Programmation Android

De la conception au déploiement avec le SDK Google Android 2

Damien Guignard

Julien Chable

Emmanuel Robles

Avec la contribution de Nicolas Sorel et Vanessa Conchodon

ISBN : 978-2-212-12587-0

EYROLLES

Table des matières

CHAPITRE 4
Communication entre applications : la classe Intent 101

CHAPITRE 5
Création d'interfaces utilisateur avancées 123

CHAPITRE 13
Géolocalisation, Google Maps et GPS 379

ANNEXE A

Avant-propos

La téléphonie mobile a connu une explosion dans les années 2000 mais aucune révolution n'a semblé arriver depuis : les appareils tendaient à tous se ressembler, les innovations n'avaient plus vraiment de saveur ; les applications étaient difficiles d'accès de par leur mode de distribution et souvent peu performantes à cause des faibles capacités des appareils.

Depuis quelques mois, les smartphones sont dotés d'une puissance plus importante et d'espaces de stockage conséquents. Les téléphones tendent à devenir des objets artistiques, presque de reconnaissance sociale, et possèdent des fonctionnalités qu'aucun téléphone ne pouvait espérer auparavant : connexion haut débit, localisation GPS, boussole, accéléromètre, écran tactile souvent multipoint, marché d'applications en ligne… Autant de qualités permettant de créer des applications innovantes et de les distribuer en toute simplicité.

La plate-forme Android apporte tout cela au consommateur, mais surtout, elle affranchit le développeur de nombreuses contraintes par son ouverture ; elle permet à n'importe quel développeur de créer ses applications avec un ticket d'entrée quasi nul. Le framework et le système d'exploitation et outils associés ont un code source ouvert, leur accès est gratuit et illimité. Plus besoin de négocier avec le constructeur du téléphone pour qu'il vous laisse développer sur sa plate-forme.

Tous les développeurs sont ainsi sur un même pied d'égalité, qu'ils soient une grande entreprise ou quelques jeunes dans un garage ; tous peuvent ajouter de la mobilité à des applications existantes.

À qui est destiné cet ouvrage ?

Cet ouvrage se veut accessible à toute personne qui souhaite créer des applications mobiles sur la plate-forme Android. Que vous soyez un développeur confirmé ou une personne débutant tout juste dans la programmation informatique, nous espérons que ce livre vous donnera l'envie et les informations nécessaires pour vous permettre de créer les applications de demain.

Cet ouvrage ne traite pas du langage ou de la plate-forme Java. Une première expérience en Java est conseillée, la plate-forme Android étant basée sur ce langage.

Achat d'un téléphone de test

Avant d'investir dans l'achat d'un téléphone Android de développement ou de vous inscrire sur le marché Android, lisez attentivement les premiers chapitres et réalisez les exemples nécessaires pour bien démarrer. Bien évidemment si vous possédez déjà un téléphone s'exécutant sous Android, cela représente déjà un avantage pour tester vos applications. Vous trouverez en annexe une partie sur la manière de configurer votre téléphone pour développer et tester directement vos applications sur ce dernier.

Versions d'Android liées à ce livre

L'évolution de la plate-forme Android est rapide : lors du projet initial de cet ouvrage, Android était en version 1.5, avant de passer rapidement en version 1.6. À l'heure de l'écriture de ce livre, Android 2.0 est le standard qui tend déjà à se répandre auprès des développeurs.

Tous les exemples de ce livre ont été créés avec Android 1.5 et la plupart vérifiés avec Android 2.0. Cependant, le rythme élevé des évolutions du SDK et les modifications réalisées, qui sont parfois non compatibles avec les versions émises précédemment, pourront nécessiter des adaptations du code. L'utilisation des exemples de ce livre ne nécessite pas l'achat d'un appareil Android : tous les développements peuvent être réalisés sur l'émulateur, exception faite des exemples du chapitre 15 sur le matériel.

Mises à jour et errata

Vous trouverez des ressources complémentaires et éventuels errata sur la fiche du livre sur le site des éditions Eyrolles et sur le site dédié au livre :

- www.android-le-livre.fr
- http://www.editions-eyrolles.com

Structure de l'ouvrage

La première partie de cet ouvrage présente la plate-forme Android et vous guide à travers l'installation de l'ensemble de l'environnement logiciel nécessaire à la mise en pratique des concepts et des exemples proposés dans ce livre.

La deuxième aborde ensuite les thèmes fondamentaux indispensables à la conception d'applications Android : composition des applications, conception et réalisation d'une première application, création d'interfaces utilisateur et enfin, présentation du mécanisme de communication entre applications (les Intents).

La troisème partie regroupe les problèmatiques qui permettront de maîtriser les techniques qui rendront votre application interactive et communicante : interfaces utilisateur avancées, persistance et exposition des données, multimédia, graphismes 3D, réseau, géolocalisation et gestion du matériel.

Enfin, la quatrième partie de ce livre vous accompagne jusqu'à la publication, sur l'Android Market, de l'application que vous aurez conçue.

À propos des auteurs

Damien Guignard est développeur Java et également formateur Java ME et Android. Il est le fondateur d'une jeune société, Neimad, au travers de laquelle il intervient auprès des sociétés qui souhaitent partager ses 10 ans de fidélité au langage Java sous toutes ses formes.

