Épisode 2 : Les outils de développement

© Alain Menu – édition 2.1, septembre 2013

Objectifs : Mise en œuvre d'une chaine de développement pour cibles Android.

Table des matières

2.1. Mise en œuvre d'une chaine de développement

Pour développer des applications Android, il faut au moins disposer du SDK et du kit de développement Java ; et pour le confort, un IDE tel qu'Eclipse est le bienvenu (d'autant qu'Android propose un *plugin* spécifique pour cet environnement) !

Ces outils sont disponibles aussi bien sous Linux que sous Windows et Mac OS (sur processeur Intel). Les applications Android étant exécutées par une machine virtuelle, il n'y a pas d'avantages particuliers à développer sur un système plutôt qu'un autre...

Le guide d'installation qui suit a été testé sur des machines Linux Ubuntu et Mac OS.

2.1.1. Composants requis

Java SDK

JRE (*Java Runtime Environement*) est le kit destiné au client pour pouvoir exécuter un programme Java. Il se compose essentiellement d'une machine virtuelle Java (JVM) capable d'exécuter le *bytecode* et les bibliothèques standard de Java.

Le kit destiné au programmeur, appelé avant JDK (*Java Development Kit*) et renommé depuis la version 1.2.2 en SDK (*Standard Development Kit*), est composé d'un JRE, d'un compilateur, de nombreux programmes utiles, d'exemples de programmes Java et des les sources de toutes les classes de l'API.

Le développement Android nécessite le SDK Java en version 5 ou supérieure, le JRE seul n'est pas suffisant. Si Java est déjà installé sur le poste, sa version peut être vérifiée en ligne de commande par : java -version

Exemple de résultat sous Linux (c'est ici une version 6 update 20) :

```
java version "1.6.0_20"
OpenJDK Runtime Environment (6b20-1.9.2-0ubuntu1~10.04.1)
OpenJDK Server VM (build 19.0-b09, mixed mode)
```

OpenJDK est une alternative libre du langage Java défini par Sun (maintenant Oracle)...

Adresse de téléchargement :http://www.oracle.com/technetwork/java/javase/downloads/index.html

Android SDK Tools

Ce kit de développement est un ensemble complet d'outils dédiés au développement d'applications Android. Il comprend notamment un débogueur, un émulateur, un moniteur temps réel, ... Ces outils sont disponibles en ligne de commande.

IDE Eclipse

Eclipse est un IDE (*Integrated Development Environment*) open-source particulièrement populaire dans le monde du développement Java.

Une version 3.6.2 (Helios) ou supérieure est nécessaire, le plugin JDT doit être ajouté (ce qui est fait par défaut pour la majorité des distributions de l'EDI).

ADT plugin pour Eclipse

ADT (*Android Development Tools*) est un module additionnel pour Eclipse qui étend les capacités de celui-ci afin de faciliter le développement Android (création, tests et débogage des applications,...). Il permet notamment d'intègrer dans l'EDI la majorité des outils du SDK Android :

- l'*Android Project Wizard*, assistant de création de projets,
- des éditeurs de manifestes, d'agencements et de ressources,
- la conversion en exécutables Android .dex, la création de fichiers de paquetage .apk et l'installation des ces fichiers dans des machines virtuelles Dalvik,
- le gestionnaire AVD (*Android Virtual Device*) qui permet de gérer les dispositifs d'hébergement virtuels,
- l'*Android Emulator* avec contrôle des connexions réseau et possibilité de simuler des appels entrants,
- le DDMS (*Dalvik Debug Monitoring Service*) comprenant la redirection de ports, la visualisation de la pile, du tas et des threads, le détail des processus et la possibilité de réaliser des captures d'écran,
- ... ...

2.1.2. Installation

Il n'est actuellement plus nécessaire d'installer séparément les différents composants ci-dessus (en dehors du SDK Java le cas échéant). Le site de développement officiel Android propose maintenant un paquetage complet (*bundle*) pour les systèmes hôtes Linux, Mac OS X et Windows :

