

Développement d'applications mobiles

Chapitre 3 : Structure d'un projet Android (2/2)



Objectifs du cours

- Gérer les ressources en fonction de la taille et de la densité d'écran
- Internationaliser son application Android

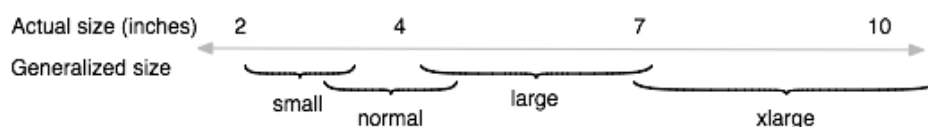
Avec la diversité des machines sous lesquelles fonctionne Android, il faut exploiter toutes les opportunités offertes par les ressources pour développer des applications qui fonctionnent sur la majorité des terminaux. Une application Android polyvalente possède un fichier XML pour chaque type d'écran, de façon à pouvoir s'adapter. En effet, si vous développez une application uniquement à destination des petits écrans, les utilisateurs de tablettes trouveront votre application illisible et ne l'utiliseront pas.

1. Taille et densité d'écran

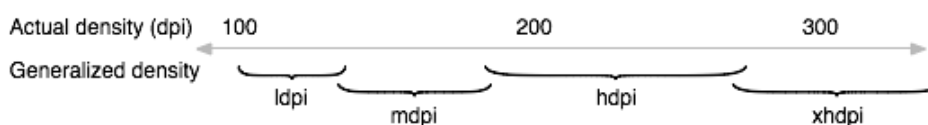
Le design d'une application Android doit être adapté à tous les types d'écran. Il ne faut pas que les vues affichées dans un appareil (smartphone ou tablette) soient décalés ou de taille inappropriée lorsque qu'elles sont affichées dans un autre. En effet, 20px sur 2 appareils de taille et/ou résolution (densité) différentes n'auront pas le même rendu, car leurs écrans n'ont ni la même taille ni la même densité d'écran.

Donc, Android fournit des techniques qui permettent d'optimiser l'interface graphique des applications pour une large variété de taille et de densité d'écran. Dans un premier temps, Android a catégorisé les différents appareils selon leur taille et leur densité :

- 4 catégories de taille : **small**, **normal**, **large** et **xlarge**



- 6 catégories de densité : **ldpi**, **mdpi**, **hdpi**, **xhdpi**, **xxhdpi** et **xxxhdpi**



Sachant que 1 pouce (inch) = 2,54 cm et dpi = dot per inch (pixels par pouce).

Selon Google Play Store (31 Août 2018) [1], le tableau suivant illustre la répartition des différents appareils Android dans le monde selon leur taille et leur densité :

	ldpi	mdpi	tvdpi	hdpi	xhdpi	xxhdpi	Total
Small	0.4 %					0.1 %	0.5 %
Normal		0.8 %	0.3 %	25.4 %	41.2 %	24.4 %	92.1 %
Large		2.4 %	1.4 %	0.3 %	0.5 %		4.9 %
Xlarge		1.6 %		0.5 %	0.4 %		2.5 %
Total	0.4 %	4.8 %	1.7 %	26.2 %	41.9 %	25.0 %	100 %

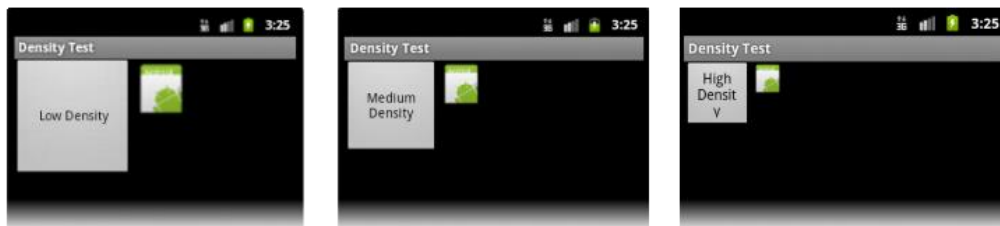
Indépendance de la densité

L'indépendance de la densité, permet de préserver la taille physique (du point de vue de l'utilisateur) des éléments graphiques lorsqu'ils sont affichés sur des écrans avec des densités différentes. En effet, un élément graphique (tel qu'un bouton) apparaît plus grand sur un écran à faible densité et plus petit sur un autre à haute densité.