Julien Chable est développeur et consultant depuis de nombreuses années auprès de PME et de grands groupes. Spécialisé dans le développement et le conseil sur les plates-formes collaboratives et mobiles, il aide les entreprises à se lancer en leur communiquant son expertise.

Emmanuel Robles se passionne dès l'enfance pour les technologies de l'informatique. Très vite, il commence à développer des applications pour ATARI, PC et maintenant pour tous types de plates-formes. Principalement dévoué à la création sur le système d'exploitation Android sur lequel Emmanuel a déjà réalisé plusieurs applications commercialisées sur l'Android Market, il crée avec Nicolas Sorel *Androlib.com* en juillet 2009

Nicolas Sorel, passionné par la programmation informatique, crée Codes-Sources en 1999 afin de permettre à tous les développeurs francophones de partager leurs connaissances en informatique. Cette communauté qui regroupe aujourd'hui plus de 1,5 million de membres offre, 10 ans après sa création, plus de 40 000 sources de code. Dès 2008, Nicolas s'intéresse de près au développement Mobile et crée avec Emmanuel Robles *Androlib.com* en juillet 2009.

Remerciements

Damien Guignard – Merci à celles et ceux qui m'ont donné mes premières ou mes secondes chances (Chrystel, Fabienne, Serge, Laurent, Sébastien, Hervé, Xavier et Christophe). Un grand merci également à tous ceux qui n'ont pas compté leurs heures sur ce livre. Et puis, c'est quand on est totalement absorbé par l'écriture ou la relecture finale qu'on s'aperçoit de l'importance de certains remerciements. Merci donc à tous ceux qui n'ont pas eu beaucoup de nouvelles et qui ne m'en tiennent pas rigueur. Et enfin, merci mon Ange, maintenant que ce livre est terminé, il est temps d'écrire les plus belles pages de notre livre de souvenirs.

Julien Chable – J'adresse mes remerciements à Damien, Nicolas et Emmanuel pour m'avoir accepté dans l'aventure. Je tiens également à remercier ma compagne sans qui ma participation n'aurait pu voir le jour. Pour terminer, je remercie bien sûr l'équipe Eyrolles : Muriel Shan Sei Fan et Vanessa Conchodon pour leur travail et leur confiance qui ont donné vie à ce livre.

Emmanuel Robles - Je remercie ma femme, ma famille et mes associés pour ce qu'ils sont : formidables ! Myriam Longuet, experte en psychologie du « Geek » et d'une patience inébranlable ainsi que toute l'équipe de Video2Brain. Reto Meier, Android Developer Advocate chez Google pour sa sympathie et sans qui Android ne serait pas ce qu'il est. Alain Herry, toujours au taquet, pour m'aider comme si sa vie en dépendait. Enfin, un remerciement spécial au « groupe ».

Nicolas Sorel - Je remercie Aude Sorel, Alice Sorel et Maxime Sorel pour la patience qu'ils ont avec moi. Grégory Renard et la société Wygwam pour leur compétence et leur aide inestimable. Étienne Jambou, directeur Marketing chez Magma Mobile pour son flegme et sa clairvoyance de chaque instant. Eclipse, l'émulateur Android et surtout l'adb toujours aussi taquins. Enfin, une dédicace spéciale à mon chien Végas.

Les sources de ce livre

Tous les codes sources des exemples de ce livre sont disponibles sous licence Apache 2.0 si vous souhaitez les réutiliser ailleurs que dans le cadre de votre formation avec cet ouvrage.

Vous trouverez les sources à télécharger sur le site des éditions Eyrolles, sur la fiche du livre, et sur le site dédié au livre :

- http://www.android-le-livre.fr
- http://www.editions-eyrolles.com

3

Création d'interfaces utilisateur

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 3-1) peut-il être vu par le développeur comme un assemblage (figure 3-2).

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

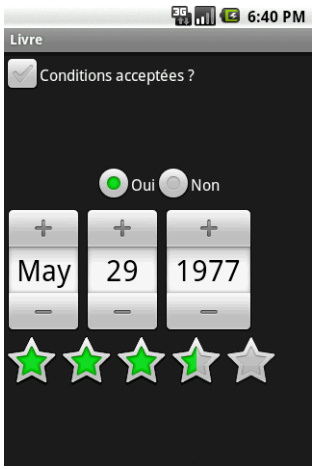


Figure 3–1
Exemple d'un écran

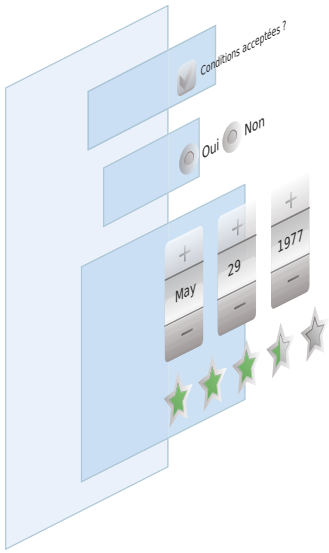


Figure 3–2
Le même écran vu par le développeur

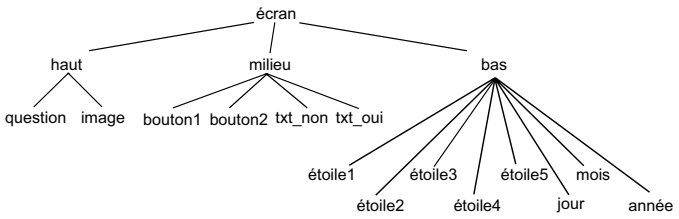


Figure 3–3
Arborescence de l'interface précédente

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`. Ainsi, dans la suite ce livre, gardez bien à l'esprit que lorsque nous créons une vue, nous créons en fait un objet graphique.