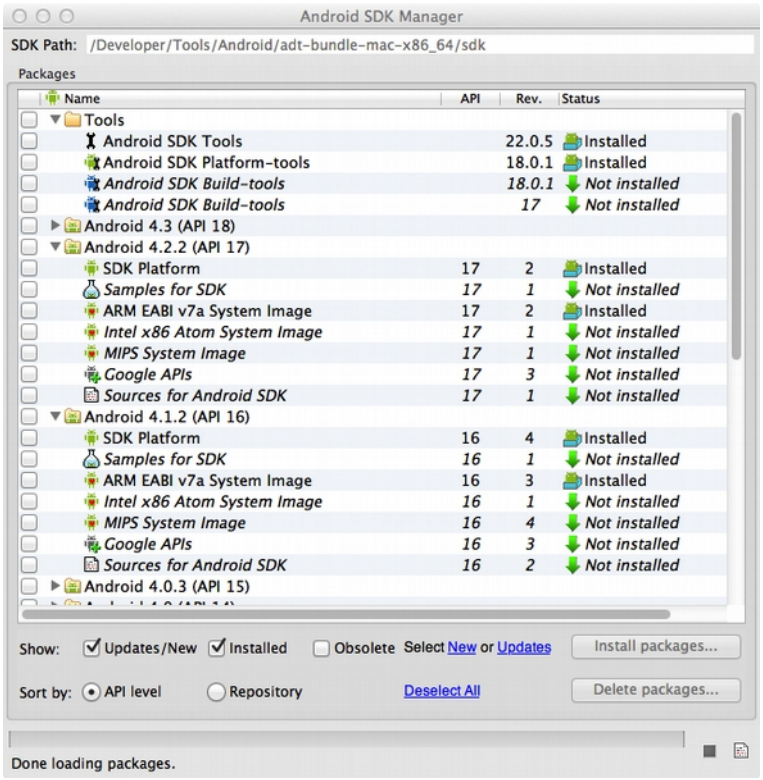
Adresse de téléchargement : http://developer.android.com/sdk/index.html

Décompresser l'archive dans un répertoire approprié (par exemple $HOME/devel/). L'application Eclipse est alors disponible dans adt-bundle-<*os_platform*>/eclipse/.

Au démarrage, Eclipse demande de choisir un répertoire de travail (*workspace*) où seront localisés les projets de développement.

2.1.3. SDK Manager

Le SDK Android est constitué de paquets modulaires qui peuvent être téléchargés séparément au moyen de l'outil SDK Manager. Cet outil est particulièrement pratique lorsqu'il est nécessaire de faire une mise à jour liée à une évolution du niveau d'API. Plusieurs niveaux d'API peuvent cohabiter.



Les paquets disponibles sont tous localisés dans le sous-répertoire adt-bundle-<*os_platform*>/sdk/.

Le SDK Manager est accessible depuis le menu principal d'Eclipse :

Window | Android SDK Manager.

L'installation de la documentation *offline* (proposée normalement pour le plus haut niveau d'API disponible) peut être une bonne chose...

Le point d'entrée de la documentation sur les classes Java est :

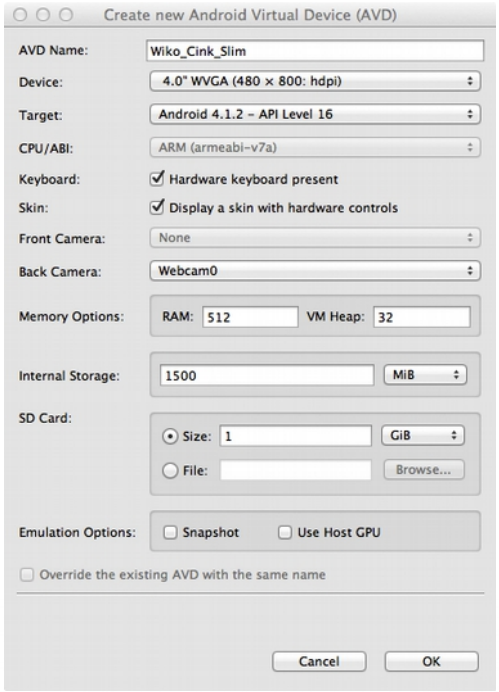
adt-bundle-<*os_platform*>/sdk/docs/reference/index.html

2.1.4. Création d'un AVD

Les applications peuvent (doivent ?) être testées sur un émulateur Android avant d'être réellement installées sur un androphone...

Il est donc nécessaire de créer au moins un AVD (*Android Virtual Device*) afin de définir les caractéristiques d'un émulateur :

- dans le menu principal d'Eclipse,
- sélectionner Window | Android Virtual Device Manager,
- choisir l'onglet Android Virtual Devices, puis cliquer sur New,
- la boite de dialogue de création d'un AVD apparaît,
- renseigner les différents champs de l'appareil à émuler (ici, un Wiko Cink Slim) →
- cliquer sur Ok.



Plusieurs AVD peuvent être créés sur le même principe, par exemple pour tester une application sous différentes version d'Android...

Sous Linux et Mac OS, les AVD sont sauvegardés dans $HOME/.android/avd/ ...

Un AVD peut aussi être créé en ligne de commande grâce à l'outil android du sous-répertoire tools :

→ `android create avd -target <target> --name <name>`

2.1.5. Préparation de l'androphone

On suppose ici disposer d'un smartphone Android réel...

Sur l'appareil, aller dans Paramètres | Applications | Développement, ou pour les versions plus récentes dans Paramètres | Options pour les développeurs :

→ activer le débogage USB.

Sous Linux, il est nécessaire de créer un fichier de règles qui contient une configuration USB pour chaque type de périphérique réel.

