**Les layouts**

1. **Introduction**

Les conteneurs (Layout) permettent de disposer d'un ensemble de widgets (et, éventuellement, desconteneurs fils) pour obtenir une bonne présentation.

Les layouts permettent de concevoir des applications plus attractives et plus ergonomiques.

On peut avoir plusieurs layouts dans une application.

**Avec xml**

Pour créer un Layout, clic droit sur **res/layout** puis **New / Androïd XML File.**

Donner un nom : **layout\_file\_name.xml**

Saisir le texte suivant dans le fichier **layout\_file\_name.xml**

**<?xmlversion=*"1.0"*encoding=*"utf-8"*?>**

**<LinearLayoutxmlns:android=*"http://schemas.android.com/apk/res/android"***

**android:layout\_width=*"fill\_parent"***

**android:layout\_height=*"fill\_parent"***

**android:orientation=*"vertical"*>**

**<TextViewandroid:id=*"@+id/text"***

**android:layout\_width=*"wrap\_content"***

**android:layout\_height=*"wrap\_content"***

**android:text=*"Hello, I am a TextView"*/>**

**<Buttonandroid:id=*"@+id/button"***

**android:layout\_width=*"wrap\_content"***

**android:layout\_height=*"wrap\_content"***

**android:text=*"Hello, I am a Button"*/>**

**</LinearLayout>**

**Code java**

**publicclass**HelloAndroid extends Activity {

**public** void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.layout\_file\_name);

}}

**Avec java**

LinearLayoutll = **new** LinearLayout(**this**);

ll.setGravity(Gravity.CENTER);

ll.setOrientation(LinearLayout.VERTICAL);

Ajouter une vue par la méthodeaddView(la\_vue) du layout.

**protected void** onCreate(Bundle savedInstanceState) {  
**super**.onCreate(savedInstanceState);  
LinearLayoutll = **new** LinearLayout(**this**);  
setContentView(ll);

TextViewtv=**new** TextView(**this**);  
tv.setText(**"ddddddddddddd"**);  
ll.addView(tv);  
}

1. **LinearLayout : placer les éléments sur une ligne**

CeLayoutpermet de mettre les vues sur une même ligne, selon une certaine orientation.

Pour configurer un LinearLayout, on peut agir sur cinq paramètres : ***l’orientation, le modèle de remplissage, le poids, la gravité et le remplissage.***

* **L'orientation**(android:orientation)

Cet attribut peut avoir deux valeurs :

* vertical pour que les composants soient placés de haut en bas (en colonne) ;
* horizontal pour que les composants soient placés de gauche à droite (en ligne).

**En java : utiliser** la méthode setOrientation() avec le paramètre constante HORIZONTAL ou VERTICAL.

* **Modèle de remplissage**

android:layout\_width et android:layout\_height.

**Exemple1**

Soit unLayoutcontenant deux boutons.

 Le LinearLayout est vertical et prend toute la place de son parent.

 Chaque bouton prend toute la place dans le parent en largeur et uniquement la taille nécessaire en hauteur (donc la taille du texte).

<LinearLayoutxmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="vertical"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

android:background="#003344">

<Button

android:id="@+id/premier"

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:text="Premier bouton" />

<Button

android:id="@+id/second"

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:text="Second bouton" />

</LinearLayout>

Changer l'orientation avec du code java.

**Exemple 2**

* Le LinearLayout est vertical et prend toute la place de son parent.
* Chacun des deux boutons prend toute la place de son parent en hauteur et uniquement la taille nécessaire en largeur.

**Exemple 3**

* Le LinearLayout est vertical et prend toute la place en largeur mais uniquement la taille nécessaire en hauteur : dans ce cas précis, la taille nécessaire sera calculée en fonction de la taille des enfants.
* Chacun des deux boutons prend toute la place possible dans le parent. Comme le parent prend le moins de place possible, ces boutons doivent faire de même.

<LinearLayoutxmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="vertical"

android:layout\_width="match\_content"

android:layout\_height="wrap\_content" >

<Button

android:id="@+id/premier"

android:layout\_width="wrap\_content"

android:layout\_height="fill\_parent"

android:text="Premier bouton" />

<Button

android:id="@+id/second"

android:layout\_width="wrap\_content"

android:layout\_height="fill\_parent"

android:text="Second bouton" />

</LinearLayout>

**Exemple 4**

* Le LinearLayout est horizontal et prend toute la place de son parent.
* Le premier bouton prend uniquement la place nécessaire.
* Le deuxième prend uniquement la place nécessaire en longueur et s'étend jusqu'aux bords du parent en hauteur.
* **Poidsdes composants :** partage de l’espace disponible

Supposons que nous avons deux champs de saisie multilignes en colonne et que nous voulions qu’ils occupent tout l’espace disponible de la colonne après le placement de tous les autres widgets.

* Pour ce faire on utilise le poids défini grâce à l'attribut android:layout\_weight.

android:layout\_width=fill\_parentou

android:layout\_height =fill\_parent

android:layout\_weight = une valeur qui indique la proportion d’espace libre qui sera affectée au widget.

