3

Création d'applications et découverte des activités

Une application Android est un assemblage de composants liés grâce à un fichier de configuration.

Avant de rentrer dans le détail d'une application Android et de son projet associé, différents concepts fondamentaux sont à préciser :

- les activités ;
- les vues et contrôles (et leur mise en page) ;
- les ressources ;
- le fichier de configuration appelé également manifeste.

Les *vues* sont les éléments de l'interface graphique que l'utilisateur voit et sur lesquels il pourra agir. Les vues contiennent des composants, organisés selon diverses mises en page (les uns à la suite des autres, en grille…).

Les *contrôles* (boutons, champs de saisie, case à cocher, etc.) sont eux-mêmes un sous-ensembles des vues (nous y reviendrons ultérieurement). Ils ont besoin d'accéder aux textes et aux images qu'ils affichent (par exemple un bouton représentant un téléphone aura besoin de l'image du téléphone correspondante). Ces textes et ces images seront puisés dans les fichiers *ressources* de l'application.

Une *activité* peut être assimilée à un écran structuré par un ensemble de vues et de contrôles composant son interface de façon logique : elle est composée d'une hiérarchie de vues contenant elles-mêmes d'autres vues. Une activité est par exemple un formulaire d'ajout de contacts ou encore un plan Google Maps sur lequel vous ajouterez de l'information. Une application comportant plusieurs écrans, possédera donc autant d'activités.

À côté de ces éléments, se trouve un fichier XML : le *fichier de configuration* de l'application. C'est un fichier indispensable à chaque application qui décrit entre autres :

- le point d'entrée de votre application (quel code doit être exécuté au démarrage de l'application) ;
- quels composants constituent ce programme ;
- les permissions nécessaires à l'exécution du programme (accès à Internet, accès à l'appareil photo...).

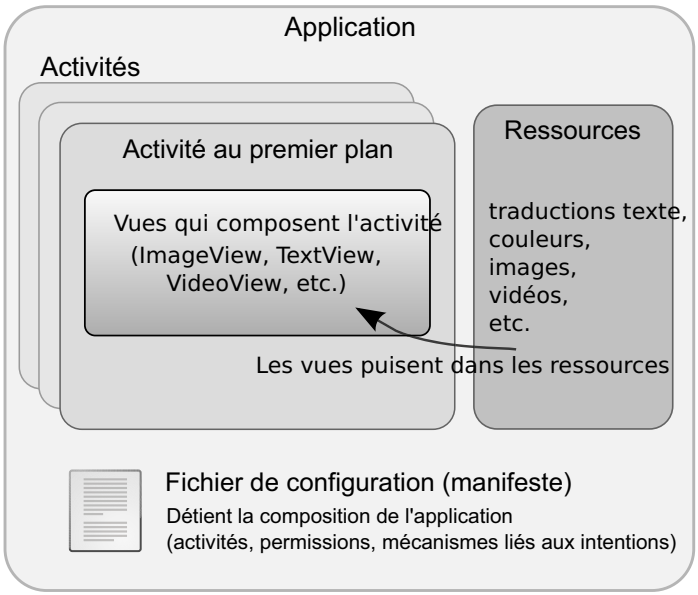


Figure 1
Composition d'une application

Le fichier de configuration Android : la recette de votre application

Chaque application Android nécessite un fichier de configuration : `AndroidManifest.xml`. Ce fichier est placé dans le répertoire de base du projet, à sa racine. Il décrit le contexte de l'application, les activités, les services, les récepteurs d'Intents (*Broadcast receivers)*, les fournisseurs de contenu et les permissions.

Structure du fichier de configuration

Un fichier de configuration est composé d'une racine (le tag `manifest` 1) et d'une suite de nœuds enfants qui définissent l'application.

