IN01
Programmation Android

02 – La plateforme Android

Yann Caron
Avec l'aide de Jean-Marc Farinone

Sommaire - Séance 02

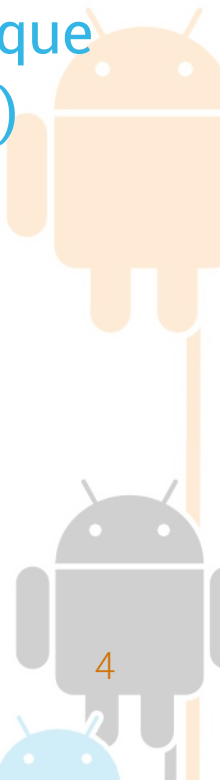
- Anatomie d'une projet Android : Manifest, Resources et qualifications, R.java, Internationalisation string.xml, Assets, Libraries
- Les composants : Activity, Service, Content Provider, Broadcast Receiver
- Cycle de vie d'une application et persistance
- Un premier projet
- Exécution et adb
- Logcat

IN01 – Séance 02

Anatomie d'un projet

Workflow

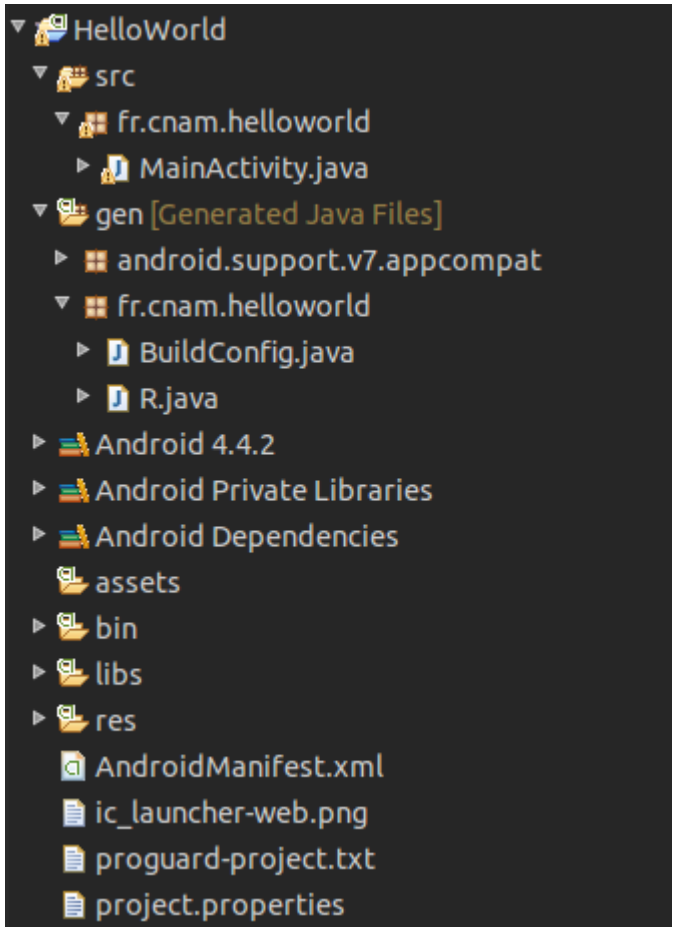
- Les phases sont :
  - Installation de l'environnement (setup)
  - Developpement (sources et ressources)
  - Déboguage et tests (DDMS et JUnit) :: Test Driven Developpment ?
  - Publication sur le Google Play (nécessite une clé cryptographique propre au fournisseur (nous), et un compte Google Developper)
- Plus de détails :: http://developer.android.com/tools/index.html



Concepts de base

- Les programmes sont écrits en Java (Ce n'est pas un hasard ! C'est un des langages les plus utilisés)
- ADK compile l'ensemble du projet (code Java, données, fichier de ressources, fichier XML) dans un package apk (une application)
- La JVM = stack based virtual machine
- La Dalvik = register based virtual machine
- Le Dexer converti les jars d'un bytecode à l'autre (et ça marche plutôt bien, 80% du code Algoid est compatible PC / Android)

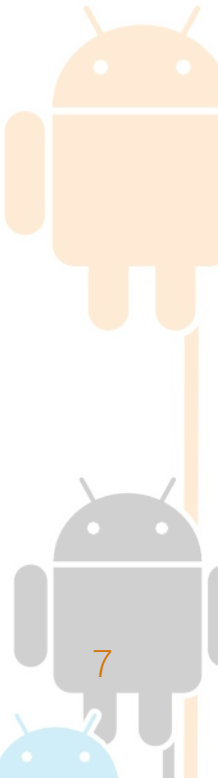
Anatomie d'un projet



- src : les sources java
  - Packages et .java
- gen : sources générées par l'IDE
  - R.java : identifie toutes les ressources
- assets : fichiers ressources additionelles et non compilées
- bin : le résultat de la compilation
- libs : les jars nécessaires
- res : les ressources du projet
- AndroidManifest.xml

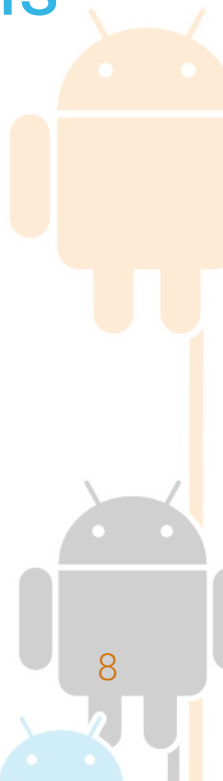
Manifest

- Fichier obligatoire (point d'entrée) ne doit pas être renommé ni déplacé
- Définit le nom de l'application, son icône, son package (identifiant unique sur le play store)
- Définit les composants (activities, services, broadcast receivers et content provider)
- Précise quels composants peuvent accueillir l'application



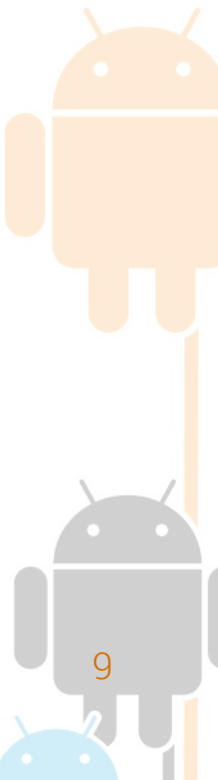
Manifest

- Déclare les permissions de l'application en interaction avec le matériel ou d'autres applications
- Déclare les permissions des autres applications avec les composants de l'application
- Des informations pour l'instrumentation (kesako ?)



Manifest

- Définit les niveaux de compatibilité (notablement l'API minimum pour que l'application s'exécute)
  - ➔ Exclu d'anciens appareils qui ne seraient pas compatibles
- Liste les librairies (android) nécessaires
- Définit les intents



AndroidManifest.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="fr.cnam.in01.db4o"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="19" />

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

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="fr.cnam.in01.db4o.MainActivity"
            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>
```

API level

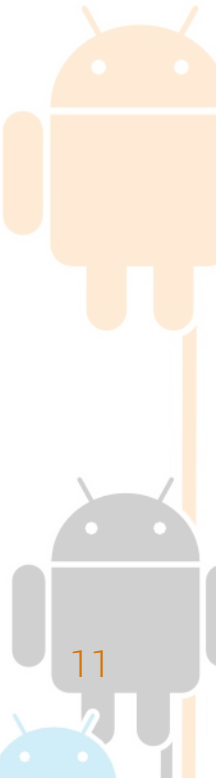
Permissions

Activities

Main Activity !

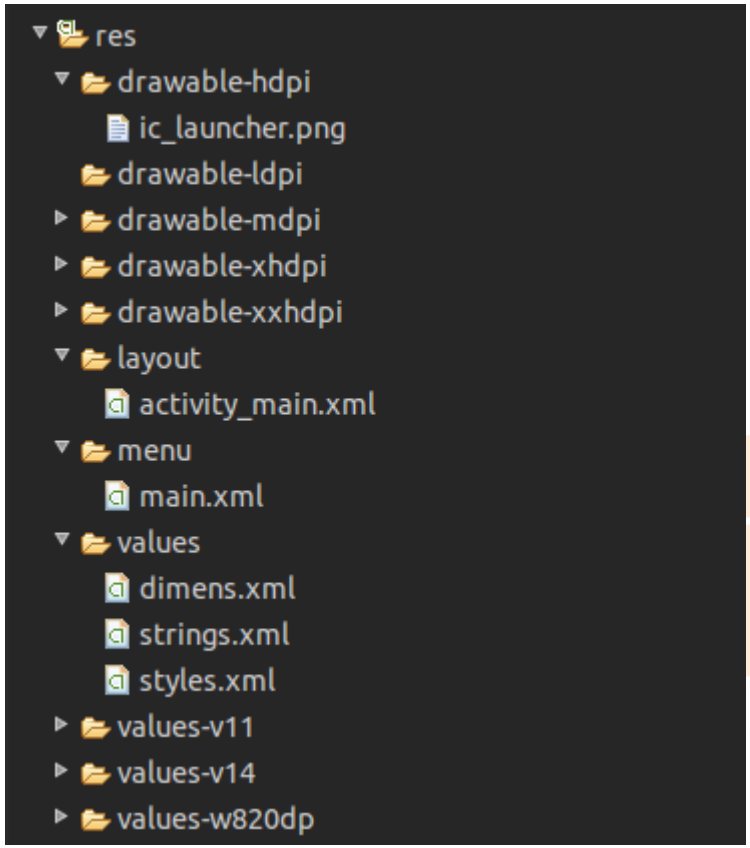
Manifest

- Toutes les activities du projet doivent être référencées dans le Manifest
- Sinon elles ne fonctionnent pas ! Exception !
- L'activity Main (le point d'entrée du programme) est déclaré grâce à un intent-filter
  - Action :: `android.intent.action.MAIN`
  - Catégorie :: `android.intent.category.LAUNCHER`
- Attention !! Un seul point d'entrée par projet



Ressources

- drawable :: images ou XMLs descriptifs
- layout :: les vues (MVC basé sur une description XML)
- menu
- values :: constantes nécessaires à l'application :: strings, dimensions, styles, couleurs



Organisation

- Organisation des ressources selon le materiel (ldpi, v14 etc)
- Ordre de priorité des **qualificateurs**
- Les paramètres par défaut sont ceux dans le repertoire sans modificateur
  - ex :
    - layout-land
    - layout

Disposition de l'écran lorsque l'appareil est en mode paysage

Defaut (portrait)

Qualificateurs

Priorité 2
Langue
fr
en
en-US
en-UK
etc ....

Priorité 3
Taille de la diagonale
small
normal
large
xlarge

Priorité 6
Orientation
port
land

Priorité 8	
Densité	
ldpi	120 dpi
mdpi	160 dpi
hdpi	240 dpi
xhdpi	320 dpi (>= API 8)
nodpi	non redimenssioné

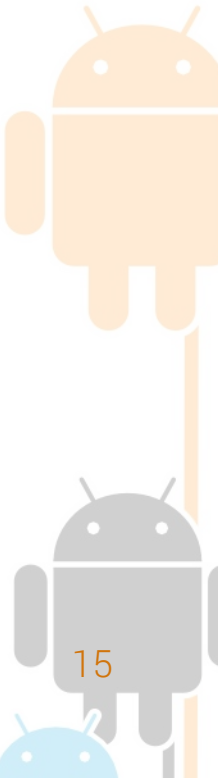
Priorité 14
Version de l'API
v3
v5
v7
v8
etc ....

- ex :: des images pour de petits écrans en portrait de petite densité
  - `drawable-small-port-ldpi`

Arborescence type

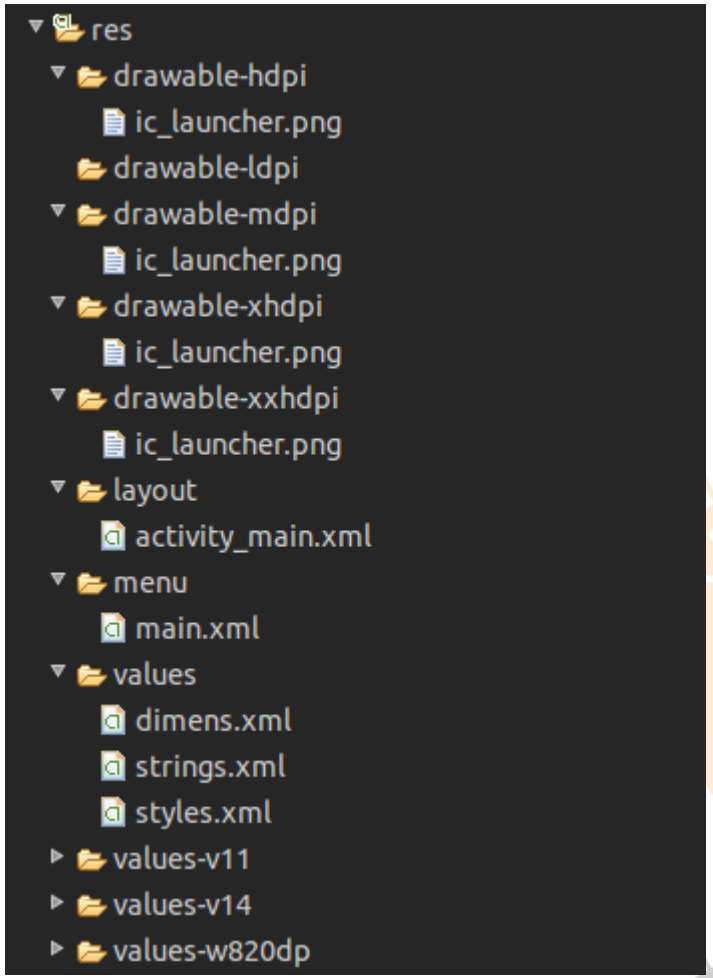
- drawable-ldpi
- drawable-mdpi
- drawable-hdpi
- drawable-xhdpi
- drawable-xxhdpi
- layout-land
- layout
- menu-fr
- menu

- values-fr
  - string.xml
- values-v11
  - style.xml
- values-w820dp
  - dimens.xml
- values
  - string.xml
  - style.xml
  - dimens.xml



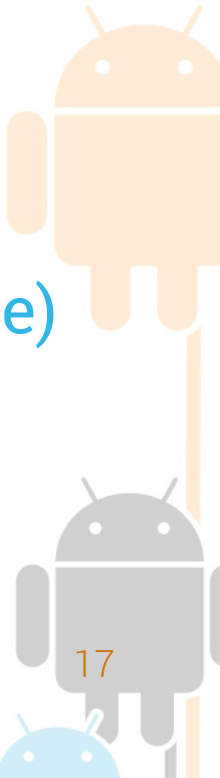
R.java