* Si cette valeur est la même pour les deux widgets (1, par exemple), l’espace libre sera partagééquitablement entre eux.
* Si la valeur est 1 pour un widget et 2 pour l’autre, le second utilisera deux fois plus d’espace libre que le premier, etc.

Le poids d’un widget est fixéà zéro par défaut.

Un autre moyen d’utiliser les poids consiste àutiliser des pourcentages. Pour utiliser cette technique avec une disposition en ligne, par exemple :

* + Initialisez à zéro les valeurs android:layout\_widthde tous les widgets du layout.
  + Initialisez avec les pourcentages les valeurs android:layout\_weightde tousles widgets du layout.

La somme de ces pourcentages doitêtreégale à 100.

**Exemple**

Pour que le bouton de droite prenne deux fois plus de place, on peut lui mettre android:layout\_weight="1" et mettre au bouton de gauche android:layout\_weight="2".

* **Où placer la vue : Gravity**

Par défaut, les widgets s’alignent à partir de la gauche et du haut. Si vous créez une ligne avec un LinearLayouthorizontal, cette ligne commencera donc à se remplir à partir du bord gauche de l’écran.

Sinon, il faut indiquer une gravitéà l’aide de la propriétéandroid:layout\_gravityou avec la méthode setGravity() afin d’indiquer au widget et à son conteneur comment l’aligner par rapport à l’écran.

***android:layout\_gravity, à ne pas confondre avec android:gravity.***

* android:layout\_gravity permet de déterminer comment se place la vue dans le parent.
* android:gravitypermet de déterminer comment se place le contenu de la vue à l'intérieur de la vue.
* **Remplissage**

Les widgets sont, par défaut, serrés les uns contre les autres. On peut augmenter l’espace entre ces vues à l’aide de la propriétéandroid:paddingou avec la méthode setPadding()du widget.

La valeur de remplissage précise l’espace situé entre le contour de la "cellule" du widget et son contenu réel.

|  |  |
| --- | --- |
| <LinearLayoutxmlns:android="http://schemas.android.com/apk/res/android"  android:orientation="horizontal"  android:layout\_width="fill\_parent"  android:layout\_height="fill\_parent" >  <Button  android:id="@+id/bouton1"  android:layout\_width="fill\_parent"  android:layout\_height="wrap\_content"  android:layout\_gravity="bottom"  android:layout\_weight="40"  android:text="Bouton 1" />  <Button  android:id="@+id/bouton2"  android:layout\_width="fill\_parent"  android:layout\_height="wrap\_content"  android:layout\_gravity="center"  android:layout\_weight="20"  android:gravity="bottom|right"  android:text="Bouton 2" />  <Button  android:id="@+id/bouton3"  android:layout\_width="fill\_parent"  android:layout\_height="wrap\_content"  android:layout\_gravity="top"  android:layout\_weight="40"  android:text="Bouton 3" />  </LinearLayout> | Trois boutons placés différemment |

1. **RelativeLayout : placer les éléments les uns en fonction des autres**

De manière totalement différente, ceLayoutsert à placer les composants les uns par rapport aux autres. Il est même possible de les placer par rapport au RelativeLayout.

On peut ainsi placer le widget X en dessous et à gauche du widget Y ou que le bord inférieur du widget Z soit aligné avec le bord inférieur du conteneur, etc.

L'intérêt de ceLayoutest de placer les éléments les uns par rapport aux autres.

* android:layout\_centerInParent="true" : placela vue au centre du Layout
* android:layout\_centerHorizontal="true"centre sur l'axe horizontal
* android:layout\_centerVertical="true" pour centrer sur l'axe vertical.
* android:layout\_alignParentBottom="true" pour aligner le plancher d'une vue au plancher du RelativeLayout ;
* android:layout\_alignParentTop="true" pour coller le plafond d'une vue au plafond du RelativeLayout ;
* android:layout\_alignParentLeft="true" pour coller le bord gauche d'une vue avec le bord gauche du RelativeLayout ;
* android:layout\_alignParentRight="true" pour coller le bord droit d'une vue avec le bord droit du RelativeLayout.

|  |  |
| --- | --- |
| <RelativeLayoutxmlns:android="http://schemas.android.com/apk/res/android"  android:layout\_width="fill\_parent"  android:layout\_height="fill\_parent" >  <TextView  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="En haut !"  android:layout\_alignParentTop="true" />  <TextView  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="En bas !"  android:layout\_alignParentBottom="true" />  <TextView  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="A gauche !"  android:layout\_alignParentLeft="true" />  <TextView  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="A droite !"  android:layout\_alignParentRight="true" />  <TextView  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="Ces soirées là !"  android:layout\_centerInParent="true" />  </RelativeLayout> | En haut à gauche, deux TextView se superposent |

Il y a deux types d'attributs :

* Ceux qui permettent de positionner deux bords opposés de deux vues différentes.

android:layout\_below : pour aligner le plafond d'une vue sous le plancher d'une autre,

android:layout\_above:pour aligner le plancher d'une vue sur le plafond d'une autre,

android:layout\_toRightOf : aligner le bord gauche d'une vue au bord droit d'une autre,

android:layout\_toLeftOf : aligner le bord droit d'une vue au bord gauche d'une autre).