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. Dans la suite de ce livre, nous parlerons ainsi de gabarit parent et de gabarit enfant (ou plus généralement d'éléments enfants voire simplement d'enfants), le gabarit enfant étant inclus dans le gabarit parent.

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 3-1 : 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).

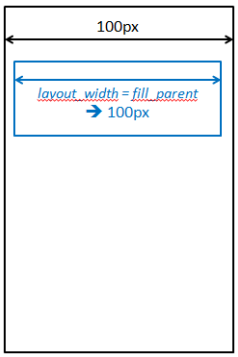


Figure 3–4
La largeur d'un composant graphique dont la propriété *layout_width* est égale à *fill_parent* correspond à la largeur de son gabarit parent.

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.

> **À RETENIR Interface et activité**
>
> À une interface utilisateur est associée une activité ; à une activité est associée une interface utilisateur.

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 (pas de lettre majuscule ou de caractère spécial, comme l'impose la convention de nommage Java).

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.

Code 3-2 : 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 3-3 : 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 3-2) 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.

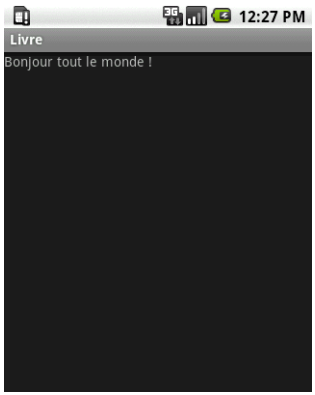


Figure 3–5
Le résultat de l'exemple 3-4

Créer une interface sans définition XML

Vous auriez pu arriver au même résultat depuis le code source de l'application, sans utiliser de définition d'interface XML. Cette technique possède quelques inconvénients, comme le fait de ne pas séparer la définition de sa présentation de la logique de cette dernière, ou encore de rendre le code plus difficile à maintenir par des profils métiers différents comme un développeur et un graphiste.

Code 3-5 : Créer une vue depuis le code source

```
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);
        TextView monTextView = new TextView(this);
        setContentView(monTextView);
        monTextView.setText("Bonjour tout le monde !");
    }
}
```

Dans cet exemple, nous n'avons pas appliqué de gabarit `LinearLayout` via `setContentView`. Nous avons uniquement instancié un composant graphique `TextView` puis utilisé la méthode `setContentView` en lui fournissant ce même `TextView`.

Ceci nous donne un résultat identique à l'exemple précédent avec fichier XML.

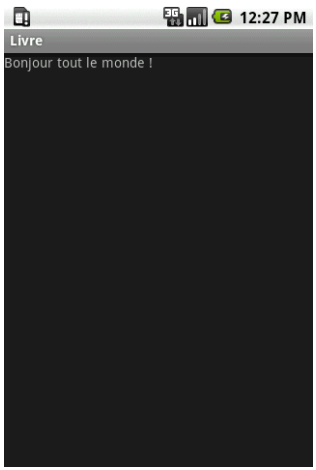


Figure 3–6
Résultat de l'exemple *3*-5

La méthode `setContentView` n'accepte qu'une seule vue. Si vous construisez votre interface depuis le code, vous aurez probablement plus d'une vue à intégrer dans votre activité. Il vous faudra donc réunir vos vues dans un gabarit de vues (par exemple le gabarit `LinearLayout`) et le transmettre à la méthode `setContentView`. Dans notre exemple, nous allons utiliser un gabarit `LinearLayout` pour intégrer plusieurs éléments `TextView` dans notre activité.

Code 3-6 : Créer un groupe de vues de façon programmatique

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.LinearLayout;
import android.widget.TextView;

public class Main extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Nous instancions un LinearLayout dans lequel nous
        // intégrerons nos différents TextView
        LinearLayout monLinearLayout = new LinearLayout(this);

        // Nous paramétrons monLinearLayout afin qu'il affiche
        // les vues les unes au-dessus des autres
        monLinearLayout.setOrientation(LinearLayout.VERTICAL);

        // Nous instancions nos deux TextViews à afficher
        TextView monTextView1 = new TextView(this);
        TextView monTextView2 = new TextView(this);

        // Nous ajoutons les deux TextViews dans notre monLinearLayout
        monLinearLayout.addView(monTextView1);
        monLinearLayout.addView(monTextView2);

        // Nous appliquons monLinearLayout sur notre activité
        setContentView(monLinearLayout);