Par conséquent, les pixels indépendants de la densité (dp : Density independant Pixel) sont un moyen de spécifier des dimensions d'écran qui s'adaptent à des matériels différents en fonction de leur densité en pixels [2]. Voici comment un dp est calculé en fonction de la densité de l'écran (dpi) : $dp = px / (dpi / 160)$.

Les deux figures suivantes montrent l'intérêt de l'indépendance de la densité :

Dépendance de la densité (px)



Indépendance de la densité (dp)



Remarque :

Il est préférable d'éviter, partout où cela est possible, d'utiliser des valeurs codées en dur (px ou dp). Plutôt, utiliser des **RelativeLayout** (décrits dans la section 2.3).

2. Prise en charge des différentes configurations

Sous Android, il est possible d'adapter les ressources de l'application (**layout**, **drawable**, **dimens**, ...), en fonction de [3] :

- Pays et réseau de la carte SIM,
- Langue et région (**ar**, **fr**, **en**, ...) (Voir la section 2.2),
- Taille de l'écran (**small**, **normal**, **large** et **xlarge**),
- Ratio largeur/longueur (**long** ou **notlong**) de l'écran,
- Orientation de l'écran (**port** pour portrait, **land** pour landscape ou **square** pour carré),
- Densité de l'écran en pixel (**ldpi**, **mdpi**, **hdpi**, **xhdpi**, **xxhdpi** ou **xxxhdpi**),
- Type d'écran tactile (**notouch** pour non sensible, **stylus** pour le stylet et **finger** pour sensible),
- Visibilité du clavier (**keyexposed** pour clavier physique apparent, **keyshidden** pour clavier physique caché ou **keysoft** pour un clavier logiciel),
- Type de clavier (**nokeys**, **qwerty** ou **12Key**).

Chacun des qualificatifs est indépendant et peut être utilisé seul ou combiné à d'autres. Pour définir une configuration spécifique pour un dossier de ressource, il suffit de suffixer le nom du dossier avec des qualificatifs séparés par des "-". Ainsi, le dossier **layout-ar-small-port-ldpi-finger** spécifie le chemin du dossier contenant tous les layouts en langue arabe (**ar**), pour un petit portable (**small**), en mode portrait (**port**), avec un écran faible en densité (**ldpi**) et sensible au toucher (**finger**).

Il faut toujours définir, pour toutes les ressources, le dossier par défaut (celui qui n'est suffixé par aucun qualificatif), par exemple **res/drawable**, **res/drawable**, **res/values**, ...

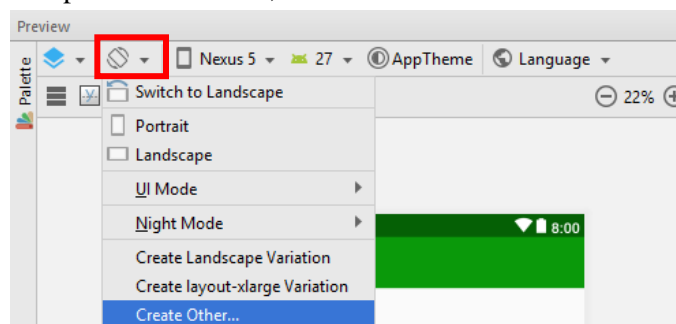
- Country Code
- Network Code
- Locale
- Layout Direction
- Smallest Screen Width
- Screen Width
- Screen Height
- Size
- Ratio
- Orientation
- UI Mode
- Night Mode
- Density
- Touch Screen
- Keyboard
- Text Input
- Navigation State
- Navigation Method
- Dimension
- Version


Les consignes d'Android concernant la prise en compte des différentes configurations sont :