```
public final class R {
  public static final class attr {}

  public static final class dimen {
    public static final int padding_large = 0x7f040002;
    public static final int padding_medium = 0x7f040001;
    public static final int padding_small = 0x7f040000;
  }

  public static final class drawable {
    public static final int ic_launcher = 0x7f020001;
  }

  public static final class id {
    public static final int menu_settings = 0x7f080000;
  }

  public static final class layout {
    public static final int activity_main = 0x7f030000;
  }

  public static final class menu {
    public static final int activity_main = 0x7f070000;
  }

  public static final class string {
    public static final int app_name = 0x7f050000;
    public static final int hello_world = 0x7f050001;
  }

  public static final class style {
    public static final int AppTheme = 0x7f060000;
  }
}
```



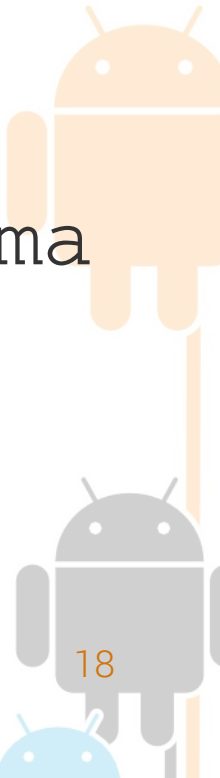
R.java

- Généré automatiquement lors de la phase de compilation du projet
- Référence toutes les ressources du projet et les rend utilisable depuis Java
- Association identifiant (`R.string.app_name`) / index (`0x7f050000`)
  - Optimise :: entiers plus efficace que des String (hashage)
  - Auto-completion :: accessible depuis les sources Java



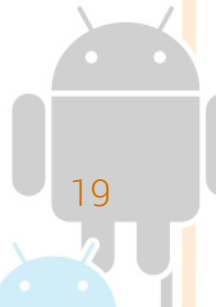
Le vues

- Les ressources se trouvent dans le repertoire `res/layout`
- Disponible depuis la class `R.java` depuis la classe interne `R.layout.*`
- Utilisable dans le projet avec `setContentView(R.layout.activity_main)`



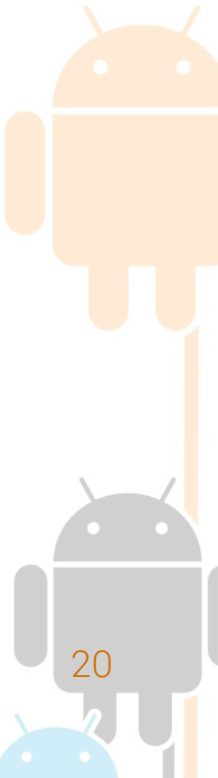
Les composants et les ids

- Les composants graphiques présents dans les layouts sont identifiés par un id
- Il sont créer dans le fichier xml de layout comme suit :: `android:id="@+id/myTextView"`
- `@+id` signifie “créer l'id”
- Il sont accessibles depuis les sources Java avec l'instruction `findViewById(R.id.myTextView)`
- ex :: `myTextView = (TextView)findViewById(R.id.myTextView);`



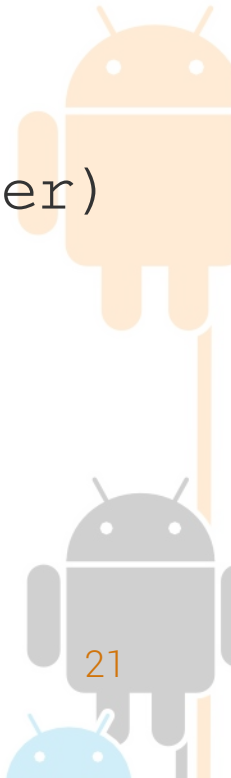
Internationalisation :: String.xml

- Les ressources se trouvent dans le fichier `res/values/strings.xml`
- Repertoires différents selon la langue (`values, values-fr, values-en, values-fr-rCA`)
- Classe interne `R.string.*`
- Utilisable dans le projet avec `getString(R.string.hello_world)`
- Utilisable depuis les fichiers XML `android:text="@string/hello_world"`



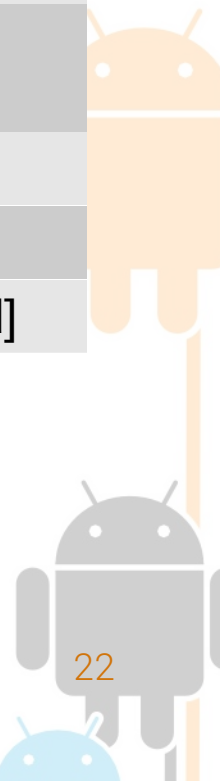
Multi-resolutions :: les images

- Les ressources se trouvent dans le repertoire `res/drawable` et dérivés
- Repertoires différents selon la résolution (`drawable-ldpi, drawable-mdpi, drawable-hdpi`)
- Classe interne `R.drawable.*`
- Utilisable dans le projet avec `getResources.getDrawable(R.drawable.ic_launcher)`
- Utilisable depuis les fichiers XML `android:icon="@drawable/ic_launcher"`



Résumé

	Source res/values/*	[id]	Java getResources().*	XML
bool	<bool>	R.bool.*	getBool([id])	@bool/[id]
integer	<integer>	R.integer.*	getInteger([id])	@integer/[id]
array	<integer-array><item> <string-array><item>	R.array.*	getIntArray([id]) getStringArray([])	@array/[id]
string	<string>	R.string.*	getString([id])	@string/[id]
types array	<array><item>	R.array.*	obtainTypedArray([id])	@array/[id]
color	<color>	R.color.*	getColor([id])	@color/[id]
style	<style><item>			@style/[id]
dimen	<dimen>	R.dimen.*		@dimen/[id]



Résumé

	Source	[id]	Java	XML
id		R.id.*	findViewById([id])	@+id/[id]
IHM	res/layout/*	R.layout.*	setContentView([id])	
dessins images	res/drawable/*	R.drawable.*	getResources().getdrawable([id])	@drawable/[id]
anim	res/anim/*	R.anim.*		@anim/[id]
menu	res/menu/*	R.menu.*	getMenuInflater().inflate([id])	@menu/[id]
style	res/values/* <	R.style.*		@style/[id]

- Liste exhaustive: http://developer.android.com/guide/topics/resources/available-resources.html

Exemple d'utilisation de R.java

```java
public class MainActivity extends ActionBarActivity {

  TextView myTextView = null;

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

    // le layout et les vues
    setContentView(R.layout.activity_main);
    myTextView = (TextView)findViewById(R.id.myTextView);
    String text = getString(R.string.hello_world);

    // les images
    Drawable d = getResources().getDrawable(R.drawable.ic_launcher);
    ImageView image = (ImageView)findViewById(R.id.image);

    // autres
    getResources().getColor(R.color.abc_search_url_text_holo);
    getResources().getDimension(R.dimen.dialog_fixed_height_major);
  }

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

  // etc ....
```

Layout

View

String

Image

Color

Dimension

Menu

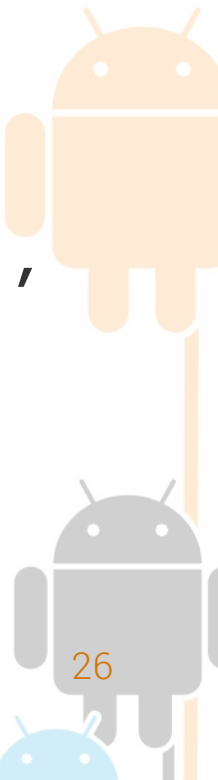
Context

- Les méthodes `setContentView, findViewById, getString, getResources et getMenuInflater` sont des méthodes héritées de la classe `Context`, dont hérite `Activity` à son tour.

- Dans d'autres classes, le contexte est passé en paramètres au constructeur (par exemple les vues)

Assets

- Similaire aux ressources, mais sans identifiant généré automatiquement
- répertoire `/assets`
- Accessible depuis la méthode `getAsset()` : `AssetManager` depuis la classe `Context`
- Plusieurs méthodes : `list (String filter)`, `open(String fileName)`, `openXmlRessource(String filename)`



Un exemple

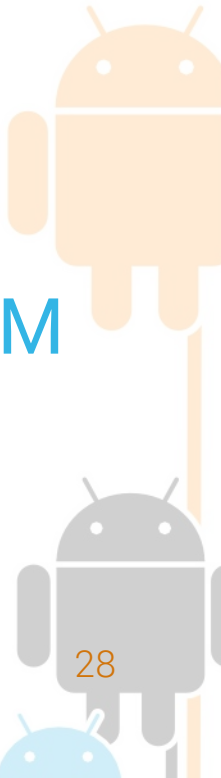
```
String line; StringBuilder sb = new StringBuilder(); BufferedReader input = null;

try {
  input = new BufferedReader(new InputStreamReader(getAssets().open("hello.txt")));

  while ((line = input.readLine()) != null) {
    sb.append(line);
  }
} catch (IOException e) {