        // Nous paramétrons un texte à afficher sur nos 2 TextViews
        monTextView1.setText("Bonjour tout le monde !");
        monTextView2.setText("Ceci est mon 2eme texte");
    }
}
```

Dans certains cas, vous serez amené à positionner des vues plutôt que de les aligner. Pour se faire, il faudra utiliser un gabarit `RelativeLayout` et le paramètre `gravity`, spécifiant l'attrait d'un composant vers un coin de l'écran.

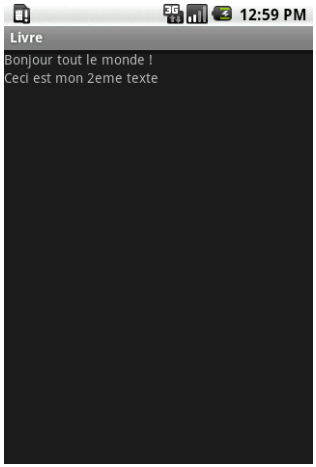


Figure 3–7
Le résultat dans l'émulateur de l'exemple *3*-6

Nous allons maintenant modifier la gravité de nos éléments `TextView` dans la définition de l'interface afin de les aligner en haut et en bas de notre gabarit.

Code 3-7 : Positionnement des vues en déclaratif

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical"
  >
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/monText"
    android:text="Texte en haut à droite"
    android:gravity="top|right"
    />
  <TextView
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="bottom|center_horizontal"
    android:id="@+id/monText2"
    android:text="Texte en bas au centre"
    />
</LinearLayout>
```

Vous noterez l'utilisation du caractère | pour spécifier une combinaison de valeurs et placer vos éléments au plus proche de vos besoins.

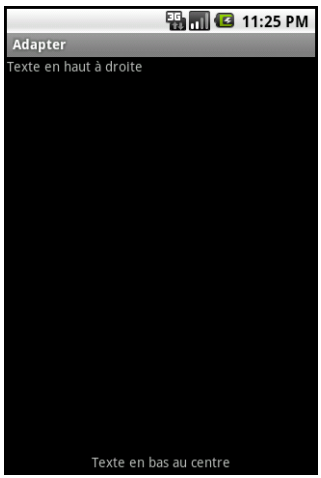


Figure 3–8
Résultat de l'exemple *3-7*

Dans cet exemple, la gravité `top | right` (haut | droite) est appliquée sur l'élément `TextView` (*texte en haut à droite*) qui occupe toute la largeur grâce à la valeur `fill_parent` du paramètre `layout_width`. Ce paramétrage indique au texte de se positionner le plus en haut et à droite de l'espace qu'il occupe. Si à la place de la valeur `fill_parent` nous avions spécifié `wrap_content`, le texte aurait été aligné visuellement à gauche car la place qu'occuperait l'élément `TextView` serait limitée à son contenu.

Pour aligner en bas le deuxième `TextView`, nous avons dû paramétrer la propriété `layout_width` et `layout_height` avec la valeur `fill_parent` et spécifier la valeur `bottom | center_horizontal` (bas | centre horizontal) à la propriété `gravity`.

Gérer les événements

Sous Android, toutes les actions de l'utilisateur sont perçues comme un événement, que ce soit le clic sur un bouton d'une interface, le maintien du clic, l'effleurement d'un élément de l'interface, etc. Ces événements peuvent être interceptés par les éléments de votre interface pour exécuter des actions en conséquence.

Le mécanisme d'interception repose sur la notion d'écouteurs, aussi appelés *listeners* dans la documentation Java. Il permet d'associer un événement à une méthode à appeler en cas d'apparition de cet événement. Si un écouteur est défini pour un élément graphique et un événement précis, la plate-forme Android appellera la méthode associée dès que l'événement sera produit sur cet élément. Par exemple, on

pourra définir un écouteur sur l'événement clic d'un bouton pour afficher un message « Bouton cliqué !». C'est justement ce que nous allons faire ci-après.

Notez que les événements qui peuvent être interceptés ainsi que les méthodes associées sont imposés. Pour un événement `OnClick` (élément cliqué), la méthode associée sera `OnClick()` : il nous suffira alors de définir cette méthode pour notre écouteur pour qu'elle soit appelée lorsque l'utilisateur cliquera sur l'élément graphique associé. Ainsi, pour que l'interface réagisse à un événement sur un élément, il faudra choisir l'élément et l'événement à intercepter, définir un écouteur sur cet événement et définir la méthode associée.

Avant de gérer un événement du type « cliquer sur un bouton », commençons par créer un gabarit avec un bouton centré au milieu de l'écran. Pour intégrer un bouton dans notre gabarit, il suffit d'ajouter une vue `Button`.

Modifiez les déclarations XML du fichier `main.xml` de l'exemple précédent pour y insérer un bouton.

Code 3-8 : Insertion d'un bouton dans l'interface

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_height="fill_parent"
  android:layout_width="fill_parent"
  android:gravity="center_vertical|center_horizontal"
  >
   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/monBouton"
      android:text="Cliquez ici !"
      >
   </Button>
</LinearLayout>
```

Une fois l'instance du bouton `monBouton` récupérée dans le code avec la référence `R.id.monBouton`, vous pouvez associer l'écouteur correspondant à l'événement désiré (ici, à l'aide de `setOnClickListener`.). Modifiez le code de l'activité `main.java`.

Code 3-9 : Création d'un écouteur sur un bouton

```
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
```

```
import android.widget.Button;
import android.widget.Toast;

public class Main extends Activity {

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      setContentView(R.layout.main);

