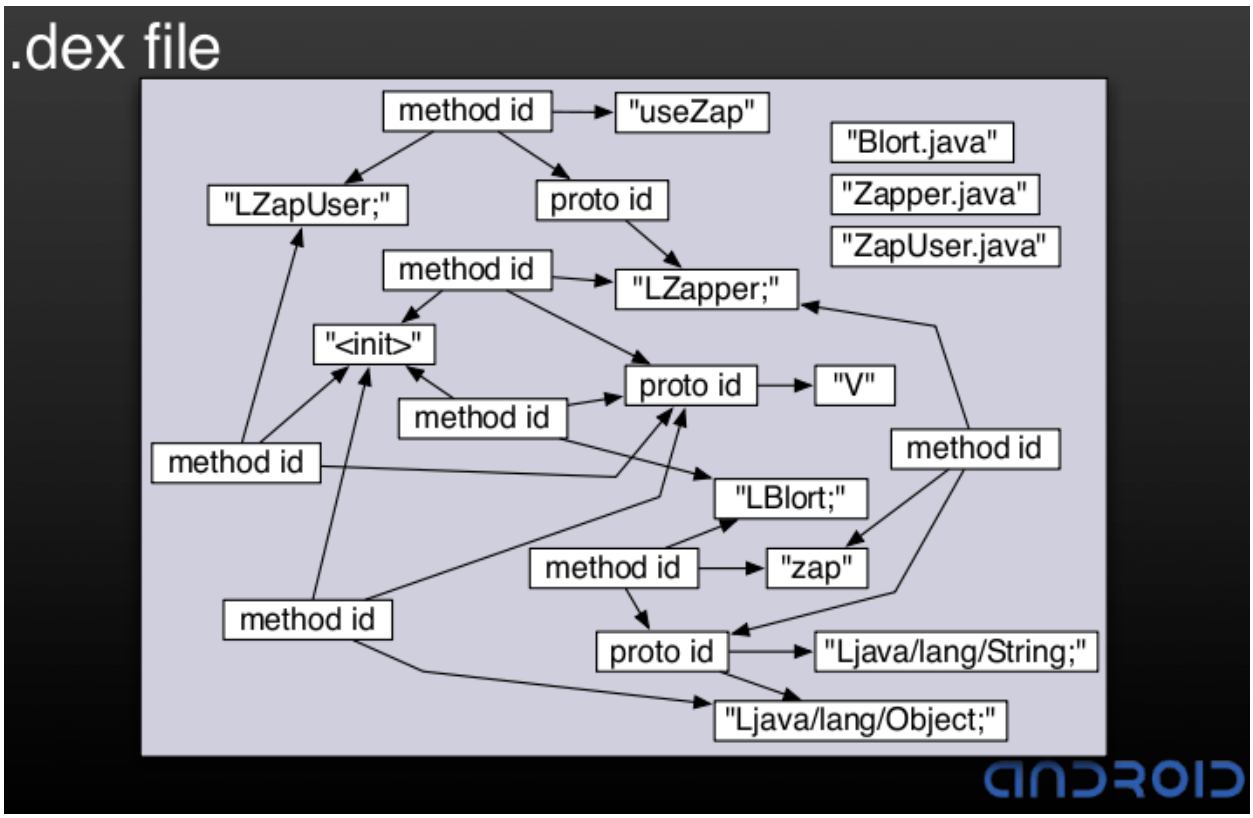
5.1. Architecture de la VM

Contrairement à la plupart des véritables machines virtuelles Java qui sont des machines a pile, la VM Dalvik basée sur une architecture par registres.

Généralement, les machines qui utilisent une pile doivent utiliser des instructions pour charger des données sur celle-ci et manipuler ses données, et par conséquent requièrent plus d'instructions que les machines à registre pour implémenter le même niveau de code. Néanmoins, les instructions dans les registres doivent encoder la source et la destination des registres et on donc tendance à être plus larges.

Cette différence est principalement d'importance aux interpréteurs de la VM pour lesquelles le dispatching des opérations deviens couteux alors que d'autres facteurs ont de l'importance pour la compilation JIT.


Original .class files
class Zapper
"Zapper"
"SourceFile"
"zap"
"java/lang/Object"
"Zapper.java"
"(Ljava/lang/String;Ljava/lang/Object;)Ljava/lang/String;"
class Blort
"Blort"
"Zapper"
method ref
"zap"
"LineNumberTable"
"Blort.java"
"(Ljava/lang/String;Ljava/lang/Object;)Ljava/lang/String;"
"Code"
method ref
"java/lang/Object"
"<init>"
"SourceFile"
"()V"
class ZapUser
"ZapUser"
"Zapper"
"ZapUser.java"
method ref
"zap"
"LineNumberTable"
"(Ljava/lang/String;Ljava/lang/Object;)Ljava/lang/String;"
"Code"
method ref
"java/lang/Object"
"<init>"
"SourceFile"
"()V"
"useZap"
"(LZapper;)V"


.dex file
method id
"useZap"
"Blort.java"
"LZapUser;"
proto id
"Zapper.java"
"ZapUser.java"
method id
"LZapper;"
"<init>"
proto id
"V"
method id
method id
method id
"LBlort;"
method id
"zap"
method id
proto id
"Ljava/lang/String;"
"Ljava/lang/Object;"
android

5.2 Différences avec VM habituelles

Étant optimisée pour les appareils avec peut de mémoire, la VM Dalvik possèdes des caractéristiques qui la différencie des autres VM standard :

- la VM à été rétrécie pour prendre moins de place;
- dalvik ne possède pas de compilateur JIT (Just In Time), mais Android 2.0 en possède un expérimental désactivé par défaut;
- la plage des constantes a été modifiée pour utiliser uniquement des index sur 32bit pour simplifier l'interpréteur;
- elle utilise son propre bytecode et pas celui de Java. À la place, un outil appelé **dx** transforme les fichiers *".class"* du Java de base en un autre format : *".dex"*.

Ce format dex est en réalité un regroupement de tout les fichiers .class ce qui permets par la même occasion de supprimer toutes les références multiples aux constantes ou aux fonctions ce qui évite une forte redondance et allège le tout ce qui permets de diminuer la taille des index. Voici le résultat de la compression au format dex pour une exécution sur la machine Dalvik.

common system libraries (U) 21445320 — 100% (J) 10662048 — 50% (D) 10311972 — 48%		(U) uncompressed jar file (J) compressed jar file (D) uncompressed dex file
web browser app (U) 470312 — 100% (J) 232065 — 49% (D) 209248 — 44%		
alarm clock app (U) 119200 — 100% (J) 61658 — 52% (D) 53020 — 44%		

Partie 4

La programmation avec Android

1. Fondements des applications

1.1. Composants des applicatifs

Les applications Android sont écrites dans le langage de programmation Java. Le code Java compilé – de même que toutes les données et fichiers ressources que l'application requiert – sont empaquetées par l'outil aapt dans un package Android, c'est à dire une archive possédant le suffixe ".apk". Ce fichier est le moyen de distribuer des application et de les installer sur les appareils mobiles. Tout ce qui est intégré dans un fichier apk est considéré être une application.

De plusieurs manières, chaque application Android vit dans son propre monde :

- par défaut, chaque application s'exécute dans son propre processus. Android lance le processus lorsque n'importe quelle portion du code a besoin d'être exécutée, et le stoppe lorsque il n'est plus demandé et que les ressources système sont requises par d'autres applications;
- chaque processus possède sa propre JVM (Java Virtual Machine), ainsi, chaque application est isolée des autres;
- par défaut, à chaque application est assigné un ID utilisateur Linux différent (n'oublions pas que c'est sur un kernel Linux que repose Android). Les permissions sont configurées de telle manière à ce que les fichiers de cette application soient visibles uniquement par cet utilisateur c'est a dire uniquement par cette application (il existe néanmoins des méthodes pour les exporter vers d'autres applications).

Il est possible de s'arranger pour que deux applications partagent le même ID utilisateur: dans ce cas elles seront capables de voir les fichiers l'une de l'autre. Pour conserver des ressources système, les applications qui partagent le même ID peuvent aussi s'arranger pour être exécutées dans le même processus Linux, partageant ainsi la même JVM.

1.2. Composants des applications

Une des caractéristiques principales d'Android est qu'une application peut se servir des éléments d'autres applications (dans le cas où ces applications le permettent). Par exemple, si une application doit afficher une liste déroulante d'images et qu'une autre application en a développé une appropriée et l'a rendue disponible aux autres, on peut utiliser cette liste pour faire le travail, plutôt que d'en développer une spécifique. L'application n'incorpore pas le code de l'autre application ni ne crée un lien vers elle: à la place, elle démarre simplement cette partie de code de l'autre application quand le besoin se fait sentir.

Pour que cela fonctionne, le système doit être capable de démarrer un processus d'application lorsque n'importe laquelle de ses parties est nécessaire, et instancier les objets Java pour cette partie. Effectivement, contrairement aux applications sur la plupart des autres systèmes, les applications

Android n'ont pas un seul point d'entrée dans l'application (pas de fonction main() par exemple). À la place, il existe des composants essentiels que le système peut instancier et exécuter au besoin.

Il existe quatre type de ces composants :

- activity (activité) : Une activity représente une interface utilisateur graphique pour une action précise que l'utilisateur peut effectuer. Par exemple, dans une application de messagerie, on peut avoir une activity qui affiche une liste de contacts a qui envoyer des messages, une activity pour écrire un message au contact choisi, une activité pour la configuration, ... Alors quelles travaillent ensemble pour former une interface utilisateur cohérente, chaque activity dépends des autres. Chacune est implémentée comme une sous classe de la classe de base "Activity";

  Chaque activity possède une fenêtre par défaut dans laquelle dessiner, Typiquement, la fenêtre occupe toute la taille de l'écran et flotte au dessus des autres fenêtres. Une activity peut aussi utiliser des fenêtres additionnelles. Par exemple, une pop-up de dialogue qui attends une réponse de l'utilisateur à l'intérieur de l'activity;

- services : Un service n'a pas d'interface graphique, mais à la place s'exécute comme une tache de fond pour une période de temps non définie. Par exemple, un service peut jouer une musique en fond alors que l'utilisateur s'occupe d'autre chose, ou sa peut chercher des données sur le réseau. Chaque service étends la classe de base "Service".

  Il est possible de se connecter à un service qui s'exécute (et de démarrer ce service si il n'est pas en train de s'exécuter). Lorsque l'on est connecté, on peut communiquer avec le service à travers une interface que le service expose (en reprenant le même exemple, changer son statut de messagerie).

  De même que les activités et les autres composants, les services s'exécutent dans le thread principal de l'application. De telle manière, ils ne bloquent pas les autres composants de l'interface utilisateur, ils créent souvent un nouveau thread pour les tâches qui consomment du temps (nous reviendrons sur les threads un peu plus tard);

- broadcast receivers : Un récepteur de broadcast est un composant qui ne fait rien d'autre que recevoir et réagir aux annonces de broadcast. Beaucoup de broadcasts viennent directement du code système - par exemple, les annonces que le fuseau horaire a changé, que la batterie est faible, qu'une photo à été prise, ... Les applications peuvent aussi émettre des broadcast pour, par exemple, avertir que des données ont été téléchargées ....

  Une application peut avoir un nombre quelconque de récepteurs de broadcast pour répondre aux annonces qu'elle considère importantes. Tout les récepteurs étendent la classe de base "BroadcastReceiver". Les récepteurs n'ont pas d'interface graphique mais ils peuvent néanmoins démarrer une activité en réponse à une information qu'ils ont reçu. Typiquement, ils utilisent une icône persistante dans la barre de statut.

- content providers : Un fournisseur de contenu rend une partie spécifique des données de l'application disponible aux autres applications. Les données peuvent être stockées dans le système de fichiers, dans une base de donnée SQLite, ou de n'importe quelle autre manière. Le fournisseur de contenu étends la classe de base "ContentProvider" pour implémenter un groupe de méthodes standard qui permets aux autres applications de récupérer et de stocker les données du type qu'il contrôle.

  Néanmoins, les applications n'appellent pas ces méthodes directement: à la place elles utilisent un objet "ContentResolver" et appellent ses méthodes. Un ContentResolver peut parler avec n'importe quel fournisseur; il coopère avec le fournisseur pour gérer toutes les communications inter-processus impliquées.

Lorsque une requête doit être gérée par un composant particulier, Android s'assure que le processus de l'application du composant est en cours d'exécution, si besoin est qu'il soit démarré, et qu'une instance appropriée du composant est disponible, créant une instance si nécessaire.

1.3. Activation des composants : Intents

Les fournisseurs de contenu sont activés lorsque ils sont ciblés par une requête d'un ContentResolver. Les trois autres composants sont activés par des messages asynchrones appelés *"intents"* (un Intent est un objet qui comporte le contenu du message) pour les activités et les services, il appelle l'action étant requise et spécifie entre autre l'URI de la donnée sur laquelle agir. Plus précisément :

- une activity est démarrée (ou se voit donner quelque chose à faire) en passent un objet Intent à *"Context.startActivity(**new** Intent(ClassAppelante.**this**, ClasseAppellée.**class**))"* ou si un résultat est attendu de cette activité *"Context.startActivityForResult(**new** Intent(ClassAppelante.**this**, ClasseAppellée.**class**),idDésignantIntentAppelé);"*. L'activité qui réponds peut voir l'intent initial qui l'a démarré en appelant la méthode *"getIntent()"*. Android appelles la méthode "onNewIntent()" de l'activité pour la passer a tout les intents suivants;
- un service est démarré (ou de nouvelles instructions sont données à un service existant) en passant un objet Intent à *"Context.startService()"*. Android appelles la méthode *"onStart()"* du service et lui passe l'objet Intent. De même, un Intent peut être passé à *"Context.bindService()"* pour établir une connexion entre le composant qui appelles et le service cible. Le service reçoit un objet Intent dans un appel à *"onBind()"*. Si le service n'est pas en cours d'exécution *"bindService()"* peut optionnellement le démarrer;
- une application peut initialiser un broadcast en passant un objet Intent à des méthodes telles que *"Context.sendBroadcast()"*, *"Context.sendOrderedBroadcast()"* et *"Context.sendStickyBroadcast()"* quelle que soit leur variation. Android fournis alors l'Intent à tout les récepteurs de broadcast intéressés on appelant leur méthode *"onReceive()"*.

1.4. Arrêt des Composants

Un fournisseur de contenu est actif seulement lorsque il réponds à une requête d'un ContentResolver. Un BroadcastReceiver est actif seulement lorsque il réponds a un message de broadcast. Il n'y a pas de besoin explicite d'arrêter ses composants.

Les activity, d'un autre coté fournissent une interface utilisateurs et peuvent donc rester actives ou en attente assez longtemps. Ceci est aussi valable pour les services. Android possède ainsi des méthodes permettant d'arrêter ceux-ci.

- une activity peut être arrêtée en appelant sa méthode *"finish()"*. Elle peut également arrêter une autre activity (qui aura été démarrée avec startActivityForResult()) en appelant *"finishActivity(**int** requestCode);"*
- un service peut être arrête en appelant la méthode *"stopSelf()"* ou en appelant *"Context.stopService(Intent name);"*.

Les composants peuvent aussi être arretés par le système lorsque ils ne sont plus utilisés ou lorsque Android a besoin de mémoire pour les composants actifs

2. Le cycle de vie d'une activity

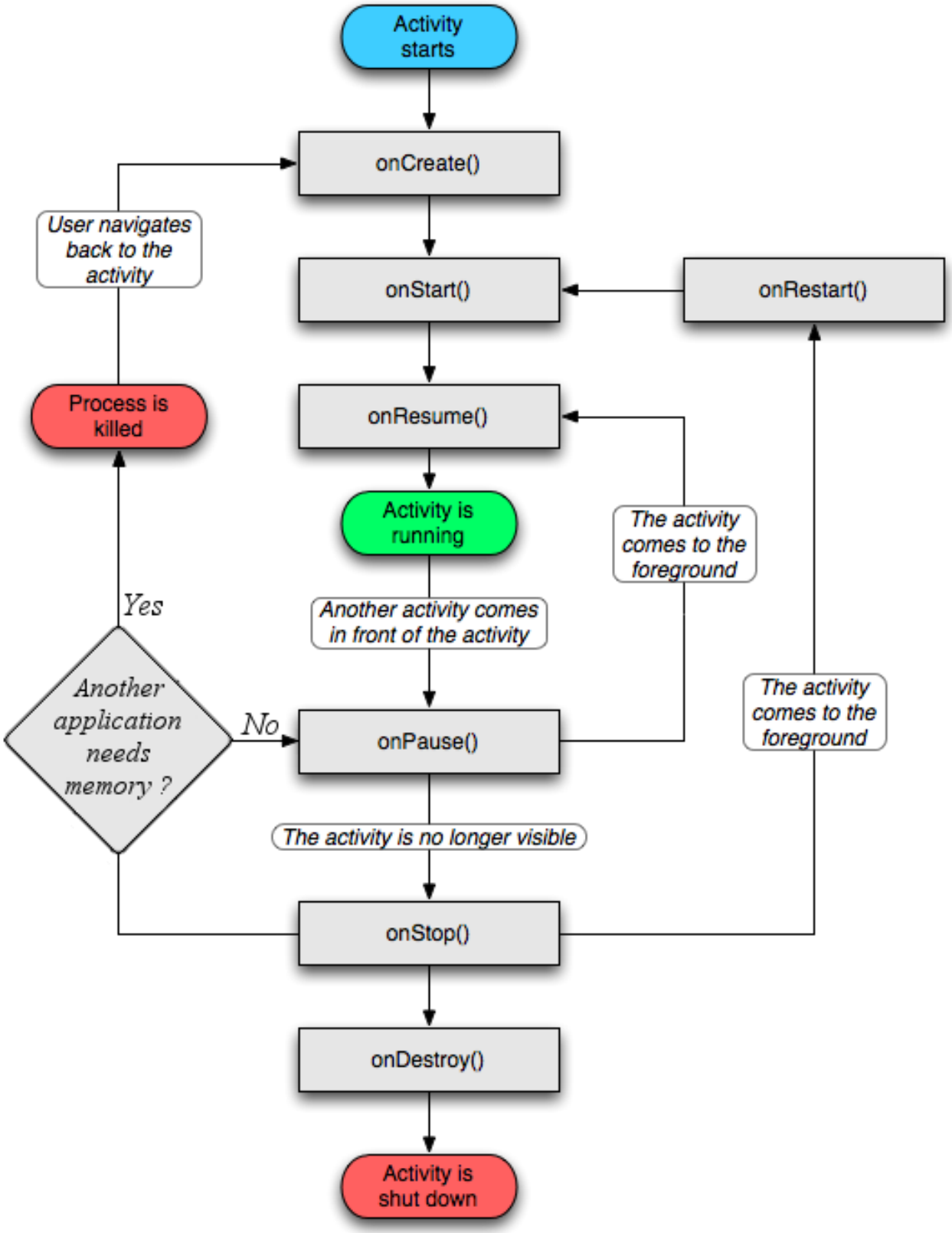
Une activity peut donc avoir trois états :

- active lorsque qu'elle est en premier plan à l'écran;
- en pause lorsque elle n'en plus en avant, mais est toujours existante en arrière plan. Elle existe toujours mais peut être tuée si le système a besoin de mémoire;
- stoppée si elle est complètement remplacée par une autre activity, elle conserve cependant son état et ses informations, elle sera sûrement d si le système à besoin de mémoire.

Comme une activity passe d'un état à l'autre, ceci est notifié par des appels aux méthodes protégées suivantes :

```
void onCreate(Bundle savedInstanceState)
void onStart()
void onRestart()
void onResume()
void onPause()
void onStop()
void onDestroy()
```

Voici la représentation schématique du cycle de vie d'une activity :


Activity starts
onCreate()
User navigates back to the activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
The activity comes to the foreground
Yes
Another activity comes in front of the activity
Another application needs memory ?
No
onPause()
The activity comes to the foreground
The activity is no longer visible
onStop()
onDestroy()
Activity is shut down

3. Le cycle de vie d'un service

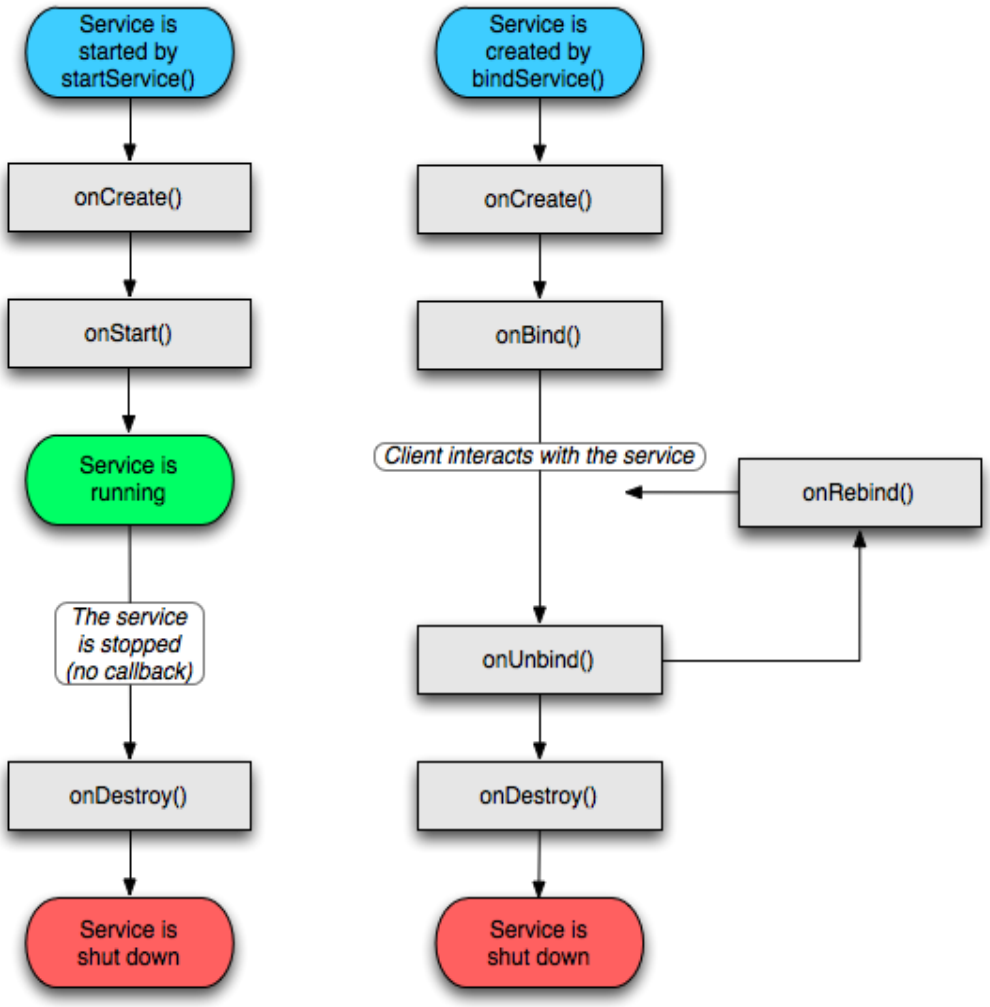
Un service peut être utilisé de deux manières :

- il peut être démarré et attendre qu'on le stoppe ou s'arrêter de lui même.
- il peut être utilisé par programmation en utilisant une interface qu'il définit et exporte. Les clients établissent alors une connexion à l'objet service et utilisent cette connexion pour appeler le service.

