Terminaux Mobiles Communicants

Programmation Android

—

P-F. Bonnefoi

Version du 4 avril 2011

TMC

Table des matières

L'OHA est un consortium regroupant plus de 70 sociétés comprenant :

- des fabricants de terminaux *Motorola*, *HTC*, *Dell*, *Garmin*... ;
- des fabricants de composants *Atheros*, *ARM*, *Gemalto*, *Broadcom*, *Intel*, *NVIDIA*...
- des opérateurs de téléphonie *Bouygues Telecom*, *Vodaphone*, ... ;
- des sociétés de développement logiciel *Google*, ... ;
- ...

Motto

A commitment to openness, a shared vision for the future, and concrete plans to make the vision a reality. To accelerate innovation in mobile and offer consumers a richer, less expensive, and better mobile experience.
Together we have developed Android™, the first complete, open, and free mobile platform.
`http://www.openhandsetalliance.com/oha_faq.html`

2 Android : matériel et logiciel combinés

- un document de référence exprime les contraintes matérielles nécessaires au support de la pile logicielle ;
- un noyau Linux qui fournie les interfaces de bas-niveau vers le matériel, la gestion de la mémoire, la gestion des processus, optimisé pour les appareils mobiles ;
- des bibliothèques opensource pour le développement d'application incluant SQLite, WebKit, OpenGL et un gestionnaire multimédia ;
- un logiciel de démarrage, *runtime* :
  - ◇ de petite taille, efficace pour une plateforme mobile
  - ◇ pour exécuter et héberger les applications Android ;
  - ◇ pour démarrer et gérer la machine virtuelle Dalvik ;
  - ◇ pour gérer et mettre à disposition les bibliothèque des fonctionnalités de base ;

- un « framework » qui expose tous les services à la couche applicative sans privilégier une application suivant son origine (éditeur tiers ou non) : gestionnaire de fenêtre, fournisseur de contenu, gestionnaire de position géographique, téléphonie et service de communication p2p ;
- un « framework » d'interface utilisateur utilisé pour héberger et lancer les applis ;
- des applications pré-installées ;
- un kit de développement logiciel pour créer des applications : outils, plug-ins et documentation.

3 Les applications natives

Il existe un certain nombre d'applications disponibles dans Android :

- un client de messagerie compatible Gmail mais aussi avec d'autres messageries ;
- une application de gestion des SMS ;
- un gestionnaire de contacts et d'agenda, un PIM *Personal Information Management*, intégré avec les services onlines de Google ;
- une application de gestion de « Google Maps » incluant StreetView, la navigation guidée par GPS, les vues satellites et l'obtention des conditions de trafic ;
- un navigateur web basé sur WebKit ;
- un client de messagerie instantanée ;
- un lecteur multimédia et un afficheur de photo ;
- un client « l'Android Marketplace » pour télécharger des applications d'éditeur tiers ;
- un client pour la boutique d'achat de musique « Amazon ».

Ces applications sont programmées en Java avec le SDK et s'exécutent sur la Dalvik.

Mais

- Les données utilisées par toutes ces applications sont rendues disponibles aux applications tierces.
- Les applications tierces peuvent recevoir et gérer des événements comme la réception d'appel téléphonique et celle de SMS.

4 Ce qu'Android n'est pas...

- une implémentation de Java ME, *Micro Edition* : une application android est écrite en Java, mais ne s'exécutent pas dans une machine virtuelle Java ME.
  Les classes Java compilées et les exécutables ne peuvent tourner directement sous android ;

- une partie du *Linux Phone Standards Forum* (LiPS) ou du *Open Mobile Alliance* (OMA) : android tourne sur un noyau Linux opensource, mais l'approche d'Android est plus global ;

- juste une couche applicative : Android recouvre la pile logiciel complète :système d'exploitation, bibliothèques et les applications également ;

- un téléphone mobile : Android inclus un document de référence pour la conception de téléphone, mais il n'existe pas « un seul » téléphone Android mais au contraire de nombreux téléphones fabriqués par différents constructeurs ;

- la réponse de Google à l'iPhone : Android est issu d'un consortium et non d'un seul constructeur.

C'est un *smartphone*, caractérisé par :

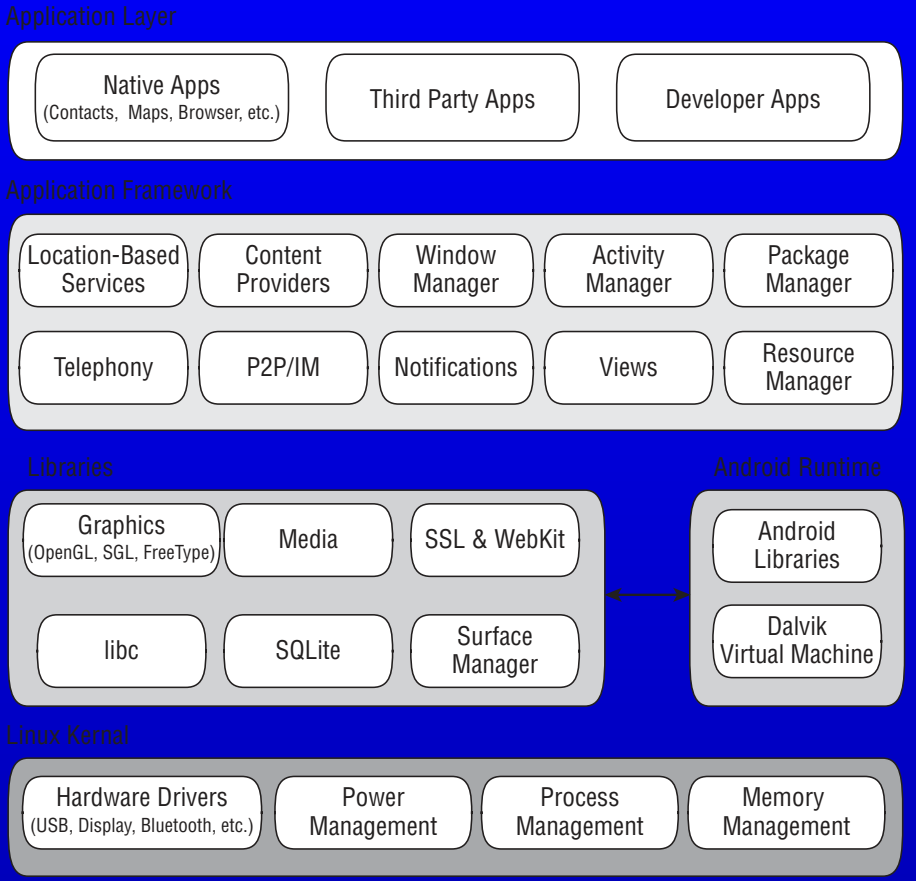
- un processeur autour de 1Ghz, *comme le Snapdragon de Qualcomm sur architecture ARM* ;
- une batterie de capacité limitée, *1 ou 2 journées d'utilisation* ;
- un écran de résolution variable suivant les modèles limitée en dimension, *QVGA : 320x240, WVGA : 800x480, etc.*
- une mémoire intégrée importante : $\sim$ 512Mo de RAM, 512Mo de flash ;
- un espace de stockage renouvelable et extensible (SDCard) ;
- écran tactile, *capacitif « multi-touch », ou encore résistif* ;
- capacités multimédia 2D, 3D, vidéo et son : différents CODECs : MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF, différentes APIs : OpenGL, FreeType, SGL ;
- l'accès à Internet en mobilité : accès au WiFi, au GSM, au EDGE, à la 3G pour la téléphonie et l'échange de données ;
- un environnement de développement riche, basé sur le langage Java permettant de créer facilement des applications mobiles connectées !