      // Nous cherchons le bouton dans notre interface
      ((Button)findViewById(R.id.monBouton))
      // Nous paramétrons un écouteur sur l'événement 'click' de ce bouton
      .setOnClickListener(new OnClickListener() {
         @Override
         public void onClick(View v) {
            // Nous affichons un message à l'utilisateur
            Toast.makeText(Main.this, "Bouton cliqué !", Toast.LENGTH_LONG).show();
         }
      });
   }
}
```

Dans la méthode `onClick` de l'écouteur que nous avons redéfinie, nous déclenchons une alerte visuelle par l'intermédiaire d'un toast (les alertes visuelles telles que les toasts seront détaillées dans le chapitre 11 traitant des services).

Figure 3–9
Résultat de l'exemple 3-9 lorsque l'utilisateur clique sur le bouton

Intégrer des éléments graphiques dans votre interface

Nous venons de voir comment intégrer un bouton sur une interface mais il existe bien d'autres éléments graphiques que nous pouvons ajouter. Nous allons en voir quelques-uns dans cette partie de façon à construire vos premières interfaces, sachant que nous aborderons la création d'interfaces plus complexes dans le chapitre 5.

Intégrer une image dans votre interface

Pour ajouter une image dans votre interface, utilisez la vue de type `ImageView`. Vous pouvez spécifier la source de l'image directement dans votre définition XML, via l'attribut `src`, ou alors utiliser la méthode `setImageResource` en passant l'identifiant en paramètre.

Pour l'exemple suivant, nous utilisons une image nommée `icon.png` qui se trouve déjà dans les ressources de votre projet (cette ressource est incluse dans le modèle de projet Android d'Eclipse et se trouve dans le dossier `/res/drawable`).

Pour spécifier la taille d'une image dans la valeur de la propriété `src`, plusieurs options s'offrent à vous : soit vous utilisez la dimension réelle de l'image (`wrap_content`) ou la taille de l'écran (`fill_parent`), soit vous spécifiez la taille exacte (exemple : 100 px, 25 dp, etc).

Mettez à jour la définition de l'interface de l'application en insérant un élément `ImageView`.

Code 3-10 : Intégrer une vue image dans une interface

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_height="fill_parent"
   android:layout_width="fill_parent"
   android:gravity="center_vertical|center_horizontal"
  >
   <ImageView
      android:id="@+id/monImage"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:src="@drawable/icon"
   >
   </ImageView>
</LinearLayout>
```

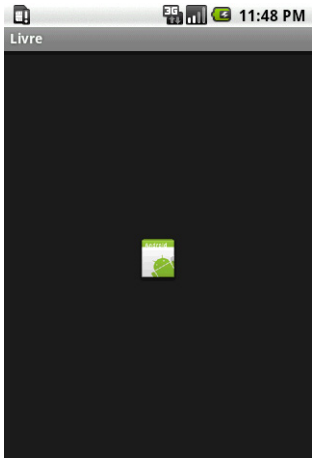


Figure 3–10
Le résultat de l'exemple *3*-10

Vous remarquerez que le code source de l'application n'a pas été changé : seul le code XML de l'interface a eu besoin d'être modifié pour afficher l'image. Si vous aviez souhaité spécifier la source de l'image directement dans le code :

```
ImageView monImage = ...
monImage.setImageResource(R.drawable.icon);
```

Intégrer une boîte de saisie de texte

Pour proposer à l'utilisateur de saisir du texte, vous pouvez utiliser la vue `EditText`.

L'exemple suivant montre l'utilisation de cette vue. Modifiez le contenu du fichier `main.xml` de votre projet pour y insérer la déclaration de la vue `EditText`.

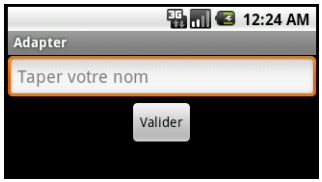
Code 3-11 : Ajout d'une vue de saisie de texte à l'interface utilisateur

```
<?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">
   <!--
      L'élément EditText permettra à
      l'utilisateur de saisir son nom.
   -->
   <EditText
      android:id="@+id/monEditText"
      android:layout_height="wrap_content"
      android:hint="Taper votre nom"
      android:layout_width="fill_parent">
```

```
    </EditText>
    <!--
       Le Bouton qui permettra à l'utilisateur
       de valider sa saisie
    -->
    <Button
       android:id="@+id/monBouton"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="Valider"
       android:layout_gravity="center_horizontal">
    </Button>
</LinearLayout>
```

Exécutez le projet à nouveau, après avoir enregistré la modification, pour voir le résultat.

Figure 3–11
Résultat de l'exemple 3-11



La propriété `hint` de l'élément `EditText` permet l'affichage en fond d'une aide. Tant qu'aucun texte n'est saisi dans la zone dédiée, cette aide apparaît grisée sur le fond. Cela peut vous éviter de mettre un texte de description avant ou au-dessus de chacune des zones de saisie et d'indiquer à l'utilisateur l'objet de sa saisie.

Voyons maintenant comment récupérer ce que l'utilisateur a inscrit lorsqu'il clique sur le bouton *Valider*.

Modifiez la méthode `onCreate` de l'activité `Main` de votre projet.