Comme une activity, un service possède des méthodes pour gérer son cycle de vie que l'on peut implémenter pour avoir un monitoring sur son état (ces méthodes sont publiques) :

void onCreate()
void onStart(Intent *intent*)
void onDestroy()

Voici la représentation schématique du cycle de vie d'un service :



4. Le fichier AndroidManifest.xml

Chaque application doit avoir un fichier AndroidManifest.xml (avec ce nom précis) dans son répertoire racine. Le manifest contient les informations essentielles à propos de l'application pour le système Android. Entre autres, voici ce que fait le manifeste :

- il déclare le nom de package Java pour l'application, ce qui sers d'identifiant unique;
- il décrit les composants de l'application tels les activity, les services, les récepteurs de broadcast, et les fournisseurs de contenu dont l'application est composée; il déclare les noms des classes du package, qui contiennent chacun de ses composants et publie leur capacités. Ces déclarations permettent au système de savoir quels composants peuvent être démarrés et dans quelles conditions;
- il détermine quel processus va contenir les composants de l'application;
- il déclare quelles permissions l'application doit avoir pour pouvoir accéder a des parties protégées de l'API et interagir avec d'autres applications;
- il déclare également les permissions que les autres doivent avoir afin d'interagir avec les composants de l'application;
- il liste les classes Instrumentation qui fournissent un profilage et d'autres informations tels l'état d'exécution de l'application; ces déclarations ne sont dans le manifest que lors de la partie développement et test de l'application;
- il déclare le niveau minimum de l'API que l'application requiert;
- il liste les librairies auxquelles l'application doit se lier.

5. Structure du fichier manifest

Dans le diagramme qui suit on peut voir la structure globale du manifest et de chaque élément qu'il contiens :

```
<?xml version="1.0" encoding="utf-8"?>
<manifest>
   <uses-permission />
   <permission />
   <permission-tree />
   <permission-group />
   <instrumentation />
   <uses-sdk />
   <uses-configuration />
   <uses-feature />
   <supports-screens />
   <application>
```

```
    <activity>
      <intent-filter>
        <action />
        <category />
        <data />
      </intent-filter>
      <meta-data />
    </activity>
    <activity-alias>
      <intent-filter> . . . </intent-filter>
        <meta-data />
    </activity-alias>
    <service>
      <intent-filter> . . . </intent-filter>
      <meta-data/>
    </service>
    <receiver>
      <intent-filter> . . . </intent-filter>
      <meta-data />
    </receiver>
    <provider>
      <grant-uri-permission />
      <path-permission />
      <meta-data />
    </provider>
    <uses-library />
  </application>
</manifest>
```

Voici à quoi cela ressemble pour une petite application basique :

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.mimp"
    android:versionCode="1"
    android:versionName="1.0">
  <application android:icon="@drawable/icon" android:label="@string/app_name">
  <uses-library android:name="com.google.android.maps" />

    <activity android:name=".screens.MapScreen" android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
```

```
    <activity android:name=".screens.AboutScreen" android:label="AboutScreen">
      <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name    ="android.intent.category.DEFAULT" />
      </intent-filter>
    </activity>

    <activity android:name=".screens.SettingsScreen" android:label="SettingsScreen">
      <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name    ="android.intent.category.DEFAULT" />
      </intent-filter>
    </activity>

    <activity android:name=".screens.ResolutionScreen" android:label="ResolutionScreen">
      <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name    ="android.intent.category.DEFAULT" />
      </intent-filter>
    </activity>

  </application>
  <uses-permission android:name="android.permission.INTERNET"/>
  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
  <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
  <uses-sdk android:minSdkVersion="4" />
</manifest>
```

5.1. Conventions

Certaines conventions et règles s'appliquent généralement à tout les élément et attributs dans le manifest :

- éléments: Seulement les éléments "<manifest>" et "<application>" son requis, chacun d'entre eux doit être présent et n'apparaitre qu'une seule fois. La plupart des autres peuvent apparaitre plusieurs fois ou pas du tout. Néanmoins il faut quand même que certains soient présents pour avoir un manifest qui ait une signification et capable de faire quelque chose;

  Les éléments au même niveau ne sont généralement pas ordonnés. Par exemple, les éléments "<activity>", "<provider>" et "<service>" peuvent être mélangés dans n'importe quelle séquence. Un élément "<activity-alias>" doit suivre le "<activity>" pour lequel il est un alias;

- attributs : Dans un sens formel, tout les attributs sont optionnels. Néanmoins, il y en a certains qui doivent être spécifiés pour que un élément puisse accomplir son action. À l'exception de certains attributs de l'élément racine "<manifest>", tout les noms attributs commencent par le préfixe "android:";
- déclaration des noms de classes : Beaucoup d'éléments correspondent a des objets Java, incluant des éléments pour l'application elle même (l'élément <application>) et ses principaux composants ("<activity>", "<service>", "<receiver>" et "<provider>").

  Si l'on définis une sous-classe, comme c'est toujours le cas pour les classes de composants (Activity, Service, BroadcastReceiver et ContentProvider) la sous classe est déclarée a travers un attribut "name". Le nom doit inclure la destination complète du package. Example :

```
<manifest . . . >
   <application . . . >
      <service android:name="com.example.project.SecretService" . . . >
         . . .
      </service>
   </application>
</manifest>
```

Néanmoins, en tant que raccourci, si le premier caractère de la chaine est un point, la chaine est ajoutée au nom de package de l'application. Exemple :

```
<manifest package="com.example.project" . . . >
   <application . . . >
      <service android:name=".SecretService" . . . >
         . . .
      </service>
   </application>
</manifest>
```

- valeurs multiples : Si plus d'une valeur peut être spécifiée, l'élément est presque toujours répété, plutôt que de lister plusieurs valeurs dans un même élément. Exemple :

```
<intent-filter . . . >
   <action android:name="android.intent.action.EDIT" />
   <action android:name="android.intent.action.INSERT" />
   <action android:name="android.intent.action.DELETE" />
   . . .
</intent-filter>
```

- valeur des ressources : Certains attributs ont des valeurs qui peuvent être affichées aux utilisateurs, tels du texte ou une icône pour une activity. Les valeurs de ces attributs devraient être localisés et donc placés depuis une ressource ou un thème (il est possible de définir des thèmes personnalisés pour une application, il existe déjà deux thèmes globaux prédéfinis qui sont un thème *"light"* et un thème *"dark",* j'y reviendrai plus tard) . Les valeurs des ressources sont exprimées dans ce format :
    - @[*package*:]*type*:*name*

  où le nom du package peut être omis si la ressource est dans le même package que l'application. "type" définis le type de ressource que l'on utilise tel "string" ou "drawable". Exemple :

```
<activity android:icon="@drawable/smallPic" . . . >
```

- chaines de caractère : Lorsque la valeur d'un attribut est une chaine, il faut utiliser des doubles backslashes ('\\') pour les caractères d'échappement par exemple, "\\n" pour un retour de ligne ou "\\u" pour un caractère Unicode.

```
<activity android:name=".org.Tabber" android:label="Ma\\tTabulation">
```

5.2. Thèmes et styles

Un style/thème est une collection de propriétés qui définissent une apparence et un format pour une vue ou une fenêtre. Un style peut spécifier des propriétés telles la hauteur, la largeur, la bordure, la couleur, ... Un style est définis dans une ressource XML séparée. Les styles dans le système Android partagent une philosophie identique aux feuilles de style en cascade dans le web design – ils permettent de séparer le design du contenu (MVC ...). Par exemple, en utilisant un style, on peut transformer se layout XML :

```
<TextView
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:textColor="#00FF00"
   android:typeface="monospace"
   android:text="@string/hello" />
```

Et le transformer en ceci :

```
<TextView
   style="@style/CodeFont"
   android:text="@string/hello" />
```

Tout les attributs relatifs au style ont été enlevés du XML de layout et ont été placés dans une définition de style appelée CodeFont qui ont dont appliquée avec l'attribut style. Pour définir un style in faut créer un fichier XML dans le répertoire *"res/values"*, peut importe son nom. Voici à quoi ressemble le contenu de se fichier :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <style name="CodeFont" parent="@android:style/TextAppearance.Medium">
      <item name="android:layout_width">fill_parent</item>
      <item name="android:layout_height">wrap_content</item>
      <item name="android:textColor">#00FF00</item>
      <item name="android:typeface">monospace</item>
   </style>
</resources>
```

Ces fichiers peuvent contenir plusieurs styles différents, ceux-ci seront alors séparés par la balise *"<style></style>"*. Chaque *"<item/>"* représente une propriété du style le nom définis la propriété que l'on souhaite configurer et son paramètre désigne la valeur que l'on désire lui donner.

Il existe deux moyens de placer un style :

- dans une vue individuelle, en ajoutant l'attribut style comme vu ci-dessus;
- dans une activité entière ou une application en ajoutant l'attribut *"android:theme"* au tag *"<activity>"* ou *"<application>"* dans le fichier AndroidManifest.xml.

Pour une application complète :

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.mimp"
    android:versionCode="1"
    android:versionName="1.0">
  <application android:theme="@style/CustomTheme"
 android:icon="@drawable/icon" android:label="@string/app_name">
  <uses-library android:name="com.google.android.maps" />
...
</manifest>
```

Pour seulement une activité :

```
<manifest
...
      <activity android:name=".screens.MapScreen" android:label="@string/app_name"
android:theme="@android:style/Theme.Translucent">
        <intent-filter>
          <action android:name="android.intent.action.MAIN" />
          <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
      </activity>
...
</manifest>
```

Il aura souvent été vu dans les différents codes présentés ces quelques lignes dans les fonctions *"onCreate(Bundle savedInstanceState)"* des différentes activités :

```
    requestWindowFeature(Window.FEATURE_NO_TITLE);
    setTheme(android.R.style.Theme_Light_NoTitleBar_Fullscreen);
```

La première de ces lignes signifie que l'on désire que la fenêtre n'affiche pas ne nom de l'activité, la deuxième force l'utilisation d'un des thèmes prédéfinis de la plateforme Android.

5.3. Permissions

5.3.1 L'utilisation

Une restriction est une limitation à l'accès à une partie de l'appareil. La limitation est imposée pour protéger des données critiques et ainsi d'éviter d'avoir du code qui provoque des dégâts ou pose des problèmes.

Chaque permission est identifiée par un nom unique. Le plus souvent le nom indique l'action qui est restreinte. Exemple :

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.mimp"
    android:versionCode="1"
    android:versionName="1.0">
  <application android:icon="@drawable/icon" android:label="@string/app_name">
  <uses-library android:name="com.google.android.maps" />
    ...
  </application>
  <uses-permission android:name="android.permission.INTERNET"/>
  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
  <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
  <uses-sdk android:minSdkVersion="4" />
</manifest>
```

Ici on requiers l'accès à internet : *"android.permission.INTERNET"*, l'accès au GPS : *"android.permission.ACCESS_FINE_LOCATION"*, la localisation par réseau telephonique ou réseau wifi : *"android.permission.ACCESS_COARSE_LOCATION"* et la permission d'écrire sur un stockage externe carte mémoire SD : *"android.permission.WRITE_EXTERNAL_STORAGE"*.

Une option peut être protégée par au maximum une permission. Si une application a besoin d'accéder à une option protégée par une permission, elle doit déclarer qu'elle requiert cette permission avec un élément <uses-permission> dans le manifest. Alors lorsque l'application est installée sur l'appareil, l'installateur détermines si il doit ou non fournir les permissions requises en vérifiant les autorités qui ont signé le certificat de l'application (nous y reviendrons juste après) et, dans certains cas, en demandant à l'utilisateur. Si la permission est donnée, l'application peut utiliser les options protégées.

5.3.2 La création

D'un autre côté, il est aussi intéressant de pouvoir sécuriser nos propres applications. Si l'application est constituées de simples Activity et d'Intent Receivers, la sécurité n'est peut être pas des plus importante. Mais si l'on utilise des ContentProvider ou des Services, il sera préférable d'implémenter une sécurité pour contrôler quelle application peut faire quoi sur vos données.

La première étape afin de sécuriser vos propres applications en utilisant les permissions est de déclarer les dites permissions, une nouvelle fois, dans le fichier AndroidManifest.xml. Dans ce cas ci, à la place de définir les permissions que nous voulons utiliser nous allons ajouter les permissions que nous requérons pour l'utilisation de nos éléments. Une nouvelle fois il est possible d'avoir entre zéro et plusieurs permissions, toutes dérivées de l'élément racine du manifeste.

Déclarer une permission est un peut plus compliqué que d'en utiliser une. Il y a trois informations qu'il faut fournir :

- le nom symbolique de la permission: pour éviter d'avoir des collisions de permissions avec celles des autres applications, il faut utiliser le namespace (le nom du package principal) Java de l'application en tant que préfixe;
- un label pour la permission: texte cours compréhensible par les utilisateurs;
- Une description de la permission: texte légèrement plus long compréhensible par les utilisateurs.

Example :

```
<permission
  android:name="me.org.secret.ALLOW_SECRET"
  android:label="Access Secret"
  android:description="Allows access to SecretAct data" />
```

Ceci ne force néanmoins pas la permission. Cela indique plutôt que c'est une permission possible; l'application doit toujours désigner les violations de sécurité lorsque celles-ci se produisent.

Pour ce faire, il existe deux méthodes pour que l'application force les permissions, en disant où et dans quelles circonstances ces permissions sont requises. Il est possible de forces ces permissions dans le code, mais la méthode la plus simple reste d'indiquer dans le manifeste où les permissions sont requises.

5.3.2.1 Dans le AndroidManifest.xml

Les Activity, Services et Intent Receivers, peuvent tous déclarer un attribut nommé android:permission, dont la valeur est le nom de la permission qui est requise pour accéder à ces éléments :

```
<activity
  android:name=".SecretAct"
  android:label="Top Secret"
  android:permission="me.org.secret.ALLOW_SECRET">
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
```

Seulement les applications qui ont requis cette permission seront capables d'accéder au composant sécurisé, dans ce cas ci, *"accéder"* signifie que :

- les Activity ne seront pas démarrées sans la permission;
- les Services ne seront ni démarrés, arretés ou liés à une Activity sans la permission;
- les BrodcastReceivers ignorent les messages envoyés avec sendBroadcast() tant que celui qui envoie ne possède la permission;

Les ContentProvider offrent deux attributs distincts : readPermission et writePermission :

```
<provider
  android:name="..SecretProv"
  android:authorities="me.org.secret.SecretProvider"
  android:readPermission="me.org.secret.READ_SECRET"
  android:writePermission="me.org.secret.WRITE_SECRET" />
```

Dans ce cas, readPermission contrôle l'accès à faire une requête de lecture au ContentProvider, pendant que writePermission contrôle l'accès aux insertions, mise à jour ou suppression des données dans celui-ci.

5.3.2.2 Dans le code

Par programmation, il existe deux moyens additionnels de mettre en place des permissions. Les Services peuvent vérifier les permissions à chaque appel à travers *"checkPermission(String permission, **int** pid, **int** uid);"*, qui retourne *"PackageManager.PERMISSION_GRANTED"* ou *"PackageManager.PERMISSION_DENIED"*, celons que l'appelant à la permission ou non.

Il est également possible d'inclure une permission lors d'un appel à *"sendBroadcast()"*, cela aura pour effet que seuls les BroadcastReceiver qui possèdent la permission voulue recevront le broadcast. Par exemple, le système Android inclus la permission RECEIVE_SMS lorsque il broadcast au système qu'un message SMS est réceptionné, ceci permets de faire en sorte que seuls les récepteurs autorisés recevront le message.

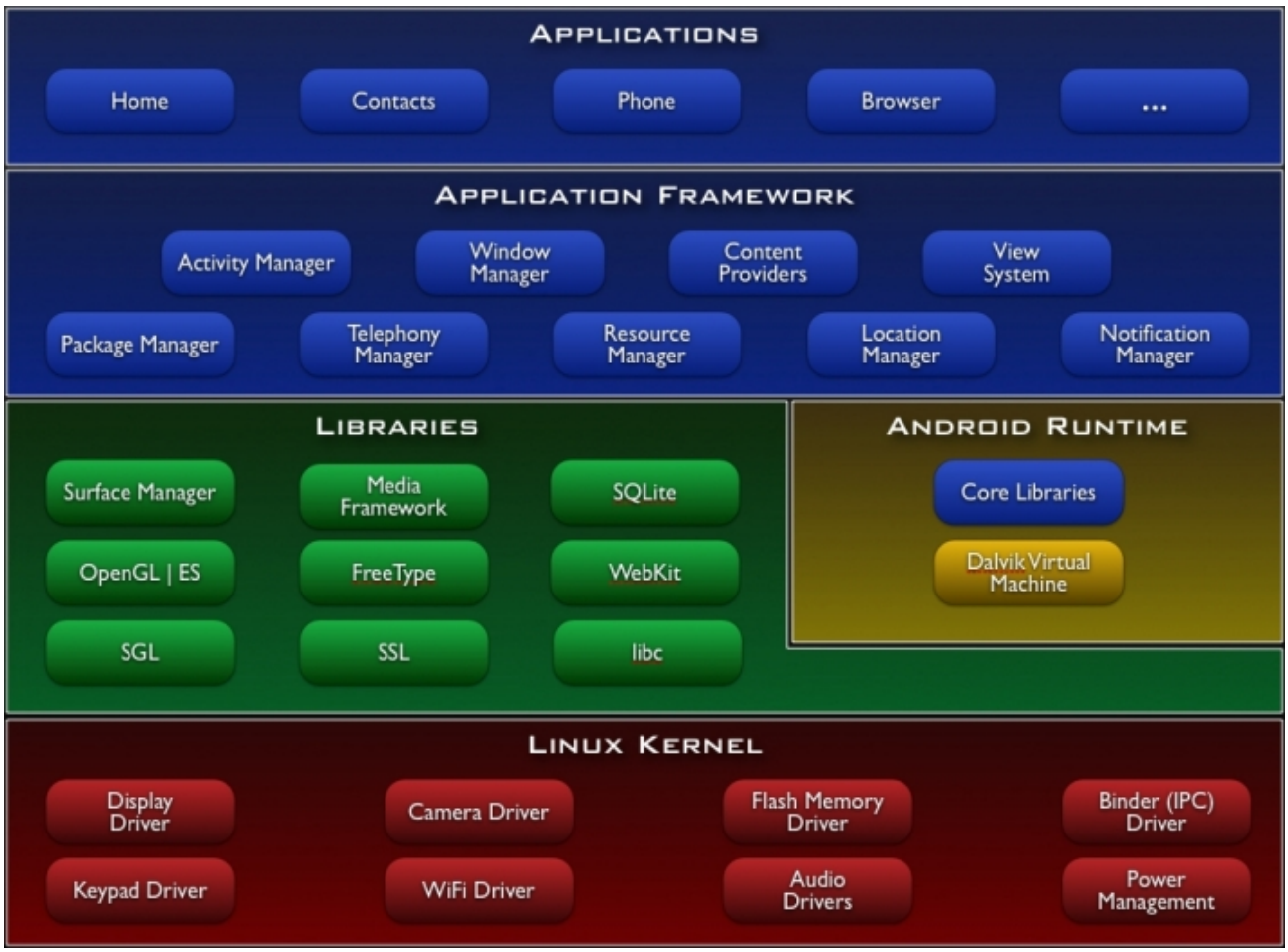
5.3.3 À l'utilisation

Il n'existe pas de mécanisme de vérification des permissions à la compilation, toutes les erreurs auront lieu à l'exécution. Il est ainsi important de documenter les permissions requises pour les API publiques, incluant les ContentProvider, Services et Activity qui sont désignées pour pouvoir être démarrées à partir d'autres Activity. Le cas contraire un programmeur qui essayera d'interfacer sera obligé de découvrir les permissions requises par essai et erreur.

De plus, il faut s'attendre à ce que les utilisateurs des applications se voient demandé de confirmer toutes les permissions requises par celle-ci. De ce fait, il faut documenter les permissions pour les utilisateurs, afin qu'ils sachent à quoi s'attendre afin qu'ils ne décident pas de ne pas utiliser l'application développée, si ils ne comprennent pas celles-ci.

6. Architecture Android

Le diagramme suivant représente les composants majeurs du système d'exploitation Android.



Android est basé sur le kernel Linux 2.6 pour la base du système tels la sécurité, gestion mémoire, gestion des processus, pile réseau, et modèle de pilotes. Le kernel joue aussi le rôle de couche d'abstraction entre le matériel et le reste de la pile logicielle.

Android inclus un groupe de librairies bas niveau (C/C++) qui fournissent la plupart des fonctionnalités disponibles dans la librairie de base du langage de programmation Java.

La machine virtuelle Dalvik se repose sur le kernel Linux pour les fonctionnalités sous-jacentes telles le threading et la gestion bas niveau de la mémoire.