6:15 PM
Google
MENU
1 2 3 4 5 6 7 8 9 0
Q W E R T Y U I O P
A S D F G H J K L DEL
Z X C V B N M .
ALT SYM @ / , ALT

7 Les avantages de la plateforme Android et de ses APIs

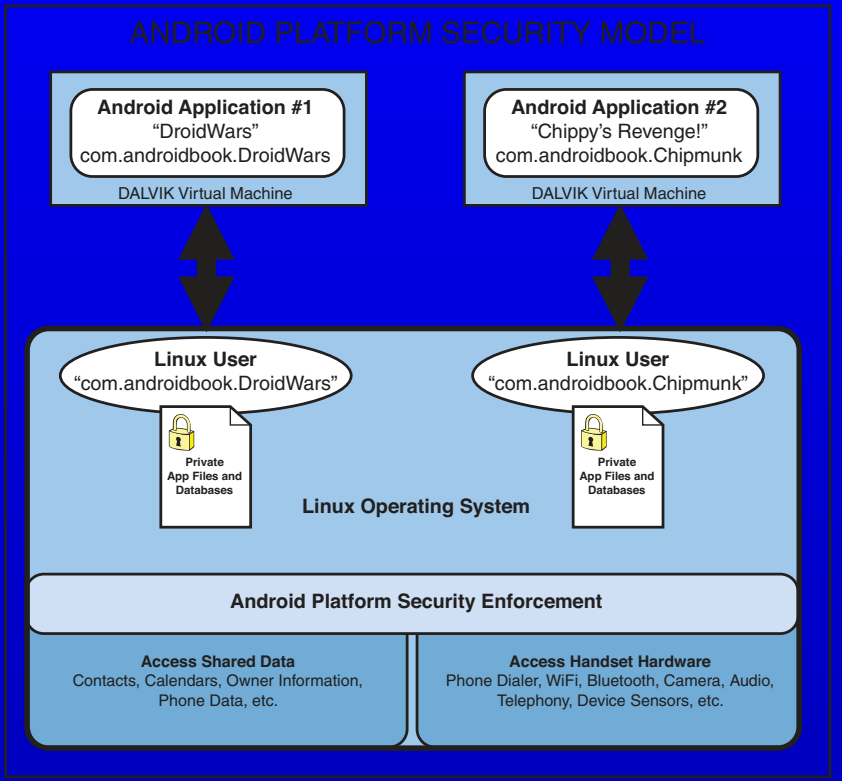
- pas de license à obtenir, pas de dépense pour la distribution et le développement *Cool pour l'Université !*
- développer des applications « location-based » en utilisant le GPS ;
- utiliser des cartes géographiques avec Google Maps ;
- recevoir et émettre des SMS, envoyer et recevoir des données sur le réseau mobile ;
- enregistrer et lire image, son et vidéo ;
- utiliser compas, accéléromètre, gyroscope ;
- des IPC, *inter process communication*, avec les *Notifications* et *Intents* pour une relation « event-driven » avec l'utilisateur et l'échange de messages entre applis ;
- desoutils de stockage de données partagés (SQLite en version *Sandbox*) ;
- *Content Provider* pour partager l'accès à ses données ;
- un navigateur que l'on peut inclure basé sur WebKit ;
- une accélération matérielle pour la 2D et la 3D ;
- des services et des applications qui peuvent tourner en tâche de fond : qui peuvent réagir au cours d'une action, à votre position dans la ville, à l'heure qu'il est, suivant l'identité de l'appelant...
- une plateforme de développement qui favorise la **réutilisation** de composants logiciels et le **remplacement** des applications fournies.

8 La pile logicielle



9 Modèle de sécurité

Chaque application Android est associée à un compte utilisateur Linux différent.



Le partage d'accès se fait ensuite suivant un système de « permissions ».

10 Une application Android

Une application Android est constituée de **composants faiblement couplés**.
Ces composants sont décrits dans un même document : le « manifest », un fichier au format XML `AndroidManifest.xml` qui décrit :

- chaque composant : *Activities*, *Services*, *Content Providers*, *Broadcast Receivers* ;
- les interactions de ces composants : *Intent Filters* et *Notifications* ;
- les accès nécessaires au matériel et logiciel (bibliothèques) : *Permissions* ;

Les composants pour définir une application :

- les « Activités » ou *activity* : correspondent à des applications sur un ordinateur de bureau.
  - ◇ elles sont démarrées à la demande de l'utilisateur ou du système ;
  - ◇ elles tournent aussi longtemps qu'elles sont utiles : le système peut tuer une activité pour récupérer de la mémoire ;
  - ◇ elles peuvent interagir avec l'utilisateur, demander des données ou des services à d'autres activités ou à des « services », au travers de requêtes et d'« intentions », *intents* ;
  - ◇ en général, une activité = un écran d'interface, appelé *View*.

- les « Services » : similaires aux « démons » sous Unix et aux « services » sous Windows : du code tournant en tâche de fond.
  - ils travaillent même lorsque lorsque l'activité de l'application n'est plus active ou visible (par exemple, un lecteur MP3 qui peut jouer de la musique même si l'interface de contrôle ne tourne plus) ;
  - ils peuvent transmettre des notifications.

- les « notifications » : c'est un moyen de signaler à l'utilisateur sans interrompre l'activité courante, ni prendre le focus.
  - la meilleur méthode pour attirer l'attention de l'utilisateur depuis un service ou un récepteur de diffusion ;
  - exemple : l'utilisateur reçoit un SMS ou un appel et il est alerté par des voyants qui clignotent, l'écoute d'un son, l'affichage du message ou une icone.

- les « fournisseurs de contenu » ou *Content providers* :
  ils ont été définis pour partager des données avec d'autres applications.
  - ils utilisent une interface standardisée sous la forme d'URI, *Uniform Resource Identifier*, pour répondre à des requêtes en provenance d'autres applications. Ces applications n'ont même pas besoin de connaitre le fournisseur de contenu qui leur répond.
  - exemple : `content://contacts/people` permet de définir une requête pour l'accès à la liste des contacts de l'utilisateur ;
    - le système d'exploitation recherche si une application s'est enregistrée comme fournisseur de contenu pour cette URI ;
    - s'il trouve une application, il la démarre si elle ne s'exécute pas déjà, et lui transmet la requête ;
    - s'il trouve plus d'une application, il demande à l'utilisateur laquelle il veut utiliser.

- les « intents » ou notifications : ils définissent un modèle par « passage de message » entre les applications.
  - Ces messages peuvent être diffusés, *broadcast*, à tout le système ou bien être envoyés à une activité ou un service particulier (le système décide des cibles) ;
  - un message a pour but de déclencher une action.

- les « récepteurs de diffusion », ou *broadcast receivers*. Ils s'enregistrent dans le système et reçoivent les messages diffusés qui correspondent à un filtre prédéfini.
  - ils peuvent déclencher une application en réponse à la réception d'un message ;
  - ils permettent d'utiliser un modèle « piloté par les événements », *event-driven*.

- les « widgets » : des composants visuels qui peuvent être ajoutés à l'écran de démarrage, *home screen*. Il correspondent à une variation du *broadcast receiver*.

L'intérêt de ces composants est de permettre de les partager avec d'autres applications (en particulier les services et fournisseurs de contenu).

Dans Android, il n'y a qu'une activité « active » à la fois, c-à-d. en « avant-plan » ou *foreground*.

Le système d'exploitation est **libre** de terminer une activité qui est en arrière-plan, lorsque la quantité de mémoire libre du système est trop basse.

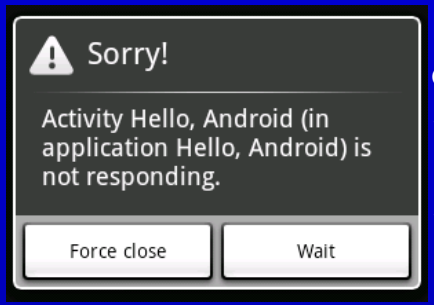
Ainsi, une application Android, lorsqu'elle n'est plus en avant-plan, doit être capable de maintenir **son état** pour garantir un fonctionnement *seamless* à l'utilisateur (retrouver les valeurs et affichage précédents lorsqu'il remet l'application en avant-plan).

C'est à l'application de gérer son **état**, ses données, et ses ressources afin d'être prête à être interrompue ou bien terminée à tout moment.

Pour répondre à ses événements, des méthodes de réponse, *callback*, sont définies : `onCreate()`, `onResume()`, `onPause()`, `onDestroy()`

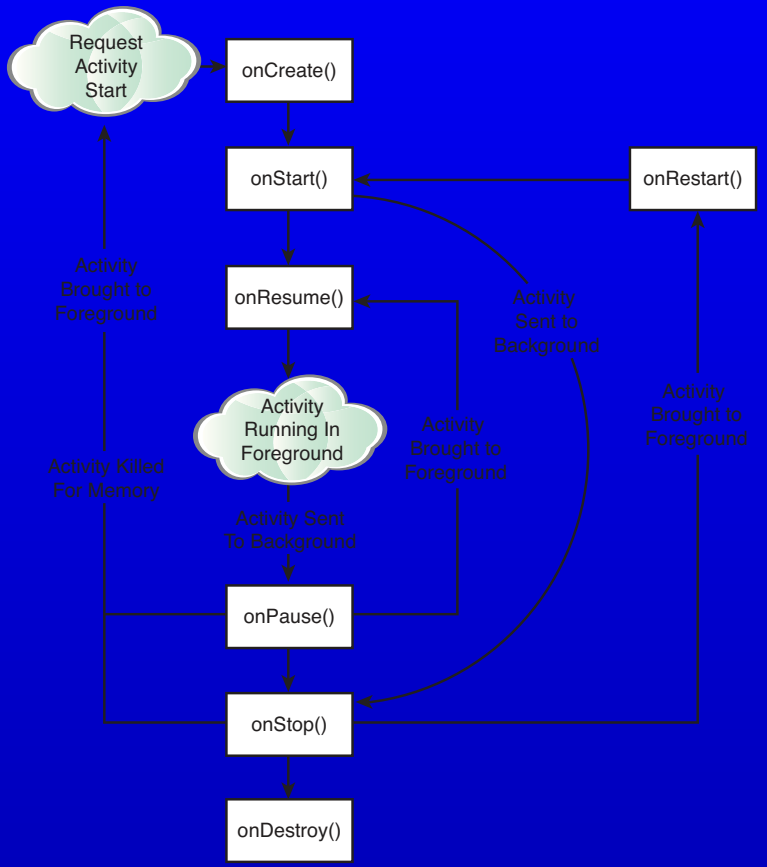
Description des différents callback et des actions à entreprendre lors de leur réception :

- `onCreate()` : est appelé quand l'application démarre ou redémarre.
  Initialiser les données statiques, établir des liens vers les données et les ressources, positionner l'interface avec `setContentView()`.
- `onResume()` : appelé quant une activité passe en avant-plan.
  Reprendre le contrôle des ressources exclusives.
  Continuer les lectures audio et vidéo ou les animations.
- `onPause()` : appelé quand l'activité quitte l'avant plan.
  Sauvegarder les données non encore sauvegardées, libérer l'accès aux ressources exclusives, stopper la lecture audio, vidéo et les animations.
- `onDestroy()` : appelé quand l'application est fermée.
  Nettoyer les données statiques de l'activité, libérer toutes les ressources obtenues.



