






Chapitre 4 : Les conteneurs

Introduction

Les conteneurs sont utilisés pour placer des éléments graphiques simples, ou des groupes ou même d'autres conteneurs. Ils **héritent** tous de la classe `ViewGroup`. Leurs propriétés sont celles décrites dans le chapitre précédant auxquelles viennent s'ajouter des propriétés spécifiques décrites dans ce chapitre.

Des `ViewGroup` particuliers sont prédéfinis : ce sont des gabarits (`layout`) qui proposent une prédisposition des objets graphiques comme le montre le tableau suivant :

Image du conteneur	Intitulé du conteneur	Usage
	<code>RelativeLayout</code>	Place des éléments les uns relativement aux autres
	<code>FrameLayout</code>	Placement en haut à gauche. Si l'on place plusieurs éléments ils se superposent, généralement <code>FrameLayout</code> est utilisé pour ne placer qu'un seul élément.
	<code>LinearLayout</code>	Place les éléments les uns à côté des autres horizontalement ou verticalement.
	<code>TableLayout</code>	Place les éléments en lignes et colonnes (Matricielle)
	<code>GridLayout</code>	Disposition matricielle avec N colonnes et un nombre infini de lignes

I. RelativeLayout

Ce type de `layout` permet d'aligner tous les Widgets qu'il contient dans une seule direction : horizontale ou verticale.

I.1. Position relative au conteneur

Les paramètres XML permettant de lier un élément à son conteneur sont :

<code>android:layout_alignParentTop="true/false"</code> <code>android:layout_alignParentBottom="true/false"</code> <code>android:layout_alignParentLeft="true/false"</code> <code>android:layout_alignParentRight="true/false"</code>	Ces options permettent de préciser si l'élément doit être aligné avec son conteneur respectivement des côtés haut, bas, gauche et droite.
<code>android:layout_centerHorizontal = "true/false"</code> <code>android:layout_centerVertical = "true/false"</code>	Ce code indique si l'élément doit être centré horizontalement et/ou verticalement dans son conteneur.
<code>android:layout_centerInParent = "true/false"</code>	Permet d'indiquer si l'élément doit être centré horizontalement et verticalement dans son conteneur.

I.2. Position relative aux autres éléments

Il est également possible de positionner un élément par rapport à un autre grâce à son identificateur (son id) :

- A la déclaration d'un élément : `android:id="@+id/idElem"`
- A l'utilisation : `@id/idElem`

<pre>✓ android:layout_above="@id/idElem1" ✓ android:layout_below="@id/idElem2" ✓ android:layout_toLeftOf="@id/idElem3" ✓ android:layout_toRightOf="@id/idElem4"</pre>	Indique que l'élément sera placé respectivement au-dessus, au-dessous, à gauche et à droite de celui indiqué par son id.
<pre>✓ android:layout_alignTop="@id/idElem1" ✓ android:layout_alignBottom="@id/idElem2" ✓ android:layout_alignLeft="@id/idElem3" ✓ android:layout_alignRight="@id/idElem4"</pre>	L'élément sera aligné du même côté (haut, bas, gauche, droite) de celui indiqué par son id
<pre>✓ android:layout_alignBaseline="@id/idElem"</pre>	Indique que les lignes de base des deux éléments sont alignées

II. FrameLayout

Ce layout empile les widgets les uns sur les autres. Chaque composant est positionné dans le coin en haut à gauche en masquant le widget précédent ce qui ne permet d'afficher qu'un seul élément. Il peut sembler inutile comme ça, mais ne l'est pas du tout. En effet, dans certains cas, `FrameLayout` peut sembler le conteneur le plus adéquat dans un contexte donné comme :

- **Un album photo** : il suffit de mettre plusieurs photos dans le `FrameLayout` et de ne laisser qu'une seule visible, en laissant les autres invisibles grâce à l'attribut `android:visibility`.
- **Un lecteur PDF** : il suffit d'empiler toutes les pages dans le `FrameLayout` et de n'afficher que la page actuelle, celle du dessus de la pile, à l'utilisateur.