Code 3-12 : Récupérer la saisie de l'utilisateur

```
...

   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);

      // nous ajoutons un écouteur de type OnClickListener sur le bouton
       // de validation.
      ((Button)findViewById(R.id.monBouton)).setOnClickListener(
                                          ➥ new OnClickListener() {

         @Override
         public void onClick(View v) {
```

```
            // On récupère notre EditText
            EditText texte = ((EditText)findViewById(R.id.monEditText));
            // On garde la chaîne de caractères
            String nom = texte.getText().toString();
            // On affiche ce qui a été tapé
            Toast.makeText(Main.this, nom, Toast.LENGTH_SHORT).show();
        }
    });
  }
}

...
```

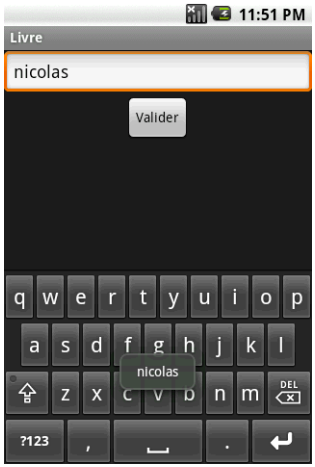


Figure 3–12
Résultat de l'exemple 3-12, après renseignement du nom et clic sur le bouton

Comme vous pouvez le voir, avec l'élément `EditText`, le clavier virtuel apparaît automatiquement dès que la zone de saisie est sélectionnée, sans que vous ayez besoin de le spécifier dans le code.

Intégrer d'autres composants graphiques

Nous allons maintenant intégrer dans notre interface plusieurs types de vues que vous serez amené à utiliser dans vos applications : case à cocher, bouton avec image, bouton radio, sélecteur de date, etc. Autant d'éléments que vous utiliserez couramment dans vos interfaces.

Dans l'exemple que nous vous proposons, nous allons ajouter les éléments suivants : une case à cocher (`CheckBox`), un bouton image (`ImageButton`), deux boutons radio (`RadioGroup` et `RadioButton`), un contrôle de saisie de date (`DatePicker`), une barre de vote (`RatingBar`) et deux horloges, l'une digitale (`DigitalClock`) et l'autre analogique (`AnalogClock`). Avant de vous expliquer comment fonctionne chacun ces éléments, voyons de façon concrète comment les intégrer dans votre application.

JARGON Widget, contrôle et composant graphique

Dans la littérature, les composants graphiques sont parfois nommés *widget* (contraction des mots anglais *window* et *gadget*) ou *contrôles* (de l'anglais *control*).
Dans ce chapitre, par soucis de clarté, nous privilégions la dénomination *composant graphique* (ou de façon plus générale *élément graphique*) mais vous retrouverez aussi les mots *widget* ou *contrôle* au détour d'une phrase dans le reste du livre. Gardez bien à l'esprit que lorsque contrôle est abordé dans le cadre d'une interface graphique, il s'agit de composant graphique.

Pour ajouter ces différents éléments, dans votre projet, modifiez le fichier de définition d'interface `main.xml`.

Code 3-13 : Diverses vues dans un même gabarit

```
<?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">
   <!-- Case a cocher -->
   <CheckBox
      android:id="@+id/CheckBox01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Conditions acceptées ?">
   </CheckBox>
   <!-- Bouton avec une Image -->
   <ImageButton
      android:id="@+id/ImageButton01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:src="@drawable/icon">
      <!-- @drawable/icon est une Image qui se trouve dans le dossier /res/
drawable de notre projet -->
   </ImageButton>
   <!-- Groupe de boutons radio -->
   <RadioGroup
      android:id="@+id/RadioGroup01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:orientation="horizontal"
      android:layout_gravity="center_horizontal">
      <!-- Radio Bouton 1 -->
      <RadioButton
         android:id="@+id/RadioButton01"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="Oui"
         android:checked="true">
```

```
         <!-- Nous avons mis checked=true par défaut sur notre 1er bouton
radio afin qu'il soit coché -->
      </RadioButton>
      <!-- Bouton radio 2 -->
      <RadioButton
         android:id="@+id/RadioButton02"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="Non">
      </RadioButton>
   </RadioGroup>
   <!-- Sélectionneur de date -->
   <DatePicker
      android:id="@+id/DatePicker01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content">
   </DatePicker>
   <!-- Barre de vote -->
   <RatingBar
      android:id="@+id/RatingBar01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content">
   </RatingBar>
   <!-- Nous ajoutons un LinearLayout
      avec une orientation horizontale
      afin d'afficher de gauche à droite les views
      que nous allons grouper dedans -->
   <LinearLayout
      android:orientation="horizontal"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="center_horizontal">
      <!-- Horloge Digital -->
   <DigitalClock
      android:text="Horloge"
      android:id="@+id/DigitalClock01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="center_vertical">
   </DigitalClock>
   <!-- Horloge Analogique -->
   <AnalogClock
      android:id="@+id/AnalogClock01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content">
   </AnalogClock>
</LinearLayout>
```

De la même façon, modifiez votre fichier `main.java` pour y placer le code suivant.