Il est nécessaire de limiter le temps pris pour traiter la réception de ces callbacks :

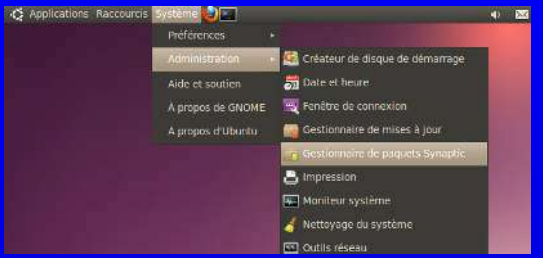
- la thread principale de l'application est celle s'occupant de l'interface : *UI thread* ;
- elle ne doit pas être bloquée plus de 5 secondes sous peine de voir le « Application Not Responding ».


Request Activity Start
onCreate()
onStart()
onRestart()
onResume()
Activity Running In Foreground
onPause()
onStop()
onDestroy()
Activity Brought to Foreground
Activity Sent to Background
Activity Brought to Foreground
Activity Killed For Memory
Activity Brought to Foreground
Activity Sent To Background

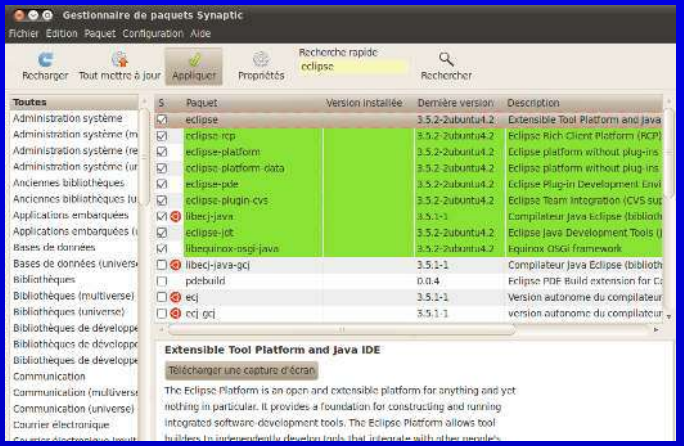
Pour pouvoir développer une application Android, on utilise :
– le kit de développement fourni par Google ;
– un des différents simulateurs de terminaux communicants.
Ces différents éléments peuvent être intégrés et utilisés dans l'IDE, « Integrated Development Environment », Eclipse.

Pour pouvoir utiliser l'API Bluetooth sur le simulateur, on utilise une bibliothèque permettant de simuler son fonctionnement en utilisant la pile de communication TCP/IP de la machine hôte.
En effet, le simulateur ne prends pas en charge la présence éventuelle d'un composant bluetooth sur la machine hôte.

Pour cela, on va installer :
– Eclipse ;
– le SDK d'Android ;
– une bibliothèque pour l'utilisation du Bluetooth ;



Sous Ubuntu, on peut utiliser le gestionnaire de paquets Synaptic :



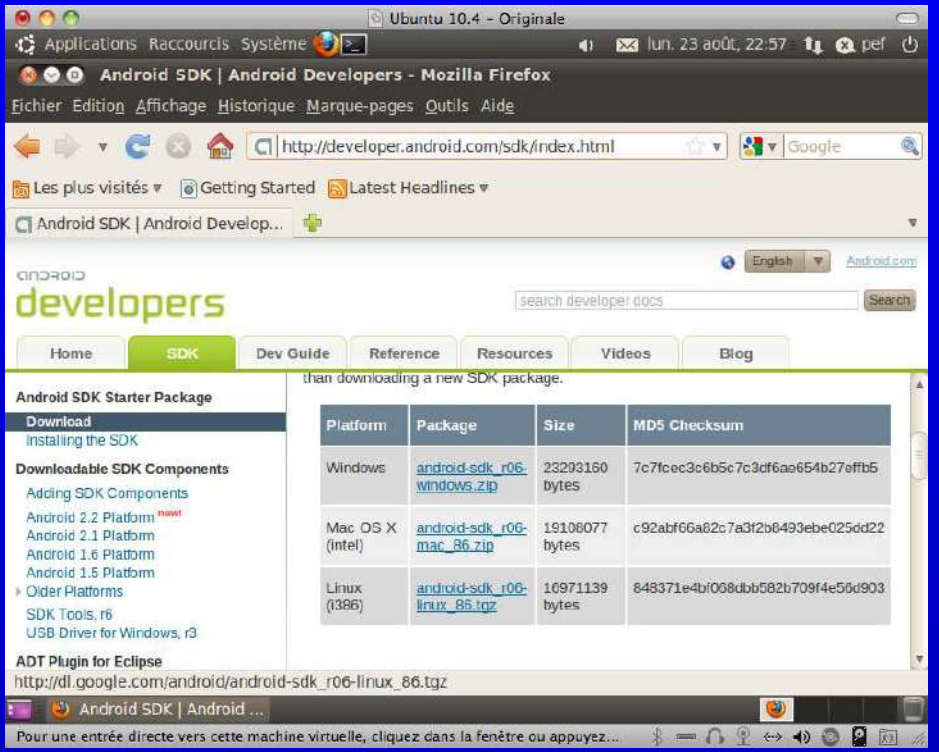
On sélectionne Eclipse, pour procéder à son installation.

On peut également utiliser la commande :

```
$ sudo apt-get install eclipse
```

12.2 Récupération du SDK Android

Le SDK d'Android est disponible à l'adresse :
`http://developer.android.com/sdk/index.html`



On choisira l'archive `android-sdk_r06-linux_86.tgz`.

12.3 Installation du SDK et intégration dans Eclipse

- récupérer l'archive `android-sdk_r06-linux_86.tgz`
- créer un répertoire `ANDROID` :

```
$ mkdir ~/ANDROID
$ tar xvfz android-sdk_r06-linux_86.tgz
$ cd android-sdk-linux_86
```

Il faut ajouter la ligne
`export PATH=${PATH}:${HOME}/ANDROID/android-sdk-linux_86/tools`
à la fin du fichier `~/.bashrc`, pour l'accès aux commandes du SDK.

- il faut ajouter le plugin « Android Developement Tools (ADT) » dans Eclipse :
  - ◇ Allez dans le menu « Help → install new software », et rajoutez le site `https://dl-ssl.google.com/android/eclipse/`
  - ◇ sélectionnez les « Developper Tools » ;
  - ◇ acceptez l'installation d'éléments non signés.
- Allez dans le menu « Window → Preferences » :
  - ◇ sélectionnez « Android » ;
  - ◇ indiquez dans le zone de saisie le chemin d'accès au SDK :
    `/home/pef/ANDROID/android-sdk-linux_86/`
    Vous mettrez votre nom de compte à la place de « pef »...

- Allez dans le menu « Window → Android SDK and AVD Manager » :
  - ◇ sélectionnez la plate-forme 2.2
  - ◇ vous pouvez retourner dans le menu « Window → Preferences » pour vérifier la présence de cette plate-forme.