III. LinearLayout

Ce layout permet de positionner les éléments dans une seule direction : verticalement ou horizontalement

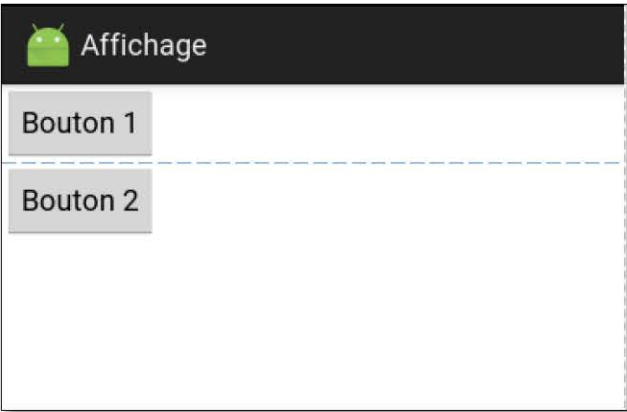
III.1. Orientation

C'est la première chose à préciser lors de la création d'un **LinearLayout**. Cette option spécifie la façon dont seront **alignés** les éléments contenus dans ce Layout.

```
android:orientation="vertical/horizontal"
```

Deux options sont possibles :

- **Vertical** : place les éléments les uns aux dessous des autres.
- **Horizontal** : place les éléments les uns à côtés des autres.

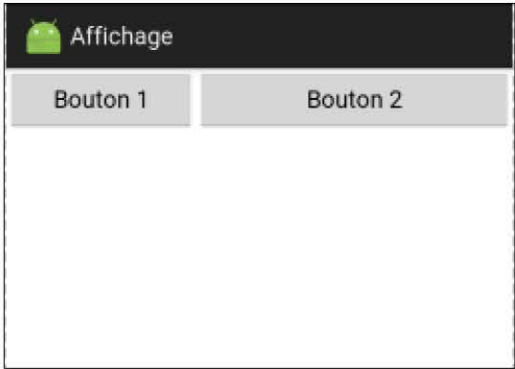
	Exemple 1	Exemple 2
En XML	<pre> <LinearLayout android:orientation="vertical" android:layout_width="match_parent" android:layout_height="match_parent" > <Button android:id="@+id/btnAfficher1" android:layout_width="wrap_content" android:layout_height="wrap_content" android:text="Bouton 1" /> <Button android:id="@+id/btnAfficher2" android:layout_width="wrap_content" android:layout_height="wrap_content" android:text="Bouton 2" /> </LinearLayout> </pre>	<pre> <LinearLayout android:orientation="horizontal" android:layout_width="match_parent" android:layout_height="match_parent" > <Button android:id="@+id/btnAfficher1" android:layout_width="wrap_content" android:layout_height="wrap_content" android:text="Bouton 1" /> <Button android:id="@+id/btnAfficher2" android:layout_width="wrap_content" android:layout_height="wrap_content" android:text="Bouton 2" /> </LinearLayout> </pre>
Résultat		

III.2. Poids

Si on observe l'exemple précédent, on remarque que les deux boutons occupent l'espace d'une façon équitable. Si on voudrait par exemple que le bouton 2 occupe le triple espace que celui du premier alors on utilise :

```
android:layout_weight='valeur'
```

Où cette valeur doit être égale au double de celle attribuée au premier bouton.

Exemple	Résultat
<pre> <LinearLayout android:orientation="horizontal" android:layout_width="match_parent" android:layout_height="match_parent" > <Button android:id="@+id/btnAfficher1" android:layout_weight="1" android:layout_width="wrap_content" android:layout_height="wrap_content" android:text="Bouton 1" /> <Button android:id="@+id/btnAfficher2" android:layout_weight="2" android:layout_width="wrap_content" android:layout_height="wrap_content" android:text="Bouton 2" /> </LinearLayout> </pre>	

III.3. Gravité

Dans un `LinearLayout` les éléments sont alignés de gauche à droite et de haut en bas. Si l'on voudrait placer un élément tout en bas ou à droite, on devrait utiliser la propriété :

```
android:layout_gravity = "valeur"
```

Où valeur peut prendre : `left`, `center_horizontal`, `center_vertical`, `top`, `bottom`, `right`, qui représentent respectivement aligner les éléments à gauche, centrer horizontalement, centrer verticalement, en haut, en bas et à droite.