  if (input != null) {
    throw new RuntimeException(e);
  }
} finally {
  try {
    if (input != null) { input.close(); }
  } catch (IOException e) {
    throw new RuntimeException(e);
  }
}
```

Nom du fichier asset

Attention aux ressources !

Librairies

- Pour ajouter une librairie (un jar) au projet, rien de plus simple.
- Il suffit de l'ajouter dans le repertoir `[project]/libs`
- Eclipse se charge de créer la référence dans le projet
- Le Dexer se chargera de convertir le bytecode JVM en bytecode Dalvik

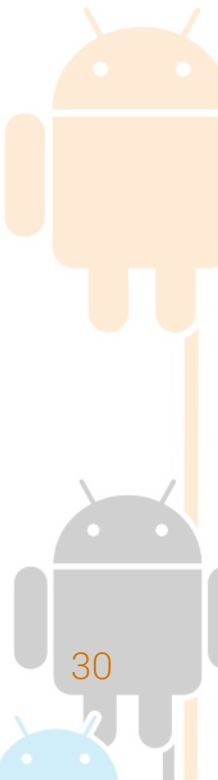


IN01 – Séance 02

Les composants

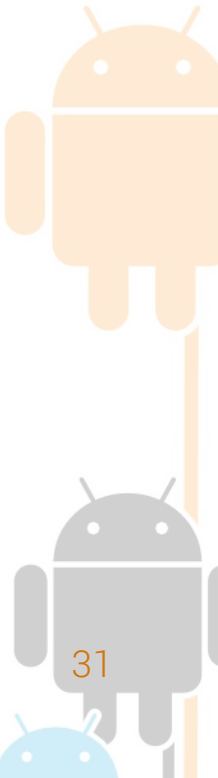
Les composants

- Une app peut-être composée d'un ou plusieurs composants
  - Des activities
  - Des services
  - Content provider
  - Broadcast receiver



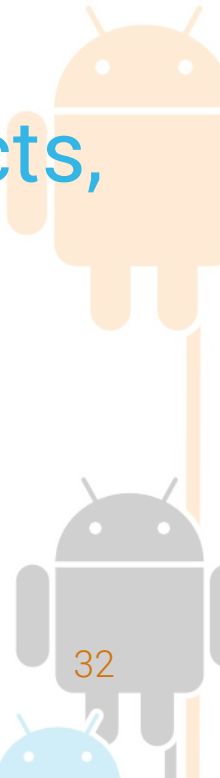
Activity & Service

- Une activity
  - Une application peut présenter une ou plusieurs Activities
  - Une “page” avec un HMI
  - Pris en charge avec le bouton back
  - Hérite de la classe `android.app.Activity`
- Un service et un composant en tâche de fond
  - Pas de HMI
  - Hérite de la classe `android.app.Service`



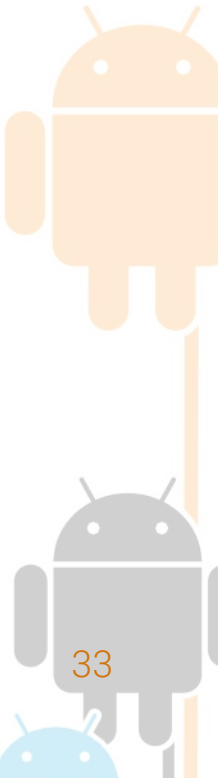
Content provider

- Fournisseur de contenu
- Donne accès à un ensemble de données structuré
- Un certain nombre de providers sont à disposition : Clock, Calendar, CallLog, Contacts, Documents, Media, Settings, Telephony
- Possibilité de créer son propre provider



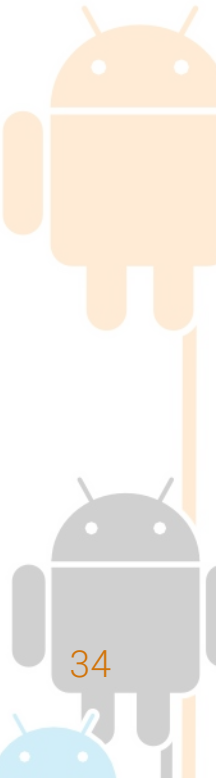
Broadcast Receiver

- Abonne l'application à un évènement applicatif ou système
- Ex : `ACTION_BOOT_COMPLETED` émet un évènement lorsque l'appareil est démarré
- Une méthode onReceive est appelée sur une classe dite “receiver”
- Processus asynchrone
- Hérite de la classe `android.content.BroadcastReveiver`
- Possibilité de créer son propre “emiter”



Résumé

- Activité == logique associée aux interfaces utilisateurs
- Service == opération s'exécutant en arrière plan
- Content Provider == fournisseur de contenu ! c'est à dire gestionnaire des données partageables et/ou persistantes
- Broadcast receiver == gestionnaire des messages systèmes ou applicatifs
- Intent == message indiquant une intention à faire une certaine action

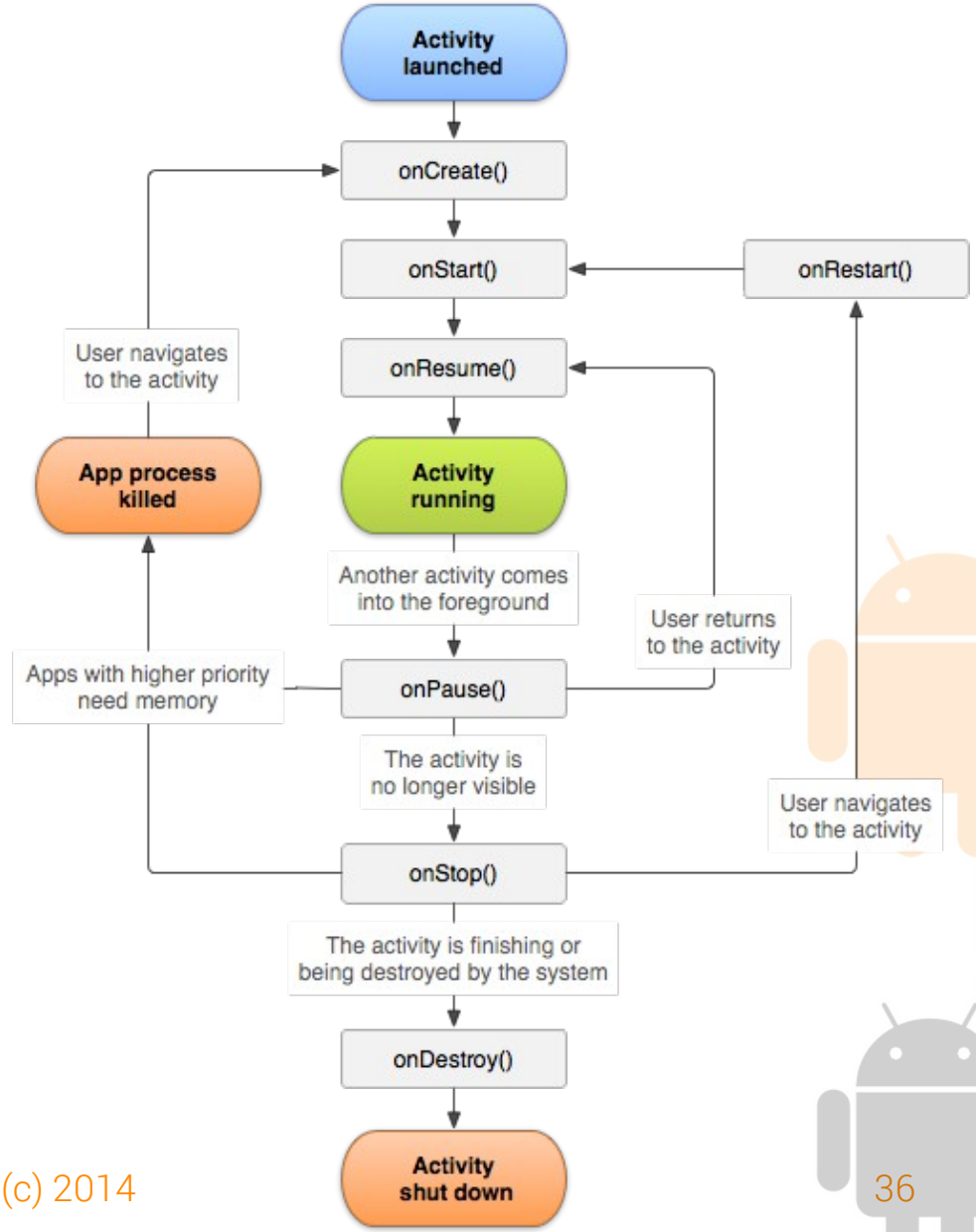


IN01 – Séance 02

Cycle de vie d'une application et persistance

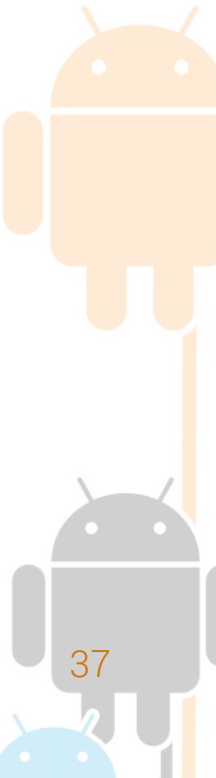
Cycle de vie

- Un certain nombre d'évènements avant et après le démarrage d'une application
- L'application est détruite `onDestroy` et recréée `onCreate` à chaque rotation de l'écran



Persistance

- Pour éviter la perte de donnée (durant la rotation, notamment), il faut les persister.
- Plusieurs options :
  - Utiliser un outil tout près, le Bundle
  - Persister en base de données (cf cours 04)
  - Serialiser des objets dans le système de fichier (dans le cas de structures complexes)

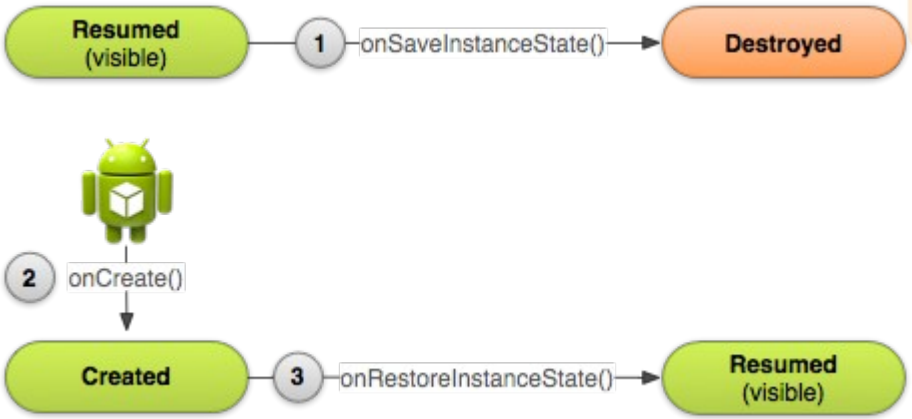


Bundle

- Bundle est un tableau associatif Clé / Valeur typé
- Des méthodes pour stocker les données : `putBoolean (String key, boolean value)`, `putInteger`, `putParceable`, `putSerializable`
- Des méthodes pour récupérer les données : `getBoolean (String key)`, `getInteger`, `getParceable`, `getSerializable`
- `Parceable` décrit comment stocker un object complexe dans un `Parcel`

Evènements

- Deux méthodes sont disponibles dans l'objet Activity pour réaliser la persistance.
- `onSaveInstanceState` et `onRestoreInstanceState`
- Il suffit de les surcharder dans notre Activity spécialisée



Un exemple

```
static final String STATE_SCORE = "playerScore";
private int currentScore = 10;

@Override
protected void onSaveInstanceState (Bundle outState) {
  outState.putInt(STATE_SCORE, currentScore);
  super.onSaveInstanceState(outState);
}