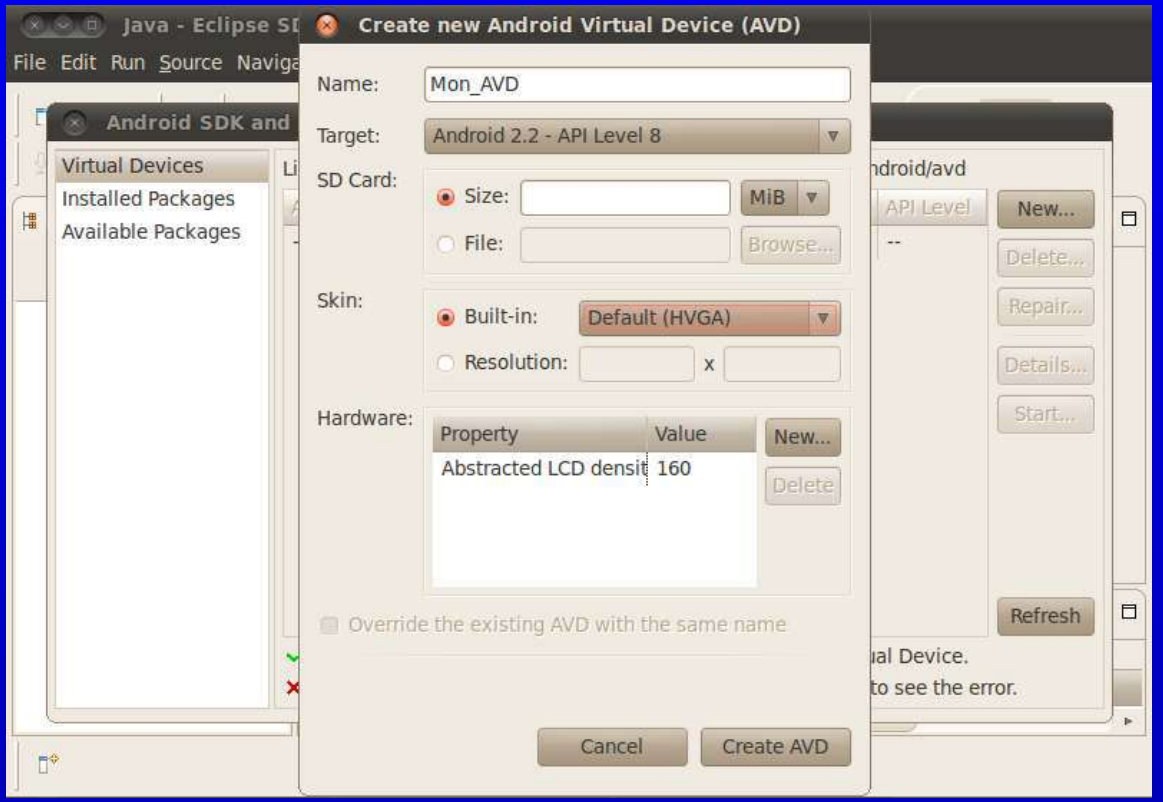
- installer une bibliothèque pour l'utilisation du Bluetooth :
  `http://github.com/cheng81/Android-Bluetooth-Simulator`

```
$ sudo apt-get install git-core
$ cd ANDROID
$ git clone git://github.com/cheng81/Android-Bluetooth-Simulator.git
$ cd Android-Bluetooth-Simulator
```

13 Le contenu du SDK

- les APIs Android ;
- les outils de développement (dans le répertoire `tools`) :
  - ◇ DDMS, *Dalvik Debug Monitoring Service* : le débogueur ;
  - ◇ AAPT, *Android Asset Packaging Tool* : outil de construction des *packages* d'application (`.apk`) ;
  - ◇ ADB, *Android Debug Bridge* : copier des fichiers, déployer des applications ;
  - ◇ SQLite3 : pour la gestion des BDs ;
  - ◇ MkSDCard : pour créer une SDCard virtuelle ;
  - ◇ dx : convertir les `.class` Java en bytecode Android `.dex`.
- l'émulateur android :
  - ◇ interactif, avec possibilité de simuler la gestion d'appel téléphonique et de SMS ;
  - ◇ plusieurs *skins* pour tester différents *form factors* (écran, orientation, *etc.*) ;
  - ◇ chaque application est exécutée dans une VM Dalvik, ce qui permet de tester et de déboguer les applications dans un environnement « réel ».
- une documentation : `http://developer.android.com/guide/index.html`
- une communauté de développeur :
  `http://developer.android.com/resources/community-groups.html`
  et sur Stack Overflow : `http://stackoverflow.com/`
- des exemples de code.

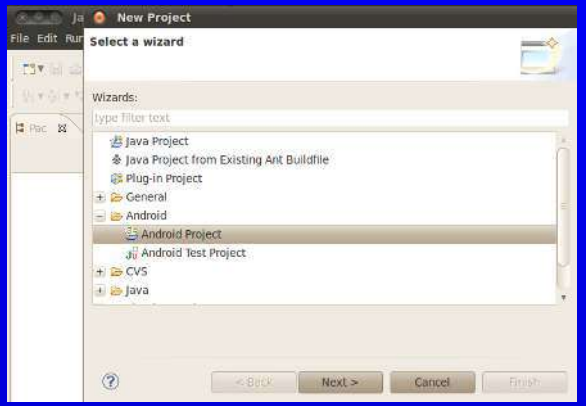
Allez dans le menu « Window → Android SDK and AVD Manager », puis prendre « Virtual Devices » (avec les choix par défaut) :



15 Connexion à l'émulateur

Il est possible de se connecter au terminal virtuel à l'aide de la commande telnet sur le port indiqué dans le nom de la fenêtre de l'émulateur :

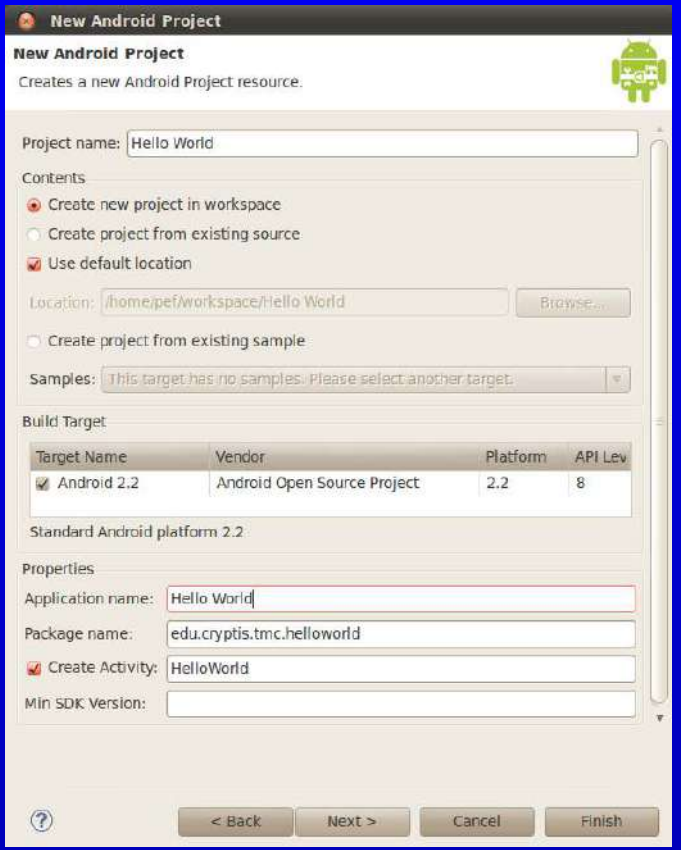
```
pef@pef-desktop:~/ANDROID/android-sdk-linux_86/tools$ telnet localhost 5554
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Android Console: type 'help' for a list of commands
OK
help
Android console command help:
    help|h|?         print a list of commands
    event            simulate hardware events
    geo              Geo-location commands
    gsm              GSM related commands
    kill             kill the emulator instance
    network          manage network settings
    power            power related commands
    quit|exit        quit control session
    redir            manage port redirections
    sms              SMS related commands
    avd              manager virtual device state
    window           manage emulator window
try 'help <command>' for command-specific help
OK
```



Puis on définit les options ⇒ de l'application :

- ◇ Package Name ;
- ◇ Activity ;
- ◇ Application name ;
- ◇ la version de la plateforme.

⇐On choisit le menu « File → New → Project »



On obtient le code ci-dessous dans le fichier `HelloWorld.java` :