En tant que *root*, créer le fichier /etc/udev/rules.d/51-android.rules et y inscrire la ligne suivante (l'exemple donné utilise la valeur hexadécimale 0bb4 qui est le *USB vendor ID* du fabricant HTC, utilisé aussi semble-t-il par Wiko...) :

```
SUBSYSTEM=="usb", SYSFS{idVendor}=="0bb4", MODE="0666"
```

Adapter les droits sur le fichier, exécuter pour cela la commande :

→ `chmod a+r /etc/udev/rules.d/51-android.rules`

La commande Linux `lsusb` peut être très utile pour déterminer le *vendor ID* lorsque un appareil est branché...

USB vendor Ids :

Acer	0502
Dell	413c
Foxconn	0489
Garmin-Asus	091e
HTC	0bb4
Huawei	12d1
Kyocera	0482
LG	1004
Motorola	22b8
Nvidia	0955
Pantech	10a9
Samsung	04e8
Sharp	04dd
Sony Ericsson	0fce
ZTE	19d2

Brancher le téléphone et vérifier qu'il est reconnu. Le répertoire plateform-tools du SDK Android contient un utilitaire nommé adb (*Android Debug Bridge*). Cet outil permet de lister les périphériques connectés (y compris les AVD...) :

→ `./adb devices`

2.2. Le traditionnel « Hello World »

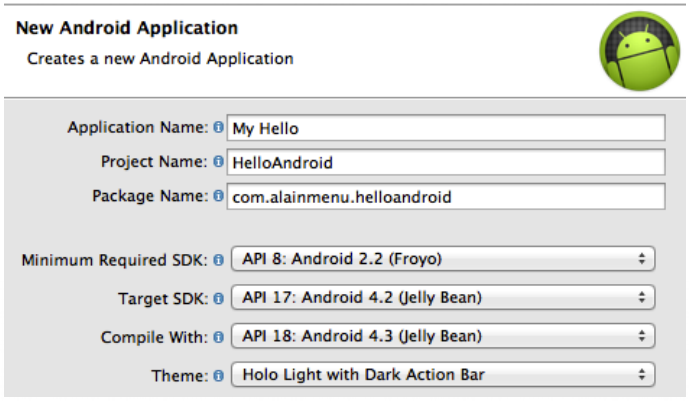
Il s'agit maintenant de tester la chaine de développement précédemment installée avec une application minimale de type « Hello World »...

2.2.1. Création d'un nouveau projet

Démarrer Eclipse et lancer File | New | Android Application Project...

Remplir les champs de la boite de dialogue :

- Application name : nom « convivial » de l'application, celui qui apparaitra sur le smartphone...
- Project name : nom du projet Eclipse (répertoire dans le dossier workspace défini lors du démarrage de l'EDI)
- Package name : nom de package Java qui doit être unique, ici peut être utilisé le *namespace* par défaut pour les exemples de la documentation (com.example), mais cet espace ne permettra pas une publication sur Google Play



- Minimum required SDK : version minimale de SDK (*API level*) qui pourra être utilisée par l'application
- Target SDK : version la plus récente du SDK sur laquelle l'application a été testée
- Compile with : version de la librairie avec laquelle compiler l'application, choisir la version la plus élevée disponible
- Theme : style de l'interface utilisateur

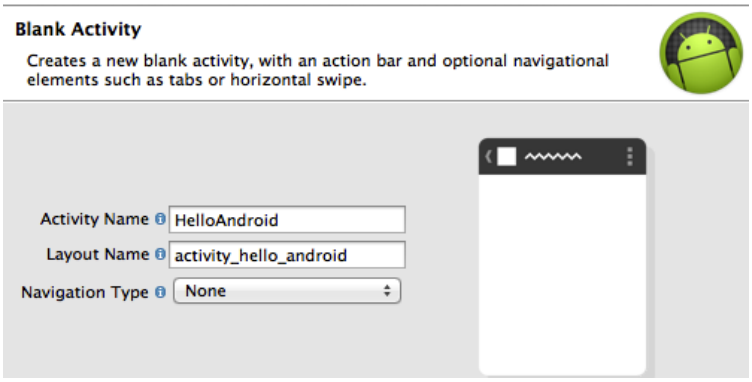
(*note : pour ces 4 derniers champs, les valeurs proposées par défaut doivent convenir...*)

Cliquer sur Next, conserver les valeurs par défaut (Create custom launcher icon, Create activity, Create Project in Workspace, ...). Cliquer à nouveau sur Next...

La fenêtre suivante permet de sélectionner l'icône de l'application. Celui proposé par défaut convient parfaitement pour ce premier exercice.

Conserver toutes les valeurs proposées par défaut et cliquer (encore) sur Next.

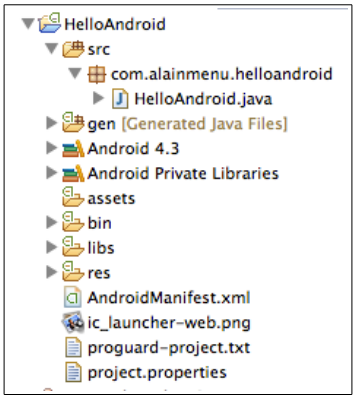
Valider la proposition de modèle d'activité (Blank Activity) et cliquer (une dernière fois) sur Next.