@Override
protected void onRestoreInstanceState(Bundle savedInstanceState) {
  super.onRestoreInstanceState(savedInstanceState);
  currentScore = savedInstanceState.getInt(STATE_SCORE);
}
```

Sauve l'état

Appel de la méthode surchargée
Stock les états au travers la hierarchie de classes

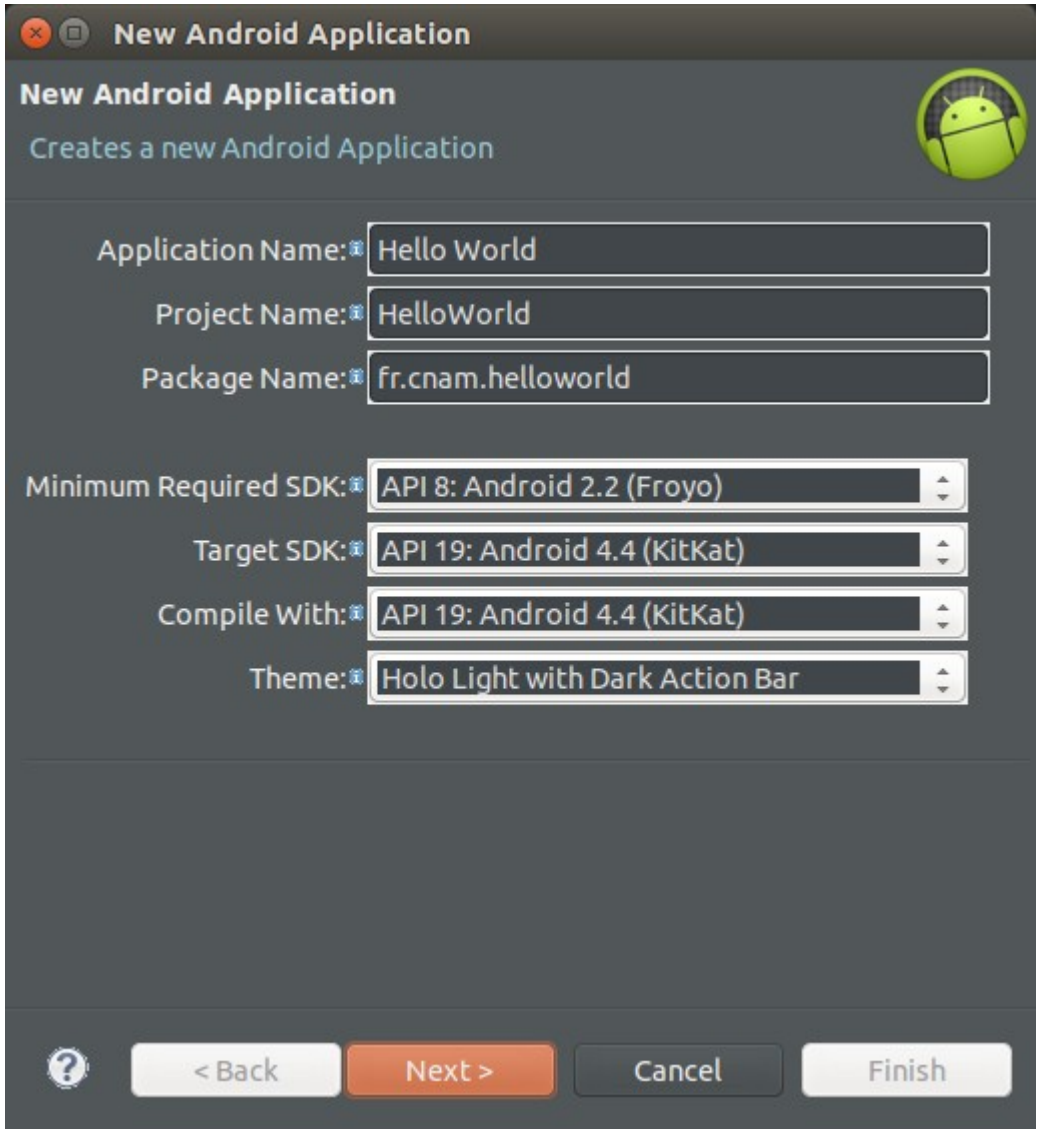
Comme précédamment

Récupère l'état

IN01 – Séance 02

Un premier projet

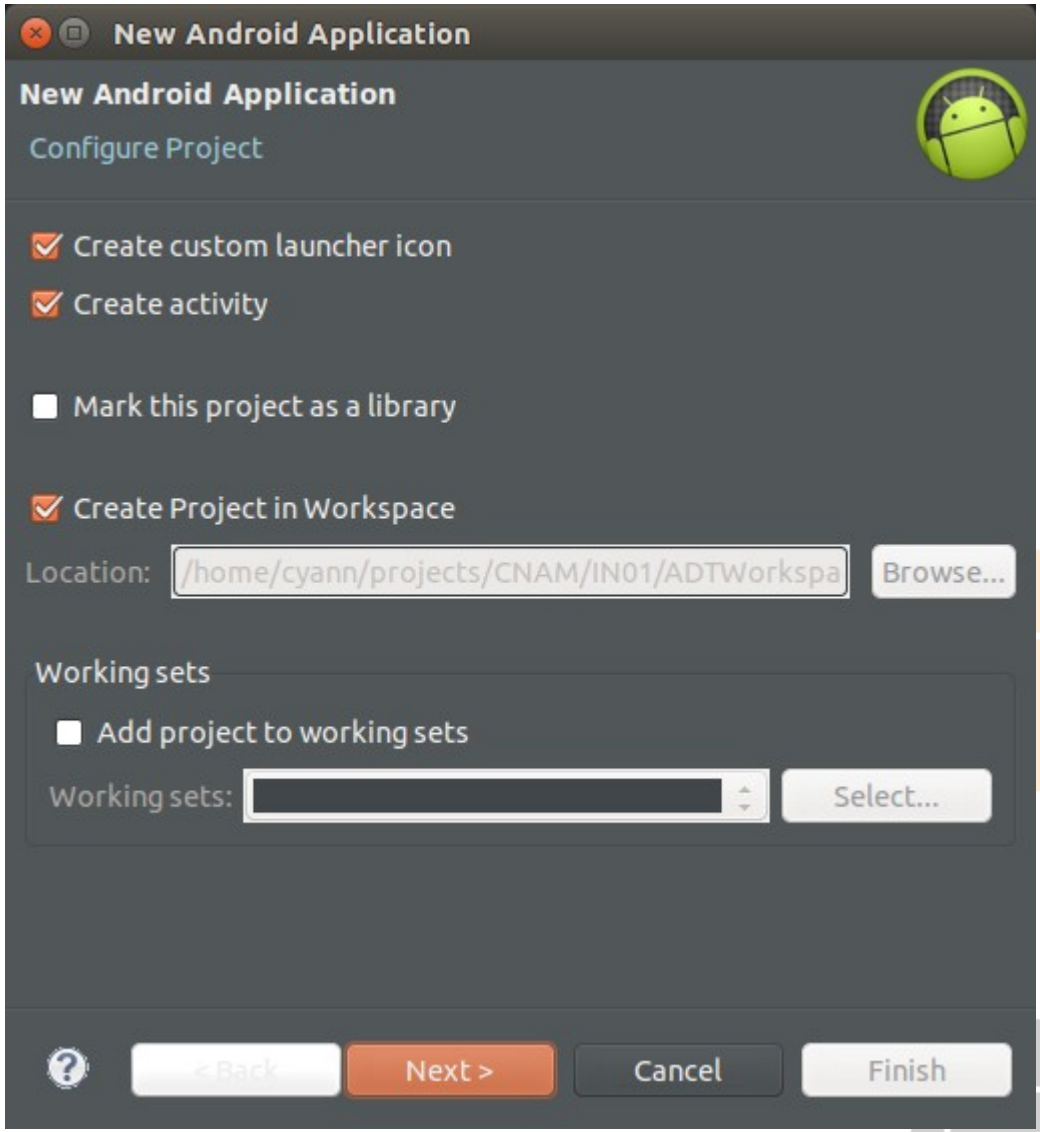
Création



- Donner un nom (celui qui s'affiche sous l'icone)
- Un nom de projet (utile pour eclipse)
- Un chemin de package fr.cnam.*
- Min, Target, Compile SDK
- Thème

Quelques options

- Créer une icône par défaut
- Créer l'activity principale (Main_Activity)
- Workspace eclipse

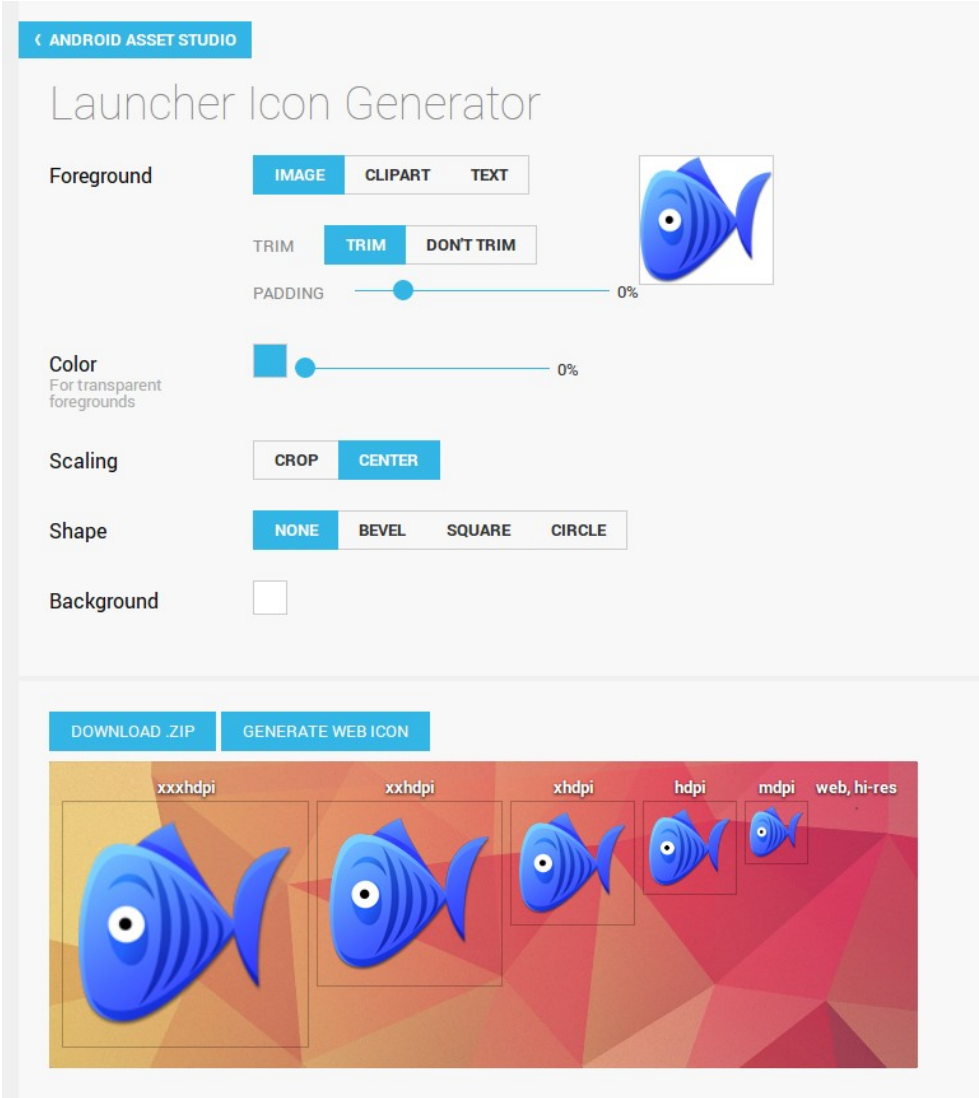


L'icône par défaut

- L'icône par défaut
- Quelques options
- Conseil, utiliser inkscape pour créer une icône vectorisée (ligne graphique ?)

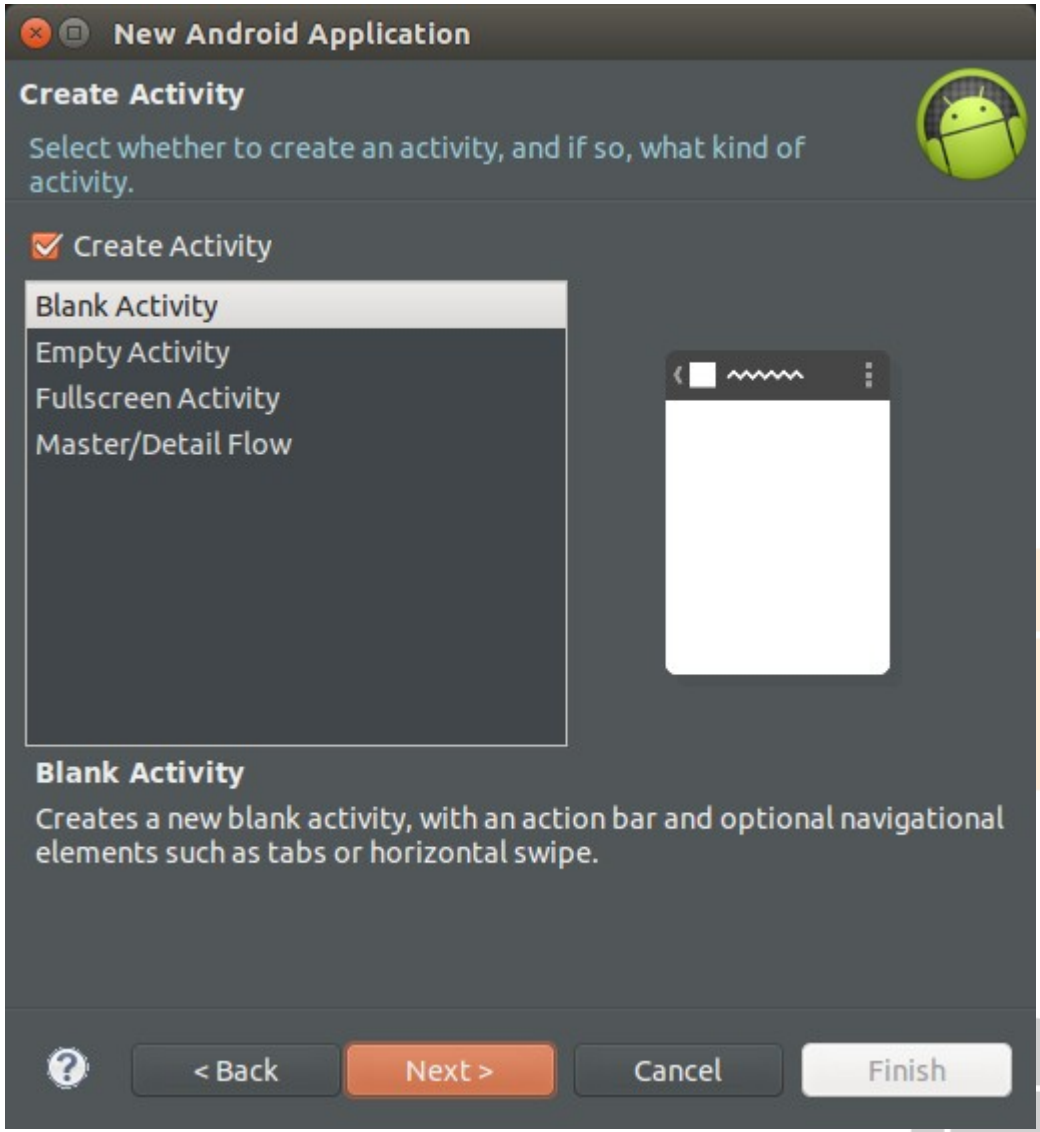
Outils webs

- Google met à disposition des outils html
- Disponible ici :: http://romannurik.github.io/AndroidAssetStudio/

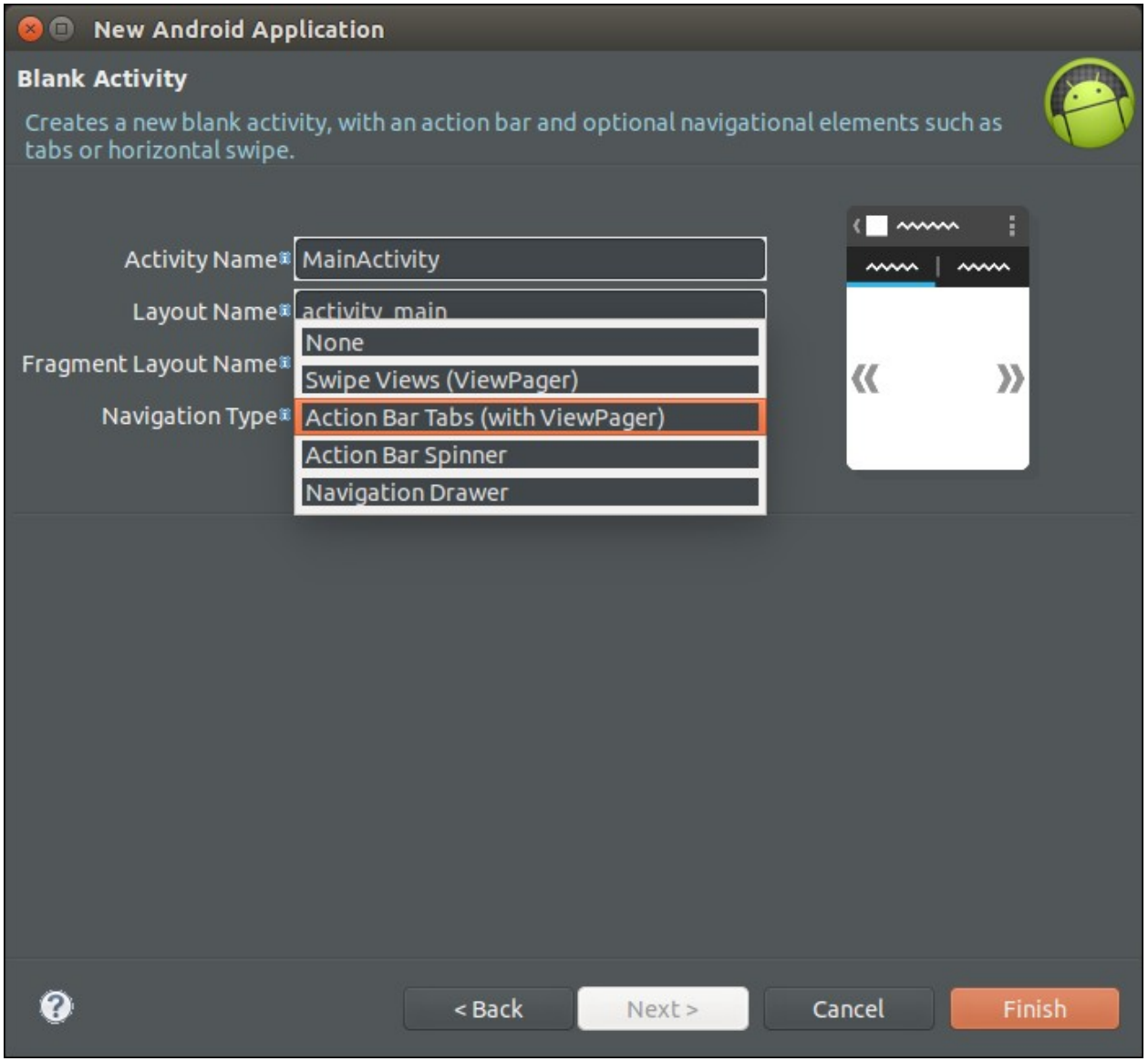


L'Activity

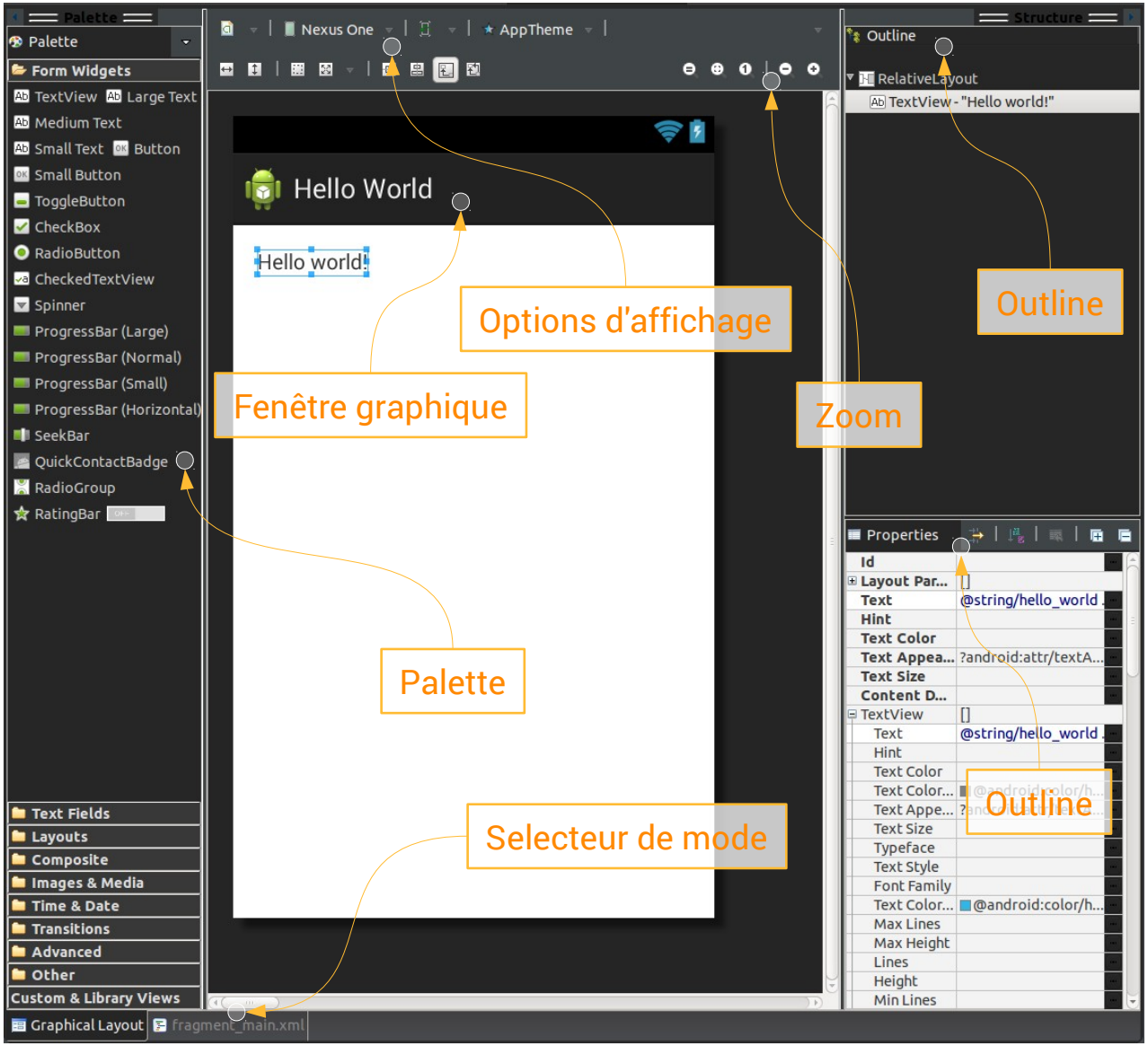
- Crée le layout de base
- Modifiable par la suite



Options de navigation



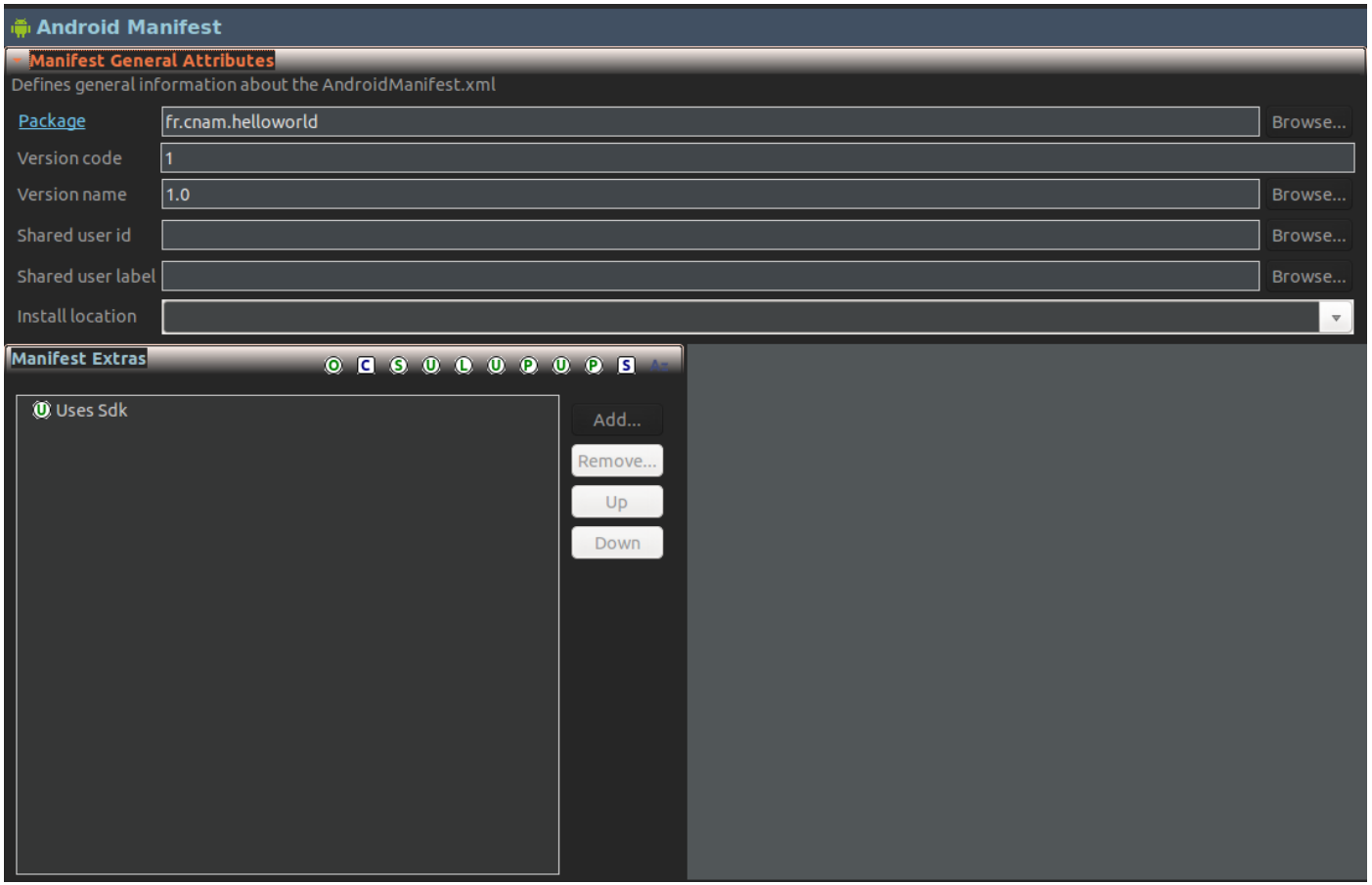
WYSIWYG



- Préférer le mode XML
  - compréhension
  - copier / coller
  - + efficace à l'usage
  - code maîtrisé (+ propre)

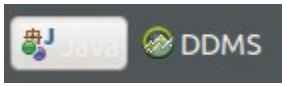
Manifest

- Un outil graphique pour éditer le manifest



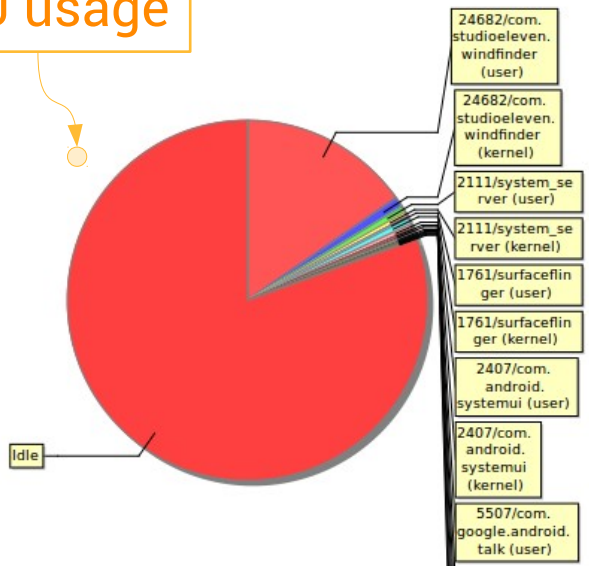
Eclipse DDMS

- La perspective DDMS d'éclipse

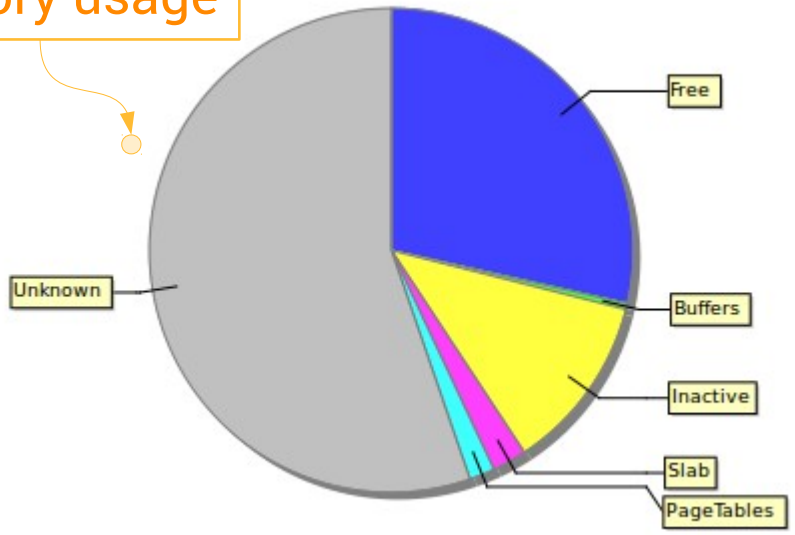


- Toutes les informations sur les processus s'exécutant sur l'appareil

CPU usage

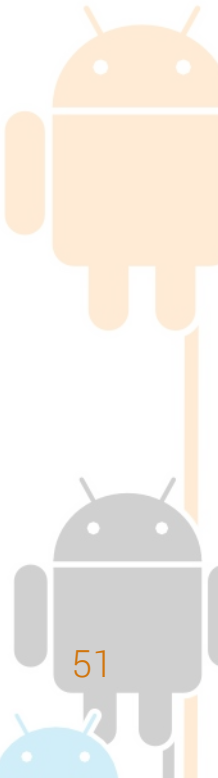


Memory usage

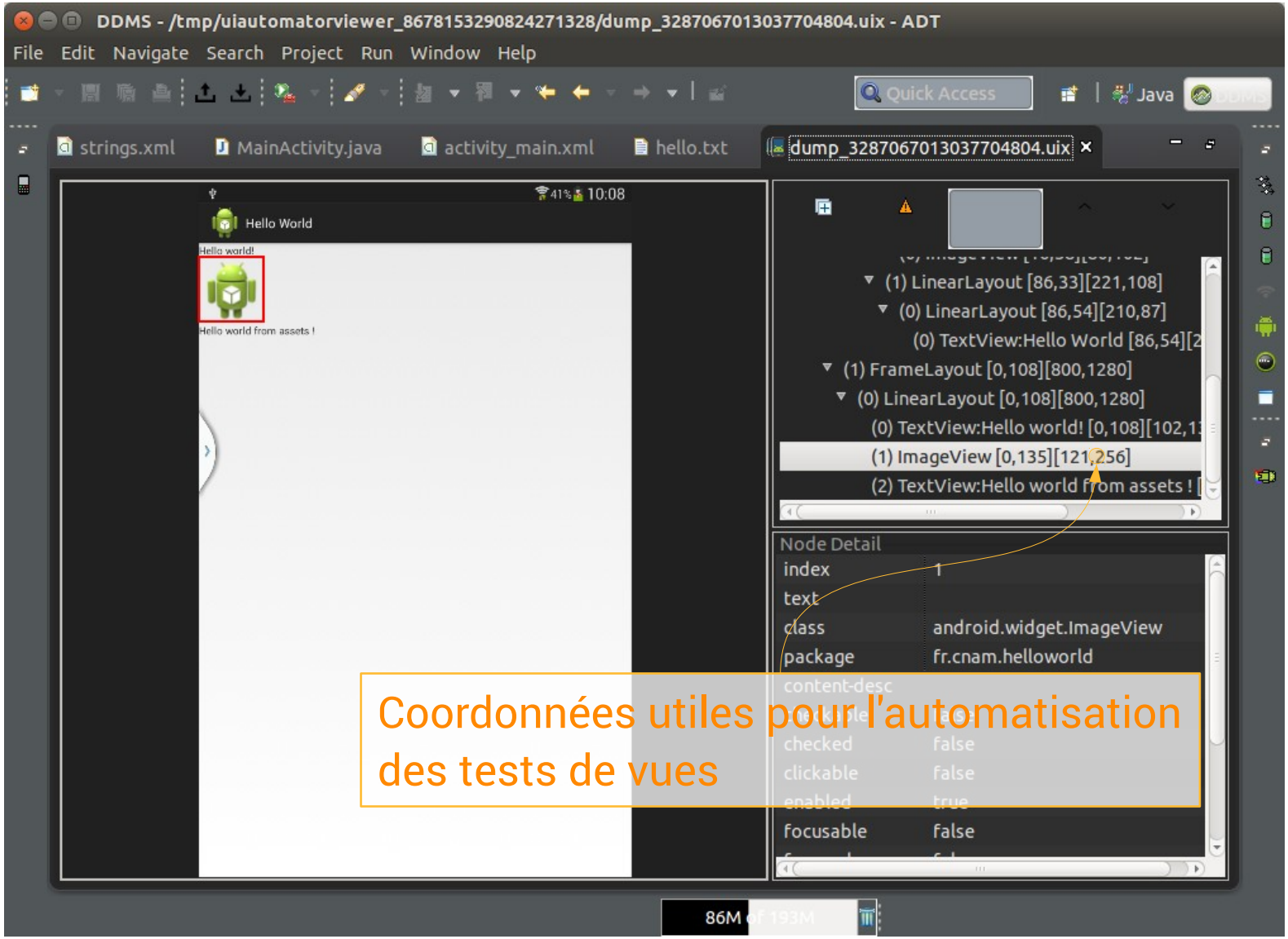


Eclipse DDMS

- Un explorateur de fichiers
- Screen Capture (utile lors de la publication sur le play store)
- Statistiques réseau
- Threads, Heap, Allocation



Dump view hierarchy



Coordonnées utiles pour l'automatisation des tests de vues

IN01 – Séance 02

Exécution et adb

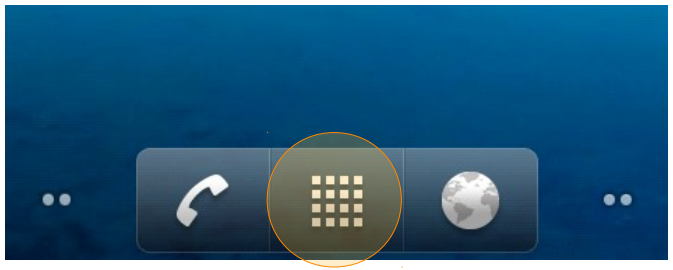