IV. TableLayout

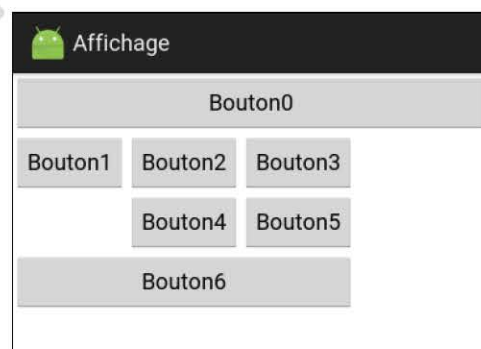
Ce layout permet de placer les éléments sous forme d'un tableau comme on pourrait le faire en HTML avec la balise `<table>`. Toutefois, `TableLayout` présente quelques différences notables par rapport à son équivalent HTML : il n'est pas possible de faire apparaître les bordures du quadrillage.

Il est possible de mettre des vues directement dans le tableau, auquel cas elles prendront toute la place possible en longueur. Cependant, si on veut un contrôle plus complet ou avoir plusieurs éléments sur une même ligne, alors il faut passer par un objet `<TableRow>` (équivalent à la balise `<tr>` en HTML).

Exemple :

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button android:text="Bouton0" />
    <TableRow>
        <Button android:text="Bouton1" />
        <Button android:text="Bouton2" />
        <Button android:text="Bouton3" />
    </TableRow>
    <TableRow>
        <Button android:text="Bouton4"
android:layout_column="1" />
        <Button android:text="Bouton5" />
    </TableRow>
    <TableRow>
        <Button android:text="Bouton6"
android:layout_span="3" />
    </TableRow>
</TableLayout>
```

Resultat :



On observe tout d'abord qu'il est possible de mettre des vues directement dans le tableau, auquel cas elles prendront toute la place possible en largeur. En fait, elles s'étendront sur toutes les colonnes du tableau. Cependant, si on veut un contrôle plus complet ou avoir plusieurs éléments sur une même ligne, alors il faut passer par un objet `<TableRow>`.

```
<Button android:text="Bouton0"/>
```

Cet élément s'étend sur toute la ligne puisqu'il ne se trouve pas dans un `<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. Dans l'exemple précédent, le tableau comporte 3 colonnes puisque la ligne avec le plus de cellule contient 3 boutons.

```
<TableRow>
    <Button android:text="Bouton1" />
    <Button android:text="Bouton2" />
    <Button android:text="Bouton3" />
</TableRow>
```

Il est possible de choisir dans quelle colonne se situe un élément avec l'attribut `android:layout_column` sachant que l'index des colonnes commence à 0. Dans notre exemple, le bouton4 se place directement à la deuxième colonne grâce à `android:layout_column="1"`.

```
<TableRow>
    <Button android:text="Bouton4" android:layout_column="1"/>
    <Button android:text="Bouton5" />
</TableRow>
```

Il est également possible d'étendre un élément sur plusieurs colonnes à l'aide de l'attribut `android:layout_span`. Dans notre exemple, le bouton6 s'étend de la première colonne jusqu'à la troisième.

```
<TableRow>
    <Button android:text="Bouton6" android:layout_span="3"/>
</TableRow>
```

V. GridLayout

`GridLayout` permet de composer des vues évoluées plus facilement : formulaires, tableaux,...Elle permet de présenter les éléments sous forme matricielle en fixant obligatoirement le nombre de colonnes (`android:columnCount`) et éventuellement le nombre de lignes (`android:rowCount`). L'utilisation est assez simple et est très similaire à l'utilisation d'un `LinearLayout` qu'on remplit ligne à ligne.

Exemple :

```
<GridLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:columnCount="3">
<Button android:text="Bouton 1" />
<Button android:text="Bouton 2" />
<Button android:text="Bouton 3" />
<Button android:text="Bouton 4" />
<Button android:layout_column="2"
        android:layout_row="2"
        android:text="Bouton 5" />
</GridLayout>
```

Resultat :



Conclusion

Dans ce chapitre, nous avons présenté les différents conteneurs (layout) pouvant être utilisés afin d'organiser les vues dans l'interface graphique d'une application mobile en détaillant pour chacun ses principales propriétés. Le chapitre suivant permet de parcourir les différents éléments graphiques utilisés dans une application Android.