Code 3-14 : Composition de diverses vues dans un même conteneur

```
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.CheckBox;
import android.widget.CompoundButton;
import android.widget.DatePicker;
import android.widget.ImageButton;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.RatingBar;
import android.widget.Toast;

public class Main extends Activity {
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);

      // On récupère notre case à cocher pour intercepter l'événement
      // d'état (cochée ou pas)
      ((CheckBox)findViewById(R.id.CheckBox01)).setOnCheckedChangeListener(
                  ➥ new CheckBox.OnCheckedChangeListener() {
        public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
           afficheToast("Case cochée ? : " + ((isChecked)?"Oui":"Non"));
         }
      });

      // On récupère notre sélectionneur de date (DatePicker) pour attraper
      // l'événement du changement de date
      // Attention, le numéro de mois commence a 0 dans Android, mais pas les jours.
      // Donc si vous voulez mettre le mois de Mai, vous devrez fournir 4 et non 5
      ((DatePicker)findViewById(R.id.DatePicker01)).init(1977, 4, 29,
                  ➥ new DatePicker.OnDateChangedListener() {
         @Override
         public void onDateChanged(DatePicker view, int year, int monthOfYear,
                  ➥ int dayOfMonth) {
            // On affiche la nouvelle date qui vient d'être changée dans notre
            // DatePicker
            afficheToast("La date a changé\nAnnée : " + year + " | Mois : "
                  ➥ + monthOfYear + " | Jour : " + dayOfMonth);
         }
      });

      // On récupère notre barre de vote pour attraper la nouvelle note que
      // sélectionnera l'utilisateur
```

```
        ((RatingBar)findViewById(R.id.RatingBar01)).setOnRatingBarChangeListener(
                    ➥ new RatingBar.OnRatingBarChangeListener() {
           @Override
           public void onRatingChanged(RatingBar ratingBar, float rating,
                    ➥ boolean fromUser) {
              // On affiche la nouvelle note sélectionnée par l'utilisateur
              afficheToast("Nouvelle note : " + rating);
           }
        });

        // On récupère notre groupe de bouton radio pour attraper le choix de
        // l'utilisateur
        ((RadioGroup)findViewById(R.id.RadioGroup01)).setOnCheckedChangeListener(
                    ➥ new RadioGroup.OnCheckedChangeListener() {
           @Override
           public void onCheckedChanged(RadioGroup group, int checkedId) {
              // On affiche le choix de l'utilisateur
              afficheToast("Vous avez répondu : "
                    ➥ + ((RadioButton)findViewById(checkedId)).getText());
           }
        });

        // On récupère notre Bouton Image pour attraper le clic effectué par
        // l'utilisateur
        ((ImageButton)findViewById(R.id.ImageButton01)).setOnClickListener(
                    ➥ new OnClickListener() {
           @Override
           public void onClick(View v) {
              // On affiche un message pour signalé que le bouton image a été pressé
              afficheToast("Bouton Image pressé");
           }
        });
     }

     // Méthode d'aide qui simplifiera la tâche d'affichage d'un message
     public void afficheToast(String text)
     {
        Toast.makeText(this, text, Toast.LENGTH_SHORT).show();
     }
}
```

Dans cet exemple, nous employons diverses vues que vous serez souvent amené à utiliser :

- la case à cocher (`CheckBox`) : propose un choix basique à vos utilisateurs : « oui » ou « non ». Pour récupérer à tout moment l'état de la case à cocher, il vous suffit de le récupérer avec la méthode `isChecked`. Dans notre exemple nous récupérons l'état dès qu'il change, grâce à l'écouteur `OnCheckedChangeListener` que nous

avons associé à la case à cocher. Ceci nous permet de récupérer immédiatement le changement d'état et si nécessaire d'exécuter du code en conséquence ;

- le bouton image (`ImageButton`) : fonctionne de la même façon qu'un bouton classique mais présenté précédemment sous la forme de l'image choisie. La principale différence avec un bouton classique est que l'on ne peut ajouter un texte à afficher : seule l'image est affichée ;
- le groupe de boutons radio et les boutons radio (`RadioGroup` et `RadioButton`) : propose une liste de choix à réponse unique (seul un des boutons du groupe de boutons pourra être coché à un instant *t*). Afin que les boutons radio (de type `RadioButton`) puissent basculer de l'un à l'autre, vous devez les regrouper dans un élément `ViewGroup` de type `RadioGroup`. Si vos boutons radio ne sont pas regroupés dans un `RadioGroup`, ils seront tous indépendants. Dans notre exemple, nous attrapons le changement d'état des boutons radio dès qu'ils sont pressés grâce à l'écouteur `OnCheckedChangeListener`que nous avons associé au `RadioGroup`. Si vous souhaitez récupérer le bouton radio pressé à n'importe quel moment il vous suffira d'appeler la méthode `getCheckedRadioButtonId` ;
- le sélectionneur de date (`DatePicker`) : affiche les boutons nécessaires pour saisir une date. Attention celui-ci s'adapte visuellement à la langue et région du téléphone. Ainsi, un téléphone en langue « Anglais US » affiche la date sous la forme Mois-Jour-Année tandis que le même téléphone en langue française l'affiche sous la forme Jour-Mois-Année. Dans notre exemple, nous récupérons le changement de date de notre sélectionneur de date dès que l'utilisateur modifie une valeur grâce à l'écouteur `OnDateChangedListner` que nous lui avons associé. Pour récupérer les valeurs sélectionnées, vous pouvez à tout moment utiliser les méthodes`getDayOfMonth`, `getMonth`, `getYear`. Attention, comme signalé en commentaire dans l'exemple, le numéro des mois commence à 0 (janvier = 0, février = 1 etc.) ce qui n'est pas le cas pour les jours et les années ;
- la barre de vote (`RatingBar`) : affiche une barre comportant des étoiles pour représenter et/ou récupérer une notation. Dans notre exemple nous récupérons le changement de note dès que l'utilisateur modifie celle-ci grâce à l'écouteur `OnRatingBarChangeListener` associé. Vous pouvez également à tout moment dans votre code récupérer la note avec la méthode `getRating`.
- l'horloge digitale (`DigitalClock`) : affiche une horloge digitale dans votre application. Par exemple : 10:05:25 ;
- l'horloge analogique (`AnalogClock`) : affiche une horloge à aiguilles dans votre application.