Choisir pour finir le nom de l'activité principale de l'application (classe qui étend la classe Activity) et le nom de la ressource de construction de l'interface utilisateur (fichier XML). Les valeurs proposées par défaut peuvent là encore être conservées.

Cliquer sur Finish.

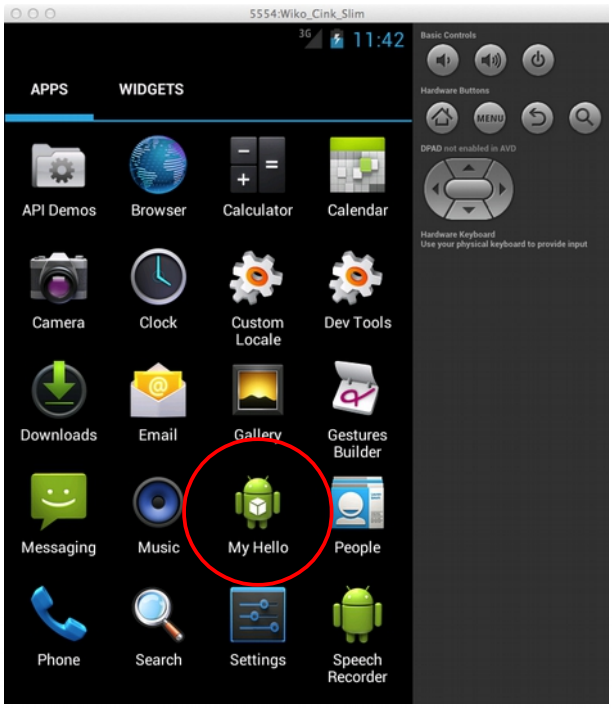
Le projet est créé...
il doit apparaître dans le panneau gauche Package Explorer d'Eclipse →



Le modèle de projet créé n'est pas vide, il implémente par défaut le traditionnel message « Hello World »...

2.2.2. Premiers tests d'exécution

Rien de plus simple !
Cliquer sur Run | Run As... Android Application.
L'AVD défini précédemment démarre (le boot peut être assez long...) et l'application s'ajoute à la liste des programmes du périphérique →



Et pour tester l'application sur un androphone réel ?
Si le travail de préparation exposé au chapitre précédent a été effectué, il suffit tout simplement de relier l'appareil au poste de développement via le câble USB...
La commande Run va installer le programme directement sur l'appareil connecté si l'AVD n'est pas en marche, ou proposer le choix entre les deux sinon...

Lors d'une demande d'exécution, le plugin ADT se charge donc :

- de compiler le projet et de le convertir en exécutable Android (.dex),
- de créer un paquetage Android (.apk) à partir de l'exécutable et des ressources externes,
- de démarrer l'appareil virtuel le cas échéant,
- d'installer l'application sur l'appareil cible,... et de la démarrer.

2.2.3. Comprendre le code

Éditer le source principal de l'application, localisé dans l'arborescence du projet sous HelloAndroid > src > com.alainmenu.helloandroid > HelloAndroid.java

Activity est la classe de base du composant visuel interactif de l'application.

La classe HelloAndroid étend cette classe et redéfinit la méthode onCreate().
L'appel setContentView() présente l'interface utilisateur en désérialisant une ressource de *layout* (arrangement géométrique).

La deuxième surcharge prépare la mise en place d'un menu déroulant d'options pour l'application.

```
package com.alainmenu.helloandroid;

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

public class HelloAndroid extends Activity {

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

	@Override
	public boolean onCreateOptionsMenu(Menu menu) {
		getMenuInflater().inflate(R.menu.hello_android, menu);
		return true;
	}

}
```

Les ressources du projet sont stockées dans le dossier res avec notamment les sous-dossiers drawable-xxx, layout et values. Elles sont référencées dans le code par la construction R.<*resourcefolder*>.<*resourcename*>.
L'icône de l'application est disponible sous différents formats pour s'adapter aux différentes résolutions d'appareils (avec en plus une version Web placée directement à la racine du dossier projet) :

ldpi :	36 x 36 pixels (*obsolète*)
mdpi :	48 x 48 pixels
hdpi :	72 x 72 pixels
xhdpi :	96 x 96 pixels
xxhdpi :	144 x 144 pixels

Le fichier res/layout/activity_hello_android.xml fournit les éléments de construction de l'interface utilisateur (UI) de l'application →

Les fichiers XML peuvent être ouverts en mode texte ou en mode graphique grâce aux onglets situés en bas de la fenêtre d'édition.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".HelloAndroid" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />

</RelativeLayout>
```

Le fichier res/values/strings.xml contient les définitions des chaines de l'application →

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">My Hello</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>

</resources>
```

Même si l'architecture de l'application semble un peu compliquée pour afficher un simple message, c'est la bonne démarche !