Code 1 : Structure vide d'un fichier de configuration d'une application

```
<manifest 1
      xmlns:android=http://schemas.android.com/apk/res/android 2
      package="fr.domaine.application"> 3
 </manifest>
```

La racine XML de la configuration est déclarée avec un espace de nom Android (`xmlns:android` 2) qui sera utile plus loin dans le fichier ainsi qu'un paquetage 3 dont la valeur est celle du paquetage du projet.

Voici le rendu possible d'un manifeste qui donne une bonne idée de sa structure. Ce fichier est au format XML. Il doit donc toujours être :

- bien formé : c'est-à-dire respecter les règles d'édition d'un fichier XML en termes de nom des balises, de balises ouvrante et fermante, de non-imbrication des balises, etc. ;
- valide : il doit utiliser les éléments prévus par le système avec les valeurs prédéfinies.

Profitons-en pour étudier les éléments les plus importants de ce fichier de configuration.

Code 2 : Structure du fichier AndroidManifest.xml extraite de la documentation

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

<manifest>

    <uses-permission /> 1
    <permission />
    <permission-tree />
    <permission-group />
    <instrumentation />
    <uses-sdk />
    <uses-configuration />
    <uses-feature />
    <supports-screens />

    <application> 2

        <activity> 3
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
            <meta-data />
        </activity>

        <activity-alias>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </activity-alias>

        <service> 4
            <intent-filter> . . . </intent-filter>
            <meta-data/>
        </service>

        <receiver> 5
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </receiver>
```

```
        <provider> 6
            <grant-uri-permission />
            <path-permission />
            <meta-data />
        </provider>

        <uses-library />

    </application>

</manifest>
```

`<uses-permission>` 1

Les permissions qui seront déclarées ici seront un prérequis pour l'application. À l'installation, l'utilisateur se verra demander l'autorisation d'utiliser l'ensemble des fonctions liées à ces permissions comme la connexion réseau, la localisation de l'appareil, les droits d'écriture sur la carte mémoire...

`<application>` 2

Un manifeste contient un seul et unique nœud application qui en revanche contient des nœuds concernant la définition d'activités, de services…

`<activity>` 3

Déclare une activité présentée à l'utilisateur. Comme pour la plupart des déclarations que nous allons étudier, si vous oubliez ces lignes de configuration, vos éléments ne pourront pas être utilisés.

`<service>` 4

Déclare un composant de l'application en tant que service. Ici pas question d'interface graphique, tout se déroulera en tâche de fond de votre application.

`<receiver>` 5

Déclare un récepteur d'objets `Intent`. Cet élément permet à l'application de recevoir ces objets alors qu'ils sont diffusés par d'autres applications ou par le système.

`<provider>` 6

Déclare un fournisseur de contenu qui permettra d'accéder aux données gérées par l'application.

Qu'est-ce qu'une activité Android ?

Une activité peut être assimilée à un écran qu'une application propose à son utilisateur. Pour chaque écran de votre application, vous devrez donc créer une activité. La transition entre deux écrans correspond (comme vu un peu plus haut dans le cycle de vie d'une application) au lancement d'une activité ou au retour sur une activité placée en arrière-plan.

Une activité est composée de deux volets :

- la logique de l'activité et la gestion du cycle de vie de l'activité qui sont implémentés en Java dans une classe héritant de `Activity` (nous reviendrons plus tard sur tous ces concepts) ;
- l'interface utilisateur, qui pourra être définie soit dans le code de l'activité soit de façon plus générale dans un fichier XML placé dans les ressources de l'application.

Voici venu le moment de voir à quoi ressemble l'activité la plus simple possible.

Code 3 : Squelette minimal pour créer une première activité

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

public class ActiviteSimple extends Activity {
   /**
    * Méthode appelée à la création de l'activité
    * @param savedInstanceState permet de restaurer l'état
    * de l'interface utilisateur
    */
```

```
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
}
```

Cycle de vie d'une activité

Tout ce que nous avons vu en parlant du cycle de vie d'une application, notamment sur la gestion des processus en fonction des ressources, a un impact direct sur les activités et notamment sur leur cycle de vie.