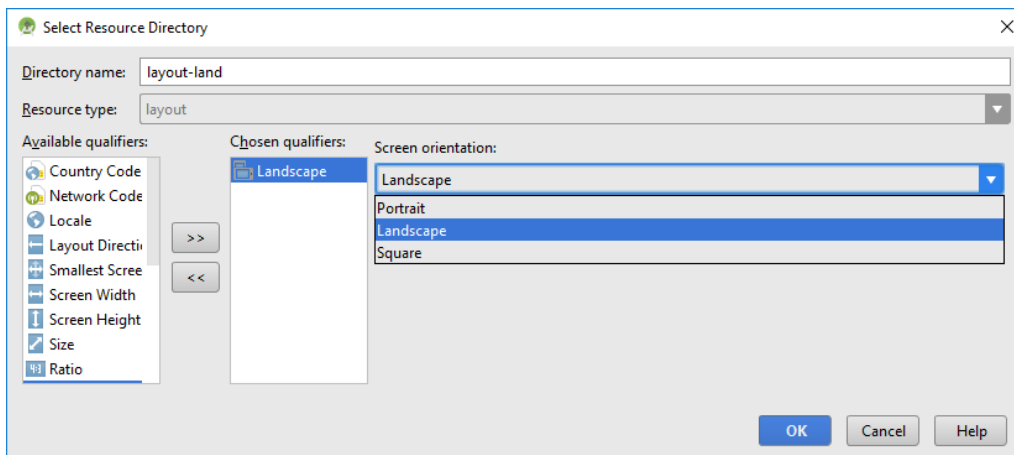
- Concevoir une application qui marche pour toutes les locales (**res/values** représente les valeurs par défaut et doit toujours exister),
- Concevoir des layouts flexibles qui s'adaptent à la taille de l'écran (en utilisant des **RelativeLayout**),
- Récupérer des ressources dynamiquement dans le code source Java, grâce à : **getResources()**, **getString()**, ...
- Tester l'application avec différentes configurations,
- Définir la liste des configurations de l'application, les tester et les noter dans le Play store lors de la publication de l'application.

Pour manipuler toutes les configurations possibles que l'application peut avoir, Android Studio fournit un outil graphique pour cela :

- 1) Ouvrir un layout, par exemple **res/layout/layout_login.xml**
- 2) En haut à droite, ouvrir la palette "Preview",



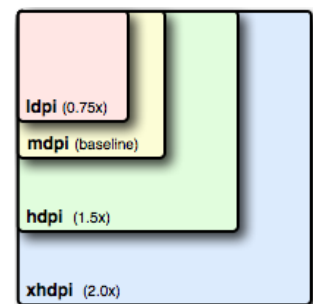
- 3) Cliquer sur l'icône , puis sur "Create Other...", une fenêtre est ouverte aussitôt
- 4) Choisir la configuration voulue, la figure suivante illustre comment peut-on choisir les versions paysage des layouts.



2.1. Drawables sur différentes densités d'écran

Les images matricielles sont toujours stockées dans **res/drawable**, cependant, une image doit être affichée sur tous les différents écran. Il faut donc, créer plusieurs variantes (plusieurs tailles) de l'image selon la densité de l'écran [4] :


- **drawable-ldpi/graphic.png** 36 x 36 (0.75x)
- **drawable-mdpi/graphic.png** 48 x 48 (1.0x baseline)
- **drawable-hdpi/graphic.png** 72 x 72 (1.5x)
- **drawable-xhdpi/graphic.png** 96 x 96 (2.0x)
- **drawable-xxhdpi/graphic.png** 144 x 144 (3.0x)
- **drawable-xxxhdpi/graphic.png** 192 x 192 (4.0x)





Remarque :

Il est possible d'utiliser des icônes vectorielles qui sont génériques (supportées par n'importe quelle densité d'écran sans perdre leur qualité). Pour les intégrer dans le projet, il faut :

- 1) Cliquer  (clic droit) sur le dossier **res/drawable**, puis choisir "New" → "Vector Asset",
- 2) Choisir soit des icônes, dites "Material", qui sont prédéfinies dans Android Studio, soit des fichiers locaux de type SVG ou PSD,
- 3) Définir le nom du fichier,
- 4) Cliquer sur "Next", ensuite sur "Finish" pour confirmer.