L'interface utilisateur pourrait en effet être construite directement dans le code comme ci-contre →

```
package com.alainmenu.helloandroid;

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

public class HelloAndroid extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TextView tv = new TextView(this);
        tv.setText("hello world !");
        setContentView( tv );
    }
}
```

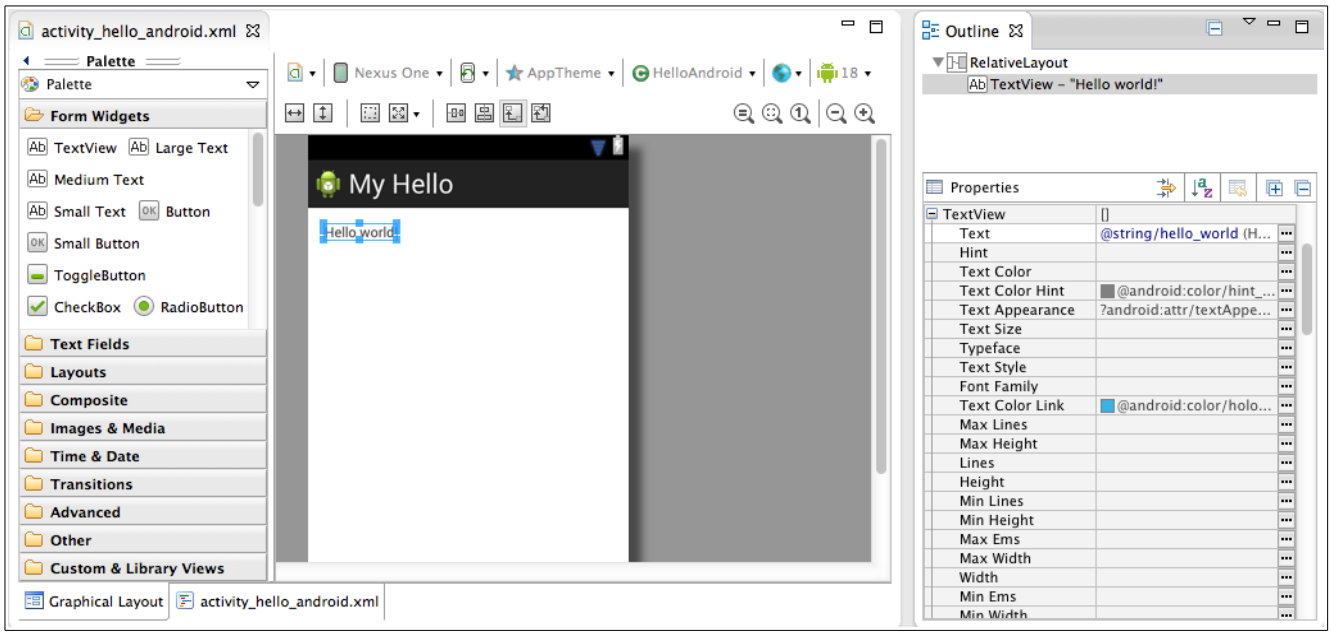
Le texte à afficher pourrait aussi être placé directement dans main.xml en écrivant :

```
android:text="hello world !"
```

Mais la structure proposée présente plusieurs avantages :

- la logique de l'application et sa présentation sont parfaitement découplées,
- le placement des éléments d'interface est beaucoup plus facile à traiter avec une arborescence XML que par de code,
- tous les textes sont centralisés, ce qui facilite l'internationalisation de l'application.

Une interface utilisateur Android est constituée d'une hiérarchie d'objets (bouton, image, label,...), chacun de ces objets correspond à une sous-classe de la classe View ; comme par exemple TextView pour les textes.
Les objets sont paramétrés par des propriétés facilement modifiables à partir de la vue graphique. La liste des propriétés de l'objet sélectionné apparaît dans la partie gauche :



Un élément défini en XML peut être facilement retrouvé dans le code, il suffit pour cela de lui attribuer un attribut id →

```
… …
<TextView
    android:id="@+id/myTxt"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/hello_world"
    />
… …
```

Le fragment suivant permet alors d'y accéder dans le code :

```
TextView myTxt = (TextView)findViewById(R.id.myTxt);
```

2.2.4. Accès à la documentation

Comment connaître les attributs et méthodes disponibles pour tel ou tel objet ?

Tout simplement en consultant la documentation du SDK sur le site http://developer.android.com/reference ! Le panneau de gauche donne une liste des points d'entrée de la doc. (espaces de noms) ; la page sélectionnée donne la liste des classes de l'espace, et chaque classe est ensuite détaillée par ses attributs et ses méthodes, héritées ou non.

La documentation Android est aussi accessible hors ligne, voir rubrique SDK Manager...

L'outil de recherche situé en haut à droite est bien utile... lorsque l'on connait le nom de ce que l'on cherche !