Exécution

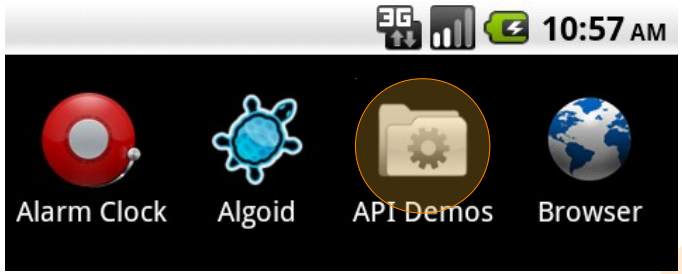
- Deux possibilités
  - l'émulateur (plus lent mais plus flexible, capable de simuler divers résolutions)
  - sur le device (les vrais performances)

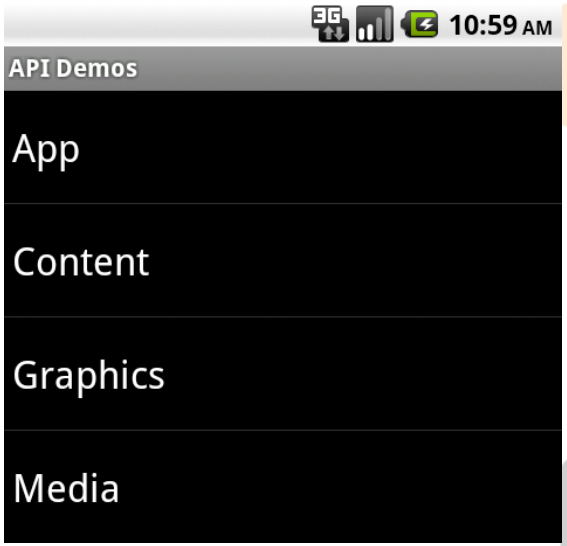
Des exemples dans l'émulateur

- Des exemples sont disponibles dans l'émulateur
- Il est possible de les exécuter depuis les menus
- Le code est accessible à partir de :

  %SDK%\sample\android-XXX\ApiDemos

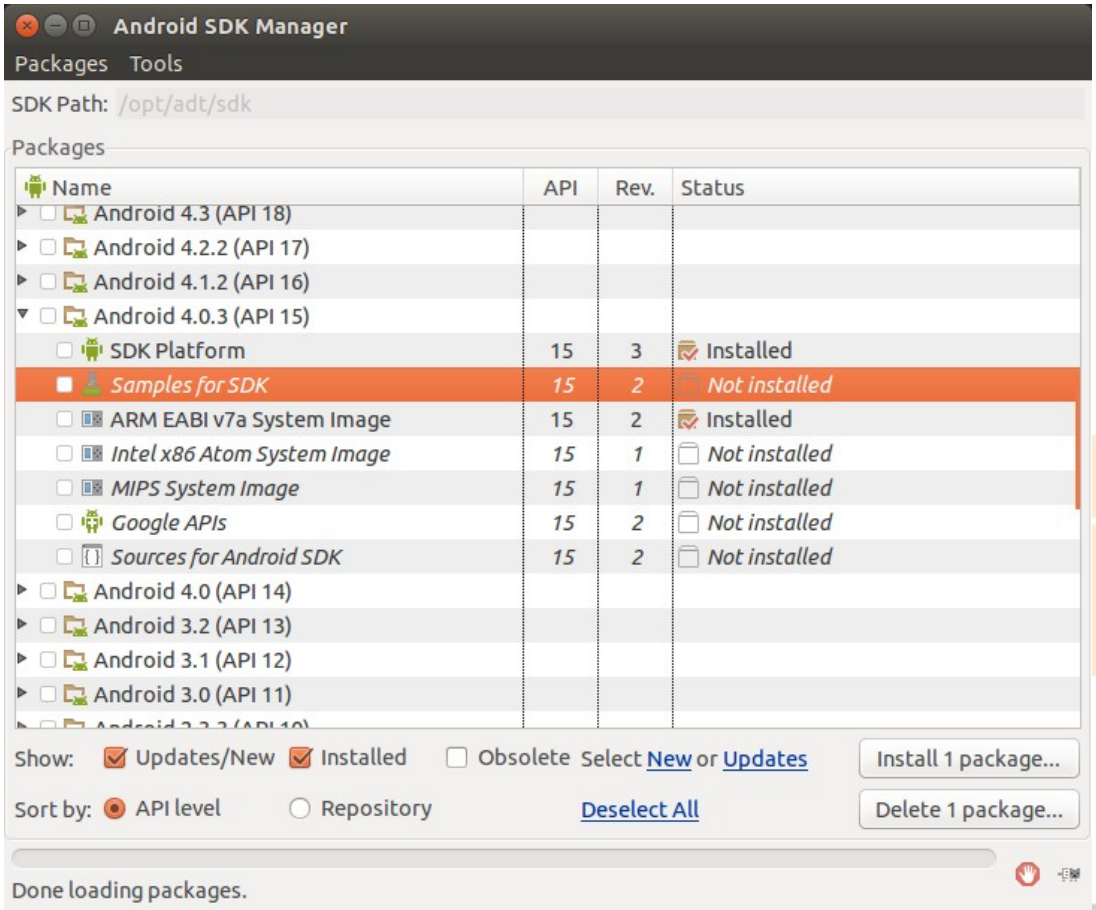






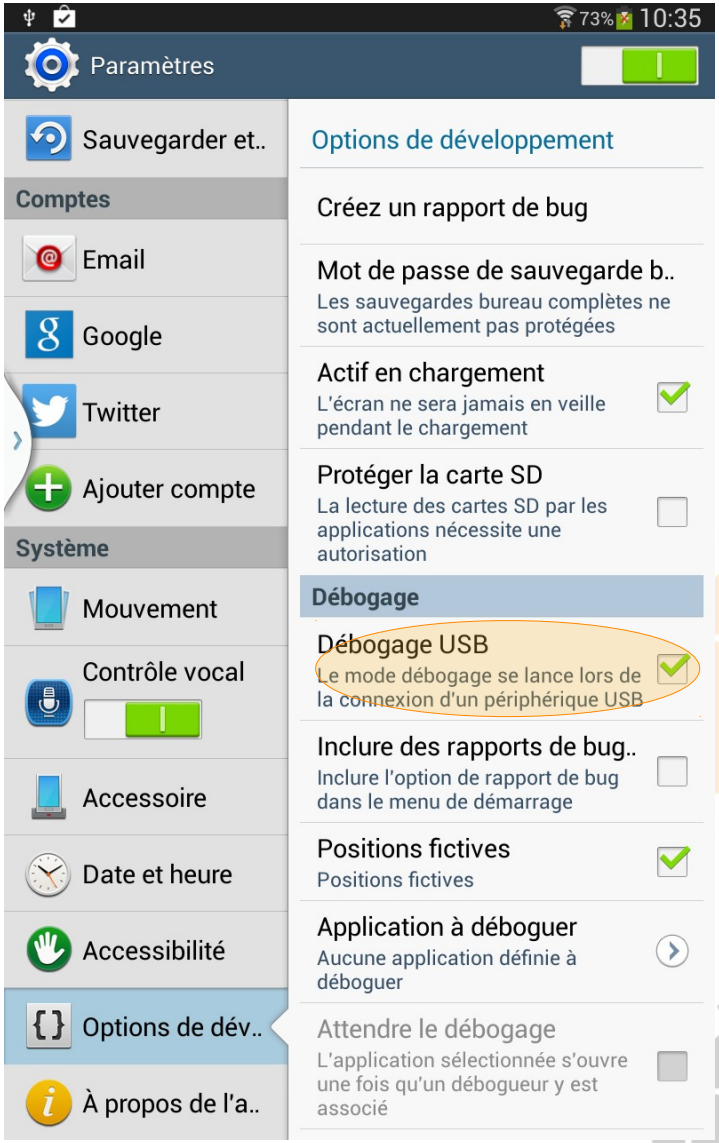
Sources

- Si les sources ne sont pas présentes dans le sdk, il faut les télécharger



Appareil - Activer le débogage

- Pour développer depuis l'appareil
- Dans le menu Options de développement, il faut activer le menu Débogage USB
- Et brancher le cable USB entre le PC et l'appareil

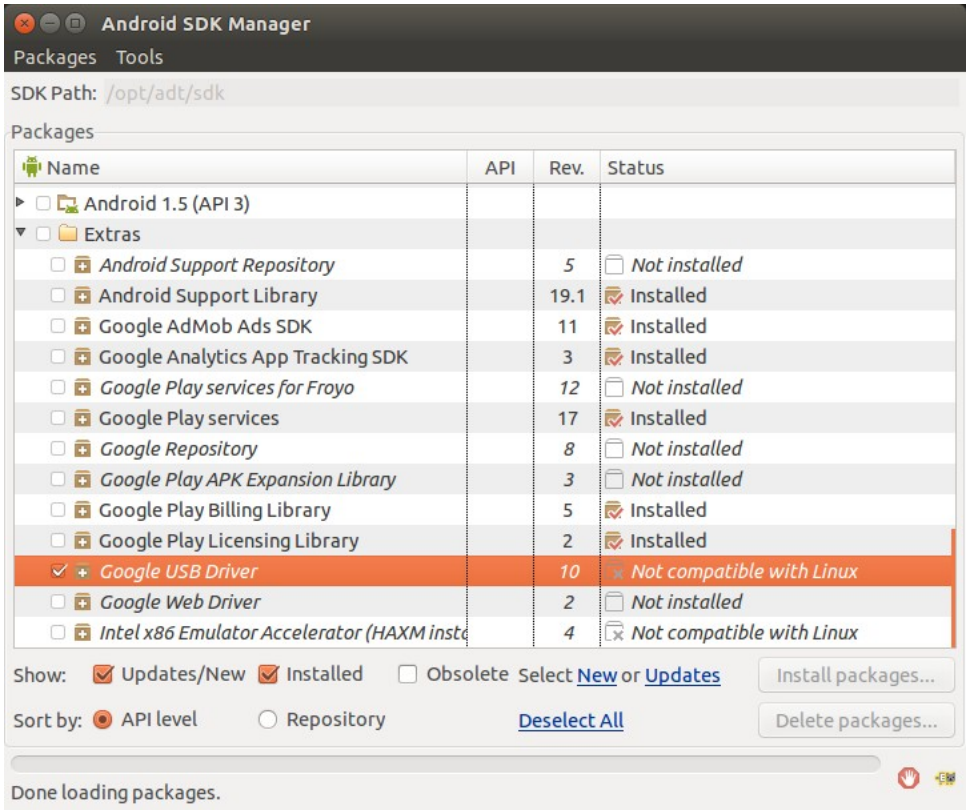


Astuce Samsung

- Sur certains appareils Samsung, le menu “Developer” est caché.
- Pour le réactiver il faut allez dans Settings/About Device et toucher 7 fois (!!!!) le menu Build number
- Sur les appareils en français, Paramètres/A propos de l'appareil, menu Numéro de version

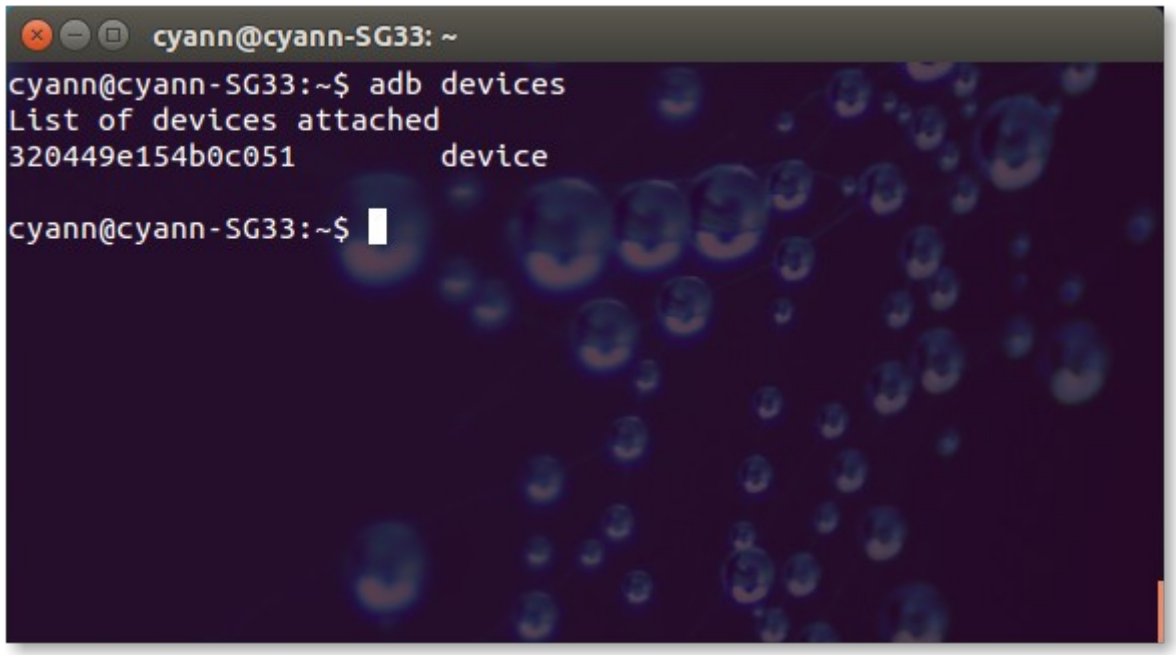
Sous windows

- Les drivers ne sont pas présents par défaut. Il fait aussi les télécharger



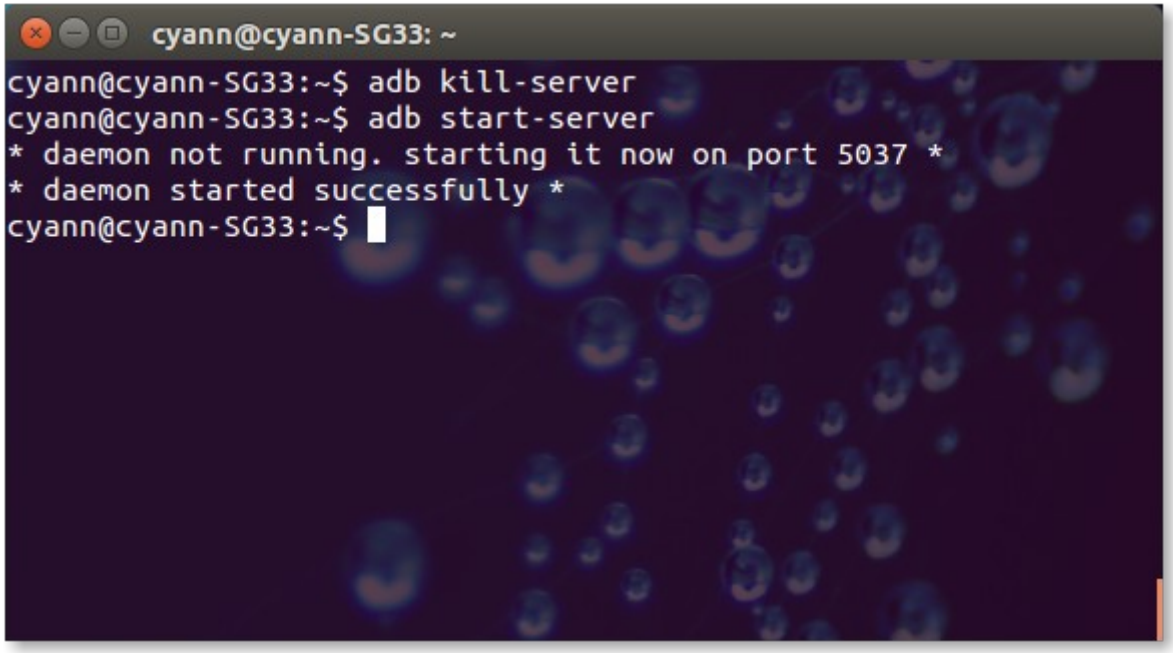
adb devices

- Android debug bridge
- Lancer une fenêtre dos ou un shell
- Tester si l'appareil est connecté `adb devices`



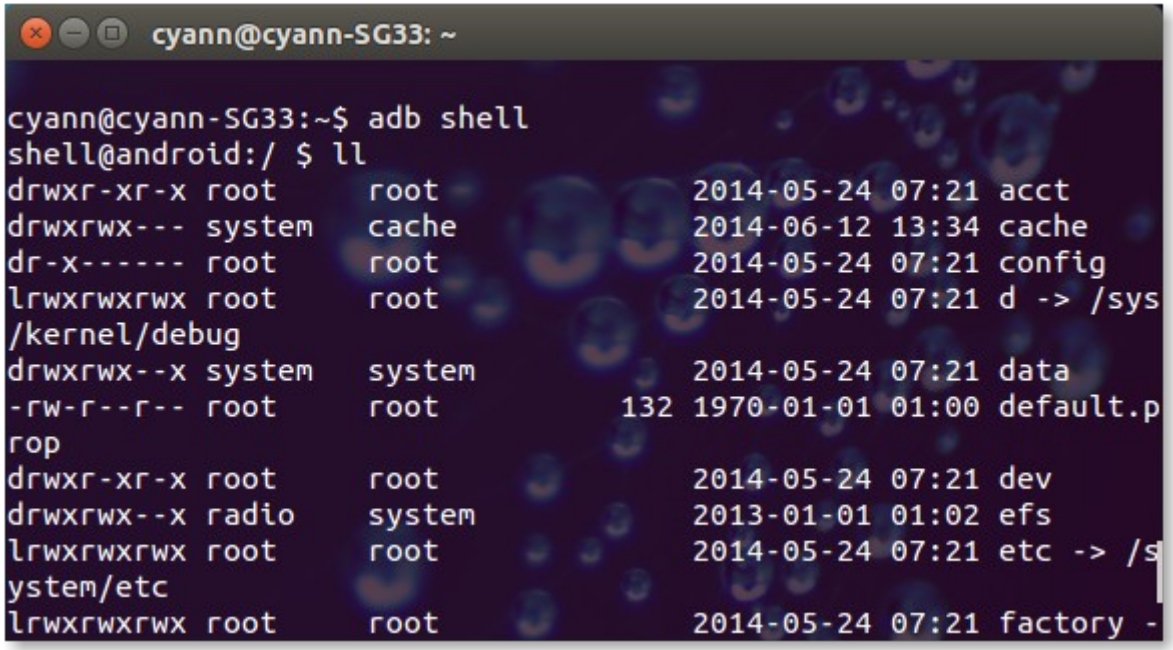
adb start, kill

- Très utile lors d'un dysfonctionnement
- `adb kill-server` et `adb start-server`



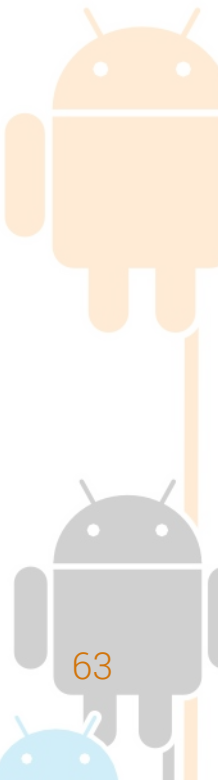
adb shell

- Un remote shell sur l'appareil depuis le PC
- `adb shell`
- Le reste, c'est du linux (`ls, mkdir, chmod, cat, exit`)



adb transfert

- Installer une apk à distance
  - `adb install c:\file.apk`
- Envoyer un fichier sur le téléphone
  - `adb push c:\file.txt /sdcard`
- Télécharger un fichier depuis le téléphone
  - `adb pull /sdcard/file.txt c:\`



IN01 – Séance 02

Logcat

android.util.Log

- Le logger interne à la plateforme
- Accessible par programmation : `android.util.Log`
- Une méthode statique générale :