* Ceux qui permettent de coller deux bords similaires ensemble.

android:layout\_alignBottom : aligne le plancher de la vue avec le plancher d'une autre.

android:layout\_alignTop : aligne le plafond de la vue avec le plafond d'une autre,

android:layout\_alignLeft:aligne le bord gauche d'une vue avec le bord gauche d'une autre

android:layout\_alignRight : aligne le bord droit de la vue avec le bord droit d'une autre.

|  |  |
| --- | --- |
| <RelativeLayoutxmlns:android="http://schemas.android.com/apk/res/android"  android:layout\_width="fill\_parent"  android:layout\_height="fill\_parent" >  <TextView  android:id="@+id/premier"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="[I] En haut à gauche par défaut" />  <TextView  android:id="@+id/deuxieme"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="[II] En dessous de (I)"  android:layout\_below="@id/premier" />  <TextView  android:id="@+id/troisieme"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="[III] dessous et à droite de (I)"  android:layout\_below="@id/premier"  android:layout\_toRightOf="@id/premier" />  <TextView  android:id="@+id/quatrieme"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="[IV] Au dessus de (V), bord gauche aligné avec le bord gauche de (II)"  android:layout\_above="@+id/cinquieme"  android:layout\_alignLeft ="@id/deuxieme" />  <TextView  android:id="@+id/cinquieme"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="[V] En bas à gauche"  android:layout\_alignParentBottom="true"  android:layout\_alignParentRight="true" />  </RelativeLayout> | Les TextView sont bien placés |

**Remarque**

L'attribut android:layout\_above de l'avant dernier TextView ne fait pas référence au dernier TextView comme aux autres, mais il précise un +.

En fait si on veut faire référence à une vue qui ne sera définie que plus tard dans le fichier XML, alors il faut ajouter un + dans l'identifiant.

En conclusion, le problème de ce Layout, est qu'une petite modification dans l'interface graphique peut provoquer de grosses modifications dans le fichier XML, il faut donc savoir par avance très précisément ce qu'on veut faire.

1. **TableLayout : placer les éléments comme dans un tableau**

DernierLayoutde base, il permet d'organiser les éléments en tableau, comme en HTML, mais sans les bordures.

Avec ceLayout,il est possible de mettre des vues directement dans le tableau, mais elles prendront toute la place possible en longueur (toute la ligne).

<TextViewandroid:text="Les items précédés d'un V ouvrent un sous-menu" />

Pour dessiner un séparateur dans un <TableLayout>:

<View

android:layout\_height="2dip"

android:background="#FF909090" />

Si on veut plusieurs éléments sur une même ligne, il faut passer par l'objet <TableRow>.

Une ligne est composée de cellules. Chaque cellule peut contenir une vue, ou être vide. La taille du tableau en colonnes est celle de la ligne qui contient le plus de cellules.

<TableRow>

<TextView

android:text="V"

/>

<TextView

android:text="Ouvre un sous-menu"

android:layout\_column="1"

android:padding="3dip"

/>

<TextView

android:text="Là si !"

android:gravity="right"

android:padding="3dip"

/>

</TableRow>

Pour laisser vide l'espace de la première colonne, on place les deux TextView dans les colonnes 1 et 2.

* Il est possible d'étendre un item sur plusieurs colonnes avecandroid:layout\_span.
* Il est possible de faire de même sur les lignes avec l'attribut android:layout\_column.

Sur le nœud TableLayout, on peut utiliser trois attributs :Il faut préciser le rang de la colonne à cibler, ou plusieurs rangs séparés par des virgules.

* android:stretchColumns : la longueur de tous les éléments de cette colonne passe en fill\_parent, donc pour prendre le plus de place possible.
* android:shrinkColumns : la longueur de tous les éléments de cette colonne passe en wrap\_content, donc pour prendre le moins de place possible.
* android:collapseColumns pour faire disparaître des colonnes du tableau.

**Exemple** : android:stretchColumns="1">

|  |  |
| --- | --- |
| <?xml version="1.0" encoding="utf-8"?>  <TableLayoutxmlns:android="http://schemas.android.com/apk/res/android"  android:layout\_width="fill\_parent"  android:layout\_height="fill\_parent"  android:stretchColumns="1">  <TextView  android:text="Les items précédés d'un V ouvrent un sous-menu"  />  <View  android:layout\_height="2dip"  android:background="#FF909090"  />  <TableRow>  <TextView  android:text="N'ouvre pas un sous-menu"  android:layout\_column="1"  android:padding="3dip"  />  <TextView  android:text="Non !"  android:gravity="right"  android:padding="3dip"  />  </TableRow>  <TableRow>  <TextView  android:text="V"  />  <TextView  android:text="Ouvre un sous-menu"  android:layout\_column="1"  android:padding="3dip"  />  <TextView  android:text="Là si !"  android:gravity="right"  android:padding="3dip"  />  </TableRow>  <View  android:layout\_height="2dip"  android:background="#FF909090"  />  <TableRow>  <TextView  android:text="V"  />  <TextView  android:text="Ouvre un sous-menu"  android:padding="3dip"  />  </TableRow>  <View  android:layout\_height="2dip"  android:background="#FF909090"  />  <TableRow>  <TextView  android:layout\_column="1"  android:layout\_span="2"  android:text="Cet item s'étend sur deux colonnes, cool hein ?"  android:padding="3dip"  />  </TableRow>  </TableLayout> | Le contenu est organisé en tableau |

1. **FrameLayout : unLayoutun peu spécial**

CeLayoutest utilisé pour afficher une unique vue. Il n'est destiné à afficher qu'un élément, mais il est possible de mettre plusieurs dedans puisqu'il s'agit d'un ViewGroup.