Eclipse propose également une aide contextuelle très aboutie : dans la vue Éditeur, pour un source java, il suffit de laisser le pointeur de la souris sur un mot (nom de classe, de méthode, ...) pour voir surgir une fenêtre contenant la page d'aide appropriée ; glisser ensuite le pointeur souris dans cette fenêtre pour pouvoir naviguer dans la page de documentation... cliquer sur un lien et la documentation apparaît dans un nouvel onglet de l'éditeur !

2.3. Outils de test et de débogage

2.3.1. Débogage d'un programme

Pour pouvoir déboguer une application, il faut d'abord valider un attribut dans la balise <application> du manifeste →

```
<application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:debuggable="true">
    … …
</application>
```

L'organisation par défaut de la fenêtre principale d'Eclipse utilisée est appropriée au développement : c'est la perspective « Java ».

Cette perspective comporte notamment un explorateur de paquetage, une zone d'édition multi-fichiers et la console de suivi de la chaine de développement.

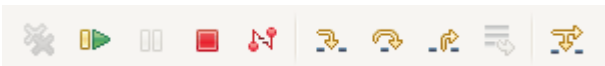
Pour le débogage, il convient d'ouvrir une nouvelle perspective nommée « Debug ».

La commande Run | Debug fait basculer l'environnement sous une autre perspective, la perspective « Debug ». Les vues Structure et Console de la perspective « Java » sont toujours présentes, mais de nouvelles vues spécifiques au débogage apparaissent :

- la vue Déboguer affiche sous la forme d'une arborescence, les différents processus en cours d'exécution ou terminés,
- la vue Variables affiche les variables utilisées dans les traitements en cours de débogage,
- la vue Points d'arrêt montre la liste des points d'arrêts définis dans l'espace de travail,
- la vue Expressions permet d'inspecter une expression en fonction du contexte des données en cours d'exécution,
- la vue Affichage permet d'afficher le résultat de l'évaluation d'une expression.

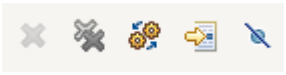
Un point d'arrêt est positionné/supprimé en double-cliquant dans la marge gauche de la ligne de code concernée.

Principales commandes de débogage :



- vue Déboguer
  - *Step into* [F5] permet de rentrer dans une méthode,
  - *Step over* [F6] permet de poursuivre l'exécution en mode pas à pas,
  - *Step return* [F7] permet de poursuivre l'exécution jusqu'à sortir de la méthode courante,
  - *Resume* [F8] permet de reprendre l'exécution normalement ou de s'arrêter uniquement si un autre point d'arrêt est rencontré.
- vue Variables
  - consultation des valeurs des variables visibles, une variable en rouge est une variable qui vient d'être modifiée
  - modification à chaud de la valeur d'une variable (par clic droit, puis Edit Value...).

- vue Points d'arrêt (*breakpoints*)
  - activation/désactivation des points d'arrêt sans les supprimer,
  - Le bouton [j!] permet de mettre un *breakpoint* sur toutes les exceptions du programme,
  - les propriétés d'un point d'arrêt sont accessibles par clic droit sur celui-ci, il est notamment possible de rendre conditionnelle sa prise en compte.