```
println(Int priority, String tag, String msg)
```

- Des priorités `DEBUG, ERROR, INFO, VERBOSE, WARN`
- Des méthodes statiques utilitaires

```
d(String tag, String msg), e, i, v ,w
```

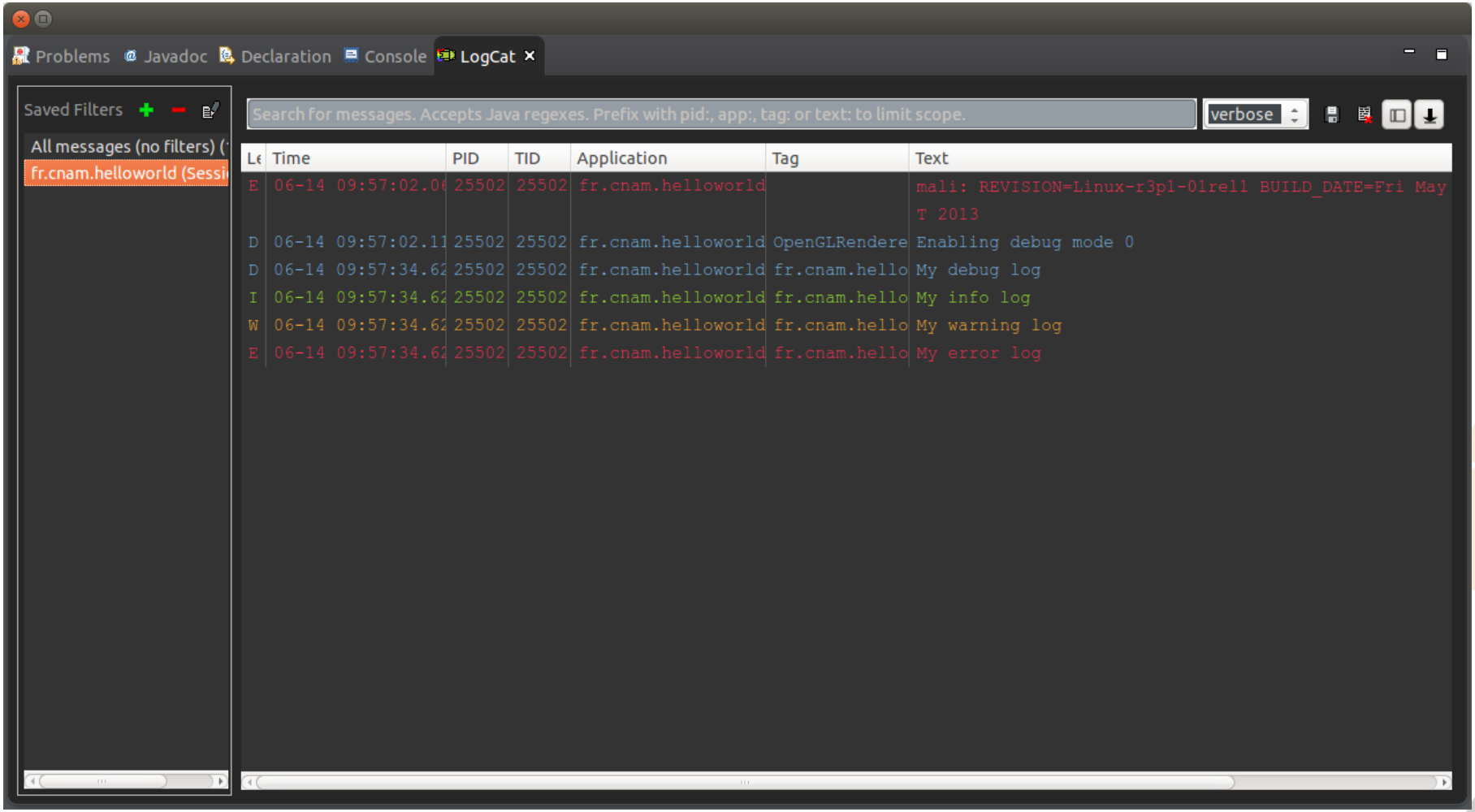
Example