Si par exemple on veut faire un album photo, il suffit de mettre plusieurs éléments dans le FrameLayout et ne laisser qu'une seule photo visible, et les autres invisibles avec l'attribut android:visibility.

Cet attribut peut prendre trois valeurs :

* visible (View.VISIBLE), la valeur par défaut.
* invisible (View.INVISIBLE) n'affiche rien, mais est pris en compte pour l'affichage duniveau spatial (on lui réserve de la place).
* gone (View.GONE) n'affiche rien et ne prend pas de place, ignoré.

L'équivalent Java de cet attribut est public voidsetVisibility (int).

Quand il y a plusieurs éléments FrameLayout les empile les uns au-dessus des autres.

1. **ScrollView : défiler le contenu d'une vue**

Il s'agit d'ajouter une barre de défilement verticale à un autre layout.

CeLayouthérite de FrameLayout, il vaut mieux ne mettre qu'une seule vue dedans.  
Il s'utilise avec LinearLayout, mais aussi avec tous les Layouts… ou bien des widgets.

**Exemple** :

<ScrollView

xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content">

<LinearLayout>

<!--contenu du Layout -->

</LinearLayout>

</ScrollView>

1. **AbsoluteLayout**

Permet de placer les éléments par positions fixes. Chaque élément d'un AbsoluteLayout indiquera sa position par :

– android:layout\_x="unité"

– android:layout\_y="unité"

1. **ConstraintLayout**

Apparu en Mai 2016, ConstraintLayout  ressemble au RelativeLayout, En effet, en plus d’hériter des contraintes de placement, le *ConstraintLayout* donne la possibilité de lier des *Widgets*, *Text*, … entre eux, d’appliquer un "poids" à la manière du *LinearLayout* ou d’attribuer un identifiant unique à plusieurs éléments pour effectuer des actions groupées.

La création de l’interface se fera avec l’utilisation du mode **Design**  recommandé pour l’utilisation du ConstraintLayout.

Un widget placé dans un ConstraintLayout doit **avoir au moins deux contraintes**, une verticale et une horizontale.

**Aligner un widget au centre**

Pour aligner un élément au centre de l’écran il suffit de lui définir les quatre contraintes autour de lui.

**Les ressources**

1. **Introduction**

Les ressources sont des informations statiques, stockées en dehors du code Java. Toutes les ressources d’une application Android (chaines de caractères, couleurs, images, layout…)se trouvent dans des sous-répertoires du répertoire res/

Il existe une grande variété de ressources différentes. Plus une application contient de ressources plus elle est flexible.

À l’exception des ressources brutes (res/raw/), tous les types de ressources sont analysés automatiquement, soit par le système des packages d’Android, soit par le système du terminal ou de l’émulateur.

Les fichiers de ressources sont des fichiers XML.

* Les images:res/drawable
* Les ressources brutes : res/raw
* Les chaînes, les couleurs, les tableaux et les dimensions : res/values
* Les fichiers XML statiques : res/xml/

1. **Aspect général des fichiers de ressources**

Les fichiers de ressources sont des fichiers XML de la forme :

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

<resources>

…

</resources>

**Exemple :**

<resources>

<colorname="opaque\_red">#f00</color>

<stringname="hello">Hello world!</string>

</resources>

Afin d'avoir un aperçu d'un fichier de ressources double-cliquez dessus ( par exemple res/values/strings.xml)

**Utiliser une ressource** L’outil AAPT (Android Asset Packaging Tool) gère l’ensemble des ressources de l'application et les compile dans une classe R.   
Quand une ressource a un identifiant, Android ajoute une référence à l'identifiant de cette ressource, dans le fichier **R.java** pour pouvoir la récupérer.

**Accès aux ressources**

Dans le code source, on peut accéder à n’importe quelle ressource déclarée dans les fichiers XML placés dans les sous répertoires du répertoire res. 

* **Avec le code Java**:**R.type\_de\_ressource.nom\_de\_la\_ressource**
* **AvecXML**: **@type\_de\_ressource/nom\_de\_la\_ressource**

Par exemple pour une chaîne de caractères nommée donnees, on fait référence en Java à l'aide de **R.strings.donnees** et en XML avec **@string/donnees**.

