# Création d'interfaces utilisateur 4

*Il n'y a pas de bonne application sans bonne interface utilisateur : c'est la condition de son succès auprès des utilisateurs.*

Les interfaces graphiques prennent une place de plus en plus importante dans le choix des applications par les utilisateurs, tant dans l'implémentation de concepts innovants qu'au niveau de l'ergonomie.

Comme sur bien des plates-formes, les interfaces d'applications Android sont organisées en vues et gabarits, avec néanmoins quelques spécificités.

## Le concept d'interface

Une interface n'est pas une image statique mais un ensemble de composants graphiques, qui peuvent être des boutons, du texte, mais aussi des groupements d'autres composants graphiques, pour lesquels nous pouvons définir des attributs communs (taille, couleur, positionnement, etc.). Ainsi, l'écran ci-après (figure 4-1) peut-il être vu par le développeur comme un assemblage (figure 4-2).

La représentation effective par le développeur de l'ensemble des composants graphiques se fait sous forme d'arbre, en une structure hiérarchique (figure 4-3). Il peut n'y avoir qu'un composant, comme des milliers, selon l'interface que vous souhaitez représenter. Dans l'arbre ci-après (figure 4-3), par exemple, les composants ont été organisés en 3 parties (*haut*, *milieu* et *bas*).

**Figure 4–1**
Exemple d'un écran

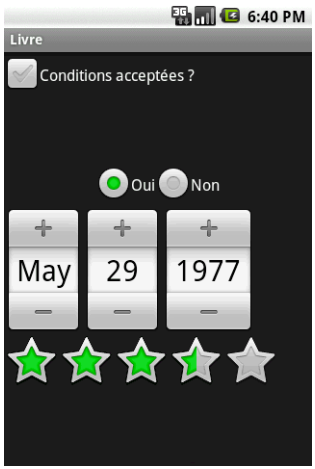



**Figure 4–2**
Le même écran vu par le développeur

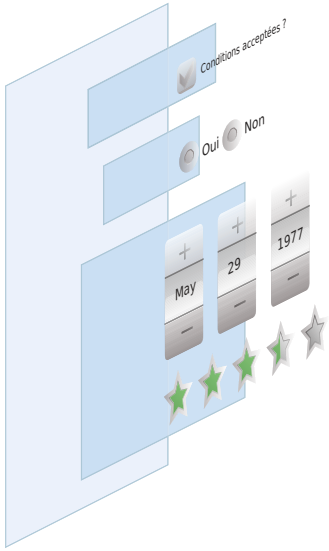
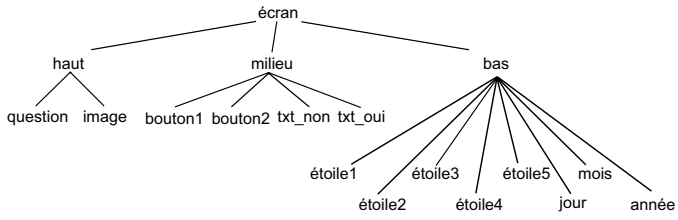



**Figure 4–3**
Arborescence de l'interface précédente



Cet exemple donne l'idée générale de l'organisation des interfaces – car les données graphiques ne sont pas exactement représentées ainsi. Nous verrons plus loin comment cela est géré pour Android.

Sous Android, vous pouvez décrire vos interfaces utilisateur de deux façons différentes (nous reviendrons plus tard en détail sur ce point) : avec une description déclarative XML ou directement dans le code d'une activité en utilisant les classes adéquates. La façon la plus simple de réaliser une interface est d'utiliser la méthode déclarative XML via la création d'un fichier XML que vous placerez dans le dossier `/res/layout` de votre projet.

## Les vues

Le composant graphique élémentaire de la plate-forme Android est la *vue* : tous les composants graphiques (boutons, images, cases à cocher, etc.) d'Android héritent de la classe `View`.

Tout comme nous l'avions fait dans l'exemple précédent, Android vous offre la possibilité de regrouper plusieurs vues dans une structure arborescente à l'aide de la classe `ViewGroup`. Cette structure peut à son tour regrouper d'autres éléments de la classe `ViewGroup` et être ainsi constituée de plusieurs niveaux d'arborescence.

L'utilisation et le positionnement des vues dans une activité se fera la plupart du temps en utilisant une mise en page qui sera composée par un ou plusieurs gabarits de vues.

# Positionner les vues avec les gabarits

Un gabarit, ou *layout* dans la documentation officielle, ou encore mise en page, est une extension de la classe `ViewGroup`. Il s'agit en fait d'un conteneur qui aide à positionner les objets, qu'il s'agisse de vues ou d'autres gabarits au sein de votre interface.