Le framework d'application offre la possibilité aux développeurs d'accéder au matériel, a la localisation, démarrer des services de fond, activer des alarmes, ajouter des notifications dans la barre de statut, ...

7. Le fichier "R.java"

Le fichier "R.java" est un index dans lequel toutes les ressources sont définies. Cette classe est utilisée dans le code source comme une sorte de raccourci pour référencer les ressources inclues dans un projet. Ceci est particulièrement efficace avec les capacités d'auto complétion du code que fournissent les IDE tels Eclipse car il laisse rapidement et de manière interactive localiser les références spécifiques recherchées.

Si dans Eclipse, on ouvre le fichier nommé R.java (qui se situe dans le répertoire gen/) (après avoir suivi l'étape création d'un projet). Il devrait ressembler a quelque chose dans le genre :

```
package com.example.helloandroid;
public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
    }
}
```

Il est possible que le vôtre semble légèrement différent que ceci (les valeurs hexadécimales sont différentes). Pour l'instant, remarquez surtout la classe interne appelée "layout", et son champ membre "main". Le plugin Eclipse à remarqué l'existence du fichier de layout XML nommé "main.xml" et à généré un champs membre pour celui-ci. Au fur et a mesure que d'autres ressources sont ajoutées au projet, on peut voir le fichier R.java changer pour s'adapter.

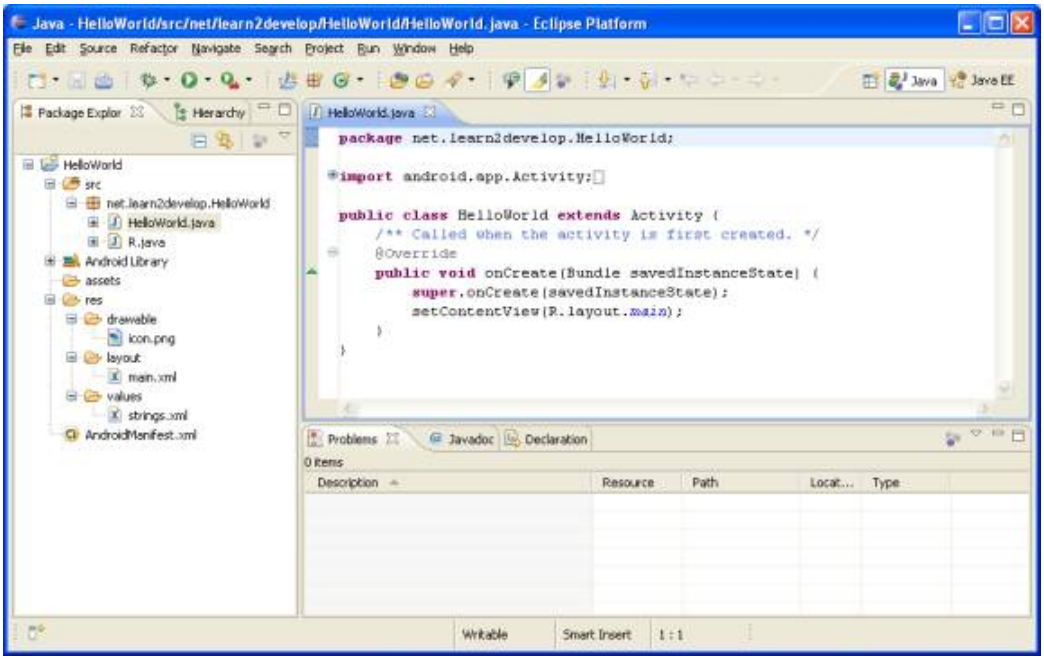
Lorsque Eclipse n'est pas utilisé, ce fichier sera généré au moment du build (avec l'outil Ant).

> Indice: Ce fichier ne doit jamais être édité à la main.

8. Création d'un projet

8.1. Hiérarchie d'un projet

Voici la représentation de la hiérarchie d'un projet Android tel qu'il est dans Eclipse :



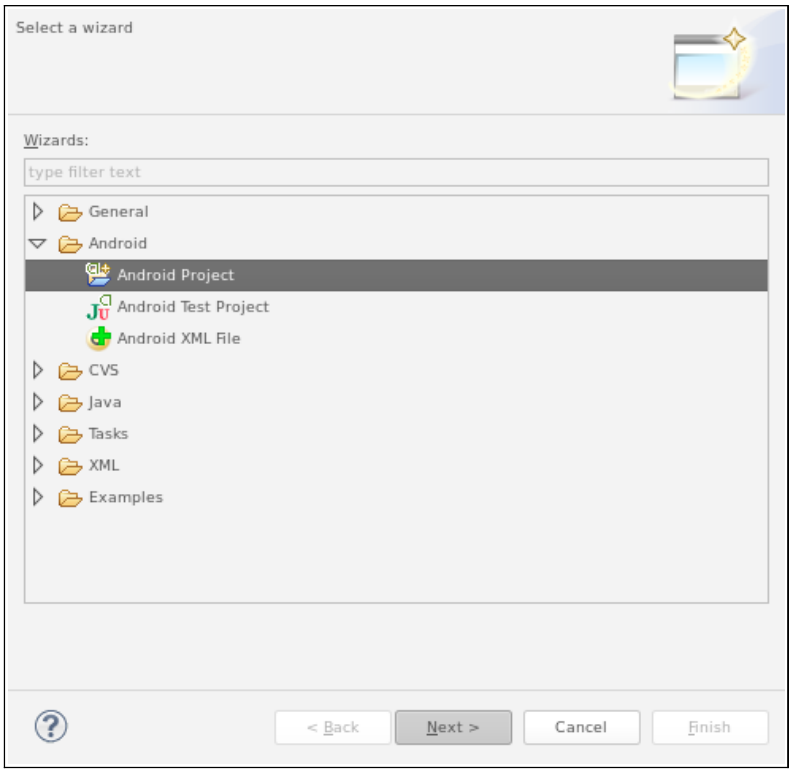
On peut observer que la racine contiens:

- un répertoire "src/" : celui-ci contiendra tout les packages de notre application ainsi que tout les fichiers source;
- un répertoire "AndroidLibrary/" : celui-ci contiendra la librairie android adaptée à la version minimum de la plate-forme qui aura été choisie pour le projet;
- un répertoire "res/" : celui-ci contiendra la plupart des ressources du projet, que ce soit les layouts ("layout/" - "layout-land/"), les images ("drawable/" - "drawable-land/"), les valeurs tels: les chaines de caractères, couleurs, ... ("values/") et les images ("drawable-ld/" - "drawable-md/" - "drawable-hd/") toutes ces ressources seront accessibles à travers la classe R;
- un répertoire "assets/" : ce répertoire n'est pas très différent en apparence au répertoire res/ néanmoins, tout ce qui est placé dans assets/ restera un fichier brut, pour le lire, il faudra utiliser la classe *"AssetManager"* afin de le lire en flux de bytes.;
- le fichier "AndroidManifest.xml".

8.2. Création du projet dans Eclipse

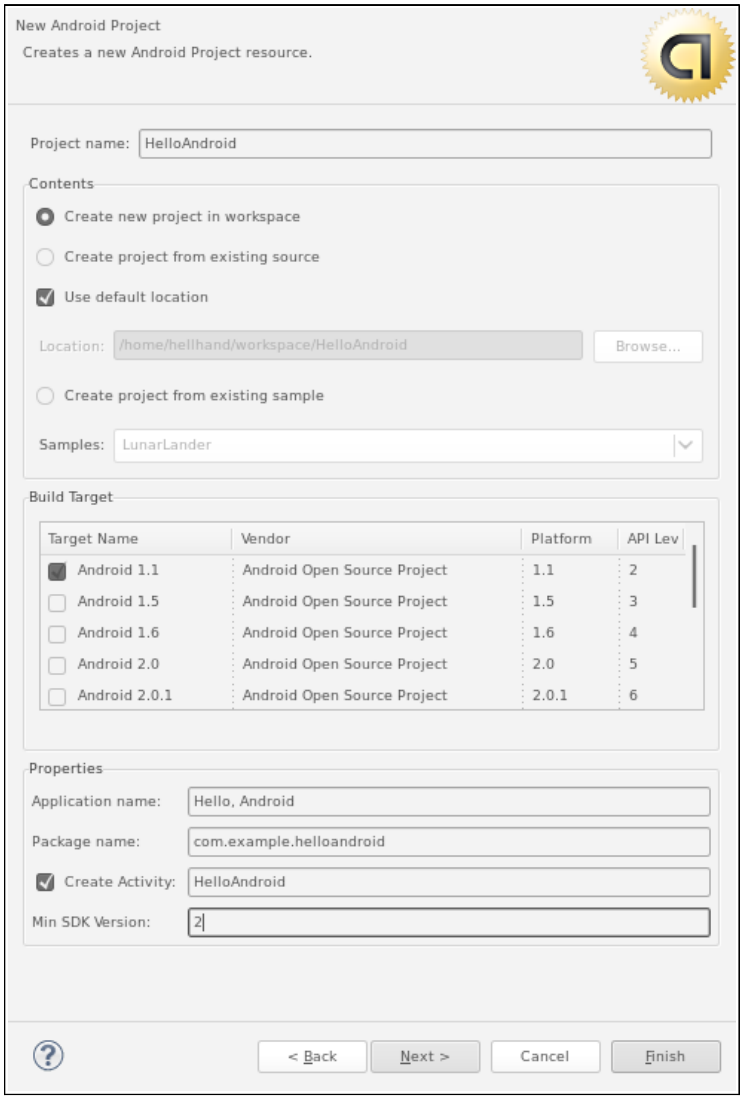
Après avoir créé un AVD (expliqué plus tôt), la prochaine étape est de créer un nouveau projet Android dans Éclipse. Voici la marche à suivre :

dans Éclipse, sélectionnez *"File > New > Project"*.
Si le plug-in ADT pour Éclipse à été correctement installé,
la boite de dialogue résultante devrait présenter un dossier nommé
"Android" qui devrait contenir *"Android Project"*.



sélectionnez *"Android Project"* et cliquez sur *"Next"*;

on obtiendra alors la fenêtre suivante :



remplissez les champs de détails du projet avec les valeurs suivantes:

"Project name" : HelloAndroid
"Application name" : Hello, Android
"Package name" : com.example.helloandroid (ou un autre espace de noms privé)
"Create Activity" : HelloAndroid
"Min SDK Version" : 2

Cliquez sur *"Finish"*;

Voici une description de chaque champs :

Project Name : C'est le nom du projet Éclipse, le nom du répertoire qui va contenir les fichiers du projet.

Application Name : C'est le titre humainement lisible de l'application; le nom qui apparaitra sur l'appareil Android.

Package Name : C'est l'espace de nom du package (qui suit les même règles que pour les packages dans le langage de programmation Java) dans lequel on veux que tout le code source réside. Il définit également le nom du package dans lequel l'activité principale sera générée.

Le nom du package doit être unique à travers tout les packages installés sur l'appareil Android. Pour cette raison, il est très important d'utiliser des noms standards de type domaine pour vos applications. L'exemple ci dessus, utilise le nom de domaine *"com.example"* qui est réservé pour les exemples. Lorsque l'on développe ses propres applications, on doit utiliser un espace de noms qui soit approprié à l'organisation ou l'entité.

Create Activity : C'est le nom de l'activité principale qui sera générée par le plugin. Cette classe fera office de *"main"*: elle sera instanciée en premier et fera office de point d'entrée pour l'application. Ce sera une sous-classe de la classe *"Activity"* d'Android. Une Activity est simplement une classe qui peut être exécutée et travailler. Elle peut créer une UI (User Interface / Interface Utilisateur) si besoin est mais ce n'est pas nécessaire. Comme la case à cocher le suggère, c'est optionnel, mais une Activity est presque toujours utilisée comme base pour une application.

Min SDK Version : Cette valeur représente le niveau d'API minimum requis pour l'application. Si le niveau de l'API entré corresponds au niveau de l'API fourni par l'appareil ciblé, alors cette cible sera automatiquement sélectionnée (dans ce cas-ci, entrer "2″ en tant que niveau d'API corresponds à une cible Android 1.1). Avec chaque nouvelle version d'une image système Android, il y a eu des ajouts ou des changements dans l'API. Lorsque cela se produit, un nouveau niveau d'API est assigné à l'image système dans le but de vérifier quelles applications sont autorisées à être exécutées. Si une applications requiers un niveau d'API supérieur que le niveau supporté par l'appareil, alors l'application ne sera pas installée.

Autre champs : La checkbox pour *"Use default location"* autorise à changer l'emplacement sur le disque où seront générés et stockés les fichiers. *"Build Target"* est la plateforme cible pour laquelle l'application sera compilée (ceci devrait être sélectionne automatiquement, basé sur la version minimale du SDK choisi). Remarquez que la "Build Target" sélectionnée utilise la version 1.1 d'Android. Ceci signifie que l'application sera compilée en utilisant les librairies de cette plateforme. Bien que l'AVD créé précédemment soit également en version 1.1, ceux ci n'ont pas besoin de correspondre; les applications Android sont compatibles vers le haut. De cette manière, une application créée avec les librairies de la plateforme 1.1 s'exécutera normalement sur une plateforme 1.5. L'inverse n'est pas vrai.

Le projet Android est maintenant près. Il devrait être visible dans l'explorateur de packages/projets sur la gauche. Ouvrez *"HelloAndroid.java"* qui se trouve dans le répertoire *"HelloAndroid > src > com.example.helloandroid"*.

Cela devrait ressembler à ceci :

```
package com.example.helloandroid;
import android.app.Activity;
import android.os.Bundle;

public class HelloAndroid extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

Notez que cette classe est basée sur la classe *"Activity"*. Une activité est une entité d'application unique qui est utilisée pour effectuer des actions. Une application peut être constituée de plusieurs activités séparées, mais l'utilisateur n'interagit qu'avec une seule d'entre elles en même temps. La méthode *"onCreate()"* va être appelée par le système Android lorsque l'activité démarre; c'est a cet endroit que l'on doit effectuer toutes les initialisations et configuration de l'UI. Une activité n'est pas requise pour avoir une interface utilisateur mais sera le plus souvent utilisée.

8.3. Création d'une interface

8.3.1. Par programmation

Une interface utilisateur Android est composée de hiérarchies d'objets appelés des "Views". Une vue est un objet affichable utilisé en tant qu'élément dans le design de l'interface utilisateur, tel un bouton, une image, ou dans le cas actuel une parcelle de texte. Chacun de ses objets est une sous classe de la classe View et la sous classe qui gère l'affichage de texte (pas l'édition) est la TextView.

Considérons le code (récupéré de la partie précédente) modifié suivant :

```
package com.android.helloandroid;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class HelloAndroid extends Activity {
    /** Called when the activity is first created. */
    @Override
```

```
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TextView tv = new TextView(this);
        tv.setText("Hello, Android");
        setContentView(tv);
    }
}
```

> Note: une manière facile d'ajouter des imports de packages est de presser **Ctrl-Shift-O** (**Cmd-Shift-O**, sur Mac). Ceci est un raccourci d'Eclipse qui identifie des packages manquant en se basant sur le code et en les ajoutant automatiquement.

Dans ce cas, une *TextView* à été créée avec le constructeur de classes, lequel acceptes une instance de la classe "*Context"* d'Android en tant que paramètre. Un contexte est une manière d'accéder au système; il fournit des services tels la résolution des ressources, l'obtention d'un accès aux bases de données et des préférences, et ainsi de suite. La classe Activity est héritée de *Context*, et parce-que notre classe HelloAndroid est une sous classe de Activity, elle est aussi un Context. De telle manière, nous pouvons passer *"this"* en tant que référence de *Context* à la *TextView*.

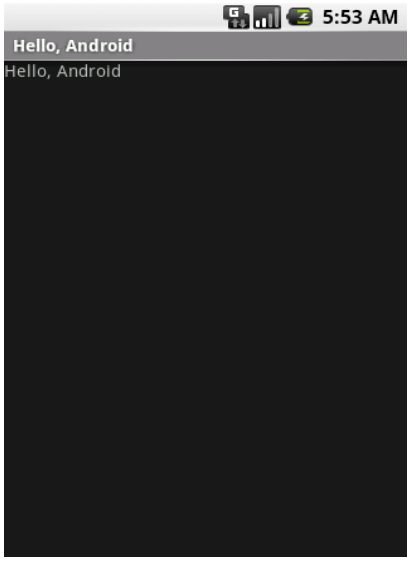
Ensuite, il faut définir le contenu textuel en utilisant *"setText (CharSequence)"* ou *"setText (int resid)"* (on parlera des ressources plus loin).

Finalement, il faut passer la TextView à *"setContentView ( )"* dans le but de l'afficher en tant que contenu pour l'interface utilisateur de l'Activity. Si l'Activity n'appelles pas cette méthode, alors il n'y aura pas d'UI présente et le système va alors afficher un écran vierge (habituellement noir).

Le plugin Eclipse rends l'exécution des applications facile, puisqu'il suffit de :

- effectuer la manœuvre suivante dans le menu *: "Run > Run";*
- sélectionner *"Android Application"*.

Eclipse ADT va alors créer automatiquement une nouvelle configuration d'exécution pour le projet et l'émulateur Android va automatiquement démarrer. Une fois que l'émulateur aura fini de démarrer, l'application va apparaître après un petit moment. On devrait maintenant voir quelque chose qui ressemble à ceci :



L'exemple "Hello, World" qui viens d'être fait utilise ce qu'on appelles un layout d'UI par "programmation". Ceci signifie que l'UI de l'application a été créée et été utilisée directement dans le code source. Si vous avez déjà fait quelque développement d'UI, vous êtes probablement familier des problèmes que cela peut apporter: de petits changements dans le layout peut résulter en de grands changements dans le code source. Il est aussi habituel d'oublier de connecter correctement des vues entre elles, ce qui peut impliquer des erreurs de layout et une perte de temps dans le debugging du code.

C'est pourquoi Android est capable de gérer un modèle de construction d'UI différent: fichiers de layout basés sur le XML. La manière la plus simple de présenter ce concept est d'utiliser un exemple. Voici un fichier de layout XML qui est identique dans son comportement à l'exemple créé par programmation:

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:text="@string/hello"/>
```

La structure générale d'un XML de layout Android est très simple: c'est un arbre d'éléments XML, dans lequel chaque élément est le nom d'une classe dérivant de View. Vous pouvez utiliser le nom de n'importe quelle classe qui étends View en tant qu'élément dans vos layouts XML, y compris vos propres classes dérivées de View que vous avez codé. Cette structure permet facilement de construire des UI très rapidement, tout en utilisant une structure et une syntaxe plus simple que ce qui se ferait par programmation. Ce modèle est inspiré par le modèle de développement web, dans lequel il est possible de séparer la présentation de la page de la partie logique; formulaires etc… Dans l'exemple au dessus, il ne se trouve qu'une vue: la TextView, laquelle possède quatre attributs XML. Voici un résumé de leur signification :

xmlns:android : C'est une déclaration d'espace de noms XML qui définit aux outils Android que vous allez faire référence à des attributs communs définis dans l'espace de noms

d'Android. Le tout premier tag dans chaque fichier de layout d'Android doit être celui-ci.

android:layout_width/height : Pour "*layout_width/height*", Cet attribut définit combien de la taille libre disponible sur l'écran cette vue doit consommer. Dans ce cas ci, c'est la vue parente, donc elle ocupera la taille de l'affichage, c'est ce que *"fill_parent"* signifie. Il est également possible d'utiliser *"wrap_content"* dans ce cas la, la taille se serait adapté au contenus de la vue.

android:text : Ceci définit le texte que la TextView va afficher. Dans cet exemple, il est utilisé une chaîne de ressources à la place d'une chaine de texte codée en dur. La chaine *hello* est définie dans le fichier *"res/values/strings.xml"*. Ceci est une pratique recommandée pour insérer des chaines de caractère dans les applications car elle rend aisée la localisation de l'application dans d'autres langages, sans avoir besoin de coder en dur les changements dans le fichier de layout.

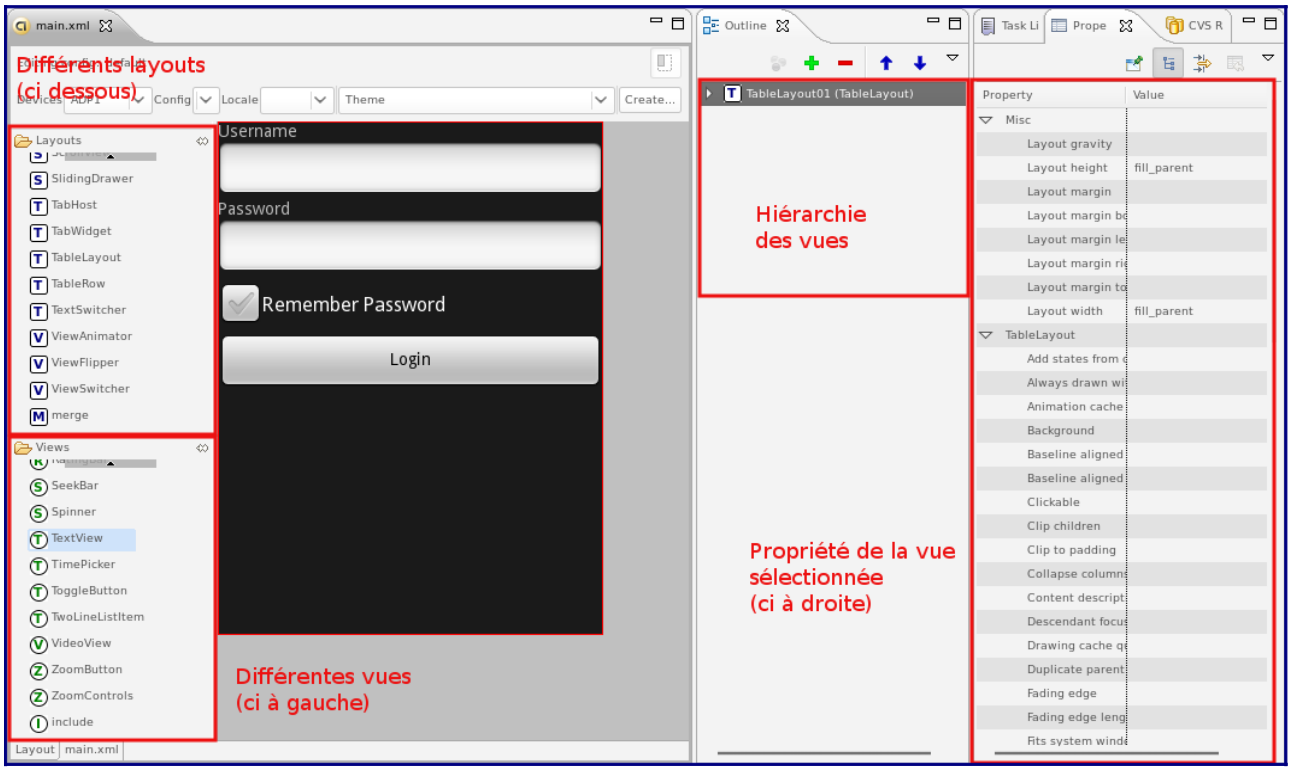
Les fichiers de layout XML se trouvent quand à eux dans le répertoire *"res/layout/"* du projet. Le *"res"* est l'abréviation de *"resources"* et le répertoire contiens toutes les données non code que l'application requiers. En plus des fichiers de layout, le dossier res/ inclus également les images, les sons/musiques et les chaînes localisées. De base, le plugin pour Eclipse génère automatiquement un de ces fichiers layout pour vous : *"main.xml"*.