Les états principaux d'une activité sont les suivants :

- active (*active*) : activité visible qui détient le focus utilisateur et attend les entrées utilisateur. C'est l'appel à la méthode `onResume`, à la création ou à la reprise après pause qui permet à l'activité d'être dans cet état. Elle est ensuite mise en pause quand une autre activité devient active grâce à la méthode `onPause` ;
- suspendue (*paused*) : activité au moins en partie visible à l'écran mais qui ne détient pas le focus. La méthode `onPause` est invoquée pour entrer dans cet état et les méthodes `onResume` ou `onStop` permettent d'en sortir ;
- arrêtée (*stopped*) : activité non visible. C'est la méthode `onStop` qui conduit à cet état.

Voici un diagramme (voir figure 2) qui représente ces principaux états et les transitions y menant.

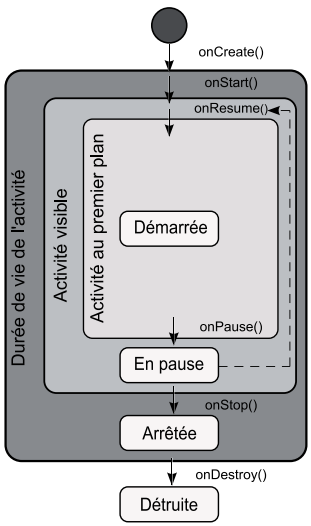


Figure 2
Cycle de vie d'une activité

Le cycle de vie d'une activité est parsemé d'appels aux méthodes relatives à chaque étape de sa vie. Il informe ainsi le développeur sur la suite des événements et le travail qu'il doit accomplir. Voyons de quoi il retourne en observant chacune de ces méthodes.