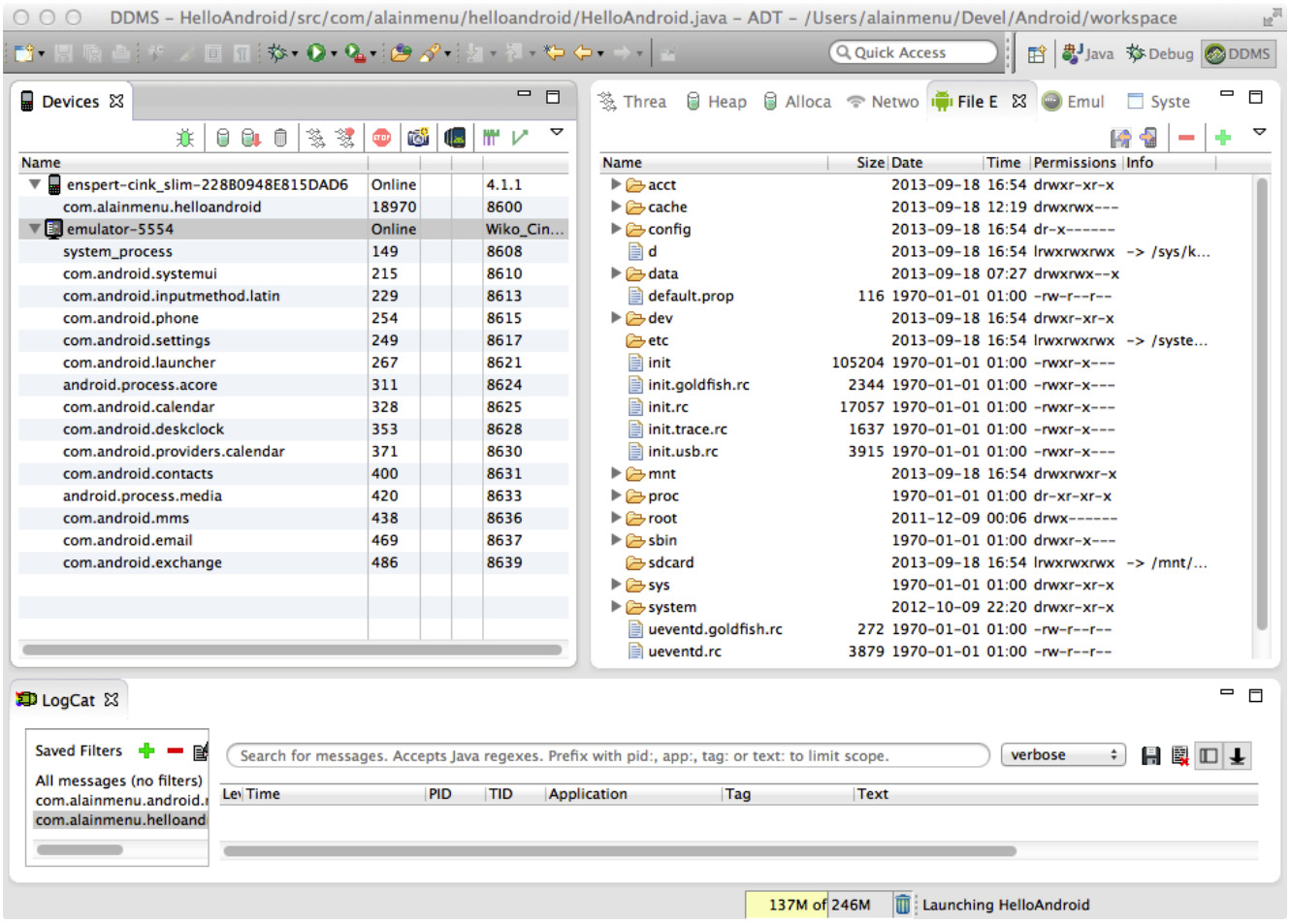


Le retour à la perspective « Java » se fait simplement en arrêtant le mode Debug, par exemple avec Resume (F8) et en sélectionnant la perspective souhaitée en haut à droite…

2.3.2. Dalvik Debug Monitor Service (DDMS)

Le plugin ADT apporte à Eclipse une autre perspective très intéressante : DDMS.

Sous Android, chaque application tourne dans son propre processus avec sa propre machine virtuelle Dalvik. DDMS permet, grâce à ADB, de connecter l'EDI aux applications en cours d'exécution sur un appareil (qu'il soit virtuel ou réel)…



La vue Devices montre les appareils disponibles, le bouton + situé à gauche permet de déployer la liste des tâches de l'appareil.

Lorsque un appareil réel ou non est sélectionné, DDMS permet de faire une capture d'écran qui peut ensuite être sauvegardée au format PNG. Il suffit de cliquer sur Screen Capture →

La vue Emulator Control permet d'envoyer et de configurer en temps réel la plateforme : vitesse du réseau, localisation, simuler l'envoi d'un appel, d'un sms, …

La vue LogCat est le journal de toutes les opérations et événements de l'appareil. Cette vue peut être agrandie par un double-clic.

La vue Threads donne la liste et l'état des threads de la VM sélectionnée. Cette liste est maintenue à jour en temps réel seulement si l'utilisateur le demande explicitement →

La vue Heap montre, sur demande, l'état de la zone d'allocation mémoire de la VM. Pour cela, activer la vue pour la tâche sélectionnée → 
puis cliquer sur Cause GC pour provoquer un *garbage collection* et mettre à jour la vue.

La vue Allocation Tracker permet de suivre les allocations de mémoire en cliquant sur Start Tracking puis Get Allocations.

La vue File Explorer donne accès au système de fichier de l'appareil sélectionné. DDMS permet de copier un fichier vers ou depuis le poste de développement, et autorise la suppression de fichiers sur l'appareil. Le *drag-and-drop* au sein de l'appareil est également permis.

2.3.3. Android Debug Bridge (ADB)

ADB est le véritable outil de gestion des appareils (réels ou non) sous Android. C'est une application client-serveur composée de trois parties :

- un client, qui tourne sur la machine de développement. Le plugin ADT et DDMS sont des clients ADB ; il est aussi possible d'en invoqué un en ligne de commande,
- un serveur, qui tourne en tâche de fond sur la machine de développement et qui assure la communication avec le démon présent sur l'émulateur ou l'androphone,
- un *daemon*, qui tourne en tâche de fond sur les appareils.

Quand un client ADB est démarré, il lance à son tour le serveur si celui-ci n'est pas présent. Le serveur écoute les requêtes des clients sur le port local TCP 5037.