Si vous voulez un design différent pour un affichage en paysage, il suffit de placer las fichiers de layout XML dans le répertoire res/layout-land. Android va automatiquement regarder a cet endroit lors d'un changement de layout pendant un changement d'orientation. Sans ce layout de paysage spécial défini, Android va étirer le layout par défaut.

8.3.2. Avec Eclipse

8.3.2.1. Éditeur d'interfaces

La première manière de créer une interface graphique avec Eclipse est d'utiliser le créateur d'interfaces. Celui-ci permets très simplement de créer des interface grâce a du simple drag & drop et par une fenêtre de propriétés pour configurer chaque partie de l'interface.



On peut voir sur la gauche le menu contenant les différents layouts (vers le haut) ainsi que les différentes vues disponibles (vers le bas). Sur la droite, on peut voir la hiérarchie des éléments par arbre avec également les propriétés pour l'élément sélectionné.

Pour obtenir un tel menu de login que celui ci-dessus, il suffit de supprimer le layout par défaut (linear layout) et le remplacer par un table layout il suffit ensuite d'ajouter un "TextView" qui contiendra le texte "Username", d'ajouter ensuite un "EditText" qui servira a accueillir l'entée utilisateur il suffit de répéter l'opération pour le "Password". On ajoutera finalement une "CheckBox" et ensuite un "Button".

8.3.2.2. À la main

Nous allons maintenant refaire la même interface que précédemment mais maintenant en utilisant l'éditeur XML inclus dans Eclipse. Cet éditeur est très simple d'utilisation car, a l'instar de la plupart des éditeurs Java, il propose de l'auto-complétion.

Il faut aussi bien comprendre que, comme c'est souvent le cas, éditer se fichier à la main est très efficace lorsque l'on veut créer une interface riche contrairement à l'utilisation de l'éditeur qui trouve vite ses limites et qui sers juste à voir si le résultat est satisfaisant.

Pour commencer on va créer le layout dont on a besoin :

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
android:id="@+id/TableLayout01"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
xmlns:android="http://schemas.android.com/apk/res/android">
```

Comme on peut le voir les champs commencent tous par "*android:*".
Pour ce qui est du champs "*id*", il désigne le nom que l'on désire donner à notre élément, attention il doit être unique dans l'application pour pouvoir le récupérer par la suite, les layouts comme celui-ci qui contiennent toute une fenêtre seront donc obligés d'avoir une id. Ce qui n'est pas le cas des autres éléments, on ne mettra pas d'id aux objets avec lesquels on ne souhaite pas avoir d'interactions tels une image ou du texte.

On ajoute ensuite les deux TextView, les deux EditText et la CheckBox

```
<TextView
android:id="@+id/TextView01"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Username" />
<EditText
android:id="@+id/EditText01"
android:layout_width="wrap_content"
android:layout_height="wrap_content" />
<TextView
android:id="@+id/TextView02"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Password" />
<EditText
android:id="@+id/EditText02"
android:layout_width="wrap_content"
android:layout_height="wrap_content" />
<CheckBox
android:id="@+id/CheckBox01"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Remember Password" />
<TextView
android:layout_width="wrap_content"
```

```
android:layout_height="wrap_content"
android:layout_weight="1"/>
<Button
android:id="@+id/Button01"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Login" />
</TableLayout>
```

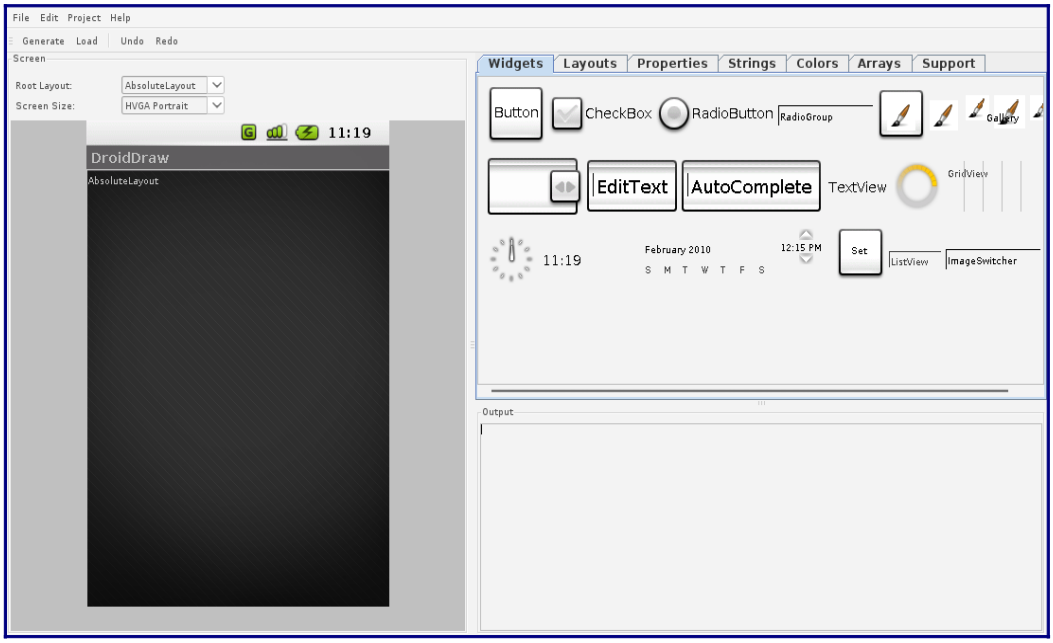
Ici, ca change légèrement on découvre le tag "*layout_weight*", celui ci accepte des valeurs en float. Il est est intéressant, car il permets de créer des limites à l'objet: imaginons que dans un layout on place une liste et que, vu que l'on désire qu'elle soit la plus grande possible on lui donne "*layout_width=fill_parent" et "layout_height= fill_parent*" on imagine bien que cette vue va prendre tout l'espace disponible (tout l'écran) et elle va également, si un bouton est placé en dessous de la fenêtre le cacher.
Pour éviter cela, on va lui donner un poids (on va la lester) pour qu'elle reste en dessous d'autres vues, ou plus précisément qu'elle n'interfère pas avec. Le bouton sera alors plus léger et aura sa place sur la fenêtre.

Voici le résultat :

8.3.3. Droid Draw

Sortons un peut d'Éclipse pour voir les autres moyens à notre disposition pour nous permettre de créer des interfaces.
Le plus connu de ses moyens s'appelle DroidDraw, c'est un utilitaire en java, donc multiplateforme, qui permets assez simplement de créer des interfaces assez élaborées.

En voici l'interface principale :



Cela ressembles certes assez fort a l'interface du créateur dans Eclipse, ici, on choisit le layout de base dans les différentes listbox en haut de l'affichage, on a ensuite accès à tout les mêmes élément également disponibles dans Eclipse sur la droite à la différence qu'ici l'on peut directement voir à quoi cela ressemble avant de le placer dans la fenêtre ce qui peut s'avérer fort utile lorsque l'on ne connaît pas encore tout les éléments disponibles de l'environnement.

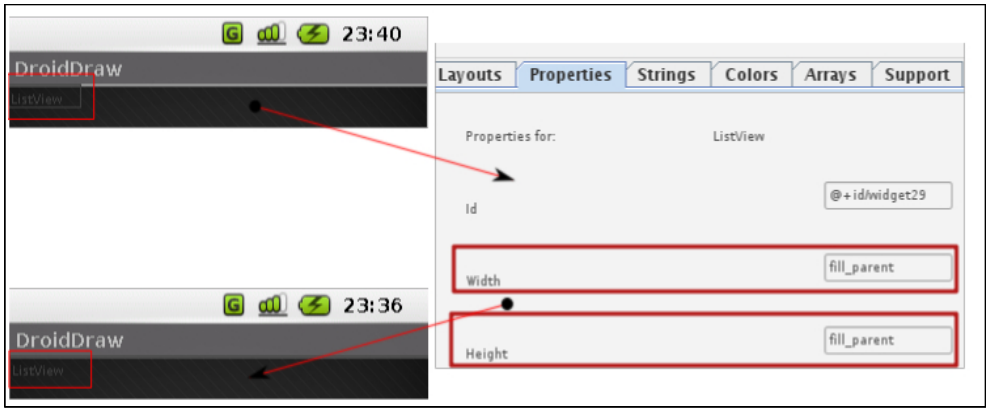
Dans le premier onglet nous avons les Widgets, autrement dit les Views.
Dans le second, les layouts, il ne faut pas oublier que un layout peut contenir un autre layout. Cela peut nous permettre de créer une page qui contiens des éléments TextView ou autre; et ensuite dans le bas de la page pourquoi pas avoir un TableLayout qui va contenir plusieurs TableRow.
Dans le troisième, se trouvent les propriétés, celles ci sont relatives à l'élément sélectionné dans la fenêtre du concepteur. Malheureusement, toutes les propriétés ne s'y retrouvent pas forcément et c'est assez dommage.
Dans le quatrième, le cinquième et le sixième, se trouvent la configuration des fichiers contenant respectivement les différentes chaînes de caractères, les différentes couleurs et finalement les tableaux.

On peut voir en bas à droite de la fenêtre un champs texte, après avoir créé une interface, il suffit de cliquer sur *"Generate"* pour en obtenir la représentation XML modifiable, il suffit alors de cliquer sur "*Load*" pour voir le résultat des modifications apportées.
Si au contraire, on préfère créer une interface à la main dès le début, et ensuite voir le résultat, il suffit de cliquer sur "*Load*" pour ce faire.

8.3.4. Droid Draw avec Eclipse

L'exemple qui suit, montre comment utiliser DroidDraw pour la partie design d'un projet tout en utilisant Eclipse pour la partie code; il sera expliqué la marche à suivre pour créer une ListView en utilisant un Array de valeurs.

Il faut en premier créer un projet Android dans Eclipse, DroidDraw peut ensuite être lancé, il suffit alors de suivre les étapes suivantes :



- sélectionner *"ListView"* dans l'onglet *"Widgets"* et la faire glisser dans la vue;
- double cliquer sur l'élément *"ListView"* dans la vues pour modifier ses propriétés;
- changer *"width"* et "*height"* en "fill_parent;
- cliquer sur *"Apply"*.

Maintenant il faut créer le tableau de valeurs qui sera utilisé pour remplir la *"ListView"*, pour cela il faut :

- se rendre dans l'onglet *"Array";*
- cliquer sur *"New"* pour créer un nouveau tableau de valeurs il faut alors lui donner un nom par example : *"myarray";*
- entrer des valeurs dans le tableau et les séparer pas une virgule;
- cliquer sur *"Save"* et enregistrer sous le nom *"arrays.xml"* et copier le fichier dans le répertoire *"res/values"* du projet.

Il faut maintenant retourner a la *"ListView"* précédemment créée pour lui dire d'utiliser le tableau qui viens d'être défini :

- modifier sa propriété *"Entry Array ID"* en *"@array/myarray"*

- cliquer sur *"Apply";*
- cliquer sur le bouton générer et enregistrer le fichier en tant que *"main.xml"* dans le répertoire *"res/layouts"* du projet;
- il s'agit maintenant de créer le code dans Eclipse qui va utiliser les éléments précédemment créés; pour cela il suffit de placer ce code dans notre Activity pour voir apparaître notre interface :

```
@Override
public void onCreate(Bundle icicle) {
    super.onCreate(icicle);this.setTitle("MyFirstApp");
    setContentView(R.layout.main);
}
```

Il est alors temps d'exécuter le projet pour voir le résultat dans l'émulateur.

9. Utilisation des Intent

Comme il l'a déjà été dis, un Intent encapsule une requête faite au système Android, afin que certaines activés ou autre Intent effectuent une action.

Si l'Activity que vous avez l'intention de démarrer est une des vôtres, vous pouvez trouver très simple de créer un Intent explicite, en nommant le composant que vous souhaitez lancer. Par exemple, à partir de l'activité, on lancera un Intent de cette manière :

```
new Intent(this, HelpActivity.class);
```

Ceci stipule que l'on veut lancer l'activité HelpActivity. Cette activity aura besoin d'être nommée dans le fichier AndroidManifest.xml. Mais pas nécessairement avec un filtre d'Intent.
Lorsque l'on a notre Intent, on a besoin de le passer à Android et de récupérer l'activité enfant a démarrer. Pour cela il existe plusieurs possibilités.

- l'option la plus simple est d'appeler startActivity() avec l'Intent ceci va faire en sorte que Android recherche l'activity la plus adaptée et lui passera l'Intent pour quelle s'en occupe. L'activité ne sera pas informée lorsque l'activité enfant est terminée;

```
startActivity(new Intent(Dispatcher.this, HelpActivity.class));
```

- il est également possible d'appeler startActivityForResult(), en lui passant l'Intent et un nombre (unique à l'activité appelante). Android recherche l'activity la plus adaptée et lui passera l'Intent. Néanmoins, l'activité courante sera avertie lorsque l'activité enfant sera terminée à travers l'appel de onActivityResult();

```
startActivityForResult(new Intent(Dispatcher.this, HelpActivity.class),100);
```

- il est également possible d'appeler sendBroadcast(), dans ce cas,Android va envoyer l'Intent a tout les BrodcastReceivers qui l'attendent;

```
sendBroadcast(new Intent(Dispatcher.this, HelpActivity.class));
```

- finalement, il est possible d'appeler sendOrderedBroadcast(). Dans ce cas ci Android va passer l'Intent à tout les BrodcastReceivers candidats un à la fois.

```
sendOrderedBroadcast(new Intent(Dispatcher.this,  HelpActivity.class), "me.org.HELPER");
```

La plupart du temps on utilisera donc startActivity() ou startActivityForResult() . Avec ce dernier, il est évident que l'on va implémenter la méthode onActivityResult() permettant de récupérer ce résultat. Celle ci recevra également le nombre unique fournis à startActivityForResult(), de telle manière, il est possible de savoir quelle activité enfant c'est terminée

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) { ... }
```

- une activité enfant retournera un résultat avec la méthode setResult(). C'est typiquement RESULT_OK ou RESULT_CANCELED, il est néanmoins possible d'utiliser d'autres codes soi même, ceux ci doivent être après RESULT_FIRST_USER;

```
setResult(101);
```

- une chaine de caractères optionnelle contenant un résultat (ex: une URL) peut également être retournée par un Intent de type ACTION_PICK;
- un bundle optionnel contenant des informations supplémentaires peut aussi être retourné.

L'envoi de paramètres a un Intent ne se limite heureusement pas a un unique entier, il est possible d'envoyer autant de données que nécessaire grâce à la fonction putExtra() qui prends en paramètres une chaine de caractères et une variable de type de base (int, string, float, ...). C'est en fait une hashtable qui sera donnée au processus enfant qui récupèrera les valeurs en utilisant getXExtra() ou X est le type de variable, elle prends en paramètre le nom du champs et une valeur par défaut.

```
startActivityForResult(new Intent(Dispatcher.this, TrackListScreen_left.class)
.putExtra("seconds", 300),15);
```

```
getIntent().getIntExtra("seconds", 100);
getIntent().getStringExtra("seconds", "100");
```

10. Internationalisation d'un projet

10.1. Au niveau théorique

Android peut être utilisé sur plusieurs appareils dans plusieurs régions. Pour atteindre la plupart des utilisateurs, l'application doit gérer le texte, les fichiers audio, les nombres, les devises et les graphiques de manière appropriée aux locales ou l'application sera utilisée.

Nous allons maintenant décrire les meilleures pratiques pour faire la localisation d'une application Android. Ces principes s'appliquent si vous développez une application en utilisant ADT avec Eclipse, outils basés sur ANT ou tout autre IDE.

Vous devriez déjà avoir des connaissances fonctionnelles en Java et être familier avec le chargement des ressources Android, la déclaration d'éléments d'interface utilisateur en XML, les considération de développement du cycle de vie des Activity et les principes généraux d'internationalisation et de localisation.

Il est de pratique courante d'utiliser le framework de ressources d'Android pour séparer l'aspect localisation de vos applications autant que possible de code Java :

- vous pouvez placer la plupart du contenu de l'interface utilisateur dans des fichiers ressources pour une gestion par le système;
- le comportement de l'interface utilisateur, d'un autre coté est dirigé par le code java. Si il faut un formatage spécial des données cela ne sera pas fait par les locales mais bien par programmation.

Les ressources sont du texte, des layouts, du son, des graphiques et toute autre donnée dont a besoin l'application Android. Une application peut inclure de multiple groupes de ressources, chacun personnalisé pour une configuration d'appareil différent. Lorsque un utilisateur va exécuter l'application, Android va automatiquement sélectionner et charger celles qui sont les plus adéquates pour la machine.

Lorsque vous écrivez votre propre application:		**Lorsque un utilisateur lance votre application:**
Vous créez un groupe de ressources par défaut, plus des alternatives pour être utilisées dans différentes locales.	→	Le système Android va choisir quelles ressources charger, basé sur la locale de la machine.

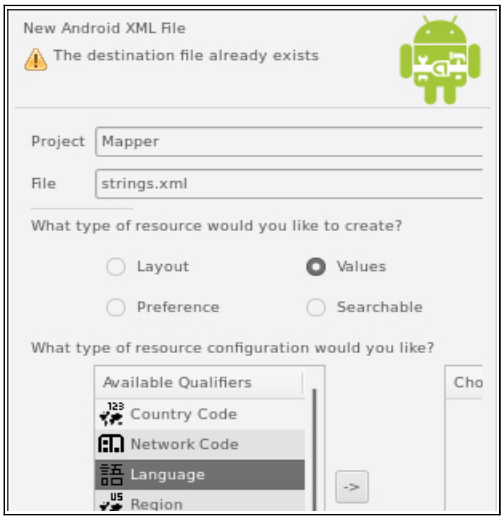
10.2. Au niveau pratique

10.2.1. Localisation des chaines de caractères

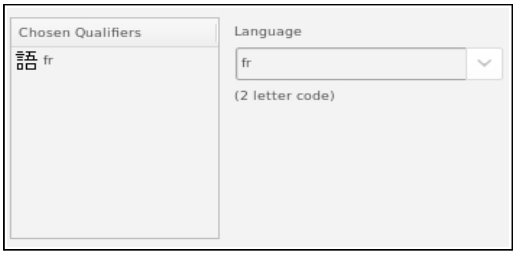
Prenons une application dont les chaines de caractères sont toutes définies dans le fichier strings.xml par défaut. Nous alors maintenant lui ajouter deux langues supplémentaires : le français, l'allemand. Pour créer ces fichiers de ressources dans Eclipse il faut :

- sélectionner "*File > New Android XML File*" ;

- sélectionnez le projet et entrez "*strings.xml*" dans le champs "*File*". Dans la liste de gauche, sélectionnez "*Language*", cliquez ensuite sur la flèche vers la droite;



- entrez "*fr*" dans le champs langue,



un nouveau fichier, "*res/values-fr/strings.xml*", apparait maintenant dans les fichiers du projet

- il suffit ensuite de répéter l'étape autant de fois que possible
- maintenant, il faut ajouter du texte localisé dans les différents fichiers. Pour ce faire il suffit d'ouvrir les fichiers localisés et d'y placer les même champs que dans le fichier non localisé.

Au final, nous nous retrouveront donc avec plusieurs fichiers qui ressembleront à ceci :

version non internationalisée

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Mapper</string>
    <string name="info">Info</string>
    <string name="voices">Voices</string>
    <string name="gps">GPS</string>
...
    <string name="minfo">Info</string>
    <string name="perspective">Perspective</string>
    <string name="settings">Settings</string>
</resources>
```

Voici maintenant pour la version localisée en français :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Mapper</string>
    <string name="info">Info</string>
    <string name="voices">Voix</string>
    <string name="gps">GPS</string>
...
    <string name="minfo">Infos</string>
    <string name="perspective">Perspective</string>
    <string name="settings">PrÃ©ferences</string>
</resources>
```

Dans le code nous auront donc quelque chose qui ressemble à ceci pour utiliser un des textes:

```
menu.add(0, POS, 0, R.string.myposition).setIcon(android.R.drawable.ic_menu_mylocation);
```

on peut donc voir que c'est le système Android lui même qui va gérer la localisation et que rien ne doit être fait au niveau du code en lui même. Il en est exactement de même en se qui concerne les icônes, celles ci peuvent changer en rapport à la locale (ex: on pourrait imaginer changer le drapeau d'une application en rapport avec la locale).