```
Log.d(MainActivity.class.getName(), "My debug log");
Log.i(MainActivity.class.getName(), "My info log");
Log.w(MainActivity.class.getName(), "My warning log");
Log.e(MainActivity.class.getName(), "My error log");
```

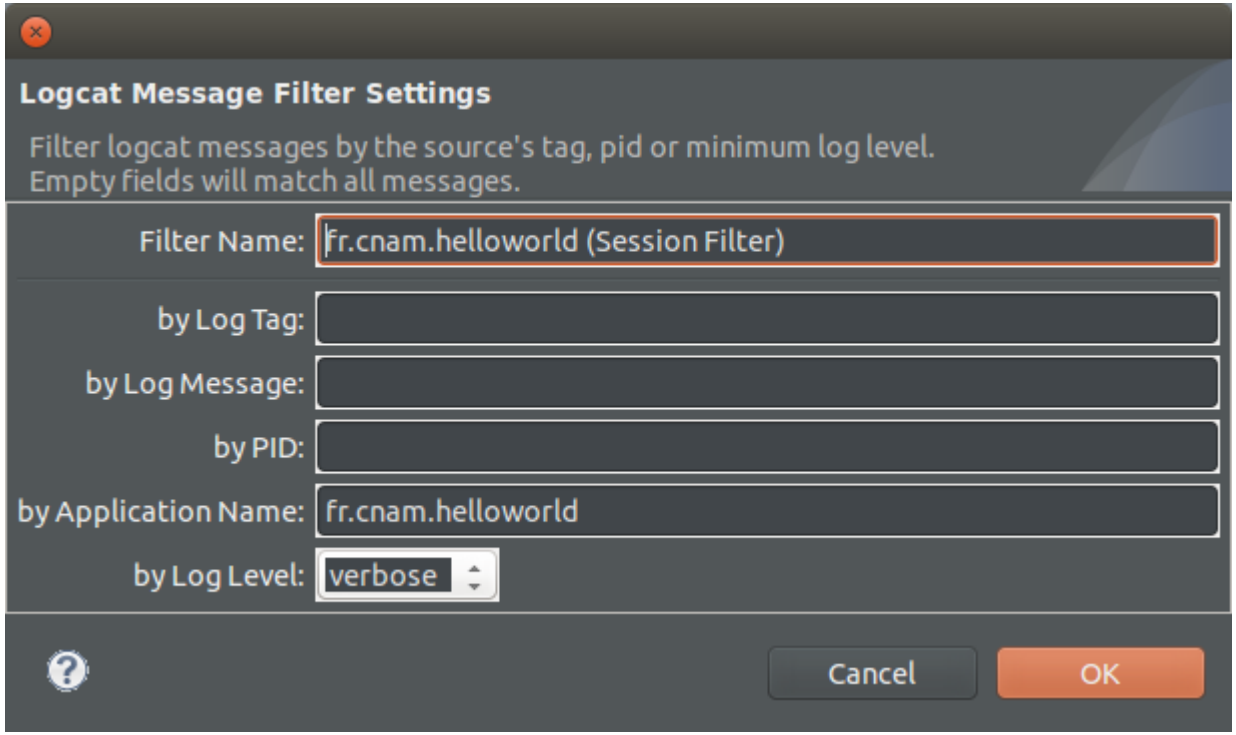
- Utile pour tracer le programme et comprendre ce qu'il se passe sans utiliser le deboggueur.