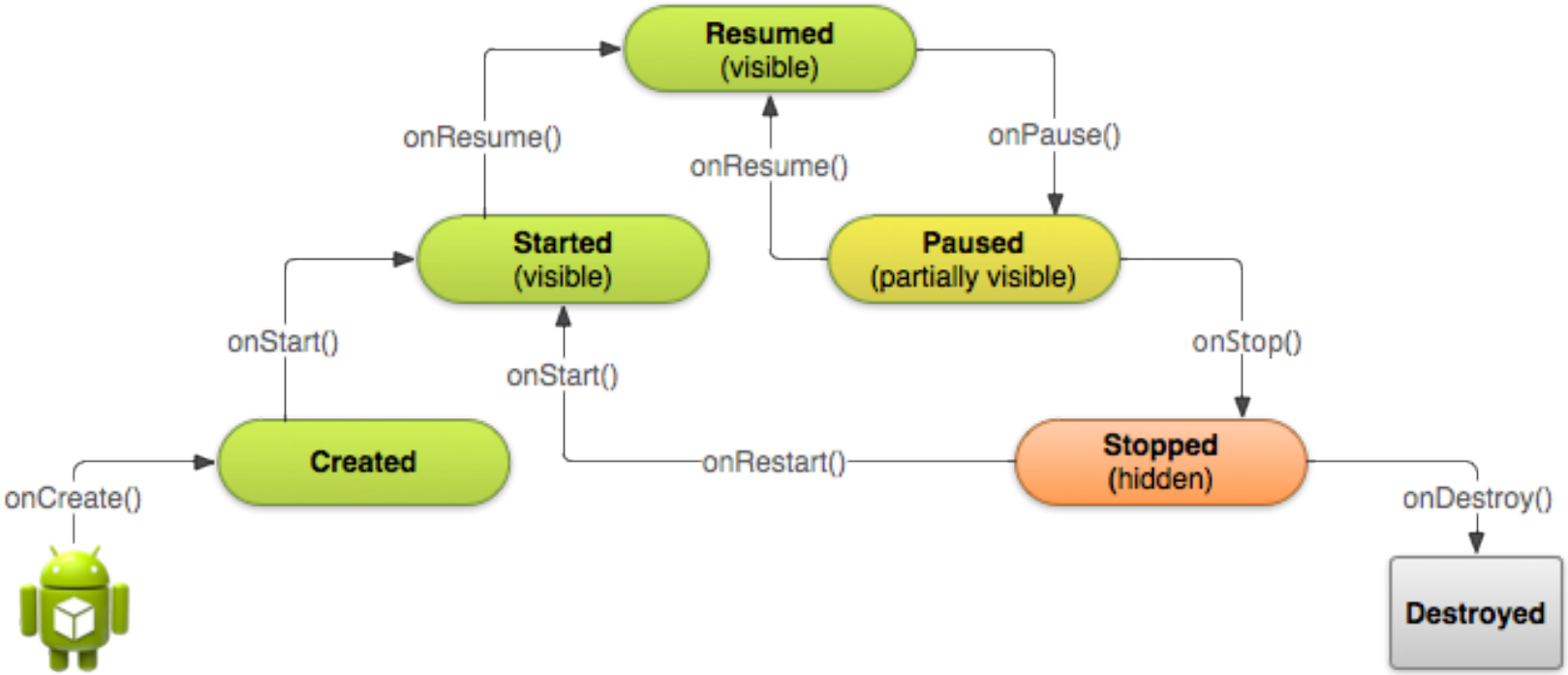
Code 4: Squelette d'une activité

```
package com.eyrolles.android.activity;

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

public final class TemplateActivity extends Activity {

   /**
    * Appelée lorsque l'activité est créée.
    * Permet de restaurer l'état de l'interface
    * utilisateur grâce au paramètre savedInstanceState.
    */
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      // Placez votre code ici
   }

   /**
    * Appelée lorsque que l'activité a fini son cycle de vie.
    * C'est ici que nous placerons notre code de libération de
    * mémoire, fermeture de fichiers et autres opérations
    * de "nettoyage".
    */
   @Override
   public void onDestroy(){
      // Placez votre code ici
      super.onDestroy();
   }
   /**
    * Appelée lorsque l'activité démarre.
    * Permet d'initialiser les contrôles.
    */
   @Override
   public void onStart(){
      super.onStart();
      // Placezvotre code ici
   }

   /**
    * Appelée lorsque l'activité passe en arrière plan.
    * Libérez les écouteurs, arrêtez les threads, votre activité
    * peut disparaître de la mémoire.
    */
```

```
@Override
public void onStop(){
   // Placez votre code ici
   super.onStop();
}

/**
 * Appelée lorsque l'activité sort de son état de veille.
 */
@Override
public void onRestart(){
   super.onRestart();
   //Placez votre code ici
}

/**
 * Appelée lorsque que l'activité est suspendue.
 * Stoppez les actions qui consomment des ressources.
 * L'activité va passer en arrière-plan.
 */
@Override
public void onPause(){
   //Placez votre code ici
   super.onPause();
}

/**
 * Appelée après le démarrage ou une pause.
 * Relancez les opérations arrêtées (threads).
 * Mettez à jour votre application et vérifiez vos écouteurs.
 */
@Override
public void onResume(){
   super.onResume();
   // Placez votre code ici
}

/**
 * Appelée lorsque l' activité termine son cycle visible.
 * Sauvez les données importantes.
 */
@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
   // Placez votre code ici
   // sans quoi l'activité aura perdu son état
   // lors de son réveil
   super.onSaveInstanceState(savedInstanceState);
}
```

```
    /**
     * Appelée après onCreate.
     * Les données sont rechargées et l'interface utilisateur.
     * est restaurée dans le bon état.
     */
    @Override
    public void onRestoreInstanceState(Bundle savedInstanceState) {
        super.onRestoreInstanceState(savedInstanceState);
        //Placez votre code ici
    }
}
```

Cycle de vie “pyramidal”



Pourquoi implémenter ces méthodes ?