Enfin, si la ressource à laquelle on essaie d'accéder est une ressource fournie par le système Android dans son SDK, il faut respecter la syntaxe :

en Java : Android.R.type\_de\_ressource.nom\_de\_la\_ressource

en XML : @android:type\_de\_ressource/nom\_de\_la\_ressource

**Accès aux ressources dans le Code java**

**Exemple :texteView**

<resources>

<stringname="hello">Hello tout le monde !</string>

</resources>

**Avec java**

TextViewmsgTextView=(TextView)findViewById(R.id.msg);

msgTextView.setText(R.string.hello);

**Avec xml**

<EditText

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

android:text=**"@string/hello"/>**

**Drowables : ImageViw**

Pour accéder à res/drowables/myimage.png utiliser le code suivant:

**R.drawable.nom\_du\_fichier\_sans\_l\_extension.**

ImageViewimageView=(ImageView)findViewById(R.id.myimageview);

imageView.setImageResource(R.drawable.myimage);

**La classe Resources**

Elle permet l’accès aux ressources répertoriées dans la classe R.

Pour accéder aux ressources avec cette classe il faut obtenir son instance par la méthode getResources() de l’activité.

• Quelques méthodes de la classe Resources (le paramètre de type int est un identifiant défini dans R de la forme R.type.nom) :

– booleangetBoolean(int)

– intgetInteger(int)

– int[] getArray(int)

– String getString(int)

– String[] getStringArray(int)

– intgetColor(int)

– float getDimension(int)

– DrawablegetDrawable(int)

**Exemple**

Resource re = getResources();

String titre=re.getString(R.string.ma\_chaine);

//String titre = R.string.ma\_chaine;

//…..utilisertitre ….

DrawablemonImage= getResources().getDrawable(R.drawable.nom\_image)

1. **Les chaînes de caractères**

**Chaînes normales**

Pour les chaînes normales on n'a besoin que d’un fichier XML situé dans le répertoire res/values.

**Caractères spéciaux**

Pour garder les caractères spéciaux il faut les précéder d'un antislash (\).

**<stringname="walili">Texte de la chaîne qui s\'appelle \" walili\"</string>**

On peut encadrer la chaîne par des guillemets afin de ne pas avoir à échapper les apostrophes, en revanche il faut toujours échapper les guillemets.

**<stringname="chaine">"Texte de la chaîne qui s'appelle \"walili\" "</string>**

**Texte stylé**

#### Application

Une chaîne de caractères qui contient des balises HTML telles que <b>, <u> et <i> ainsi que des guillemets et des apostrophes.

**Exemple**

**"**Vous connaissez l'histoire de <b>"**walili**"</b> ?**"**.

##### Etapes

* Convertir notre String en Spanned. C'est une classe qui représente les chaînes de caractères qui contiennent des balises HTML et qui peut les interpréter pour les afficher comme dans un navigateur. Cette transformation se fait à l'aide de la méthode statique Spanned**Html.fromHtml** (String source).
* On récupère ce Spanned comme texte avec la méthode setText(CharSequencetext).

Les caractères spéciaux < et > doivent être écrits en code HTML. Au lieu d'écrire < on doit écrire &laquo; et &raquo; à la place de >.

Le fichier string.xml

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

<resources>

<**string** name="hello">Hello World, mon Activity!</string>

<**string**name="histoire">Vous connaissez l\'histoire de <b>\"ait Ali\"</b> ?</string>

<string name="app\_name">mon nom</string>

</resources>

**Le code java associé**

public class StringExampleActivity extends Activity {

String hist = null;

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

// On récupère la ressource au format String

hist = getResources().getString(R.string.histoire);

// On le convertit en Spanned

Spanned sp = Html.fromHtml(hist);

texte.setText(sp);

}}

**Les ressources alternatives**

L'application devrait fournir des ressources alternatives pour soutenir les configurations de périphériques spécifiques.

Par exemple, vous devez inclure les ressources drowables alternatives (i.e.images) pour une résolution d'écran différente et des ressources de chaîne alternatives pour différentes langues...

Lors de l'exécution, Android détecte la configuration actuelle du périphérique et charge les ressources appropriées pour l'application.

Pour cela :

Créer un nouveau répertoire dans res/nom sous la forme <res\_name> - <config\_qualifier>. Ici res\_name sera l'une des ressources telles que la mise en page, etc.

Les fichiers de ressources doivent être nommés exactement comme les fichiers de ressources par défaut, mais ces fichiers auront un contenu spécifique à l'alternative.

#### Formater des chaînes de caractères

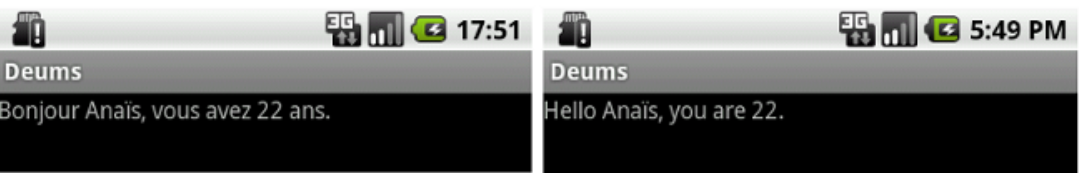
Les chaînes de caractères en tant que ressources sont statiques, elles ne sont pas destinées à être modifiées.