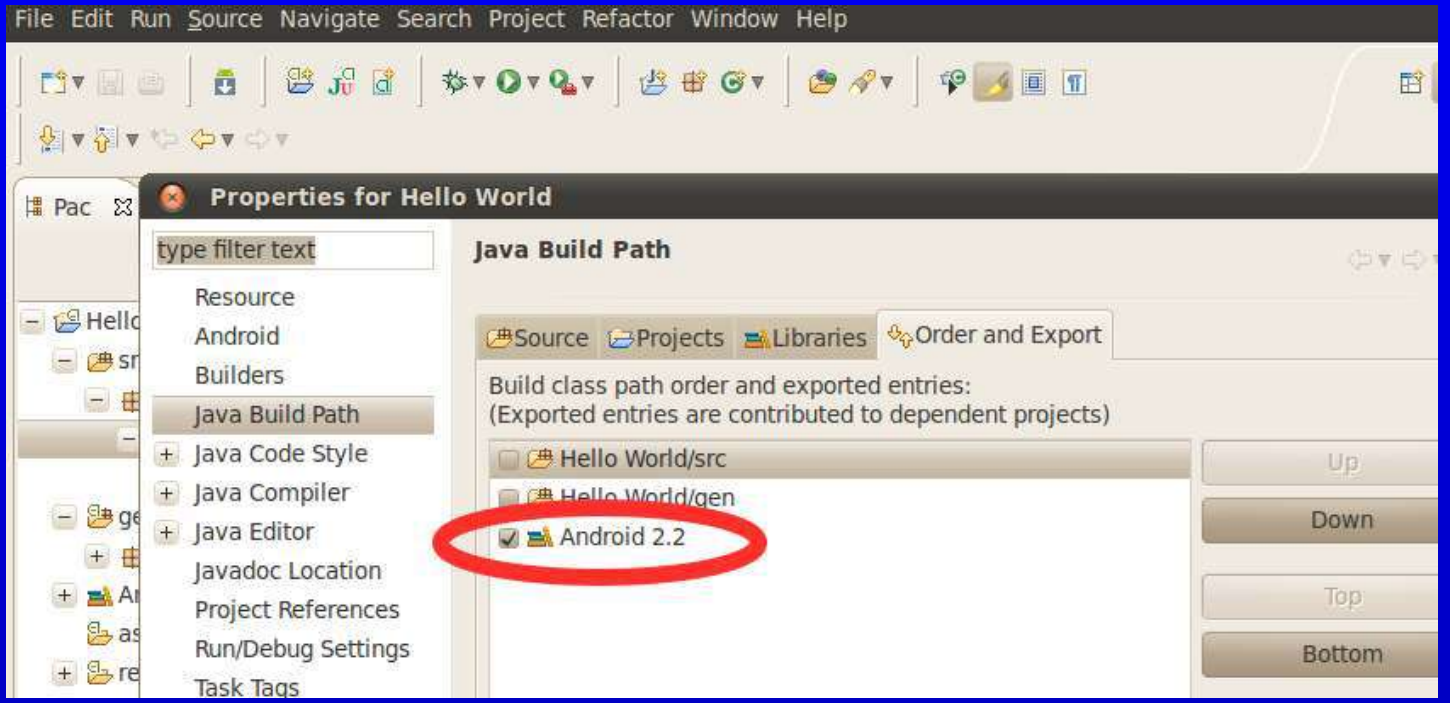
```
package edu.cryptis.tmc.helloworld;

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

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

Ce code :

– définit une *activity*, par héritage de la classe *Activity* ;

– positionne son interface utilisateur par l'utilisation de `setContentView()`.

Menu « Project→Properties », vérifiez que :



Elle est définie par :

– l'utilisation d'un *container* vertical qui remplit tout l'écran

  ◇ qui contient une étiquette faisant référence au contenu d'une chaîne définie dans les ressources (répertoire `res/`).

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

et celui du fichier `res/values/strings.xml` :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, HelloWorld!</string>
    <string name="app_name">Hello World</string>
</resources>
```

20 Le fichier AndroidManifest.xml

Le contenu du fichier res/layout/main.xml :

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="edu.cryptis.tmc.helloworld"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".HelloWorld"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

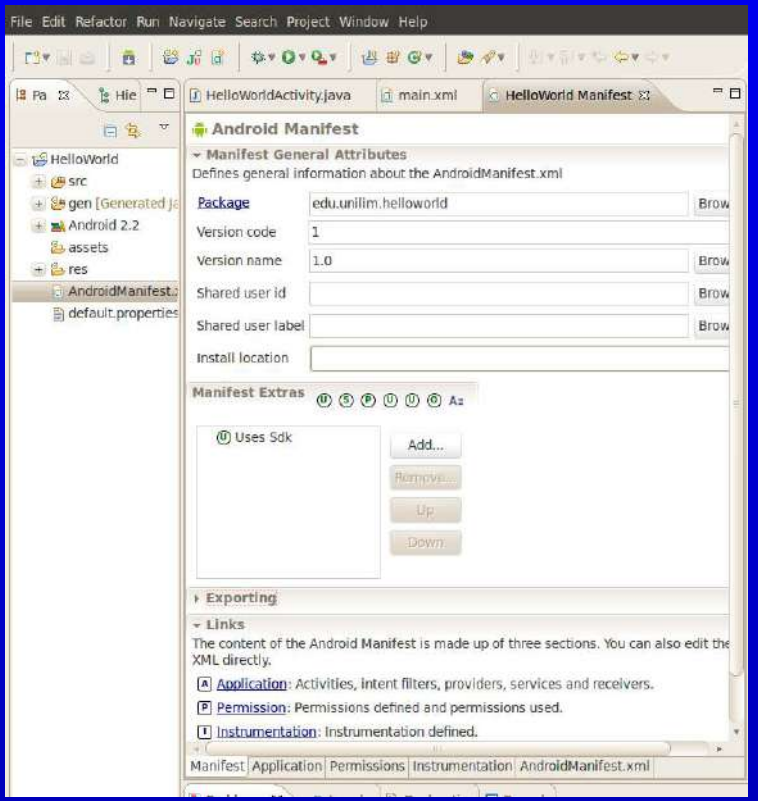
Un *namespace* est définie `xmlns:android` qui est **uniquement utilisé** pour les attributs du fichier XML et non pour les éléments, avec l'exception de l'attribut `package` de l'élément `manifest`...

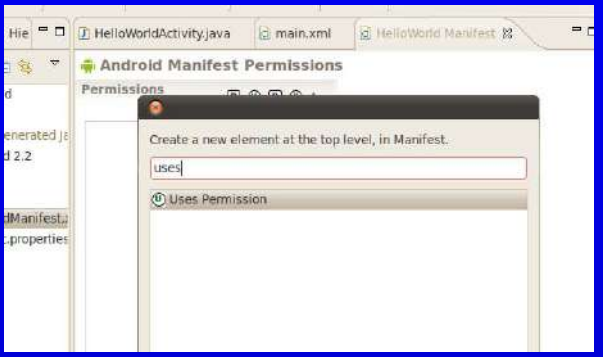
D'autres éléments :

- `uses-permissions` : les permissions nécessaires à l'application ;
- `permission` : déclarer des permissions nécessaires aux autres applications dans le but d'avoir accès au données de l'application ;
- `instrumentation` : indiquer le code à appeler lors de la réception d'événements systèmes comme le lancement d'une activité, pour *monitoring* ou du *logging* ;
- `uses-library` : utiliser d'autres composants, comme le service de localisation ;
- `uses-sdk` : `<uses-sdk minSdkVersion="2"/>` pour des sous-versions du SDK ;
- `application` : définir l'application elle-même.

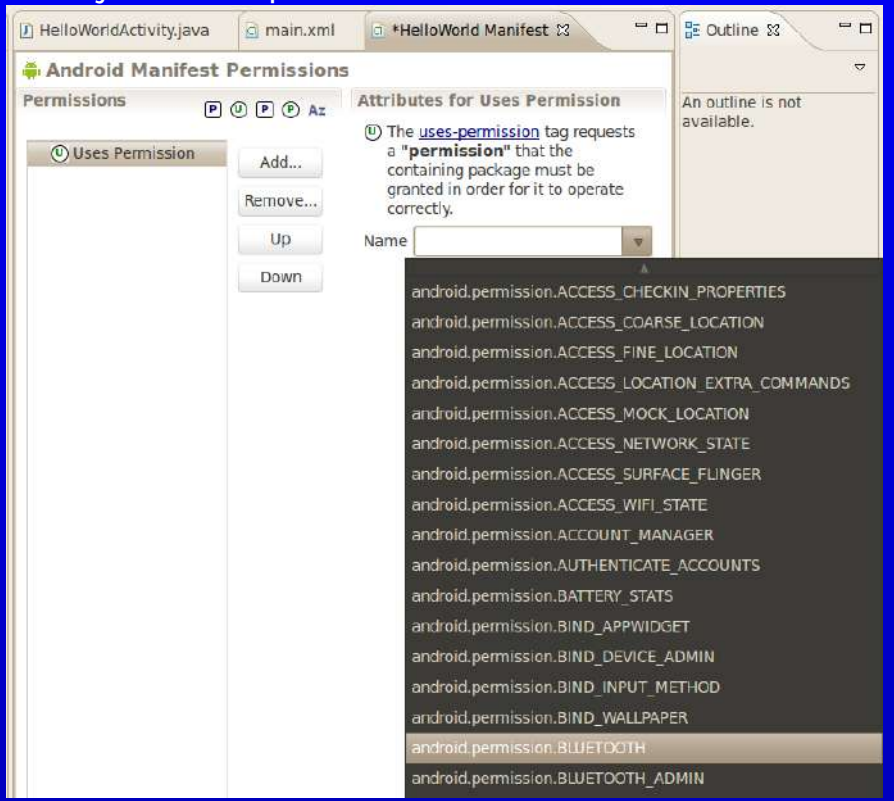
```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="edu.cryptis.tmc.monappli">
<uses-permission
android:name="android.permission.ACCESS_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_GPS" />
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission
<application> ... </application>
</manifest>
```

Il est possible pour définir le contenu du fichier `AndroidManifest.xml` d'utilisé l'éditeur spécialisé fourni par Eclipse :





Par exemple pour rajouter une permission :



22 Application Android : différents fichiers et répertoires

- `AndroidManifest.xml` ;
- `R.java` : contient les références, « ID », à différents éléments de l'application ;
- `build.xml` et `default.properties` : utilisé par Ant ;
- `bin/` : contient l'application une fois compilée ;
  - `bin/classes/` : les classes Java compilées ;
  - `bin/classes.dex` : l'exécutable créé à partir de ces classes ;
  - `bin/appli.ap_` : contient les ressources de l'application en fichier Zip ;
  - `bin/appli-debug.apk` ou `bin/appli-unsigned.apk` : l'application Android ; *Ce fichier* `.apk` *est signé électroniquement par une clé avec* `jarsigner`.
- `libs/` : les bibliothèques tierces nécessaires à l'application ;
- `assets/` : les fichiers multimédias et autres fichiers nécessaires à l'application ;
- `src/` : les sources de l'application ;
- `res/` pour les ressources :
  - `res/layout/` : les fichiers au format XML pour définir l'interface.
  - `res/drawable/` : les fichiers images pour l'interface ;
  - `res/menu/` : fichier de définition en XML des menus ;
  - `res/values/` : pour les chaînes de caractères, les constantes (permet de facilement internationaliser une application) ;
  - `res/xml/` : pour des fichiers au format XML.