Cela est important afin que votre application fonctionne correctement :

- Réception d'une appel et basculement sur une autre application ;
- Ne pas consommer trop de ressources système ;
- Ne pas avoir de problème lors de la création/restauration de l'application par le système (lors d'une rotation de l'écran par exemple).

Remarque : avant de distribuer une application, Il est donc important d'avoir fait des tests dans les situations évoquées ci-dessus.

Les vues

Les vues sont les briques de construction de l'interface graphique d'une activité Android. Les objets `View` représentent des éléments à l'écran qui permettent d'interagir avec l'utilisateur via un mécanisme d'événements.

Plus concrètement, chaque écran Android contient un arbre d'éléments de type `View` dont chaque élément est différent de par ses propriétés de forme, de taille...

Bien que la plupart des éléments dont nous ayons besoin – textes, boutons... – soient fournis par la plate-forme, il est tout à fait possible de créer des éléments personnalisés.

Les vues peuvent être disposées dans une activité (objet `Activity`) et donc à l'écran soit par une description XML, soit par un morceau de code Java.

Tous ces aspects de création d'interfaces graphiques sont abordés en détails dans le chapitre suivant.

Les ressources

Le but est ici de présenter les différents types de ressources prises en charge et les concepts généraux concernant leur utilisation (syntaxe, format, appel…) dans les projets. La mise en pratique plus concrète de ces notions se fera tout au long des exemples développés sur chaque thème et les concepts plus avancés comme l'internationalisation sont développés plus en détails dans le chapitre traitant des interfaces utilisateur avancées.

> **À RETENIR** **Les ressources**
>
> Les ressources sont des fichiers externes – ne contenant pas d'instruction – qui sont utilisés par le code et liés à votre application au moment de sa construction. Android offre un support d'un grand nombre de fichiers ressources comme les fichiers images JPEG et PNG, les fichiers XML…

L'externalisation des ressources en permet une meilleure gestion ainsi qu'une maintenance plus aisée. Android étend ce concept à l'externalisation de la mise en page des interfaces graphiques (*layouts*) en passant par celle des chaînes de caractères, des images et bien d'autres...

Physiquement, les ressources de l'application sont créées ou déposées dans le répertoire `res` de votre projet. Ce répertoire sert de racine et contient lui-même une arborescence de dossiers correspondant à différents types de ressources.

Tableau 1 Les types majeurs de ressources avec leur répertoire associé

Type de ressource	Répertoire associé	Description
Valeurs simples	`res/values`	Fichiers XML convertis en différents types de ressources. Ce répertoire contient des fichiers dont le nom reflète le type de ressources contenues : 1. `arrays.xml` définit des tableaux ; 2. `string.xml` définit des chaînes de caractères ; 3. ...
Drawables	`res/drawable`	Fichiers `.png`, `.jpeg` qui sont convertis en bitmap ou `.9.png` qui sont convertis en "9-patches" c'est-à-dire en images ajustables. Note : à la construction, ces images peuvent être optimisées automatiquement. Si vous projetez de lire une image bit à bit pour réaliser des opérations dessus, placez-la plutôt dans les ressources brutes.
Layouts	`res/layout`	Fichiers XML convertis en mises en page d'écrans (ou de parties d'écrans), que l'on appelle aussi gabarits.
Animations	`res/anim`	Fichiers XML convertis en objets animation.
Ressources XML	`res/xml`	Fichiers XML qui peuvent être lus et convertis à l'exécution par la méthode `resources.getXML`.
Ressources brutes	`res/raw`	Fichiers à ajouter directement à l'application compressée créée. Ils ne seront pas convertis.

Toutes ces ressources sont placées, converties ou non, dans un fichier de type `APK` qui constituera le programme distribuable de votre application. De plus, Android crée une classe nommée `R` qui sera utilisée pour se référer aux ressources dans le code.