Le serveur recherche les appareils disponibles (émulateurs et réels) sur les numéros de port impairs dans la plage 5555 .. 5585. Quand le serveur détecte un démon ADB, il établit une connexion sur le port concerné. Chaque instance d'appareil utilise une paire adjacente de numéros de port : un numéro pair pour la connexion console et un numéro impair pour la connexion ADB.

L'utilitaire ADB est présent dans le répertoire plateform-tools du SDK Android. ADB permet notamment de : démarrer/arrêter le serveur, vérifier les appareils disponibles, installer un paquetage (.apk), copier des fichiers de/vers un appareil, ouvrir un *shell* sur un appareil, ...

La documentation complète d'ADB est disponible à l'adresse :
http://developer.android.com/guide/developing/tools/adb.html

2.3.4. Utilisation des Log.x

Le SDK Android met à disposition du développeur la classe android.util.Log qui contient un certain nombre de méthodes d'aide au développement par injection de messages de journalisation. Ces messages peuvent être visualisés dans la vue LogCat du DDMS.

- Log.v : mode *verbose* (bavard), messages provisoires de traçage de code ;
- Log.d : mode *debug*, messages provisoires de débogage ;
- Log.e : mode *error*, messages d'erreur ;
- Log.w : mode *warning*, messages d'avertissement ;
- Log.i : messages d'information.

Toutes ces méthodes ont le même prototype : Log.*x* (String tag, String msg) ;

Les deux arguments seront retrouvés dans les colonnes homonymes de la vue LogCat en plus d'un horodatage et du PID du processus.

L'argument tag est à utiliser comme une signature de la portion de code d'où est émis le *log*. Il s'agit le plus souvent d'une constante définie en private static final, ou encore d'une référence au nom de la classe courante par getClass().getSimpleName().

Exemple : private static final String LOGTAG = "MyClass" ;

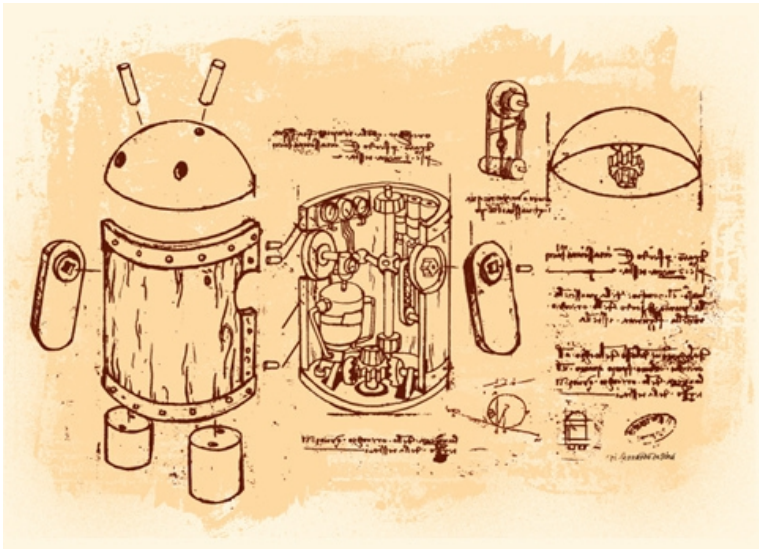
Attention à l'abus d'utilisation des *logs* ! Cela peut engendrer d'énormes ralentissements d'exécution des applications. Les méthodes Log.v et Log.d sont notamment proscrites pour les versions *release* des applications (par exemple pour publication sur l'*Android Market*), elles sont réservées aux phases de développement et doivent être supprimées ensuite.

Pour activer ou désactiver facilement l'émission des *logs*, la meilleure solution consiste sans doute à rendre l'appel aux méthodes Log.x conditionnel. Il est par exemple possible de définir une variable booléenne (globale ou pas...) →

```
static final boolean DEBUGMODE = true ;
… …
if ( DEBUGMODE ) {
    String msg = "My debug message number " + i ;
    Log.d(LOGTAG, msg ) ;
}
```

Cette technique est particulièrement efficace sachant que le compilateur Java est capable de ne pas générer le code de l'alternative précédente si la constante est fausse (donc l'objet String n'est pas non plus construit inutilement dans ce cas)...

Sur le même principe, il est évidemment possible de mettre en place plusieurs niveaux d'émission de messages de journalisation ; il suffit pour cela de gérer autant de drapeaux booléens et d'écrire les conditions en conséquence.



Android vu par Léonard de Vinci
(source : http://www.isteamandy.com/)

Références

Site Google officiel : http://www.android.com/
Site officiel de développement Android : http://developer.android.com/
Site officiel des sources Android : http://source.android.com/
Site de l'OHA : http://www.openhandsetalliance.com/
Communauté francophone autour d'Android : http://www.frandroid.com/,
et le site collaboratif associé : http://wiki.frandroid.com/wiki/Accueil
« Développement d'applications professionnelles avec Android 2 », Reto Meier – Pearson – ISBN : 978-2-7440-2452-8