Pour ajouter un fichier de n'importe quel type à l'application :

- le mettre dans le répertoire `res/raw` ;
- synchroniser l'IDE Eclipse :
  - ◇ dans le menu contextuel du répertoire sélectionner l'option *refresh* ;
  - ◇ l'IDE va créer les références manquantes dans le fichier R.java.

Attention

Chaque fichier doit avoir un nom unique en excluant son extension : toto.jpg et toto.dat vont entrer en conflit.

Pour référencer une ressource

Il faut utiliser la méthode `getResources().openRawResource(R.raw.ma_ressource);` où `ma_ressource` correspond au nom du fichier.

Il est possible de disposer d'un système de gestion de préférences dont l'accès est partagé entre les différents éléments constituants l'application.
Pour accéder aux préférences, on utilisera la méthode `getSharedPreferences()`.

Ces préférences sont organisées sous forme d'objet de la classe `SharedPreferences` et chacun de ces objets peut recevoir un nom.
Ensuite, pour un de ces objets on peut définir un ensemble d'associations de type (clé, valeur).

```
1 SharedPreferences reglages = getSharedPreferences("MesPrefs", MODE_PRIVATE);
2 SharedPreferences.Editor editeurPref = reglages.edit();
3 editeurPref.putString("Nom", "toto");
4 editeurPref.putBoolean("Inscrit", true);
5 editeurPref.commit();
```

Pour récupérer les valeurs de ces préférences :

```
1 SharedPreferences reglages = getSharedPreferences("MesPrefs", MODE_PRIVATE);
2 String nom = reglages.getString("Nom", "Titi");
```

Dans la ligne 2, le deuxième argument de la méthode `getString` *est la valeur à retourner dans le cas où il n'y aurait pas de valeur associée à la clé « Nom ».*

Nommer l'application

On ajoutera dans le *manifest* :

```
1 <application android:label="Le nom de mon application">
```

Ou en faisant appel à une ressource :

```
1 <application android:label="@string/app_name">
```

L'application portera comme nom, le contenu de la chaîne appelée `app_name` *et définie dans le fichier* `strings.xml`

Associer une icone à l'application

On ajoutera dans le *manifest* :

```
1 <application android:icon="@drawable/mon_icone">
```

Pour définir l'icone, il faut :

- définir cette icone au format PNG suivant 3 tailles différentes : 36x36, 48x48 et 72x72 suivant la densité de l'écran qui peut être basse, moyenne ou grande ;
- stocker la version 36x36 dans `res/drawable-ldpi/` sous le nom `mon_icone.png`
- stocker la version 48x48 dans `res/drawable-mdpi/` sous le nom `mon_icone.png`
- stocker la version 72x72 dans `res/drawable-hdpi/` sous le nom `mon_icone.png`

Android choisira automatiquement la taille la mieux adaptée.

Fournir une description de l'application

Cette description peut être utilisée dans le *MarketPlace* :

```
1 <application    android:label="le nom de l'application"
2              android:description="@string/app_desc">
```

Pour déboguer l'application

Pour pouvoir déboguer l'application dans Eclipse, il est nécessaire d'ajouter l'attribut `android:debuggable="true"` dans l'élément `application` dans le *manifest*.
Lors de la diffusion de l'application, cet attribut doit être modifié.

26 Journalisation des erreurs

Android définit la classe `android.util.Log` et les méthodes suivantes :

- `Log.e()` Logs errors
- `Log.w()` Logs warnings
- `Log.i()` Logs messages de description
- `Log.d()` Logs messages de déboguage
- `Log.v()` Logs messages détaillés

Chaque message de déboguage doit être étiquetté avec un *TAG* :

```
1 private static final String TAG = "MonApplication";
2 ...
3 Log.i(TAG, "Dans la methode Truc()");
```

Le TAG permet ensuite de filtrer au niveau du débogueur, LogCat d'Eclipse, les messages que l'on veut suivre.

À qui cela correspond ?

Une application Android est constituée de composants logiciels faiblement couplés qui correspondent chacune à une activité indépendante.
Parmi ces activités, l'une est plus importante que les autres : elle apparait dans le « launcher ».
On utilise un message, un *intent* pour demander au système de déclencher une activité.
Cette « intention » peut être plus ou moins spécifique, ce qui laisse une plus grande liberté pour le choix de ou des activités qui vont la satisfaire.

Analogie avec le Web

Une analogie existe avec le protocole HTTP : un verbe + URL vers une ressource (page web, image, programme serveur). Le verbe correspond à GET, POST *etc.*

Un **intent** représente une action + un contexte et il existe plus d'actions et plus de composants pour les réaliser.
À la manière du navigateur web qui sait traiter le verbe+URL, Android sait trouver l'activité correspondante.

Intent	action + données
Données	les données sont indiquées sous forme d'une URI, comme `content://contacts/people/1`
Action	les actions sont des constantes, comme `ACTION_VIEW` qui déclenche une visionneuse, `ACTION_EDIT` pour éditer une ressource, `ACTION_PICK` pour sélectionner un élément suivant une URI représentant une collection comme `content://contacts/people`.

Si on crée un intent avec l'action `ACTION_VIEW` *et l'URI* `content://contacts/people/1` *: Android va chercher et ouvrir une activité capable d'afficher cette ressource.*

D'autres critères peuvent être ajoutés dans l'intent

categorie	l'activité principale sera dans la catégorie `LAUNCHER`, mais les autres activités seront dans `DEFAULT` ou `ALTERNATIVE`.
type MIME	pour indiquer le type de ressources sur lequel l'action va être faite
un component	pour indiquer la classe de l'activité qui est supposée recevoir l'intent. *Réduit la portée de l'intent mais permet de simplifier le contenu de l'intent.*
Extras	permet d'ajouter des données à transmettre au récepteur.

On indique quels sont les intents attendus par un composant dans le fichier manifest, à l'aide d'`intent-filter` :

```
1 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
2 package="com.commonsware.android.skeleton"> <application>
3 <activity android:name=".GO" android:label="Go">
4 <intent-filter>
5    <action android:name="android.intent.action.MAIN" />
6    <category android:name="android.intent.category.LAUNCHER" />
7 </intent-filter>
8 </activity> </application></manifest>
```

Rendre plus spécifique le récepteur

Lorsque le système génère des événements, il est possible de déclencher :

- un service plutôt qu'une activité ;
- une activité suivant un contexte spécifique : le même intent sur une certaine donnée déclenche une activité mais sur une autre donnée déclenche une autre activité.

On utilise des **récepteurs d'intent** implémentant l'interface `BroadcastReceiver`.

Ces récepteurs sont des objets qui reçoivent des intents, surtout diffusés en broadcast, et qui réagissent en envoyant d'autres intents pour déclencher une activité, un service...

Elle est constituée de seulement une méthode : `onReceive()`.

Les objets implémentants cette interface doivent être déclarés dans le manifest :

```
1 <receiver android:name=".MaClasseRecepteur" />
```

Cet objet **n'est exécuté que jusqu'au retour de la méthode** et sera enlever du système : ils ne peuvent attendre de *callback* (pas d'ouverture de *dialog* par exemple).

Exception de traitement

Si on veut que le récepteur soit attaché à un composant logiciel avec une durée de vie plus longue, comme celle du système :

- il ne faut pas déclarer le récepteur dans le manifest ;
- il faut enregistrer le récepteur à l'aide de la méthode `registerReceiver()` lors du callback `onResume()` de l'activité ;
- il faut dé-enregistrer le récepteur à l'aide de la méthode `unregisterReceiver()` lors du callback `onPause()` de l'activité ou lorsque l'on ne désire plus recevoir d'intent associé.

Attention

L'utilisation d'objet Intent pour passer des données quelconques n'est possible que si le récepteur est actif (possibilité de manquer des intents quand inactif).