2.2. Internationalisation

Pour internationaliser une application Android, il faut que le dossier de ressources contienne la langue associée, par exemple : **res/values-ar** pour l'arabe et **res/values-fr** pour le français, ainsi de suite. "**ar**" et "**fr**" sont, respectivement, les valeurs suffixes des langues arabe et française lors de l'internationalisation. La langue anglaise est utilisée par défaut, si le dossier de ressources n'est suffixé par aucune langue.

Voici un exemple de fichier **strings.xml** :

res/values/strings.xml

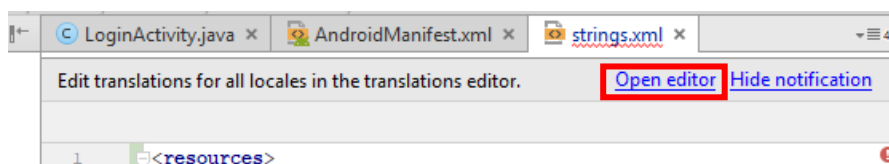
```
1 <resources>
2     <string name="hello"> Hello </string>
3     ...
4 </resources>
```

res/values-ar/strings.xml

```
1 <resources>
2     <string name="hello"> سلام </string>
3     ...
4 </resources>
```

Afin de manipuler les strings de l'application, Android Studio offre un outil graphique pour bien les gérer. Pour ce faire il faut :

- 1) Ouvrir le fichier **res/values/strings.xml**
- 2) En haut, cliquer sur "Open editor"



Afin de changer la langue de l'application par programmation, il suffit de rajouter à votre code les instructions suivantes, sachant que la langue de destination est l'arabe (ligne 2).

java/LoginActivity.java

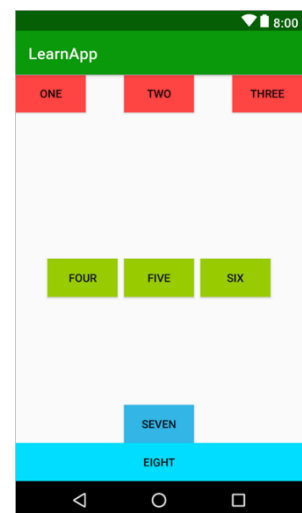
```
1 ...
2 Locale myLocale = new Locale("ar");
3 DisplayMetrics dm = getResources().getDisplayMetrics();
4 Configuration conf = getResources().getConfiguration();
5 conf.locale = myLocale;
6 getResources().updateConfiguration(conf, dm);
7 ...
8 Intent refresh = new Intent(this, LoginActivity.class);
9 startActivity(refresh);
10 finish();
11 ...
```

2.3. RelativeLayout

Un **RelativeLayout** [5] propose de placer des vues les unes par rapport aux autres. Il est même possible de les placer par rapport au conteneur (**RelativeLayout**).

Voici les différents moyens qui sont à la disposition du développeur pour le placement des vues :

- Aligner par rapport une autre vue ("@[+][package:]type:name")
 - **android:layout_above**, **android:layout_below**
 - **android:layout_alignStart**, **android:layout_alignEnd**
 - **android:layout_alignTop**, **android:layout_alignBottom**
 - **android:layout_alignLeft**, **android:layout_alignRight**
- Aligner par rapport au parent ("true" ou "false")
 - **android:layout_alignParentStart**, **android:layout_alignParentEnd**
 - **android:layout_alignParentTop**, **android:layout_alignParentBottom**
 - **android:layout_alignParentLeft**, **android:layout_alignParentRight**
 - **android:layout_alignWithParentIfMissing**
- Centrer par rapport au parent ("true" ou "false")
 - **android:layout_centerHorizontal**, **android:layout_centerVertical**
 - **android:layout_centerInParent**
- Positionner par rapport à une autre vue ("@[+][package:]type:name")
 - **android:layout_toStartOf**, **android:layout_toEndOf**
 - **android:layout_toLeftOf**, **android:layout_toRightOf**



Exercice

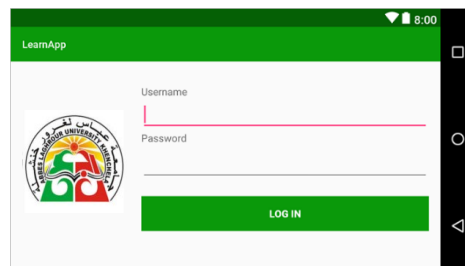
Dans l'exercice suivant, il est demandé de créer une vue d'activité, appelée **activity_complex**, en utilisant un **RelativeLayout**. Cette vue doit contenir 8 boutons organisés selon la figure suivante.