Imaginons une application qui salue quelqu'un, lui donne son âge, et s'adapte à la langue de l'utilisateur. Il faudrait qu'elle dise : « Bonjour Anas, vous avez 22 ans » en français et « Hello Anas, you are 22 » en anglais.

Pour indiquer dans une chaîne de caractères à quel endroit se situe la partie dynamique, on va utiliser un code.

**Exemple**

Dans l'exemple, on pourrait avoir ***Bonjour %1$s, vous avez %2$d ans*** en français et ***Hello %1$s, you are %2$d*** en anglais.



La première partie du code correspond à une position dans une liste d'arguments (à fournir) et la seconde partie à un type (int, float, string, bool, …) c-à-d, un code en deux parties :

* %n avec n un entier qui sert à indiquer le rang de l'argument à insérer (%1 correspond au premier argument, %2 au deuxième argument, etc.) ;
* $x type d'information ajouter ($s pour String et $d pour int, …).

Insérer les arguments en utilisant la ressource :

Resources res = getResources();

String chaine = res.getString(R.string.hello, "Anas", 22);

**Réalisation**

1. Pour représenter strings.xml On a besoin de deux fichiers: un dans le répertoire values et un dans le répertoire values-en qui contiendra le texte en anglais.

**values/strings.xml**

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

<resources>

<string name="hello">Bonjour %1$s, vous avez %2$d ans.</string>

</resources>

**values-en/strings.xml**

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

<resources>

  <string name="hello">Hello %1$s, you are %2$d.</string>

</resources>

1. **Mettre un texteView**

  <TextView

    android:id="@+id/vue"

    android:layout\_width="fill\_parent"

    android:layout\_height="wrap\_content" />

</LinearLayout>

1. Récupérer TextView et afficher le texte correct pour une personne s’appelant Anas et de 22 ans **en français ou en anglais selon la configuration du terminal :**

import android.app.Activity;

import android.content.res.Resources;

import android.os.Bundle;

public class DeumsActivity extends Activity {

  public void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);

    setContentView(R.layout.main);

    Resources res = getResources();

    String chaine = res.getString(R.string.hello, "Anas", 22);

    TextView vue = (TextView)findViewById(R.id.vue);

    vue.setText(chaine);

  }}

## **Gestion du pluriel**

Pour permettre de gérer le pluriel des chaines de caractères, android intègre une balise très pratique (plurals).

**Exemple**

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

<resources>

<pluralsname="nombre\_de\_tutos">

<item quantity="zero">Aucun tutoriel n'est disponible</item>

<item quantity="one">un tutoriel est disponible</item>

<item quantity="other">Plusieurs tutoriaux sont disponibles</item>

</plurals>

</resources>

Pour pouvoir récupérer ces chaines de caractères en Java, on utilise la méthode **getQuantityString**.

public String getQuantityString(int id, int quantity);

Cette méthode retourne la chaine de caractère souhaitée.

* Le premier argument est l’identifiant de la chaine.
* Le deuxième argument est la quantité souhaitée.

Resourcesres = getResources();

String numberOfTutos = res.getQuantityString(R.plurals.nombre\_de\_tutos, 2);

1. **Les drawables**

Le dossier **drawable** rassemble tous les fichiers dessins et images. Il contient l'icône de l'application et cela dans les résolutions (ldpi = basse résolution, mdpi = résolution moyenne, hdpi = haute résolution, xhdpi = très haute résolution).

#### Les images matricielles

Android supporte trois types d'images : les PNG, les GIF et les JPEG, mais ces formats n'ont pas les mêmes usages :

* Les GIF sont peu recommandés. On les utilise sur internet pour les images de moindre qualité ou les petites animations. A éviter le plus souvent.
* Les JPEG sont utilisés en photographie ou pour les images dont on veut conserver la haute qualité. Ce format ne gère pas la transparence, toutes vos images seront rectangulaires.
* Les PNG sont un bon compromis entre compression et qualité d'image. De plus, ils gèrent la transparence. Ce format est conseillé.

**Remarque**

Le nom du fichier déterminera l'identifiant du drawable, il peut contenir toutes les lettres minuscules, tous les chiffres et des underscores( \_ ), mais pas de majuscules.

On peut récupérer le drawable par :

R.drawable.nom\_du\_fichier\_sans\_l\_extension.

# L'utilisation des drawables pour des vues

Chaque Drawable est stocké en tant que fichier dans un des dossiers res/drawable: -mdpi, -hdpi, -xhdpi, -xxhdpi.

Les drawables sont désignés en XML par @drawable/filename (filename sans l'extension)

Par exemple, pour accéder à l'image res/drawable/image1.png :

Avec XML : @drawable/image1 :

<TextView

android:id="@+id/textView1"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:background="@drawable/image1"/>

**Avec java :** R.drawable.image1);

TextView t = (TextView)findViewById(R.id.*vue*);

t.setBackgound(R.drawable.image1);

ImageViewimageView = (ImageView) findViewById(R.id.myimageview);

imageView.setImageResource(R.drawable.image1);

**Exemple**

Voici un exemple qui spécifie les images pour un écran par défaut et des images de remplacement pour écran haute résolution.