30 L'interface graphique

Il existe deux manières de définir une interface graphique :

- à l'aide de fichier de description rédigé suivant le format XML
  - ◇ avantages : simplicité de conception, indépendance avec le code Java ;
  - ◇ inconvénients : définition statique qui ne peut évoluer au cours de la vie de l'application ;
- avec du code Java
  - ◇ avantages : peut évoluer de manière dynamique durant la vie de l'application (exemple : définir le contenu d'un menu permettant de choisir un élément parmi une liste construite par l'application) ;
  - ◇ inconvénients : difficultés d'organisation, le code est mélangé à celui de l'application.
- *tiré parti des deux...*
- *et aussi, utiliser une interface au format HTML avec le widget basé sur WebKit*

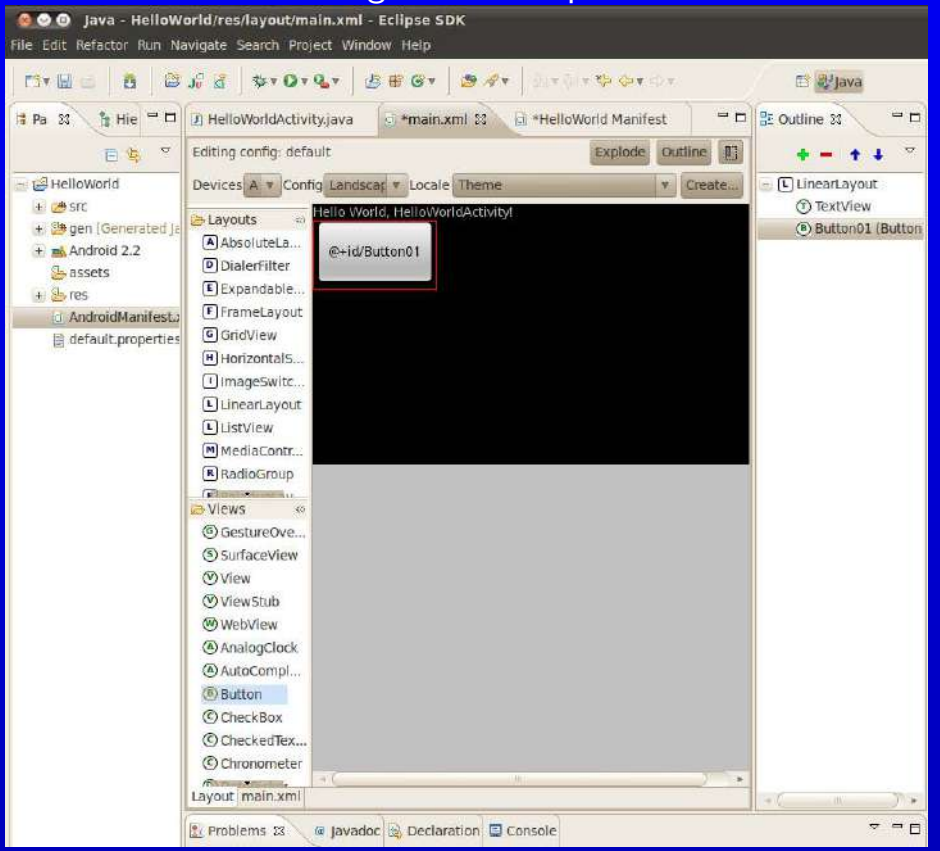
31 Utilisation d'XML

Les fichiers au format XML :

- sont stockés en tant que ressource dans : `res/layout` ;
- définissent une hiérarchie entre les *widgets* :
  - ◇ un arbre d'éléments de widgets et de *containers* ;
  - ◇ les attributs de ces éléments définissent les propriétés :
    - ⋆ l'aspect d'un widget comme celui d'un bouton avec `textStyle="bold"`
    - ⋆ le comportement d'un container.
  - ◇ l'outil `aapt` *parse* ces fichiers et construit le fichier `R.java` contenant des références pour désigner un widget dans le code de l'application.

32 Utilisation d'Eclipse

Il est possible d'utiliser l'éditeur intégré dans Eclipse :



33 Le fichier XML correspondant

Ce fichier définie une arborescence :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
<Button android:text="@+id/Button01" android:id="@+id/Button01"
   android:layout_width="wrap_content" android:layout_height="wrap_content">
   </Button>
</LinearLayout>
```

Sur cet exemple, les widget :

- sont de la classe `Button` et `TextView` qui héritent de la classe `android.view.View` ;
- disposent d'un identifiant avec l'attribut `android:id` ;
- affichent à l'initialisation le texte qui est définie par l'attribut `android:text`.
  Pour le `TextView` *on remarque qu'il fait référence à une chaîne définie dans le fichier* `strings.xml` *et référencée par* `"@string/hello`.
- les attributs `android:layout_width` et `android:layout_height` sont définis à `fill_parent` ;

À Quoi correspondent les « @ » ? À définir une identité pour une référence ultérieure : « `@+id/mon_id` »

Le fichier `R.java` contient des références sur les éléments de l'interface :

```
1  /* AUTO-GENERATED FILE.  DO NOT MODIFY.
2   *
3   * This class was automatically generated by the
4   * aapt tool from the resource data it found.  It
5   * should not be modified by hand.
6   */
7  package edu.unilim.helloworld;
8  public final class R {
9     ...
10     public static final class layout {
11         public static final int main=0x7f030000;
12     }
13   public static final class id {
14       public static final int Button01=0x7f050000;
15     }
16 }
```

- la méthode `setContentView` permet d'appeler l'interface définie dans un fichier XML : `setContentView(R.layout.main)`
- on utilise la méthode `findViewById()` à laquelle on passe l'identifiant numérique défini dans le fichier `R.java`.
  Exemple : `Button btn = (Button) findViewById(R.id.button01)`
  On utilise le nom définie dans le fichier XML d'interface, ici « button01 ».
- on accroche un *listener* : `btn.setOnClickListener(...)` pour obtenir l'événement (clic du bouton).

Il existe différentes classes et sous-classes :

- `ImageView` : avec les attributs `android:src`, comme par exemple : `android:src="@drawable/logo"`
  Il est également possible d'utiliser la méthode `setImageURI()` `pour utiliser un Content Provider`
- `ImageButton`
- `EditText` : pour une zone de saisie. Avec les attributs :
  - `android:numeric` pour ne rentrer que des chiffres ;
  - `android:singleLine="false"` pour indiquer si la saisie se réalise sur une ligne ou plus ;
  - `android:password` : pour l'entrée masquée d'un mot de passe.

Lorsque l'on veut intéragir avec l'utilisateur de manière rapide et simple, sans avoir à créer une activité dédiée, on peut créer une fenêtre de *Dialog* :

- `Activity.showDialog()` : afficher un dialogue ;
- `Activity.onCreateDialog()` : une fonction de callback lors de la création du dialogue et son ajout à la liste des autres dialogues de l'activité ;
- `Activity.onPrepareDialog()` : permet de mettre à jour un dialogue juste avant son affichage ;
- `Activity.dismissDialog()` : ferme un dialogue et retourne à l'activité ;
- `Activity.removeDialog()` : supprime le dialogue de la liste des dialogues.

On peut définir l'apparence du dialogue à l'aide :

- d'un fichier *layout* au format XML ;
- la méthode `setContentView()`.

Il existe des dialogues prédéfinis : `AlertDialog`, `CharacterPickerDialog`, `DatePickerDialog`, `ProgressDialog` et `TimePickerDialog`.

On peut récupérer les références vers les éléments de l'interface à l'aide de `Dialog.findViewById()`.

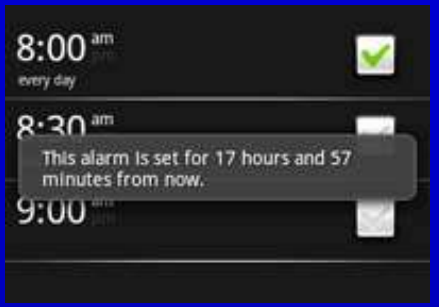
37 Notifications Toast

C'est un message qui va venir en premier plan et disparaitre automatiquement après un temps limité.