10.3. L'importance des ressources par défaut

Lorsque l'application s'exécute dans une locale pour laquelle il n'a pas été fournis de texte spécifique, Android va charger les chaines de caractère par défaut a partir de "res/values/strings.xml". Si ce fichier par défaut est absent, ou si il lui manque une chaine de caractères dont l'application à besoin, l'application ne va pas s'exécuter et va afficher une erreur. Voici ce qui se passe lorsque le fichier par défaut est incomplet :

Une application Java dont le code référence juste deux chaines de caractères, text_a et text_b. Cette application inclus un fichier de ressources localisé qui définis text_a et text_b en Anglais. Cette application inclus un fichier de ressources par défaut qui contiens uniquement text_a:

- cette application peut compiler sans problèmes, il se peut que des erreurs ne soient pas détectées;
- lorsque cette application est démarrée avec un appareil dont la locale est mise sur Anglais, l'application peut s'exécuter sans aucun problème, car le fichier de localisation contiens les deux chaines;
- néanmoins, si l'application est démarrée sur un appareil avec une locale différente, l'application ne se lancera pas.

11. Localisation et Cartes

Les services et applications basés sur la location et la cartographie attirent énormément les utilisateurs d'appareils mobiles. Ces capacités peuvent être plus ou moins simplement introduites dans vos applications en utilisant des classes du package "*android.location*" et de la librairie externe Google Maps.

11.1. Localisation

Android permets aux applications d'accéder aux services de localisation supportés par l'appareil à travers les classes du package "*android.location*". Le composant central du framework de localisation est le service système LocationManager, lequel fournis une API pour déterminer la position et avertir l'appareil.

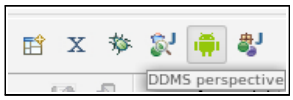
De même que pour les autres services système, on n'instancie pas directement un LocationManager directement. À la place, on requiers une instance de LocationManager au système en faisant appel à "*getSystemService(Context.Location_Service)*". Cette méthode retourne un gestionnaire d'une nouvelle instance de LocationManager.

Une fois que vôtre application à un gestionnaire d'instance de LocationManager, vôtre application sera capable de faire ces trois choses:

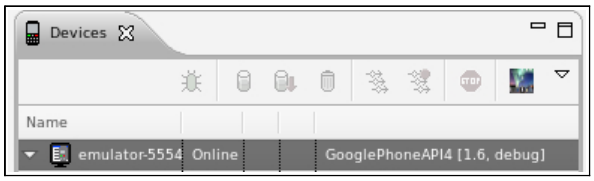
- faire une requête de la liste de tout les LocationProviers connus du LocationManager pour récupérer la dernière localisation;
- activer ou désactiver les mises à jour périodiques concernant la position actuelle à partir d'un LocationProvier;
- activer ou désactiver le démarrage d'un Intent prédéfinis si l'appareil arrive à une certaine distance d'une longitude/latitude donnée.

Néanmoins lors du développement dans l'émulateur, il n'est pas possible d'accéder a de vraies données de GPS, il existe un moyens de donner une fausse position :

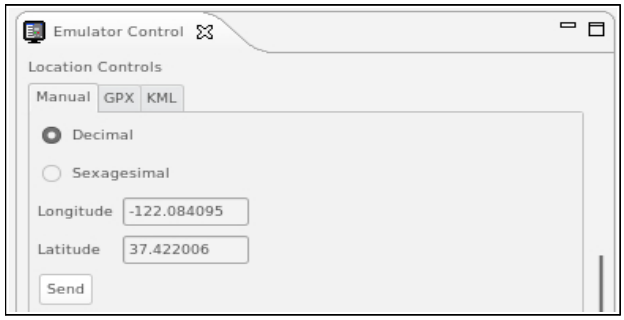
- activez la perspective ddms;



- ensuite, a gauche, il faut sélectionner l'émulateur (démarré) qu'il faut configurer dans la partie "*Devices*";



- il suffit maintenant de configurer les coordonnées fictives dans la partie "*Emulator Control*" et de cliquer sur "*send*";



- l'émulateur retournera désormais ces coordonnées comme position actuelle;

Voici comment faire au niveau programmation pour activer le gestionnaire de position après avoir implémenté la classe "LocationListener" qui nous permettra de gérer les événements générés par l'instance de l'objet LocationManager :

```
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   ...
   Mapiew mapView = (MapView) findViewById(R.id.mapView);
   LocationManager lm = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
   MapLocationOverlay mlo = new MyLocationOverlay(getApplicationContext(), mapView);
   mapView.getOverlays().add(mlo);
   mlo.runOnFirstFix(new Runnable() {
      public void run() {
         mc.animateTo(mlo.getMyLocation());
      }
   });
   mapView.getOverlays().add(mlo);
   mapView.postInvalidate();
}

public void enableLocationListener(){
   locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);
   locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,1l, 5, this);
   mapLocationOverlay.enableMyLocation();
   mapView.postInvalidate();
}

public void disableLocationListener() {
   mapLocationOverlay.disableMyLocation();
   locationManager.removeUpdates(this);
   mapView.postInvalidate();
}

@Override
public void onLocationChanged(Location location) { goTo(location); }
@Override
public void onProviderDisabled(String provider) { ... }
@Override
public void onProviderEnabled(String provider) { ... }
@Override
public void onStatusChanged(String provider, int status, Bundle extras) { ... }
```

"*MapLocationOverlay*" est une classe qui implémente "*ItemizedOverlay*, qui est un élément qu'il est possible d'ajouter sur une carte. Son constructeur prends en argument le contexte et la carte qu'il utilisera "*MyLocationOverlay(getApplicationContext(), mapView);*" il permets, lorsque il est activé en utilisant sa fonction "*enableMyLocation();"* d'afficher un point bleu sur la carte, celui-ci désigne la position actuelle de l'appareil ou configurée dans l'émulateur.

Autour de ce point, on retrouve un cercle bleu plus clair et transparent, celui-ci représente la précision de l'appareil pour la position actuelle donnée.

On remarque également que la fonction "requestLocationUpdates (LocationManager.*GPS_PROVIDER*, 1l, 5, **this**);" du LocationManager permets de définir l'intervalle de temps à laquelle on désire être avertit des mises à jour de la position. Des mises a jour fréquentes sont bonnes pour le suivi mais très pénalisantes au niveau de l'autonomie de l'appareil, il faut ici faire un choix, 5 secondes d'intervalles semblent un bon compromis en général.

Les fonctions "onLocationChanged(Location location)", "onProviderDisabled(String provider)", "onProviderEnabled(String provider)", "onStatusChanged(String provider, int status, Bundle extras)" réagissent elles aussi aux événements du LocationManager, ceci permets savoir si des changements on eu lieu et peut-être d'en avertir l'utilisateur.

11.2. Cartographie

Afin de faciliter le développement d'applications cartographiques, Google fournis une librairie Maps externe qui inclus le package *"com.google.android.maps"*. Les classes de ce package, fournissent des capacités de téléchargement, d'affichage, ainsi qu'une variété d'options d'affichage et de contrôles.

La classe clef dans le package Maps est *"com.google.android.maps.MapView"*, une sous-classe de *"ViewGroup"*. Une *"MapView"* affiche une carte avec des données obtenues à partir du service Google Maps. Lorsque la MapView est au premier plan, elle va capturer les pressions et les gestes afin de se déplacer et de zoomer sur la carte automatiquement, tout en incluant automatiquement les requêtes réseau pour la récupération des dalles de la carte. Il est également possible d'utiliser par programmation les contrôles de la MapView et de dessiner un certain nombre d'icônes par dessus (*"Overlays"*).

En général, la classe MapView fournis un contrôleur autour de l'API Google Maps qui permets de contrôler les données cartes Google à travers des méthodes de classe, et permets de travailler avec celles-ci comme pour tout autre type de vue.

La librairie externe Maps ne fait pas partie de la librairie Android standard, ceci implique quelle ne sera pas présente sur certains appareils Android. De manière identique, la librairie Maps n'est pas inclue dans la librairie standard fournie dans le SDK. Pour rendre possible le développement en utilisant des classes du package com.google.android.maps, cette librairie externe est rendue disponible à travers les add-on de l'API Google pour le SDK Android.

Voici comment gérer une MapActivity :

```
public class MapScreen extends MapActivity implements OnClickListener, LocationListener {

   private MapView mapView;
   private MapController mc;

   @Override
   protected void onCreate(Bundle savedInstanceState) {

      LinearLayout zoomLayout = (LinearLayout)findViewById(R.id.zoom);
      ZoomControls zoomView = (ZoomControls) mapView.getZoomControls();
      zoomLayout.addView(zoomView,
         new LinearLayout.LayoutParams(
         LayoutParams.WRAP_CONTENT,
         LayoutParams.WRAP_CONTENT));
      mapView.displayZoomControls(true);
      mc = mapView.getController();
      mapView.setSatellite(true);
      mapView.setStreetView(true);
   }

   public void goTo(GeoPointer geoPointer) {
      mc.animateTo(
         new GeoPoint(geoPointer.getLatitudeE6(), geoPointer.getLongitudeE6()));
   }

   public void goTo(Location location) {
      if (location != null) {
         Double lat = location.getLatitude()*1E6;
         Double lng = location.getLongitude()*1E6;

         GeoPoint point = new GeoPoint(lat.intValue(), lng.intValue());
         mc.animateTo(point);
      }
   }

   @Override
   protected boolean isRouteDisplayed() {
      return false;
   }
```

Le *"LinearLayout"* utilisé contiendra les boutons de zoom de la carte: les *"ZoomControls"*, il n'y a ici rien à gérer de manière directe, Le *"MapController"*, lui permets d'interagir directement sur le contenu de la MapView fournis par Google tels le centre de la carte *"setCenter(point)"*, les déplacements *"animateTo(point)"*, les zooms *"setZoom(zoomLevel)"*, etc ... C'est néanmoins sur la MapView elle même qu'il est possible de changer l'apparence de celle-ci, il est possible d'utiliser la vue de type carte papier avec *"setSatellite(**false**)"*, vue satellite *"setSatellite(**true**)"* ainsi que Street View. La dernière fonction n'a pas d'utilité sinon renvoyer des informations a Google pour faire un recensement des applications qui affichent une route ou non. Il est "conseillé" de laisser la valeur du return à false.

Le XML :

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent">

   <org.me.ExtendedMapView
      android:id="@+id/mapView"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:enabled="true"
      android:clickable="true"
      android:apiKey="..."/>

   <LinearLayout
      android:id="@+id/zoom"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentBottom="true"
      android:layout_centerHorizontal="true" />
</RelativeLayout>
```

Ici on peut voir qu'il faut placer dans le XML une clef d'API fournie Google pour pouvoir afficher une carte, pour obtenir une de ses fameuses clef il faut :

- se rendre à cette url : http://code.google.com/intl/fr/apis/base/signup.html;
- il faut alors accepter le contrat et de sélectionner le type de produit pour lequel une clef est désirée;
- il ne reste alors que à s'enregistrer.

Pour configurer la MapView, il suffira alors de recopier la clef dans le XML à cet emplacement : *"android:apiKey="..."*". Sans l'ajour d'une clef, une exception est lancée, le fond de la carte sera néanmoins affiché mais non la carte. Remarquez également que je ne fait pas appel à la classe

Google dans le XML mais bien à la classe précédemment implémentée, si nous utilisons la classe de base de Google à la place nous aurions donc com.google.android.maps.MapView. Il n'est néanmoins pas possible d'utiliser la classe de Google dans le XML et d'ensuite faire un cast dessus par programmation, cela poserait problème et lancerait une exception. Voici finalement la classe d'extension de la MapView :

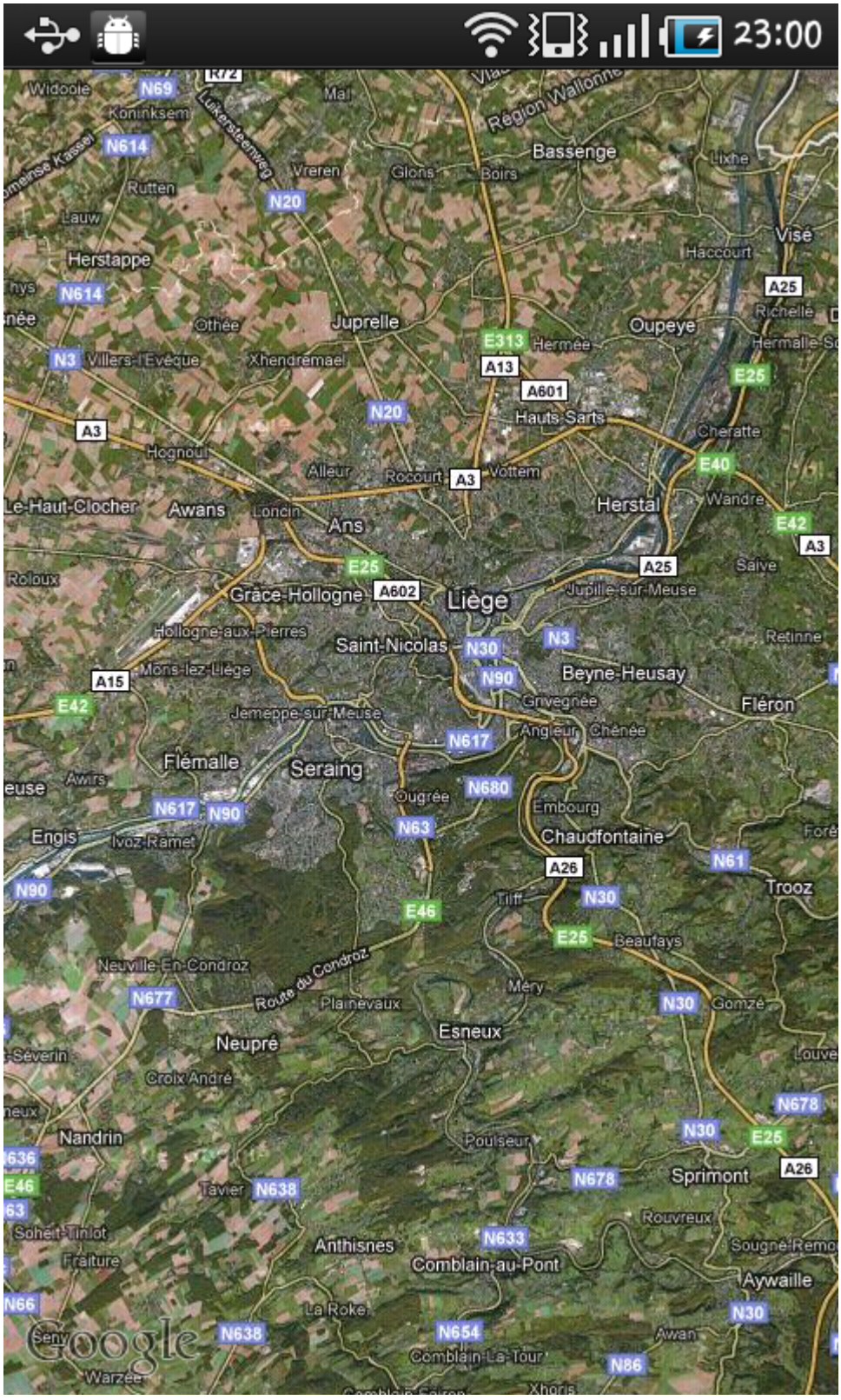
```
public class ExtendedMapView extends MapView {
   private long lastTouchTime = -1;

   public ExtendedMapView(Context context, AttributeSet attrs) {
      super(context, attrs);
   }

   @Override
   protected void onFinishInflate() {
      super.onFinishInflate();
      applyMapViewListener();
   }

   @Override
   public void draw(Canvas canvas) {
      super.draw(canvas);
   }

   protected void applyMapViewListener() {
      // System.out.println("~~~~~~~~~~~~~~~~~ New GestureDetector");
      final GestureDetector gd = new GestureDetector(
            new GestureDetector.SimpleOnGestureListener() {
               @Override
               public boolean onFling(MotionEvent e1, MotionEvent e2,float velocityX, float
velocityY) {
                  return true;
               }

               @Override
               public void onLongPress(MotionEvent e) {
                  super.onLongPress(e);
                  System.out.println("onLongPress()");
               }

               @Override
               public boolean onDoubleTapEvent(MotionEvent e) {
                  getController().zoomInFixing((int) e.getX(),(int) e.getY());
                  return super.onDoubleTapEvent(e);
               }
```

```
            });
        this.setOnTouchListener(new OnTouchListener() {
            @Override
            public boolean onTouch(View v, MotionEvent ev) {
                System.out.println(ev);
                return gd.onTouchEvent(ev);
            }
        });
    }
}
```

On peut voir ici que un nouveau gestionnaire d'évènements au toucher est utilisé, ici, il à plusieurs utilités :

- permettre de gérer le double clic sur la carte pour effectuer un zoom, en effet cette fonctionnalité n'existe pas de base sur la MapView;
- permettre de supprimer l'effet de lancer: lorsque l'on déplace la carte à une certaine vitesse, la carte bouge toujours un certain temps, ceci est la vélocité, que nous supprimons en retournant directement true dans la méthode onFling().
- permettre d'utiliser un événement de presse longue pour par exemple permettre de choisir un lieu.

Voici le résultat lorsque le code est exécuté, on se retrouve avec une carte qui permets de se déplacer et de zoomer en double cliquant sur celle-ci :


23:00
Widooie
N69
Koninksem
N614
Rutten
Lauw
Herstappe
N614
Othée
Juprelle
N3
Villers-l'Evêque
Xhendremael
N20
A3
Hognoul
Alleur
Rocourt
A3
Vottem
Le-Haut-Clocher
Awans
Loncin
Ans
E25
Roloux
Grâce-Hollogne
A602
Liège
Hollogne-aux-Pierres
Saint-Nicolas
N30
N3
A15
Mons-lez-Liège
N90
Beyne-Heusay
E42
Jemeppe-sur-Meuse
Grivegnée
Fléron
N617
Angleur
Chênée
Flémalle
Seraing
Awirs
N680
Ougrée
N617
N90
Embourg
Engis
Ivoz-Ramet
N63
Chaudfontaine
A26
N61
N90
Tilff
N30
Trooz
E46
E25
Beaufays
Neuville-En-Condroz
Route du Condroz
N677
Méry
Plainevaux
N30
Gomzé
Esneux
Neupré
Croix André
Nandrin
N678
N30
E25
Pouleur
Sprimont
A26
E46
Tavier
N638
N678
Rouvreux
Anthisnes
N633
Comblain-au-Pont
Aywaille
N66
La Roke
N30
N638
N654
Awan
Comblain-La-Tour
N86
Warzée
Google

11.3. Geocoding et Geocoding inverse

Si la latitude et la longitude d'un lieu est connue, il est possible de trouver son adresse en utilisant un procédé connu sous le nom de Geocoding. Les Google Maps dans Android supportent cette fonction Via la classe *"Geocoder"*. Le code suivant montre comment il faut s'y prendre pour retrouver l'adresse du lieu touché sur la carte grâce à la méthode getFromLocation() :

```
class MapOverlay extends com.google.android.maps.Overlay
{
    @Override
    public boolean draw(Canvas canvas, MapView mapView, boolean shadow, long when)
    { ... }

    @Override
    public boolean onTouchEvent(MotionEvent event, MapView mapView)
    {
      if (event.getAction() == 1) {                //---when user lifts his finger---
        GeoPoint p = mapView.getProjection().fromPixels((int) event.getX(),(int) event.getY());
        Geocoder geoCoder = new Geocoder( getBaseContext(), Locale.getDefault());
        try {
          List<Address> addresses = geoCoder.getFromLocation(p.getLatitudeE6()  / 1E6,
            p.getLongitudeE6() / 1E6, 1);
          String add = "";
          if (addresses.size() > 0)  {
            for (int i=0; i<addresses.get(0).getMaxAddressLineIndex();
              i++)
              add += addresses.get(0).getAddressLine(i) + "\n";
          }
          Toast.makeText(getBaseContext(), add, Toast.LENGTH_SHORT).show();
        }
        catch (IOException e) {
          e.printStackTrace();
        }
        return true;
      }
      else  return false;
    }
  }
```

Si l'adresse d'un lieu est connue mais que l'on désire connaitre sa latitude et sa longitude, il est possible de le faire en utilisant le Geocoding inverse. De nouveau, ceci est possible grâce à la classe Geocoder. Le code suivant montre comment trouver la localisation exacte de l'empire state building en utilisant la méthode getFromLocationName() :

```
Geocoder geoCoder = new Geocoder(this, Locale.getDefault());
try {
   List<Address> addresses = geoCoder.getFromLocationName("empire state building", 5);
   String add = "";
   if (addresses.size() > 0) {
      p = new GeoPoint(
            (int) (addresses.get(0).getLatitude() * 1E6),
            (int) (addresses.get(0).getLongitude() * 1E6));
      mc.animateTo(p);
      mapView.invalidate();
   }
} catch (IOException e) {
   e.printStackTrace();
}
```

11.4. Affichage d'un tracé et de points sur une carte

Dans certains cas lorsque vous travaillerez avec des cartes vous allez devoir afficher des tracés soit pour représenter un chemin a suivre, le chemin en cours ou toute autre chose. Pour ce faire, rien de plus simple, il suffit de définir une peinture et d'avoir une liste de coordonnées (géographiques ou écran) qu'il suffit alors de relier. Voici comment faire :

```
public class LineMapOverlay extends Overlay {

        private Vector<GeoPoint> geoPoints;
        private Point screenPointa = new Point();
        private Point screenPointb = new Point();
        private Paint pathPaint = new Paint();
        private Projection projection;
        private Path thePath = null;

   public void setLineMapOverlay(Context context, List<GeoPoint> geoPoints, int height, int
width) {
      this.geoPoints = new Vector<GeoPoint>(geoPoints);
                this.pathPaint = new Paint();
                this.pathPaint.setAntiAlias(false);
                this.pathPaint.setStrokeWidth(4);
                this.pathPaint.setARGB(100, 113, 105, 252);
        }

   public void setLineMapOverlay(Context context, Vector<GeoPointer> geoPoints, int height,
int width) {
        this.geoPoints = new Vector<GeoPoint>();
```

```
        for (int i=0; i < geoPoints.size() ;i++) {
                this.geoPoints.add(geoPoints.get(i));
        }
    this.pathPaint = new Paint();
                this.pathPaint.setAntiAlias(false);
                this.pathPaint.setStrokeWidth(4);
                this.pathPaint.setARGB(100, 113, 105, 252);
        }