**res/**

drawable/

image1.png

background1.png

**drawable-hdpi/**

image2.png

background2.png

Ici un autre exemple qui spécifie la mise en page pour une langue par défaut et la mise en page alternative pour la langue arabe.

**res/**

drawable/

icon.png

background.png

**drawable-hdpi/**

icon1.png

background1.png

layout/

activity\_main.xml

info.xml

**layout-ar/**

main.xml

# Drawables XML

## Shape drawables

Les "shape drawables" sont des fichiers XML permettant de définir un objet géométrique avec des couleurs, des bordures et des gradients qui peuvent être assignés aux vues.

Le code suivant montre un exemple de « shape drawable » :

<?xml version="1.0" encoding="UTF-8"?>

<shape

xmlns:android="http://schemas.android.com/apk/res/android"

android:shape="rectangle">

<stroke

android:width="2dp"

android:color="#FFFFFFFF" />

<gradient

android:endColor="#DDBBBBBB"

android:startColor="#DD777777"

android:angle="90" />

<corners

android:bottomRightRadius="7dp"

android:bottomLeftRadius="7dp"

android:topLeftRadius="7dp"

android:topRightRadius="7dp" />

</shape>

On peut mettre ce drawable dans la propriété arrière-plan de mise en page :

<LinearLayout

…

android:background="@drawable/myshape"

## State drawables

Les « state drawables » permettent de définir des états et pour chaque état, on peut assigner à la vue un drawable différent.

On utilise la balise selector dans laquelle on ajoute tous nos Drawable sous la forme de balises item.

Pour finir, on choisit les différentes valeurs des attributs  android:state  et  android:drawable  qui permettent d'associer le Drawable à un état (fichier mybutton.xml mis dans res/drawable).

Par exemple, le code suivant définit les différents drawables pour un bouton en fonction de son état  (fichier mybutton.xml):

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

<selector xmlns:android="http://schemas.android.com/apk/res/android">

<item android:drawable="@drawable/button1"

android:state\_pressed="true" />

<item android:drawable="@drawable/button2"

android:state\_checked="true" />

<item android:drawable="@drawable/button\_default" />

</selector>

La dernière étape consiste à modifier la propriété android:background du Button pour que ce dernier change totalement d'apparence:

<Button

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:background="@drawable/mybutton"

android:text="New dummy content" />

1. **Les styles et les thèmes**

**Les styles**

Un **style** est une collection de propriétés qui spécifie le design d’une vue, d’un élément ou d’une application. Il peut spécifier différentes propriétés :

* Padding,
* Margin,
* Largeur,
* Couleur du texte,
* Taille du texte,
* etc…

Un **style** est défini dans un fichier de ressource XML séparé du fichier XML de la vue. Ce fichier se nomme en général ***styles.xml*** et se situe dans le dossier ***values***.

<resources>

<style name="nom\_du\_style" parent="nom\_du\_parent">

<item name="propriete\_1">valeur\_de\_la\_propriete\_1</item>

<item name="propriete\_2">valeur\_de\_la\_propriete\_2</item>

<item name="propriete\_3">valeur\_de\_la\_propriete\_3</item>

…

<item name="propriete\_n">valeur\_de\_la\_propriete\_n</item>

</style>

</resources>

**Règles à respecter :**

* Définir un nom unique pour le style, puisqu'il y aura une variable pour y accéder.
* Il est possible d'ajouter des propriétés physiques par <item>. Le nom de <item> correspond à un des attributs destinés aux Vues. Par exemple pour changer la couleur d'un texte, on va utiliser l'attribut android:textColor.
* on peut faire hériter un style d'un autre style.

**Exemple**

Le style suivant permet de mettre du texte en cyan :

<style name="texte\_cyan">

<item name="android:textColor">#00FFFF</item>

</style>

Pour attribuer un style à une vue en XML on utilise l'attribut style.style="identifiant\_du\_style". style=*"@style/texte\_cyan"*/>

**Exemple**

1. **Définir le fichier styles.xml**

<style name=*"texte\_cyan"*>

<item name=*"android:textColor"*>#FF0000</item>

<item name=*"android:background"*>#FF00FF</item>

</style>

1. **Dans le fichier main.xml**

<Button

android:text=*"bouton1"*

style=*"@style/texte\_cyan"*/>

# **Notion d'héritage**

Il est possible de faire hériter un style d'un autre style. Si on veut créer un style forçant une vue à prendre la taille de son parent et à centrer son contenu :

Les deux styles suivants héritent du style précédent en rajoutant d'autres attributs :

<style name="texte\_cyan\_grand" parent="texte\_cyan">

  <!-- On récupère la couleur du texte définie par le parent -->

  <item name="android:textSize">20sp</item>

</style>

<style name="texte\_rouge\_grand" parent="texte\_cyan\_grand">

  <!-- On écrase la couleur du texte définie par le parent, mais on garde la taille -->

  <item name="android:textColor">#FF0000</item>

</style>

Cependant, un style ne s'applique pas de manière dynamique en Java, il faut alors préciser le style à utiliser dans le constructeur.