Vous pouvez imbriquer des gabarits les uns dans les autres, ce qui vous permettra de créer des mises en forme évoluées.

Comme nous l'avons dit plus haut, vous pouvez décrire vos interfaces utilisateur soit par une déclaration XML, soit directement dans le code d'une activité en utilisant les classes adéquates. Dans les deux cas, vous pouvez utiliser différents types de gabarits. En fonction du type choisi, les vues et les gabarits seront disposés différemment :

**`LinearLayout`** : permet d'aligner de gauche à droite ou de haut en bas les éléments qui y seront incorporés. En modifiant la propriété `orientation` vous pourrez signaler au gabarit dans quel sens afficher ses enfants : avec la valeur `horizontal`, l'affichage sera de gauche à droite alors que la valeur `vertical` affichera de haut en bas ;

**`RelativeLayout`** : ses enfants sont positionnés les uns par rapport aux autres, le premier enfant servant de référence aux autres ;

**`FrameLayout`** : c'est le plus basique des gabarits. Chaque enfant est positionné dans le coin en haut à gauche de l'écran et affiché par-dessus les enfants précédents, les cachant en partie ou complètement. Ce gabarit est principalement utilisé pour l'affichage d'un élément (par exemple, un cadre dans lequel on veut charger des images) ;

**`TableLayout`** : permet de positionner vos vues en lignes et colonnes à l'instar d'un tableau.

Voici un exemple de définition déclarative en XML d'une interface contenant un gabarit linéaire (le gabarit le plus commun dans les interfaces Android).

**Code 4-1 : Interface avec un gabarit LinearLayout**

```
<!-- Mon premier gabarit -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
</LinearLayout>
```

Chaque gabarit possède des attributs spécifiques, et d'autres communs à tous les types de gabarits. Parmi les propriétés communes, vous trouverez les propriétés `layout_weight` et `layout_height`. Celles-ci permettent de spécifier le comportement du remplissage en largeur et en hauteur des gabarits et peuvent contenir une taille (en pixels ou dpi) ou les valeurs constantes suivantes : `fill_parent` et `wrap_content`.

La valeur `fill_parent` spécifie que le gabarit doit prendre toute la place disponible sur la largeur/hauteur. Par exemple, si le gabarit est inclus dans un autre gabarit – parent (l'écran étant lui-même un gabarit) – et si le gabarit parent possède une largeur de 100 pixels, le gabarit enfant aura donc une largeur de 100 pixels. Si ce gabarit est le gabarit de base – le plus haut parent – de notre écran, alors ce dernier prend toute la largeur de l'écran.

Si vous souhaitez afficher le gabarit tel quel, vous pouvez spécifier la valeur `wrap_content`. Dans ce cas, le gabarit ne prendra que la place qui lui est nécessaire en largeur/hauteur.

**À RETENIR Les unités de mesure**

Pour spécifier les dimensions des propriétés de taille de vos composants graphiques (largeur, hauteur, marges, etc), vous pouvez spécifier plusieurs types d'unité. Le choix de l'utilisation d'une unité par rapport à une autre se fera en fonction de vos habitudes et si vous souhaitez spécifier une taille spécifique ou relative, par exemple : 10 px, 5 mm, 20 dp, 10 sp.

Voici les unités prises en charge par Android :

- pixel (px) : correspond à un pixel de l'écran ;
- pouce (in) : unité de longueur, correspondant à 2,54 cm. Basé sur la taille physique de l'écran ;
- millimètre (mm) : basé sur la taille physique de l'écran ;
- point (pt) : 1/72 d'un pouce ;
- pixel à densité indépendante (dp ou dip) : une unité relative se basant sur une taille physique de l'écran de 160 dpi. Avec cette unité, 1 dp est égal à 1 pixel sur un écran de 160 pixels. Si la taille de l'écran est différente de 160 pixels, les vues s'adapteront selon le ratio entre la taille en pixels de l'écran de l'utilisateur et la référence des 160 pixels ;
- pixel à taille indépendante (sp) : fonctionne de la même manière que les pixels à densité indépendante à l'exception qu'ils sont aussi fonction de la taille de polices spécifiée par l'utilisateur. Il est recommandé d'utiliser cette unité lorsque vous spécifiez les tailles des polices.

Ces deux dernières unités sont à privilégier car elles permettent de s'adapter plus aisément à différentes tailles d'écran et rendent ainsi vos applications plus portables. Notez que ces unités sont basées sur la taille physique de l'écran : l'écran ne pouvant afficher une longueur plus petite que le pixel, ces unités seront toujours rapportées aux pixels lors de l'affichage (1 cm peut ne pas faire 1 cm sur l'écran selon la définition de ce dernier).