  @Override
public void draw(Canvas canvas, MapView mapView, boolean shadow) {
    super.draw(canvas, mapView, shadow);
    if (shadow) {
        return;
    }
                thePath = new Path();
                projection = mapView.getProjection();
                projection.toPixels(geoPoints.get(0), screenPointa);
                thePath.moveTo(screenPointa.x,screenPointa.y);
                for (int i=0; i < geoPoints.size() ;i++) {
              projection.toPixels(geoPoints.get(i), screenPointb);
    thePath.lineTo(screenPointb.x, screenPointb.y);
    }
                this.pathPaint.setStyle(Paint.Style.STROKE);
                canvas.drawPath(thePath, pathPaint);
        }
```

La classe LineMapOverlay étends la classe Overlay ce qui va lui permettre d'être facilement affichable sur une carte en utilisant la fonction getOverlays().add(Overlay) de la MapView.

On peut voir que la fonction draw reçoit un boolean qui définit si il s'agit de l'ombre de notre élément ou non, ici on n'en n'affiche pas une on va donc en profiter pour économiser des ressources en quittant directement la fonction. Dans ce cas ci, des coordonnées géographiques étant utilisées elles vont être traduites en coordonnées écran, un nouveau objet Path est alors créé, et nous allons placer son point de départ avec moveTo(X,Y) en coordonnées écran, nous allons ensuite nous déplacer sur chaque point du tracé. Pour finir on fait appel a la fonction drawPath(Path,Paint) du canvas pour dessiner le tracé alors calculé.

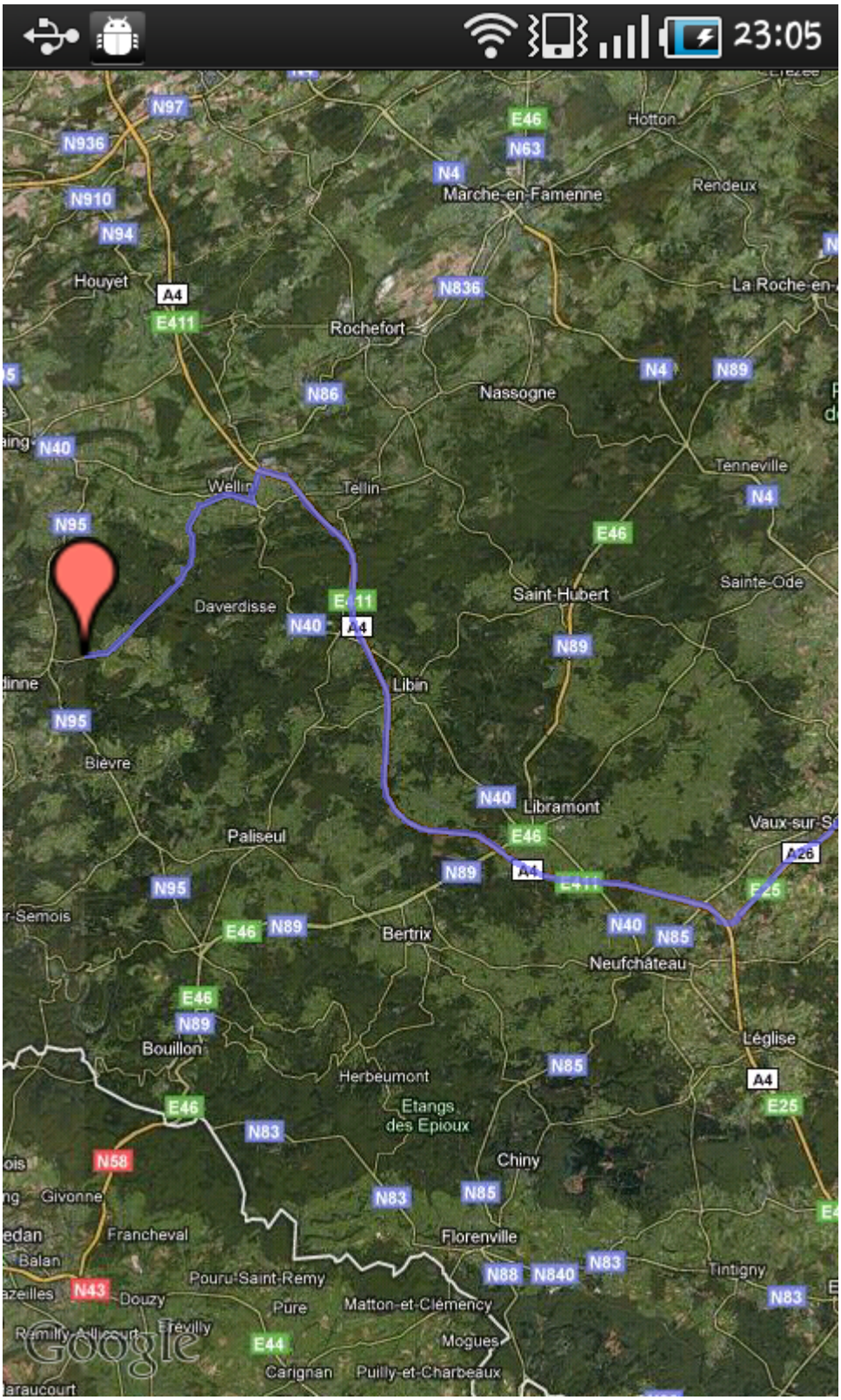
En se qui concerne l'affichage d'une image sur une carte, des méthodes semblables on été mises en place. Dans ce cas ci, il a également été prévu de gérer la position des différents éléments, qu'ils soient sur la droite, centrés ou sur la gauche par rapport aux coordonnées. De cette manière on peut sans problème afficher le type d'icône que l'on désire avec le décalage voulu.
Voici comment cela se présente :

```
public class DrawableMapOverlay extends Overlay {

   public final static int LEFT = -1;
   public final static int CENTER = 0;
   public final static int RIGHT = 1;
   private GeoPointer geoPointer;
   private Context context;
   private int drawable;
   private int type;
   private int height = 0;
   private int width = 0;

   public void setImageMapOverlay(Context context, GeoPointer geoPoint, int drawable, int type,
int height, int width) {
     ....
   }

   @Override
   public boolean draw(Canvas canvas, MapView mapView, boolean shadow, long when) {
      super.draw(canvas, mapView, shadow);
      // Convert geo coordinates to screen pixels
      Point screenPoint = new Point();
      mapView.getProjection().toPixels(new GeoPoint(geoPointer.getLatitudeE6(),
geoPointer.getLongitudeE6()), screenPoint);
      // Read the image
      Bitmap markerImage = BitmapFactory.decodeResource(context.getResources(), drawable);
      // Draw it around the given coordinates
      if (type == LEFT)
         canvas.drawBitmap(markerImage,screenPoint.x - markerImage.getWidth(),screenPoint.y -
markerImage.getHeight(), null);
      else if (type == CENTER)
         canvas.drawBitmap(markerImage,screenPoint.x - markerImage.getWidth()/2,screenPoint.y
- markerImage.getHeight(), null);
      else if (type == RIGHT)
         canvas.drawBitmap(markerImage,screenPoint.x,screenPoint.y - markerImage.getHeight(),
null);
      return true;
   }

   @Override
   public boolean onTap(GeoPoint p, MapView mapView) {
      // Handle tapping on the overlay here
      return true;
   }
}
```

Cette classe DrawableMapOverlay étends elle aussi la classe Overlay pour être facilement affichable sur une MapView. Ici il n'y a que un seul point à calculer: la coordonnée d'origine du dessin de l'image. Il suffit alors d'instancier un objet Bitmap qui représente notre image, pour cela on utilise la classe BitmapFactory qui va nous permettre de créer cet objet à partir d'un fichier ressource.

Viens ensuite le positionnement de l'image en relation aux paramètres reçus. Il suffit alors d'utiliser les dimensions de l'image pour effectuer les calculs. Ici seul trois modes sont possibles mais il est facile d'imaginer de permettre le positionnement dans un cadre de neuf possibilités autour du point.

Voici le résultat d'un LineMapOverlay sur carte Google en utilisant des DrawableMapOverlay pour définir le point de départ et le point d'arrivée :


23:05
N97
N936
N910
N94
E46
N63
N4
Marche-en-Famenne
Hotton
Rendeux
Houyet
A4
E411
N836
La Roche-en-
Rochefort
N4
N89
N86
Nassogne
N40
Tenneville
N4
N95
Wellin
Tellin
E46
Saint-Hubert
Sainte-Ode
Daverdisse
N40
A4
N89
Libin
N95
Bièvre
N40
Libramont
Paliseul
E46
A26
N89
A4
N95
E25
E46
N89
Bertrix
N40
N85
Neufchâteau
E46
N89
Bouillon
Herbeumont
N85
Léglise
A4
E25
E46
N83
Etangs
des Epioux
N58
Chiny
Givonne
N83
N85
Francheval
Florenville
Balan
N88
N840
N83
Tintigny
N43
Douzy
Pouru-Saint-Remy
Pure
Matton-et-Clémency
N83
Mogues
E44
Carignan
Puilly-et-Charbeaux
Google

11.5. Affichage d'une info bulle

Si il y a bien une chose qui manque à l'api fournie par Google, c'est la possibilité de créer une info bulle pouvant contenir des informations, tel ce qui est fait sur l'application Maps en donnant l'adresse du lieu sélectionné.

C'est un élément qui peut paraitre simple mais qui est loin de l'être, en effet il faut prendre en considération le nombre de lignes qui seront affichés, la taille de la police la taille des lignes etc ... Pour avoir un élément qui s'intègre parfaitement quel que soit la situation, il va donc falloir faire des calculs en tenant compte de tout ces points et encore de différentes autres choses telles que des conversions de taille entre DIP et Pixel de telle façon à ce que les bulles aient la même taille quelque soit la taille de l'écran utilisé ainsi que de sa résolution et ce pour des raisons de lisibilité.

Voici donc à quoi peut ressembler le code d'une info bulle (il existe surement de nombreuses autres méthodes) (chaque partie sera expliquée séparément) :

```
public class BubbleOverlay extends Overlay {

	private GeoPoint selectedMapLocation;
	private Paint innerPaint;
	private Paint borderPaint;
	private Paint textPaint;
	private Paint mainTextPaint;
	private List<String> mAddress;
	private Context mContext;
	private float scale;
	private int mainFontSize;
	private int altFontSize;

	private int INFO_POINTER_WIDTH = 20;
	private int INFO_POINTER_HEIGHT = 20;
	private int INFO_WINDOW_WIDTH = 30;
	private int INFO_WINDOW_HEIGHT = 25;
	private int TEXT_OFFSET_X = 10;
	private int TEXT_OFFSET_Y = 5;

	public BubbleOverlay(List<String> address, GeoPoint location, Context context) {
		mAddress = address;
		selectedMapLocation = location;
		mContext = context;
		scale = mContext.getResources().getDisplayMetrics().density;
		mainFontSize = (int)(scale * 15 + 0.5f);
		altFontSize = mainFontSize -5;
		generateWindowDimensions();
	}
```

Ici il s'agit principalement de configurer les variables, les élément importants sont les variables qui définissent la taille par défaut de la bulle et de sa pointe : INFO_POINTER et INFO_WINDOW. Il y a aussi scale qui représente la densité de l'écran et qui va permettre de faciliter certains calculs tels que la taille du texte pour un certains nombre de DPI (ici 15) : (**int**)(scale * 15 + 0.5f).

Nous allons maintenant calculer la taille optimale pour la bulle en prenant en compte le nombre de lignes et la longueur de celles-ci :

```
	private void generateWindowDimensions() {
		if (mAddress.size() > 0) {
			Rect rect = new Rect();
			getMainTextPaint().getTextBounds(mAddress.get(0),0, mAddress.get(0
).length(),rect);
			INFO_WINDOW_WIDTH = rect.width();
			INFO_WINDOW_HEIGHT = mainFontSize;
			for (int i=1; i < mAddress.size() ;i++) {
				getTextPaint().getTextBounds(mAddress.get(i),0, mAddress.get(i
).length(),rect);
				if (INFO_WINDOW_WIDTH < rect.width()) {
					INFO_WINDOW_WIDTH = rect.width();
				}
				INFO_WINDOW_HEIGHT += altFontSize;
			}
			INFO_WINDOW_WIDTH += TEXT_OFFSET_X * 2;
			INFO_WINDOW_HEIGHT += TEXT_OFFSET_Y * mAddress.size();
			INFO_WINDOW_HEIGHT += TEXT_OFFSET_Y * 2;
		}
	}
```

La partie importante ici est la fonction getTextBounds() de la classe Paint qui permets d'obtenir la taille qu'a le texte passé en paramètre à l'intérieur d'une instance de la classe Rect.

De telle manière il est facilement possible de savoir quelle est la ligne la plus grande a afficher et utiliser cette taille comme longueur maximale. Il en va de même pour la hauteur. Pour finnir on ajoute les offsets de manière à ce que le texte ne soit pas collé à la bulle mais soit à l'intérieur.

Après cela, il suffit de dessiner la bulle ainsi que sa pointe :

```
	@Override
	public void draw(Canvas canvas, MapView mapView, boolean shadow) {
		super.draw(canvas, mapView, shadow);
		if (shadow == false) { // bored to draw a shadow if you do it send me the code ;)
			//  First determine the screen coordinates of the selected MapLocation
			Point selDestinationOffset = new Point();
			mapView.getProjection().toPixels(selectedMapLocation,
```

```
selDestinationOffset);

                    // Setup the info window position pointer
                    RectF infoPointerRect = new
RectF(0,0,INFO_POINTER_WIDTH,INFO_POINTER_HEIGHT);
                    int infoPointerOffsetX = selDestinationOffset.x-
INFO_POINTER_WIDTH/2;
                    int infoPointerOffsetY = selDestinationOffset.y-INFO_POINTER_HEIGHT
- 10;
                    infoPointerRect.offset(infoPointerOffsetX,infoPointerOffsetY);

                    //  Setup the info window with the right size & location
                    RectF infoWindowRect = new
RectF(0,0,INFO_WINDOW_WIDTH,INFO_WINDOW_HEIGHT);
                    int infoWindowOffsetX = selDestinationOffset.x-
INFO_WINDOW_WIDTH/2;
                    int infoWindowOffsetY = selDestinationOffset.y-
INFO_WINDOW_HEIGHT - 30;
                    infoWindowRect.offset(infoWindowOffsetX,infoWindowOffsetY);

                    // Draw Pointer Border
                    Path path = new Path();
                    path.moveTo(infoPointerOffsetX,infoPointerOffsetY);
                    float border[] = {infoPointerOffsetX+10,infoPointerOffsetY+20,
                                      infoPointerOffsetX+20,infoPointerOffsetY,
                                      infoPointerOffsetX,infoPointerOffsetY};

                    for (int i=0; i < border.length ;i+=2)
                            path.lineTo(border[i], border[i+1]);
                    canvas.drawPath(path, getBorderPaint());

                    //  Draw inner info window
                    canvas.drawRoundRect(infoWindowRect, 5, 5, getInnerPaint());

                    //  Draw border for info window
                    canvas.drawRoundRect(infoWindowRect, 5, 5, getBorderPaint());

                    // Draw Pointer inner
                    path = new Path();
                    infoPointerOffsetY = infoPointerOffsetY - 1;
                    path.moveTo(infoPointerOffsetX,infoPointerOffsetY);
                    float inner[] = {infoPointerOffsetX+10,infoPointerOffsetY+20,
                                      infoPointerOffsetX+20,infoPointerOffsetY,
                                      infoPointerOffsetX,infoPointerOffsetY};
```

```
                        for (int i=0; i < border.length ;i+=2)
                                path.lineTo(inner[i], inner[i+1]);
                        canvas.drawPath(path, getInnerPaint());

                        if (mAddress.size() > 0) {
                                int height = TEXT_OFFSET_Y + mainFontSize;
                                canvas.drawText(mAddress.get(0
),infoWindowOffsetX+TEXT_OFFSET_X,infoWindowOffsetY+height,getMainTextPaint());
                                for (int i=1; i < mAddress.size() ;i++) {
                                        height += TEXT_OFFSET_Y + altFontSize;
                                        canvas.drawText(mAddress.get(i
),infoWindowOffsetX+TEXT_OFFSET_X,infoWindowOffsetY+height,getTextPaint());
                                }
                        }
                }
        }
```

On peut voir que le dessin des différents éléments de la bulle se font dans un ordre particulier: les bords de la pointe les bords de la bulle, l'intérieur de la bulle et l'intérieur de la pointe, de cette manière, la bulle est homogène c-a-d que les bords de la pointe ne dépassent pas dans la bulle et que la bordure de la bulle au niveau de la pointe n'est pas visible.

Pour définir l'endroit ou dessiner le texte la même méthode est utilisée afin de prendre en compte la taille de chaque police ainsi que de prendre en compte chaque offset.

Maintenant, tout ce qui concerne les couleurs, c'est en effet à travers celles-ci que la taille et l'apparence du texte sera définie de même que l'apparence de la bulle :

```
        private Paint getInnerPaint() {
                if ( innerPaint == null) {
                        innerPaint = new Paint();
                        innerPaint.setARGB(255, 231, 235, 231); //gray
                        innerPaint.setAntiAlias(true);
                }
                return innerPaint;
        }

        private Paint getBorderPaint() {
                if ( borderPaint == null) {
                        borderPaint = new Paint();
                        borderPaint.setARGB(255, 0, 0, 0);
                        borderPaint.setAntiAlias(true);
                        borderPaint.setStyle(Style.STROKE);
                        borderPaint.setStrokeWidth(2);
                }
                return borderPaint;
```

```
	}

	private Paint getMainTextPaint() {
		if ( mainTextPaint == null) {
			mainTextPaint = new Paint();
			mainTextPaint.setARGB(255, 0, 0, 0);
			mainTextPaint.setAntiAlias(true);
			mainTextPaint.setTextSize(mainFontSize);
			mainTextPaint.setFakeBoldText(true);
		}
		return mainTextPaint;
	}

	private Paint getTextPaint() {
		if ( textPaint == null) {
			textPaint = new Paint();
			textPaint.setARGB(255, 0, 0, 0);
			textPaint.setAntiAlias(true);
			textPaint.setTextSize(altFontSize);
		}
		return textPaint;
	}
```

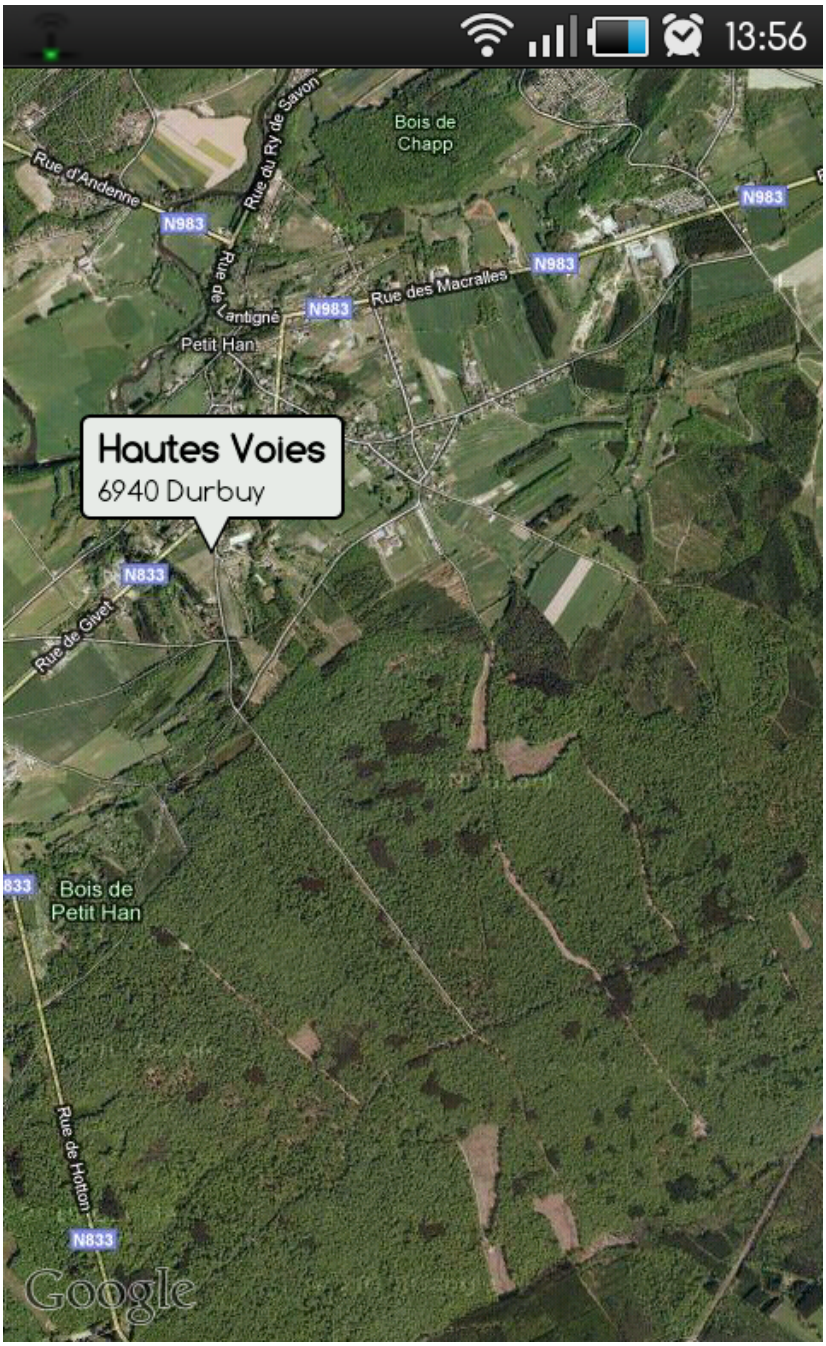
Maintenant, il ne reste plus qu'a gérer l'action a effectuer lorsque l'on tape sur la bulle, enfin presque, il subsiste néanmoins un problème, la fonction onTap() récupère l'événement qu'il aie lieu n'importe ou sur l'écran et pas seulement sur la bulle. Il va alors falloir définir un cadre dans lequel se trouve la bulle et qui sert à délimiter le lieu couvert par le onTap() :