Regardez [le constructeur d'une vue](http://developer.android.com/reference/android/view/View.html#View%28android.content.Context,%20android.util.AttributeSet%29) : public View (Context contexte, AttributeSetattrs). Le paramètre attrs est facultatif, et c'est lui qui permet d'attribuer un style à une vue.

**Exemple** :

Button bouton = new Button (this, R.style.texte\_rouge\_grand);

**Exemple : personnaliser tous les textes**

Soit une interface contenant un **TextView**, un **EditText** et un **Button**. Nous allons personnaliser tous les textes qui composent cette interface.

1. **Définir le style**

<resources>

    <stylename="GreenText"parent="@android:style/TextAppearance">

        <itemname="android:textColor">@color/textColor</item>

        <itemname="android:textSize">@dimen/textSize</item>

        <itemname="android:textStyle">italic</item>

        <itemname="android:typeface">serif</item>

    </style>

</resources>

<LinearLayout

…..

<TextViewandroid:layout\_width="wrap\_content"

**android:textAppearance="@style/GreenText"**

        android:layout\_height="wrap\_content"

android:text="@string/hello"/>

    <EditTextandroid:layout\_width="wrap\_content"

**android:textAppearance="@style/GreenText"**

        android:layout\_height="wrap\_content"

android:text="@string/hello"/>

    <Buttonandroid:layout\_width="wrap\_content"

**android:textAppearance="@style/GreenText"**

        android:layout\_height="wrap\_content"

android:text="@string/hello"/>

</LinearLayout>

## Les thèmes

Les thèmes ne sont rien d'autre que des styles s'appliquant à l'ensemble des balises de layouts.

Pour résoudre le problème de android:layout\_[width|height] obligatoires, on peut par exemple, imaginer un thème initialisant l'ensemble de ces valeurs à fill\_parent.

On commence par créer un style **monTheme** héritant du thème Android Theme.Black. Généralement, bien qu'un style soit un thème, on place le thème dans un fichier nommé **res/values/themes.xml.**

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

<resources>

<style name="monTheme" parent="@android:style/Theme.Black">

<item name="android:layout\_height">fill\_parent</item>

<item name="android:layout\_width">fill\_parent</item>

</style>

</resources>

On applique ce thème à l'ensemble de l'application en ajoutant l'attribut XML**android:theme="@style/monTheme"à la balise <application /> du fichierAndroidManifest.xml**.

Le code du layout sera donc de la forme:

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

<TextView

android:gravity="center"

android:text="Bonjour !" />

## Les attributs de style

Appliquer une valeur par défaut à l'ensemble des balises de l'application est quelque peu inutile. Les thèmes prennent tout leur sens lorsqu'ils sont utilisés en conjonction avec les attributs de style.

**Exemple**

Application d'un attribut de style à une <TextView />.

On commence par créer un nouvel attribut nommé myTextViewAttr dans le fichier res/values/attrs.xml :

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

<resources>

<attr name="myTextViewAttr" />

</resources>

Pour appliquer ce style à un layout on fait :

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

<TextView

**style="?attr/myTextViewAttr"**

android:text="Bonjour !" />

Ce <TextView /> dispose de l'attribut de style ?attr/myTextViewAttr. L'utilisation du point d'interrogation permet d'informer le système que la ressource de style recherchée n'est pas myTextViewAttr mais un style "pointé" par cet attribut.

Pour définir le style, il suffit de l'initialiser dans le thème de l'application.

<style name="Theme1" parent="@android:style/Theme.Black">

<item name="myTextViewAttr">@style/un\_style</item>

</style>

L'application est maintenant fonctionnelle et applique le style @style/un\_style à l'ensemble des balises disposant de l'attribut de style ?attr/myTextViewAttr.

# **Etendre des thèmes Android**

On peut créer ses propres styles ou étendre et modifier les thèmes existants d'Android.

Si par exemple, vous souhaitez développer une application à dominante "noir sur blanc", il suffit de créer un thème héritant de @android:style/Theme.Light et de modifier les attributs à votre convenance.

Ici un exemple de thème Android supprimant la fameuse ombre en haut de l'écran et modifiant le style par défaut de la titlebar :

<style name="Theme.Custom\_3" parent="@style/Theme1">

<item name="android:windowContentOverlay">@null</item>

<item name="android:windowTitleStyle">@style/WindowTitle</item>

<item name="android:windowTitleBackgroundStyle">@style/WindowTitleBackground</item>

</style>

Les styles associés sont définis de la façon suivante :

<style name="WindowTitleBackground">

<item name="android:background">@drawable/title\_bar</item>

</style>

<style name="WindowTitle" parent="@android:style/WindowTitle">

<itemname="android:textAppearance">@style/TextAppearance.WindowTitle</item>

</style>

<style name="TextAppearance.WindowTitle"parent="@android:style/TextAppearance.WindowTitle">

<item name="android:textStyle">bold|italic</item>

</style>