```
D | 06-14 09:57:34.62 | 25502 | 25502 | fr.cnam.helloworld | fr.cnam.hello | My debug log
I | 06-14 09:57:34.62 | 25502 | 25502 | fr.cnam.helloworld | fr.cnam.hello | My info log
W | 06-14 09:57:34.62 | 25502 | 25502 | fr.cnam.helloworld | fr.cnam.hello | My warning log
E | 06-14 09:57:34.62 | 25502 | 25502 | fr.cnam.helloworld | fr.cnam.hello | My error log
```

DDMS LogCat


Problems
Javadoc
Declaration
Console
LogCat
Saved Filters
All messages (no filters)
fr.cnam.helloworld
Search for messages. Accepts Java regexes. Prefix with pid:, app:, tag: or text: to limit scope.
verbose
Le Time PID TID Application Tag Text
E 06-14 09:57:02.06 25502 25502 fr.cnam.helloworld mali: REVISION=Linux-r3p1-01rel1 BUILD_DATE=Fri May
T 2013
D 06-14 09:57:02.11 25502 25502 fr.cnam.helloworld OpenGLRendere Enabling debug mode 0
D 06-14 09:57:34.62 25502 25502 fr.cnam.helloworld fr.cnam.hello My debug log
I 06-14 09:57:34.62 25502 25502 fr.cnam.helloworld fr.cnam.hello My info log
W 06-14 09:57:34.62 25502 25502 fr.cnam.helloworld fr.cnam.hello My warning log
E 06-14 09:57:34.62 25502 25502 fr.cnam.helloworld fr.cnam.hello My error log

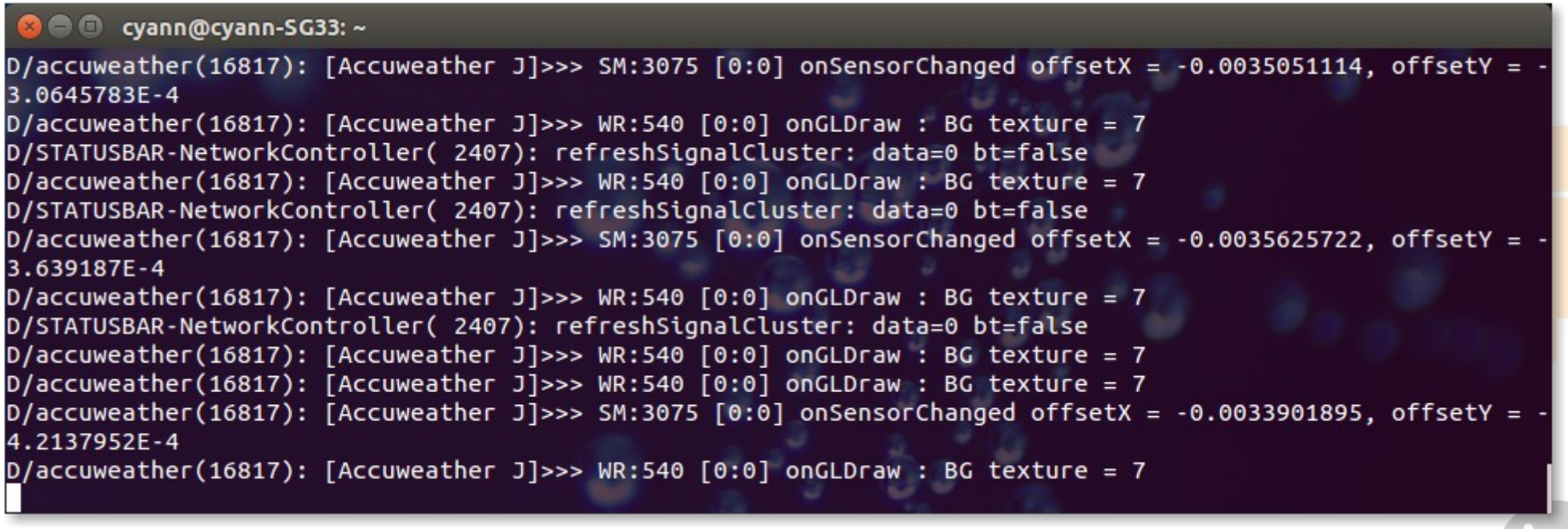
Filtres



- Eclipse filtre automatiquement sur le nom de l'application

adb logcat

- Accès au Logcat en ligne de commande
- `adb logcat`



Fin

- Merci de votre attention
- Des questions ?