```
	@Override
	public boolean onTap(GeoPoint p, MapView mapView) {
		super.onTap(p, mapView);
		if (isTapOnElement(p, mapView)) {
			String[] array = new String[mAddress.size()];
			for (int i=0; i < mAddress.size() ;i++) {
				array[i] = mAddress.get(i);
			}
			int[] coords =
{selectedMapLocation.getLatitudeE6(),selectedMapLocation.getLongitudeE6()};
			mContext.startActivity(new Intent(mContext, BubbleInteractionScreen.class
)
				.putExtra("address",array).putExtra("coords",coords));
		}
		return false;
	}
```

```
        public boolean isTapOnElement(GeoPoint p, MapView mapView) {
                Point screenCoords = new Point();
                mapView.getProjection().toPixels(selectedMapLocation, screenCoords);
                RectF hitTestRecr = new RectF();
                hitTestRecr.set(-INFO_WINDOW_WIDTH/2,-
(INFO_WINDOW_HEIGHT+INFO_POINTER_HEIGHT),INFO_WINDOW_WIDTH/2,0);
                hitTestRecr.offset(screenCoords.x,screenCoords.y);
                mapView.getProjection().toPixels(p, screenCoords);
                if (hitTestRecr.contains(screenCoords.x,screenCoords.y)) {
                        return true;
                }
                return false;
        }
}
```

La fonction onTap() n'est pas très importante ici, c'est la fonction isTapOnElement() qui est importante c'est elle qui vérifie que les coordonnées du point tapé sont bien dans la limite de la bulle et ce en utilisant la fonction contains() de la classe RectF. Pour cela on crée un objet RectF à la bonne taille et on lui donne le même offset que la bulle.

Voici le résultat d'une info bulle :


13:56
Bois de
Chapp
Rue du Ry de Savon
Rue d'Andenne
N983
Rue des Macralles
Rue de Lantigné
Petit Han
Hautes Voies
6940 Durbuy
N833
Rue de Givet
Bois de
Petit Han
Rue de Horion
Google

12. Stockage des données

Un système d'exploitation typique fournis un système de fichiers commun que toute application peut utiliser pour stocker des fichiers qui peuvent être lus par d'autres applications. Android, quand à lui, utilise un système différent: sur Android, toutes les données (ceci inclus les fichiers) sont privées à cette application.

Néanmoins, Android fournis un moyens pour que des applications puissent exposer leur données privées aux autres applications et ce à travers des fournisseurs de contenu. Un fournisseur de contenu est un composant optionnel d'une application qui donne accès en lecture/écriture aux données d'une application. Ces fournisseurs de contenu implémentent une syntaxe standard pour les requêtes et les modifications sur les données, ainsi que un mécanisme standard pour lire les données retournées. Android fournis un certain nombre de fournisseurs de contenu pour les types de données standard tels les fichiers images, vidéos, audio et les informations sur les contacts personnels.

Lorsque l'on désire ou non exporter des données, il faut un moyen pour les stocker, pour cela, Android fournit les quatre méthodes suivantes : Préférences, Fichiers, Bases de Données et le Réseau.

12.1. Préférences

Les préférences sont un système léger pour stocker et récupérer par clef des paires de type de données primitives, cela ressemble for à l'utilisation d'une HashTable/Map. Elles sont typiquement utilisées pour stocker les préférences d'une application tel le type de police ou de couleur désirée lorsque l'application démarre. Il suffit d'appeller Context.getSharedPreferences() pour lire et écrire ces valeurs. Il faut assigner un nom au groupe de préférences si l'on désire les partager avec d'autres composants dans la même application, ou utiliser Activity.getPreferences() sans nom pour les garder privées à l'activité appelante. Il n'est pas possible de partager des préfrences entre les appllications, à l'exception de l'utilisation d'un content provider.

Voici comment gérer les préférences :

```
public class Calc extends Activity {
public static final String PREFS_NAME = "MyPrefsFile";

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

    // Restore preferences
    SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
    boolean silent = settings.getBoolean("silentMode", false);
    setSilent(silent);
  }
```

```
    @Override
    protected void onStop(){
      super.onStop();
     // Save user preferences. We need an Editor object to
     // make changes. All objects are from android.context.Context
     SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
     SharedPreferences.Editor editor = settings.edit();
     editor.putBoolean("silentMode", mSilentMode);
     // Don't forget to commit your edits!!!
     editor.commit();
    }
}
```

Tip : Il ne faut pas oublier que toute Activity est un Intent et un Context; donc malgré que certaines de ces méthodes sont prévues pour l'un ou l'autre précisément, elles restent néanmoins toutes accessibles

12.2. Fichiers

Il est possible de stocker des fichiers directement sur l'appareil mobile ou sur un média externe tel une carte SD. Par défaut les autres applications n'ont pas accès a ces fichiers.

Pour lire des données à partir d'un fichier, il faut appeler la méthode *"Context.openFileInput()"* et lui passer le chemin et le nom du fichier. Il retourne un objet FileInputStrem java standard. Pour écrire, il suffira d'appeller *"Context.openFileOutput()"* avec le chemin et le nom. Il retournera un FileOutputStream. Faire appel à ses fonctions à partir d'une autre application ne fonctionnera pas; il est seulement possible d'accéder aux fichiers locaux.

Si il y a un fichier statique tel : *"myfile"* pour packager avec l'application, à la compilation, il suffit de le sauver dans le répertoire *"res/raw/"* du projet, et ensuite de l'ouvrir avec *"Resources.openRawRessource(R.raw.myfile)"*. Il retournera un objet InputStream utilisable pour lire à partir du fichier.

Voici un exemple qui montre comment accéder a des fichiers sur la carte SD ce qui sera donc le plus courant si l'on désire travailler sur des fichiers fournis par l'utilisateur et/ou créer des fichiers destinés à celui-ci :

```
    /**
     * getting the root folder of the external storage (SD card)
     */
    File root = Environment.getExternalStorageDirectory();

    try {
       root = new File(root.getAbsolutePath() + File.separator + "myfolder");
       root.mkdir();
    } catch (Exception e) {
```

```
        /**
         * displaying popup message
         */
        Toast toast = Toast.makeText(context, "Unable To Access SD-Card",
        Toast.LENGTH_LONG);
        toast.setDuration(5000);
        toast.show();
        return null;
    }
```

La fonction *"root.mkdir()"* doit être remplacée par *"root.mkdirs()"* si le chemin contiens plus de un répertoire qui peut ne pas exister.

12.3. Bases de Données

L'API Android supporte la création et l'utilisation de bases de données en utilisant SQLite. Chaque base de données est privée en rapport à l'application qui l'a créée. L'objet *"SQLiteDatabase"* représente une base de données et possède les méthodes pour interagir avec elle en faisant des requêtes et en gérant les données. Pour créer la base de données, il suffit d'appeler la méthode *"SQLiteDatabase.create()"*, et également créer une sous-classe de *"SQLiteOpenHelper"*.

Android est fournis avec l'outil de base de données sqlite3, lequel permets de naviguer dans le contenu des tables, d'exécuter des commandes sql et d'effectuer d'autres actions utiles sur les bases de données SQLite. Toutes les bases de données, SQLite et autres sont stockées sur l'appareil dans le répertoire *"/data/data/package_name/databases."*.

Voici un exemple qui montre comment utiliser ses classes et méthodes :

```
public class NotesDbAdapter {

    public static final String KEY_TITLE = "title";
    public static final String KEY_BODY = "body";
    public static final String KEY_ROWID = "_id";

    private static final String TAG = "NotesDbAdapter";
    private DatabaseHelper mDbHelper;
    private SQLiteDatabase mDb;

    /**
     * Database creation sql statement
     */
    private static final String DATABASE_CREATE =
            "create table notes (_id integer primary key autoincrement, "
                    + "title text not null, body text not null);";
```

```
private static final String DATABASE_NAME = "data";
private static final String DATABASE_TABLE = "notes";
private static final int DATABASE_VERSION = 2;

private final Context mCtx;

private static class DatabaseHelper extends SQLiteOpenHelper {

    DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {

        db.execSQL(DATABASE_CREATE);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        Log.w(TAG, "Upgrading database from version " + oldVersion + " to "
                + newVersion + ", which will destroy all old data");
        db.execSQL("DROP TABLE IF EXISTS notes");
        onCreate(db);
    }
}

/**
 * Constructor - takes the context to allow the database to be
 * opened/created
 *
 * @param ctx the Context within which to work
 */
public NotesDbAdapter(Context ctx) {
    this.mCtx = ctx;
}

/**
 * Open the notes database. If it cannot be opened, try to create a new
 * instance of the database. If it cannot be created, throw an exception to
 * signal the failure
 *
 * @return this (self reference, allowing this to be chained in an
 *         initialization call)
 * @throws SQLException if the database could be neither opened or created
```

```
     */
    public NotesDbAdapter open() throws SQLException {
        mDbHelper = new DatabaseHelper(mCtx);
        mDb = mDbHelper.getWritableDatabase();
        return this;
    }

    public void close() {
        mDbHelper.close();
    }


    /**
     * Create a new note using the title and body provided. If the note is
     * successfully created return the new rowId for that note, otherwise return
     * a -1 to indicate failure.
     *
     * @param title the title of the note
     * @param body the body of the note
     * @return rowId or -1 if failed
     */
    public long createNote(String title, String body) {
        ContentValues initialValues = new ContentValues();
        initialValues.put(KEY_TITLE, title);
        initialValues.put(KEY_BODY, body);

        return mDb.insert(DATABASE_TABLE, null, initialValues);
    }

    /**
     * Delete the note with the given rowId
     *
     * @param rowId id of note to delete
     * @return true if deleted, false otherwise
     */
    public boolean deleteNote(long rowId) {

        return mDb.delete(DATABASE_TABLE, KEY_ROWID + "=" + rowId, null) > 0;
    }

    /**
     * Return a Cursor over the list of all notes in the database
     *
     * @return Cursor over all notes
     */
```

```
    public Cursor fetchAllNotes() {

        return mDb.query(DATABASE_TABLE, new String[] {KEY_ROWID, KEY_TITLE,
                KEY_BODY}, null, null, null, null, null);
    }

    /**
     * Return a Cursor positioned at the note that matches the given rowId
     *
     * @param rowId id of note to retrieve
     * @return Cursor positioned to matching note, if found
     * @throws SQLException if note could not be found/retrieved
     */
    public Cursor fetchNote(long rowId) throws SQLException {

        Cursor mCursor =

                mDb.query(true, DATABASE_TABLE, new String[] {KEY_ROWID,
                        KEY_TITLE, KEY_BODY}, KEY_ROWID + "=" + rowId, null,
                        null, null, null, null);
        if (mCursor != null) {
            mCursor.moveToFirst();
        }
        return mCursor;

    }

    /**
     * Update the note using the details provided. The note to be updated is
     * specified using the rowId, and it is altered to use the title and body
     * values passed in
     *
     * @param rowId id of note to update
     * @param title value to set note title to
     * @param body value to set note body to
     * @return true if the note was successfully updated, false otherwise
     */
    public boolean updateNote(long rowId, String title, String body) {
        ContentValues args = new ContentValues();
        args.put(KEY_TITLE, title);
        args.put(KEY_BODY, body);
        return mDb.update(DATABASE_TABLE, args, KEY_ROWID + "=" + rowId, null) > 0;
    }
}
```

13. Réseau

Le réseau est la partie la plus facile sur Android, en effet c'est un système désigné pour les appareils mobiles donc ça semble logique. La programmation réseau est donc 100% identique a de la programmation réseau en Java Desktop, il est donc possible d'utiliser les classes J2SE, il existe également les classes natives Android (qui ressemblent fort aux classes J2ME cela dis), et qui utilisent également des possibilité de sérialisation sur des médias tels le Bluetooth.

14. Listes déroulantes

Les ListView (liste déroulantes) sont un des widget les plus utilisés dans le système Android. De base, elles sont assez simple a utiliser, elles sont également très flexibles et puissantes. Elles ne sont malheureusement pas évidentes à être utilisées de manière correcte assez rapidement.

Si nous voulons simplement afficher du texte dans une ListView, rien de plus facile il suffit de procéder comme suit :

```
setListAdapter(new ArrayAdapter<String>(context, android.R.layout.simple_list_item_1,
STRINGARRAY));
```

android.R.layout.simple_list_item_1 représente le type de liste, ici se sera une liste par défaut du système d'une taille normale, android.R.layout.simple_list_item_2 représente le même type de liste simple à la différence que elle sera moins haute et le texte sera plus petit.

Mais assez parlé des choses simples et basiques, d'ailleurs la plupart du temps les listes simples ne seront pas utilisées. Si l'on veut une liste un peut plus avancée, comprendre : avoir une liste personnalisée telle une liste avec des images ou des boutons. Dans ce cas, il va falloir recréer un modèle complet de liste.

Ceci comporte trois étapes :

- création d'une vue en XML représentant l'élément désiré dans la liste;
- création d'une classe qui représentera les éléments de la vue.
- création d'une classe qui étends *"BaseAdapter"* et qui va nous permettre de remplir notre liste.

Dans ce cas ci, nous allons créer une ListView dont le premier élément contiens des boutons et est non sélectionnable et dont le reste permets de contenir deux TextView, une normale et une qui sert de description. La première des deux verra sa position changer indépendamment de l'existence de la deuxième. Voici les classes représentant les différentes vues :

```
        private class BubbleInfoListHeaderAdapter {
                public TextView pLocation;
                public TextView pDescription;
                public ImageButton pMapButton;
                public ImageButton pDirectionButton;
        }

        private class BubbleInfoListBodyAdapter {
      public String[] pInfoTextNew =
mContext.getResources().getStringArray(R.array.entries_list_bubble_interactions_body_info_new
);
      public String[] pInfoText =
```

```
mContext.getResources().getStringArray(R.array.entries_list_bubble_interactions_body_info);
        public String[] pDescrTextNew =
mContext.getResources().getStringArray(R.array.entries_list_bubble_interactions_body_descr_ne
w);
        public String[] pDescrText =
mContext.getResources().getStringArray(R.array.entries_list_bubble_interactions_body_descr);
                    public TextView pInfoTextView;
                    public TextView pDescrTextView;
            }
```

La première classe contiens donc deux TextView ainsi que deux ImageButton, elle représente la première ligne de la liste. La deuxième classe, elle représente le reste de la liste, elle va chercher ses valeurs dans le fichier "array.xml".

Voici a quoi ressemble le contenu du fichier XML représentant le haut de la ListView :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:background="#ffffff"
   android:orientation="vertical">
   <TableRow
      android:layout_width="fill_parent"
      android:layout_height="wrap_content">
      <TextView
        android:id="@+id/bubble_interactions_header_location"
        android:paddingLeft="5dip"
        android:paddingTop="5dip"
        android:textSize="20dip"
        android:textStyle="bold"
        android:textColor="#000000" />
   </TableRow>
   <TableRow
      android:layout_width="fill_parent"
      android:layout_height="wrap_content">
      <TextView
        android:id="@+id/bubble_interactions_header_descr"
        android:paddingLeft="5dip"
        android:paddingTop="5dip"
        android:textColor="#000000" />
   </TableRow>
   <TableLayout
      android:stretchColumns="0,1,2,3"
      android:layout_width="fill_parent"
```

```
        android:layout_height="fill_parent"
        android:layout_marginLeft="5dip"
        android:layout_marginRight="5dip"
        android:layout_marginBottom="5dip"
        android:background="#ffffff">
        <TableRow
          android:layout_width="fill_parent"
          android:layout_height="wrap_content"
          android:paddingTop="10dip">
          <ImageButton
            android:id="@+id/bubble_interactions_header_map"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:paddingLeft="5dip"
            android:scaleType="center"
            android:src="@android:drawable/ic_menu_mapmode" />
          <ImageButton
            android:id="@+id/bubble_interactions_header_directions"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:paddingLeft="5dip"
            android:scaleType="center"
            android:src="@android:drawable/ic_menu_directions" />
          <ImageButton
            android:visibility="invisible"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:paddingLeft="5dip"
            android:scaleType="center"
            android:src="@android:drawable/ic_menu_directions" />
          <ImageButton
            android:visibility="invisible"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:paddingLeft="5dip"
            android:scaleType="center"
            android:src="@android:drawable/ic_menu_directions" />
        </TableRow>
      </TableLayout>
    </LinearLayout>
```

Cette première ligne de la ListView sera un *"LinearLayout"* qui va contenir deux *"TextView"* une au dessus de l'autre. Ensuite elle va contenir quatre boutons sur la même *"TableRow"*, ceux ci auront chacun la même taille, la propriété *android:stretchColumns="0,1,2,3"* ayant été appliquée au *"TableLayout"* qui les contiens.

En se qui concerne les autres ligne c'est beaucoup plus simple vu qu'elles ne contiennent que deux TextView.

Voici a quoi ressemble le contenu du fichier XML représentant le haut de la ListView :

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/bubble_interactions_body_layout"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:paddingLeft="10dip"
   android:paddingRight="10dip" >
   <TextView
      android:id="@+id/bubble_interactions_body_heightwrapper"
      android:layout_height="65dip"
      android:layout_width="wrap_content" />
   <TextView
      android:id="@+id/bubble_interactions_body_info"
      android:textSize="20dip"
      android:textColor="#000000"
      android:textStyle="bold"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:layout_toRightOf="@+id/bubble_interactions_body_heightwrapper"
      android:layout_alignWithParentIfMissing="true"
      android:gravity="center_vertical"
      android:layout_alignParentRight="true"
      android:layout_marginTop="10dip" />
   <TextView
      android:id="@+id/bubble_interactions_body_descr"
      android:layout_toRightOf="@+id/bubble_interactions_body_heightwrapper"
      android:layout_below="@+id/bubble_interactions_body_info"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:textColor="#000000"
      android:layout_alignParentRight="true"
      android:ellipsize="marquee"
      android:lines="2" />
</RelativeLayout>
```

Ici, il peut paraitre étrange d'utiliser un *"RelativeLayout"*, néanmoins, cela à pour but de permettre de centrer les deux *"TextView"* indépendamment et également de centrer la première des deux au milieu du layout lorsque la deuxièmement est manquante afin d'avoir une interface consistante.

Nous allons maintenant créer une classe BubbleInfoListAdapter qui étends BaseAdapter, celle ci permets de gérer la manière dont sera rendu chaque composant interne à la ListView. Elle va nous permettre d'instancier chaque élément de manière indépendante Voici comment elle se présente :

```
public class BubbleInfoListAdapter extends BaseAdapter {

	private Context mContext;
	private BubbleInfoListHeaderAdapter mHeaderAdapter;
	private BubbleInfoListBodyAdapter mBodyAdapter;
	private LayoutInflater mInflater;
	private boolean mExists;
	private String[] mAddress;

	public BubbleInfoListAdapter(Context context, boolean exists) {
		mContext = context;
		mExists = exists;
		mInflater = LayoutInflater.from(mContext);
		mAddress = ((Activity)mContext).getIntent().getStringArrayExtra("address");
	}

	@Override
	public int getCount() {
		return 3;
	}

	@Override
	public Object getItem(int arg0) {
		return arg0;
	}

	@Override
	public long getItemId(int arg0) {
		return arg0;
	}

	@Override
	public View getView(int position, View convertView, ViewGroup parent) {
		if (position == 0) {
			if (convertView == null) {
				convertView = mInflater.inflate(R.layout.bubbleinfolistheader, null);
				mHeaderAdapter = new BubbleInfoListHeaderAdapter();
				mHeaderAdapter.pLocation = (TextView)
convertView.findViewById(R.id.bubble_interactions_header_location);
				mHeaderAdapter.pDescription = (TextView)
convertView.findViewById(R.id.bubble_interactions_header_descr);
```

```
                mHeaderAdapter.pMapButton = (ImageButton)
convertView.findViewById(R.id.bubble_interactions_header_map);
                mHeaderAdapter.pDirectionButton = (ImageButton)
convertView.findViewById(R.id.bubble_interactions_header_directions);
                        }