Vous pouvez spécifier une taille précise en px (pixel) ou dip (dpi). Notez cependant que si vous avez besoin de spécifier une taille, notamment si vous avez besoin d'intégrer une image avec une taille précise, préférez les valeurs en *dip* à celles en *px*. En effet, depuis la version 1.6, Android est devenu capable de s'exécuter sur plusieurs tailles d'écrans. Les valeurs en dip permettent un ajustement automatique de vos éléments alors que les valeurs en px prennent le même nombre de pixels quelle que soit la taille de l'écran, ce qui peut très vite compliquer la gestion de l'affichage sur la multitude d'écrans disponibles.

# Créer une interface utilisateur

La création d'une interface se traduit par la création de deux éléments :

- une définition de l'interface utilisateur (gabarits, etc.) de façon déclarative dans un fichier XML ;
- une définition de la logique utilisateur (comportement de l'interface) dans une classe d'activité.

Cela permet d'avoir une séparation stricte entre la présentation et la logique fonctionnelle de votre application. De plus, un intégrateur graphique pourra modifier l'interface sans interférer avec le code du développeur.

## Définir votre interface en XML

Une bonne pratique est de définir intégralement votre interface dans la déclaration XML. Retenez néanmoins qu'il est aussi possible (et bon nombre de scénarios ne pourront s'en passer) d'instancier dynamiquement des vues depuis votre code.

Les fichiers de définition d'interface XML sont enregistrés dans le dossier `/layout` de votre projet. Prenez garde à ce que le nom du fichier ne comporte que des lettres minuscules et des chiffres .

Chaque fichier de définition d'interface, pour peu qu'il se trouve bien dans le répertoire `res/layout` de votre projet, possède un identifiant unique généré automatiquement par l'environnement de développement. De cette façon, vous pouvez y faire référence directement dans votre code. Par exemple, si le fichier se nomme `monLayout`, vous pouvez y faire référence dans votre code grâce à la constante `R.layout.monLayout`. Comme vous le verrez, l'utilisation des identifiants est très courante : ils vous serviront à récupérer des éléments, à spécifier des propriétés et à utiliser les nombreuses méthodes des activités (`findViewById`, `setContentView`, etc.).

Dans votre projet Android, créez un fichier `main.xml` dans le dossier */layout* et placez-y le contenu suivant.

**Code4-2 : Définition d'interface**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:id="@+id/monText"
    />
</LinearLayout>
```

Cette interface définit un gabarit de type `LinearLayout` et un composant graphique de saisie de texte de type `TextView`. Vous remarquerez que nous avons défini un attribut `android:id` avec la valeur `@+id/monText`. Cette valeur nous permettra de faire référence à cet élément dans le code avec la constante `R.id.montext`. Pour le moment nous n'allons pas détailler les éléments et les attributs : nous y reviendrons plus loin dans ce chapitre.

Le complément ADT génère automatiquement un identifiant pour cette définition d'interface. Nommée `main.xml` et placée dans le répertoire `res/layout`, cet identifiant sera `R.layout.main`.

## Associer votre interface à une activité et définir la logique utilisateur

Dans une application, une interface est affichée par l'intermédiaire d'une activité. Vous devez donc avant toute chose créer une activité en ajoutant une nouvelle classe à votre projet dérivant de la classe `Activity`.

Le chargement du contenu de l'interface s'effectue à l'instanciation de l'activité. Redéfinissez la méthode `onCreate` de l'activité pour y spécifier la définition de l'interface à afficher via la méthode `setContentView`. Cette méthode prend en paramètre un identifiant qui spécifie quelle ressource de type interface doit être chargée et affichée comme contenu graphique. Nous utilisons l'identifiant généré automatiquement pour spécifier l'interface que nous avons précédemment créée.

**Code 4-3 : Spécifier une vue à l'activité**

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

public class Main extends Activity {


    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

L'interface (code 4-2) a défini un composant graphique `TextView` vide. Pour y accéder depuis le code et pouvoir le manipuler, vous devez d'abord récupérer une instance de l'objet avec la méthode `findViewById` de l'activité. Une fois l'objet récupéré, vous pourrez le manipuler de la même façon que n'importe quel objet, par exemple pour modifier son texte. L'affichage répercutera le changement.

Le code suivant récupère le composant graphique `TextView` que nous avons nommé `monText` dans l'interface, puis utilise la méthode `setText` pour changer le contenu de son texte.

**Code 3-4 : Récupérer un élément de l'interface et interagir avec**

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
public class Main extends Activity {
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);

     TextView monTexte = (TextView)findViewById(R.id.monText);
      monTexte.setText("Bonjour tout le monde !");
   }
}
```

Le résultat dans l'émulateur Android est celui de la figure 3-5.

**Figure 4–5**
Le résultat de l'exemple 3-4