```
1  Context context = getApplicationContext();
2  CharSequence text = "Hello toast!";
3  int duration = Toast.LENGTH_SHORT;

5  Toast toast = Toast.makeText(context, text, duration);
6  toast.show();
```

Ou plus simplement :

```
1  Toast.makeText(context, text, duration).show();
```



Avec Bluetooth, il est possible de :
- chercher d'autres appareils, *devices*, à proximité ;
- établir une communication de type flux à l'aide d'une socket bluetooth.

Il existe 4 classes pour gérer le bluetooth :
- `BluetoothAdapter` : représente l'interface matérielle bluetooth, le *bluetooth device* sur lequel tourne Android ;
- `BluetoothDevice` : représente chaque appareil distant avec lequel on peut communiquer, *remote bluetooth device* ;
- `BluetoothSocket` : permet de déclencher une requête de connexion vers un appareil distant et d'établir une communication ;
- `BluetoothServerSocket` : permet de définir un serveur en attente de connexions en provenance des appareils distants.

Pour accéder à l'interface bluetooth :

```
1 BluetoothAdapter bluetooth = BluetoothAdapter.getDefaultAdapter();
```

Il faut également les permissions suivantes dans le *manifest* :

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

1. pour déclencher une découverte du voisinage, *discovery*, établir des communications
2. pour modifier les propriétés de l'interface.

39 Bluetooth : mise en activité

Par défaut, l'interface Bluetooth n'est pas activée pour économiser la batterie, et pour des questions de sécurité.

Il est nécessaire de l'activer :

— soit avec la méthode `enable()`, si l'on dispose de la permission `BLUETOOTH_ADMIN` ;

— soit à l'aide d'une sous-activité asynchrone dont on attendra le résultat :

```
1 String enableBT = BluetoothAdapter.ACTION_REQUEST_ENABLE;
2 startActivityForResult(new Intent(enableBT), 0);
```

L'utilisateur sera prévenu et une demande d'autorisation lui sera affichée.

Dans le premier cas, il vaut mieux prévenir l'utilisateur de la mise en activité de l'interface.

L'utilisation de Bluetooth se fait au travers d'un **service**.
Pour obtenir une référence vers ce service, il faut utiliser la méthode `getSystemService` de la classe `Context` qui permet d'accéder au système :

```
1 String nom_service = Context.BLUETOOTH_SERVICE;
2 BluetoothDevice bluetooth = (BluetoothDevice)getSystemService(nom_service);
```

L'objet BluetoothDevice

Ensuite, il est possible d'utiliser les méthodes suivantes :
- `enable` et `disable` : activer ou désactiver l'interface ;
- `getName` et `setName` : pour manipuler le nom de l'interface (si elle est active) ;
- `getAddress` : pour obtenir l'adresse de l'interface ;
- `get/setMode` et : pour configurer le mode de découverte « `discoverable` » ou « `connectable` »
- `get/setDiscoverableTimeout` : pour configurer le temps utilisé pour cette découverte ;

```
1 bluetooth.setName("YOBIBO");
2 bluetooth.setMode(BluetoothDevice.MODE_DISCOVERABLE);
```

Pour la découverte, *discovery*, des autres appareils bluetooth, il faut :

- être en mode découverte ;
- démarrer un cycle de découvrable :
  - `bluetooth.startPeriodicDiscovery();`
  - `bluetooth.startDiscovery(true);`

  Ces deux méthodes sont asynchrones, et diffusent un intent `REMOTE_DEVICE_FOUND_ACTION` *à chaque découverte d'un appareil bluetooth.*
- obtenir la liste des appareils découverts : `listRemoteDevices` qui retourne un tableau de chaîne de caractères, contenant les adresses de chacun de ces appareils ;
- obtenir le nom d'un appareil : `getRemoteName` en donnant en paramètre l'adresse de l'interface.

Pour le pairing

Il faut utiliser les méthodes suivantes :

- `createBonding` : pour établir le lien (`cancelBonding` pour l'abandonner) ;
- `setPin` : pour indiquer le PIN de sécurité à utiliser (`cancelPin` pour le redéfinir).

Une fois un lien établie avec un appareil, ce lien est ajouté à la BD de l'appareil et sera utilisé de nouveau lors d'une prochaine rencontre du même appareil.

```
1  String[] devices = bluetooth.listRemoteDevices();
2  for (String device : devices) {
3  if (!bluetooth.hasBonding(device)) {
4  // Set the pairing PIN. In real life it's probably a smart
5  // move to make this user enterable and dynamic.
6  bluetooth.setPin(device, new byte[] {1,2,3,4});
7  bluetooth.createBonding(device, new IBluetoothDeviceCallback.Stub() {
8     public void onCreateBondingResult(String address, int result) throws RemoteException{
9        if (result == BluetoothDevice.RESULT_SUCCESS) {
10          String connectText = "Connected to " + bluetooth.getRemoteName(address);
11          Toast.makeText(getApplicationContext(), connectText, Toast.LENGTH_SHORT);
12          }
13       }
14    public void onEnableResult(int _result) throws RemoteException {}
15    });
16    }
17 }
```

Pour écouter les requêtes de pairing, il faut définir un `BroadcastListener` *filtrant les intents* `ACTION_PAIRING_REQUEST`.

41 Surveiller la découverte

```
1  BroadcastReceiver discoveryMonitor = new BroadcastReceiver() {
2  String dStarted = BluetoothAdapter.ACTION_DISCOVERY_STARTED;
3  String dFinished = BluetoothAdapter.ACTION_DISCOVERY_FINISHED;
4  @Override
5  public void onReceive(Context context, Intent intent) {
6      if (dStarted.equals(intent.getAction())) {
7        // Discovery has started.
8        Toast.makeText(getApplicationContext(),
9                       "Discovery Started...", Toast.LENGTH_SHORT).show();
10     }
11     else if (dFinished.equals(intent.getAction())) {
12       // Discovery has completed.
13       Toast.makeText(getApplicationContext(),
14                      "Discovery Completed...", Toast.LENGTH_SHORT).show();
15     }
16   }
17 };
18 registerReceiver(discoveryMonitor, new IntentFilter(dStarted));
19 registerReceiver(discoveryMonitor, new IntentFilter(dFinished));
```

Il faut modifier le fichier `AndroidManifest.xml` pour rajouter la permission d'utiliser l'accès à Internet (en plus de celle d'accès à bluetooth et à son administration) :

```
1 <uses-permission android:name="android.permission.INTERNET">
2 </uses-permission>
```

En effet, la simulation de bluetooth passe par l'utilisation de la pile TCP/IP...

Modifications du source de l'application

Pour l'accès à la bibliothèque, il faut :

- changer l'importation :
  `import android.bluetooth`⇒`import dk.itu.android.bluetooth`
- appeler `BluetoothAdapter.SetContext(this);` par exemple dans la méthode `onCreate()` appelé lors de l'initialisation de l'*activity*.

Lancer le serveur de liaison

Dans Eclipse, il faut lancer `btemu` ou directement sur la ligne de commande :

```
$ java -jar btsim-server.jar
Server started on port 8199
ADB command: adb
press any key to exit
```

Ce serveur TCP/IP s'occupe de faire la liaison entre les différents AVDs.

43 Utilisation des fonctions de hashage

On peut faire appel aux fonctions de hashage MD5 et SHA1 :

```
1    // Obtain a message digest object.
2    MessageDigest md = MessageDigest.getInstance("MD5");
3    // Calculate the digest for the given bytes array
4    md.update(saisie.getText().toString().getBytes(), 0, saisie.length());
5    val_hash = new BigInteger(1,md.digest());
6    mon_texte.setText(String.format("%1$08X", val_hash));
```

Remarques :

- ligne 4 :
  - la variable `saisie` correspond à un *widget* de type `EditText` ;
  - dans la méthode `getBytes()`, il est possible d'indiquer un encodage comme `"ISO-8859-1"` ou `"UTF-8"` pour traduire les éventuels caractères accentués ;
- ligne 5 : on traduit la valeur retournée par la fonction de hashage en un entier de taille variable, `BigInteger` ;
- ligne 6 : ce grand entier est ensuite affiché sous forme hexadécimal, puis affecté au *widget* de type `TextView` appelé `mon_texte`.

44 Utilisation de clés RSA nouvellement créées

```
1 // Get an instance of the RSA key generator
2 KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
3 // Generate the keys - might take sometime on slow computers
4 KeyPair myPair = kpg.generateKeyPair();
```

45 Utilisation de clés RSA déjà créées

Il est possible de récupérer des clés RSA à l'intérieur d'une application Android :

- les convertir dans une représentation DER, c-à-d. en notation ASN.1 ;
- placer les fichiers les contenant dans le répertoire `res/raw` ;
- utiliser la méthode `getResources()` pour obtenir des références vers ces fichiers ;
- utiliser la classe `KeyFactory` pour importer les clés.

```
1  KeyFactory kf = KeyFactory.getInstance("RSA");
2  InputStream fic_cle = getResources().openRawResource(R.raw.ma_cle_publique);
3  byte[] cle_pub_asn1 = new byte[fic_cle.available()];
4  fic_cle.read(cle_pub_asn1);
5  fic_cle.close();
6  X509EncodedKeySpec cle_pub = new X509EncodedKeySpec(cle_pub_asn1);
7  kf.generatePublic(cle_pub);

9  fic_cle = getResources().openRawResource(R.raw.ma_cle_privee);
10 byte[] cle_priv_asn1 = new byte[fic_cle.available()];
11 fic_cle.read(cle_priv_asn1);
12 fic_cle.close();
13 PKCS8EncodedKeySpec cle_privee = new PKCS8EncodedKeySpec(cle_priv_asn1);
14 kf.generatePrivate(cle_privee);
```