                        mHeaderAdapter.pLocation.setText(mAddress[0]);
                        if (mAddress.length > 1)
                                mHeaderAdapter.pDescription.setText(mAddress[1]);
        mHeaderAdapter.pMapButton.setOnClickListener((OnClickListener) mContext);
        mHeaderAdapter.pDirectionButton.setOnClickListener((OnClickListener) mContext);

        convertView.setId(position);
        convertView.setTag(mHeaderAdapter);
                }
                else {
        if (convertView == null) {
                convertView = mInflater.inflate(R.layout.bubbleinfolistbody, null);
           mBodyAdapter = new BubbleInfoListBodyAdapter();
           mBodyAdapter.pInfoTextView = (TextView)
convertView.findViewById(R.id.bubble_interactions_body_info);
           mBodyAdapter.pDescrTextView = (TextView)
convertView.findViewById(R.id.bubble_interactions_body_descr);
        }

        if (mExists) {
                mBodyAdapter.pInfoTextView.setText(mBodyAdapter.pInfoText[position-1]);
                mBodyAdapter.pDescrTextView.setText(mBodyAdapter.pDescrText[position-1]);
        }
        else {
                mBodyAdapter.pInfoTextView.setText(mBodyAdapter.pInfoTextNew[position-1]);
                mBodyAdapter.pDescrTextView.setText(mBodyAdapter.pDescrTextNew[position-
1]);
        }

        convertView.setId(position);
        convertView.setTag(mBodyAdapter);
     }
                return convertView;
        }
```

C'est ici dans la fonction getView() que nous allons générer l'élément qui sera rendu dans chaque ligne de la ListView. En tant que paramètres, on reçoit la position dans la liste de la vue à rendre, la vue existante à cette position (ou null si elle n'existe pas) et la vue parente. Si la vue existante est nulle, on va alors la créer sur base du fichier XML que l'on désire rendre à cette position, une

instance de la classe correspondant à cette vue précise sera créée après avoir généré un objet correspondant au XML en utilisant la fonction inflate() de la classe LayoutInflater. Après avoir configuré notre classe correspondante avec le layout récemment utilisé pour générer la convertView, on donnera alors l'instance notre classe en tant que tag à celle ci de même qu'une ID qui est le nulero de la ligne ici.

Si la vue existe il n'est plus nécessaire de recréer une instance de la classe correspondante mais il est néanmoins nécessaire de réinitialiser les valeurs qu'elle est censée contenir, la liste pouvant recycler une autre vue qui contiendrait les mauvaises données.

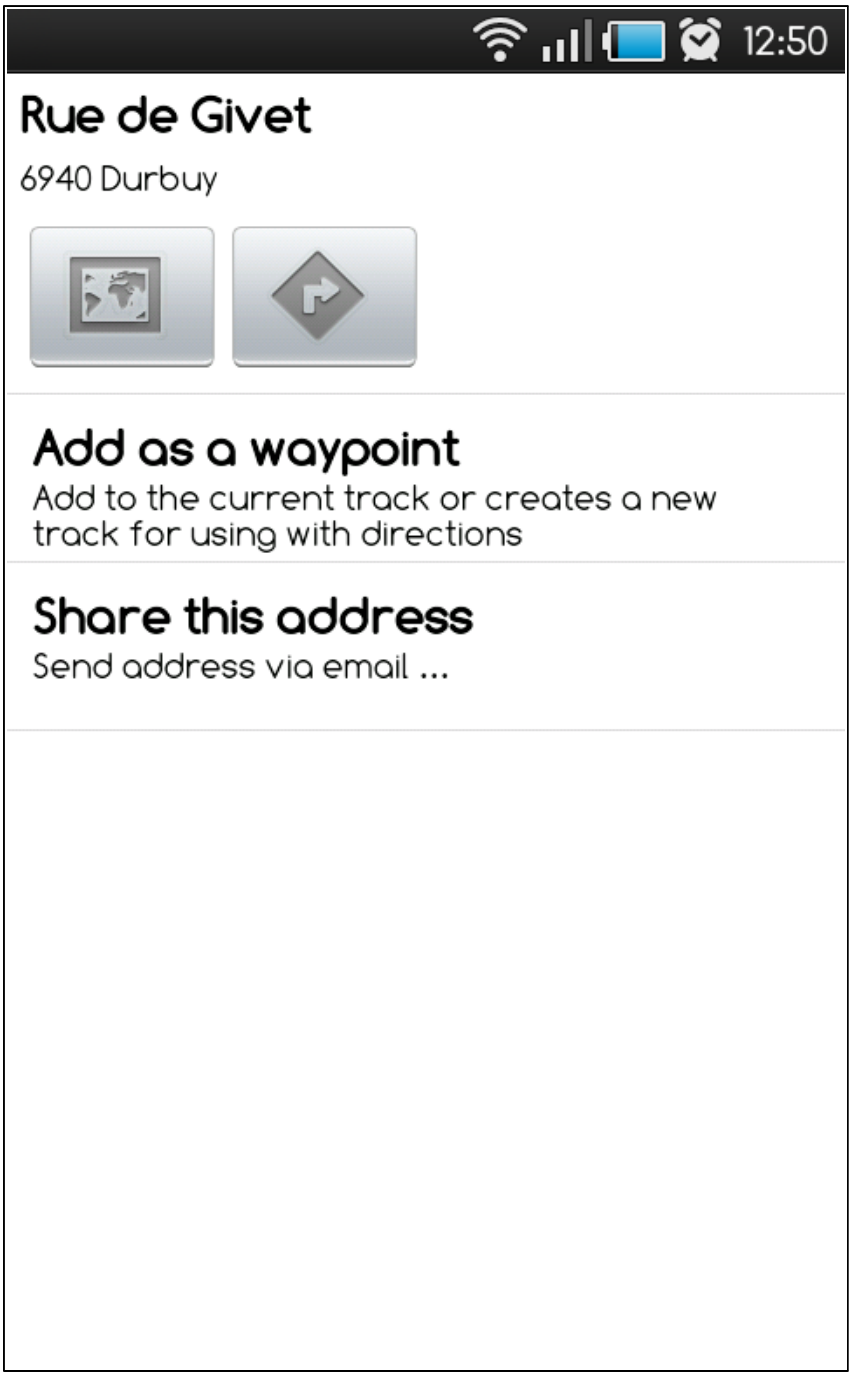
Cette fonction sera appelée autant de fois que nécessaire pour remplir la ListView et ce en rapport direct avec la valeur retournée par la fonction getCount().

Voici comment la fenêtre contenant la liste fait appel à paramètre celle ci pour utiliser le BaseAdapter récemment créé :

```
mListView = (ListView) findViewById(R.id.bubble_info_list);
mBubbleInfoListAdapter = new BubbleInfoListAdapter(this, false);
mListView.setAdapter(mBubbleInfoListAdapter);
```

La ListView déclarée dans le XML est une ListView normale du fait que c'est l'Adapter qui se charge du rendu, elle même ne se charge que d'empaqueter le résultat avec des délimiteurs et de donner la possibilité de faire dérouler la liste. Le **this** passé à SettingsListAdapter est un Context et vu que se code se trouve dans une Activity passer **this** en paramètre suffit.

Voici le résultat de cette liste déroulante :



15. Écrans des Préférences

Créer un (des) écran(s) de préférence est extrêmement simple, c'est en effet une possibilité offerte automatiquement par le système, il existe ici plusieurs manières de faire, celle qui sera retenue ici c'est celle qui se présente la plus flexible et également la plus proche du modèle MVC, elle consiste en la création d'un fichier XML dans le répertoire /res/xml/, ce dernier fonctionne de la même manière que les fichiers de Layout, à la différence que les différents éléments ont une propriété *"android:key"* qui permets de directement lier l'élément à une préférence de l'application.

Cela se passe en deux parties, la première étant le code le l'Activity qui deviens ici PreferenceActivity :

```
public class SettingsScreen extends PreferenceActivity {

   @Override
   public void onCreate(Bundle savedInstanceState) {
      requestWindowFeature(Window.FEATURE_NO_TITLE);
         super.onCreate(savedInstanceState);
      showSettingsScreen();
   }

   private void showSettingsScreen() {
         addPreferencesFromResource(R.xml.preferences);
         }

         /**
          * Handle physical Back button pression
          */
         public void onBackPressed() {
                  finish();
         }
}
```

On peut voir que à l'exception du fait d'utiliser PreferenceActivity et l'utilisation de la fonction addPreferencesFromResource() pour sélectionner le XML a instancier, il n'y à pas de grande différence avec une Activity normale.

Il suffit alors de créer le fichier XML correspondant à ce que l'on désire obtenir :

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
   <PreferenceCategory android:title="@string/inline_preferences_maps">
      <ListPreference
         android:key="map_provider_name"
         android:title="@string/settings_maps"
         android:summary="@string/settings_summary_maps"
```

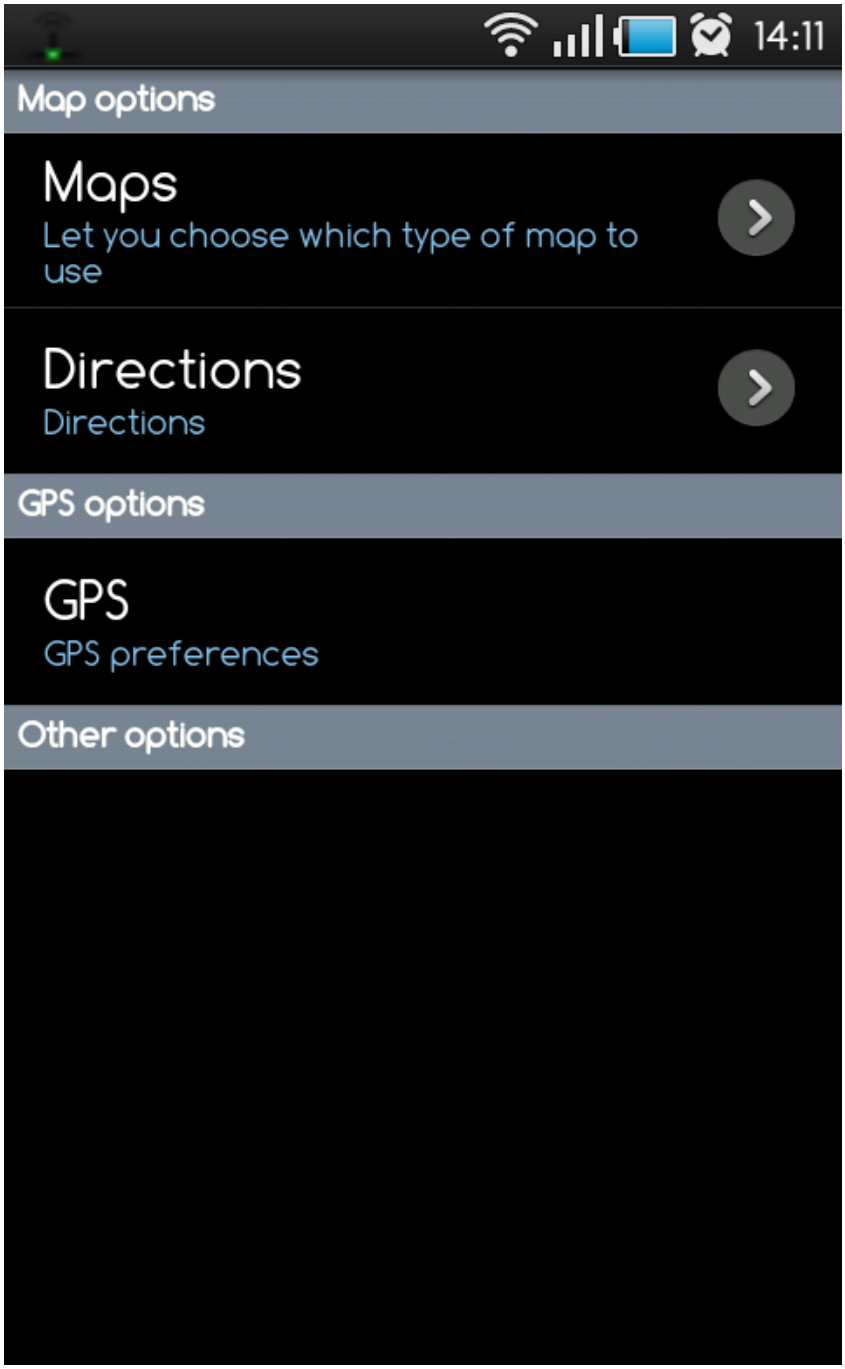
```
        android:dialogTitle="@string/settings_dialog_list_title"
        android:entries="@array/entries_list_map_names"
        android:entryValues="@array/entryvalues_list_map_names" />
    <ListPreference
        android:key="map_directions_mode"
        android:title="@string/settings_directions"
        android:summary="@string/settings_summary_directions"
        android:entries="@array/entries_list_directions_modes"
        android:entryValues="@array/entryvalues_list_directions_modes" />
  </PreferenceCategory>
  <PreferenceCategory android:title="@string/inline_preferences_gps">
    <PreferenceScreen
        android:key="gps_screen_preference"
        android:title="@string/settings_gps"
        android:summary="@string/settings_summary_gps">
        <CheckBoxPreference
            android:key="follow_user"
            android:title="@string/settings_gps_follow"
            android:summary="@string/settings_summary_gps_follow" />
        <CheckBoxPreference
            android:key="show_precision"
            android:title="@string/settings_gps_precision"
            android:summary="@string/settings_summary_gps_precision" />
    </PreferenceScreen>
  </PreferenceCategory>
  <PreferenceCategory android:title="@string/inline_preferences_other">
  </PreferenceCategory>
</PreferenceScreen>
```

Tout d'abord on remarquera que l'on ne définit pas de Layout mais *"PreferenceScreen"*. *"PreferenceCategory"* représente un séparateur, "ListPreference" qui permets de sélectionner un élément d'une liste, ces éléments sont contenus dans un array, ces derniers se retrouvent dans le fichier /res/values*/arrays.xml et se présentent sous cette forme ci :

```
<resources>

  <string-array name="entries_list_map_names">
    <item>Google</item>
    <item>Open Street Map</item>
    <item>Open Cycle Map</item>
  </string-array>
...
</ressources>
```

Chose très intéressante c'est qu'il est possible de créer une hiérarchie de fenêtres, ici tout ce qui se trouve à l'intérieur d'un *"PreferenceScreen"* interne désigne une nouvelle fenêtre ce qui est très pratique si on ne veut pas un écran surchargé et/ou devoir gérer plusieurs fichiers de préférence en même temps. Ici, les "CheckBoxPreference" représentent des cases à cocher.

Voici ce que cela donne dans l'application :



16. Interactions Systèmes (Emails, appels, ...)

Lorsque vous désirez pouvoir envoyer des e-mail ou effectuer des appels a partir de votre application, rien de plus simple, il suffit d'utiliser les applications système dédiées à ces actions. Le point positif est que pour une actions demandée, toutes les applications capables de l'effectuer seront proposées à l'utilisateur, il aura donc la possibilité de choisir celons ses préférences ce qui est toujours un plus.

Voici comment faire pour envoyer un mail :

```
	private void sendEmail() {
		mMessage = "Le message à envoyer";
		emailIntent = new Intent(android.content.Intent.ACTION_SEND);
		emailIntent.setType("plain/text");
		emailIntent.putExtra(android.content.Intent.EXTRA_EMAIL, new String[]
{"bob@email.com"});
		emailIntent.putExtra(android.content.Intent.EXTRA_SUBJECT, "Sujet");
		emailIntent.putExtra(android.content.Intent.EXTRA_TEXT, mMessage);
		startActivity(Intent.createChooser(emailIntent, "Send mail ..."));
	}
```

En premier lieu, il faut créer un Intent qui effectue l'action désirée, ici pour envoyer un mail on utilise android.content.Intent.*ACTION_SEND* vu que dans ce cas ci on envoie un mail si l'on voulait effectuer un appel on aurait utilisé *android.content.Intent.ACTION_DIAL*, ... tous commencent par *android.content.Intent.ACTION_**.

Il faut à présent définir certains paramètres (ceci n'est pas obligatoire) qui définissent le message à envoyer et le destinataire. Pour ce faire c'est très simple il suffit de faire comme avec tout intent et utiliser la fonction **putExtra()** en lui donnant un nom de propriété ainsi que la valeur. Toutes las propriétés commencent par *android.content.Intent.EXTRA_**. Ici on remplira donc l'adresse destinataire le sujet du message et le message lui-même.

Pour démarrer l'Activity associée, on va faire appel à la fonction startActivity(). On utilise ici Intent.createChooser() pour permettre à l'utilisateur de faire son choix, il verra donc une popup apparaitre à l'écran lui demandant quelle Activity (Application de gestion des mails dans ce cas) il choisis pour l'action représentée par cet Intent. Le deuxième paramètre représente le titre de cette popup.

Glossaire

Explication des termes souvent utilisés

Add-on : c'est un terme général qui comprends les plugin, les extensions, et les thèmes et peut faire référence à un élément matériel ou logiciel d'un ordinateur, qui si il est installé, ajoute ou améliore les fonctionnalités de l'application de base pour laquelle il est destiné. En peut donner en exemple les Add-on pour les navigateurs internet qui permettent de visionner des vidéos, du flash, ...

API (Application Programming Interface / Interface de Programmation d'Applications) : est une interface implémentée par un logiciel pour lui permettre d'interagir avec un autre logiciel, de la même manière qu'une interface utilisateur facilite l'interaction entre un humain et un ordinateur.

Les API sont implémentées par des applications, librairies et les systèmes d'exploitation pour déterminer le vocabulaire et les conventions d'appels que le programmeur devra utiliser pour utiliser ses services.

Bugzilla **:** désigne des outils de test et de recherche de bugs, orienté web il fût originellement développé et utilisé par le projet Mozilla (Firefox, Thunderbird, ...), et placé sous la licence "*Mozilla Public Licence"*. Il fut fournit en tant que projet open source par Netscape Communications en 199, il à été adopté par de nombreuses organisations afin d'être utilisé pour le suivi des problèmes pour les applications open source aussi bien que pour les applications commerciales. http://www.bugzilla.org/

OSI (Open Source Initiative) : est une organisation dédiée à la promotion des logiciels open-source. http://opensource.org/ http://www.opensource.org/licenses/eclipse-1.0.php

GDK (Gimp Drawing Kit) : est une librairie graphique informatique qui agit en tant qu'adaptateur autour des fonctions de dessin bas niveau et des fonctions des fenêtres du sous-système graphique. Originellement développé pour GIMP sur le X Window System (système de gestion d'écran Unix), GDK se retrouve entre le X Window System et les librairies GTK+ (librairie qui permets de créer des fenêtres), ce qui facilite la création de celles-ci. http://library.gnome.org/devel/gdk/

Mylyn (Eclipse) : est un sous système d'Eclipse qui se charge de la gestion des tâches.

Mylyn est une implémentation Open Source d'une interface orientée tâche (voir Task-Focused Interface). Il fournit une API pour les outils utilisant cette interface. Pour les développeurs de logiciels, cela les aide à travailler de manière plus efficace avec différentes tâches (tels les bugs, rapports d'erreurs ou nouvelles options).

Les tâches sont intégrées a l'intérieur de Mylyn; pour chaque tâche ayant été intégrée, Mylyn effectue un monitoring des actions de l'utilisateur et essaye d'identifier les informations relatives à l'action effectuée.

Il utilise le contexte de cette tâche pour mettre en avant dans l'interface d'Eclipse les informations relatives. Mylyn peut s'intégrer à des dépôts tels que Bugzilla, Trac, Redmine et JIRA.

Il se concentre sur l'amélioration de la productivité en réduisant les recherches, le scrolling, et la navigation. En faisant le contexte des taches explicite, Mylyn à aussi pour but de faciliter le multitâche, le planning, réutilisation des efforts passés et partage de l'expérience.

Task-Focused Interface (Interface Orientée Tâches) : est un type d'interface utilisateur qui étends la métaphore du bureau de l'interface utilisateur afin de créer des tâches, pas des fichiers ou dossiers, l'unité d'interaction primaire. À la place de montrer une hiérarchie entière d'informations, telle un arbre de documents, une interface orientée tâche montre la sous partie de l'arbre qui est relative à la tâche exécutée. Ceci s'occupe de la surcharge d'informations lorsque l'on se charge d'une grande hiérarchie, tel celle d'un software ou un grand nombre de documents.

Package (paquet) : Un package logiciel se réfère à un logiciel se trouvant empaqueté dans une archive désignée a être installée par un gestionnaire de paquets ou être auto installé.

Les distributions Linux sont normalement segmentées en packages. Chacun contenant une application spécifique ou un service.

Equinox (Eclipse) : Equinox est le nom d'un projet Eclipse qui fournit une implémentation certifiée de la spécification du framework du core R4 OSGI. Equinox est simplement un système de plugins qui permets aux développeurs d'implémenter une application en tant que paquets en utilisant des infrastructures et services communs. Dans sa version 3.0, Eclipse utilise OSGI pour remplacer la technologie de plugin dans les anciennes versions d'Eclipse .

OSGI framework **:** OSGI est un une plateforme de services et de modules systèmes pour le langage JAVA qui implémente un modèle de composants complet et dynamique, c'est quelque chose qui n'existe pas dans les versions seules de JAVA et de l'environnement de la VM. Les Applications ou composants peuvent être installés, démarrés, stoppés, mis à jour et désinstallés de manière distante sans requérir à un redémarrage.

DDMS (Dalvik Debug Monitor Server) : DDMS agis comme un intermédiaire pour connecter l'IDE à l'application tournant sur l'appareil (aussi bien physique que l'emulateur). Sur Android, chaque application tourne dans son propre processus, chacun d'une eux accueille sa propre machine virtuelle. Et chaque processus écoute le débugger sur un port différent.

Quand il démarre, DDMS se connecte à adb et démarre un service de monitoring d'appareillage entre les deux qui va envoyer des notifications à DDMS lorsque un appareil est branché ou débranché. Lorsque un appareil est branché, le service de monitoring de la VM est créé entre adb et DDMS, lequel va notifier DDMS lorsque une VM sur la machine est démarrée ou stoppée.

Lorsque une VM est lancée, DDMS récupère l'ID du processus de la VM (pid), à travers adb, et ouvre une connexion au débugger de la VM, à travers le démon de adb (adbd) sur l'appareil, DDMS peur dorénavant parler à la VM en utilisant un protocole câblé modifié.

Namespace (espace de noms) : Un namespace est un conteneur ou environnement abstrait créé pour contenir un groupement logique d'identifiants ou se symboles uniques (les noms). Un identifiant définis dans un espace de noms est associé avec ces espace de noms. Le même identifiant peut être définis indépendamment dans plusieurs espaces de noms. Ce qui fait que la signification associée a un identifiant dans un espace de noms peut ou peut ne pas avoir la même signification que le même identifiant dans un autre espace de noms.

Une représentation simple peut être le fait qu'il est possible d'avoir sur un même disque dur deux fichiers avec le même nom, il suffit qu'ils se trouvent dans un répertoire différent (leur namespace).

Bibliographie

Livres :

Apress – Beginning Android - Mark L. Murphy

Apress – Beginning Android 2 - Mark L. Murphy

Apress – Pro Android - Sayed Y. Hashimi | Satya Komatineni

Apress – Pro Android 2 - Sayed Hashimi | Satya Komatineni | Dave MacLean

Sites Web :

Google Dev :
http://developer.android.com/index.html

Android Wizard:
http://androidwizard.net/

StackOverflow
http://stackoverflow.com/

netthreads
http://ccgi.arutherford.plus.com/blog/wordpress/

AndDev
http://www.anddev.org/

HelloAndroid
http://www.helloandroid.com/

Juri Strumpflohner's TechBlog
http://blog.js-development.com/

DroidNova
http://www.droidnova.com/

Warrior Point
http://www.warriorpoint.com/blog/

Mobiforge
http://mobiforge.com/

The Open Mob
http://wiki.andmob.org/samplecode

Code Mobiles
http://www.codemobiles.com/home

Android Competency Center
http://www.androidcompetencycenter.com/

Source des examples :

http://code.google.com/p/mapinmypocket/