Solution

```
res/layout/activity_complex.xml
1  <RelativeLayout
2      android:layout_width="match_parent"
3      android:layout_height="match_parent">
4      <Button android:text="ONE"
5          android:layout_width="wrap_content"
6          android:layout_height="wrap_content"
7          android:id="@+id/Button01" />
8      <Button android:text="TWO"
9          android:layout_width="wrap_content"
10         android:layout_height="wrap_content"
11         android:id="@+id/Button02"
12         android:layout_centerHorizontal="true" />
13     <Button android:text="THREE"
14         android:layout_width="wrap_content"
15         android:layout_height="wrap_content"
16         android:id="@+id/Button03"
17         android:layout_alignParentRight="true" />
18     <Button android:text="FOUR"
19         android:layout_width="wrap_content"
20         android:layout_height="wrap_content"
21         android:id="@+id/Button04"
22         android:layout_toLeftOf="@+id/Button05"
23         android:layout_centerVertical="true" />
24     <Button android:text="FIVE"
25         android:layout_width="wrap_content"
26         android:layout_height="wrap_content"
27         android:id="@+id/Button05"
28         android:layout_centerInParent="true" />
29     <Button android:text="SIX"
30         android:layout_width="wrap_content"
31         android:layout_height="wrap_content"
32         android:id="@+id/Button06"
33         android:layout_toRightOf="@+id/Button05"
34         android:layout_centerVertical="true" />
35     <Button android:text="SEVEN"
36         android:layout_width="wrap_content"
37         android:layout_height="wrap_content"
38         android:id="@+id/Button07"
39         android:layout_above="@+id/Button08"
40         android:layout_centerHorizontal="true" />
41     <Button android:text="EIGHT"
42         android:layout_width="match_parent"
43         android:layout_height="wrap_content"
44         android:id="@+id/Button08"
45         android:layout_alignParentBottom="true"/>
46 </RelativeLayout>
```

3. Travail pratique (TP1b) : Vue paysage

Le TP 1 consiste à implémenter une activité d'authentification sous Android. Dans la partie 2 du TP 1, il est demandé de proposer une vue paysage de l'activité **LoginActivity** comme dans la capture d'écran suivante (Voir l'énoncé du TP 1).



Liens utiles

Les étudiants peuvent consulter ces références pour approfondir leurs connaissances dans ce cours :

- Configurations d'écran : developer.android.com/guide/topics/resources/providing-resources.html
- Drawables : <http://vogella.developpez.com/tutoriels/android/utilisation-drawable/>
- Android Drawable Import (Plugin) : <https://plugins.jetbrains.com/plugin/7658-android-drawable-importer>

Références

- [1] Google Inc., «Google Play,» 4 Septembre 2017. [En ligne]. Available: <https://developer.android.com/about/dashboards/>.
- [2] Android Developer, «Supporting Multiple Screens,» [En ligne]. Available: https://developer.android.com/guide/practices/screens_support.html. [Accès le 2017].
- [3] M. Seguy et Y. Bergès, Android, A Complete Course, From Basics To Enterprise Edition, Édition Française éd., 2011, p. 279.
- [4] S. Walter, «Android script Photoshop pour exporter en ldpi, hdpi et xhdpi,» 5 Janvier 2012. [En ligne]. Available: <https://blog.stephaniewalter.fr/android-script-photoshop-pour-exporter-en-ldpi-hdpi-et-xhdpi/>.
- [5] Android Developer, «References - Android Platform API 26 : Relative Layout,» [En ligne]. Available: <https://developer.android.com/guide/topics/ui/layout/relative.html>. [Accès le 2017].