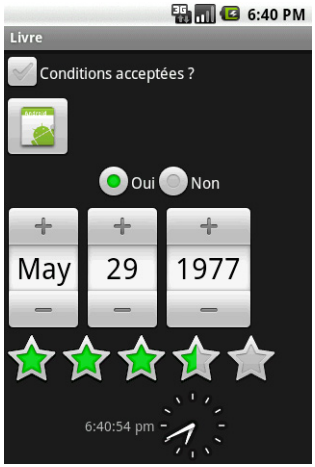


Figure 3–13
Résultat du code 3-14 dans l'émulateur Android

En ce qui concerne la barre de vote, vous pouvez définir la graduation de la note en modifiant le code XML comme ceci.

Code 3-15 : changement du pas pour la notation dans la barre de vote

```
<RatingBar
    android:id="@+id/RatingBar01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:stepSize="1"
</RatingBar>
```

Par défaut la vue `RatingBar` possède un pas de 0,5. Ce qui vous permet de récupérer des notes à 0,5 près, comme 3,5 par exemple. Dans le code ci-dessus, en précisant la propriété `stepSize=1` nous paramétrons le pas à 1. Ce qui veut dire que l'utilisateur ne pourra donner qu'une note entière.

Découper ses interfaces avec include

Comme dans les interfaces web, votre application comportera souvent des portions d'interfaces communes à l'ensemble de l'application (menu de navigation, en-tête et pied de page, etc). Ces portions d'interfaces communes – par exemple pour une application mobile, un bandeau supérieur contenant des informations – devront être dupliquées dans toutes les définitions de l'interface. Ce travail sera fastidieux et n'en sera que plus compliqué à maintenir puisque chaque modification devra être répercutée pour chaque définition d'interface.

Afin de combler cette lacune, la plate-forme Android dispose d'une instruction `include` permettant d'inclure dans la définition d'une interface, des éléments contenus dans une autre interface. Avec ce mécanisme, il suffit juste de modifier la partie de l'interface qui est incluse pour que la modification soit répercutée dans l'ensemble des interfaces.

Afin d'illustrer ce mécanisme, nous allons créer deux fichiers XML (un en-tête de page et un pied de page) qui seront ensuite inclus dans une autre interface.

Créez un premier fichier XML (l'en-tête de page) et nommez-le `include_haut.xml`.

Code 3-16 : Gabarit du haut de page

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content">
   <ImageView
      android:id="@+id/ImageView01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:src="@drawable/icon">
   </ImageView>
   <TextView
      android:id="@+id/TexteHaut"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Ceci est mon haut de page"
      android:layout_gravity="center_vertical">
   </TextView>
</LinearLayout>
```

Puis créez une second fichier XML (le pied de page) nommé `include_bas.xml`.

Code 3-17 : Gabarit du pied de page

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

   <TextView
      android:id="@+id/TextView01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Ceci est mon pied de page"
      android:layout_gravity="center_vertical">
   </TextView>
```

```
    <ImageView
        android:id="@+id/ImageView01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/icon">
    </ImageView>

</LinearLayout>
```

Pour inclure une interface dans une autre, utilisez l'élément `<include>` à l'endroit où vous souhaitez insérer les éléments communs de l'interface et renseignez l'attribut `layout` pour spécifier l'identifiant de l'interface à inclure à cet emplacement.

Assemblons nos deux définitions d'interface d'en-tête et de pied de page (code 3-16 et 3-17) dans un fichier `main.xml`.

Code 3-18 : Inclusion d'interfaces dans le gabarit d'une interface

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">
    <!-- Inclusion de l'interface comportant l'en-tête de page -->
    <include
        android:id="@+id/include01"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        layout="@layout/include_haut">
    </include>

    <TextView
        android:text="Ici se trouve le texte et tout ce que\nje souhaite
        afficher\ndans mon application"
        android:id="@+id/TextView01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
    </TextView>

    <!-- Insertion du pied de page -->
    <!--
        Afin de positionner le pied de page en bas de l'écran, nous
        l'intégrons dans un LinearLayout prenant toute la place
        restante et paramétrons l'élément include avec le paramètre
        android:layout_gravity="bottom" pour le coller en bas.
    -->
```

```
    <LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      >
      <!-- Insertion de l'interface comportant le pied de page -->
       <include
          android:id="@+id/include02"
          android:layout_width="fill_parent"
          android:layout_height="wrap_content"
          layout="@layout/include_bas"
          android:layout_gravity="bottom">
       </include>
    </LinearLayout>
</LinearLayout>
```

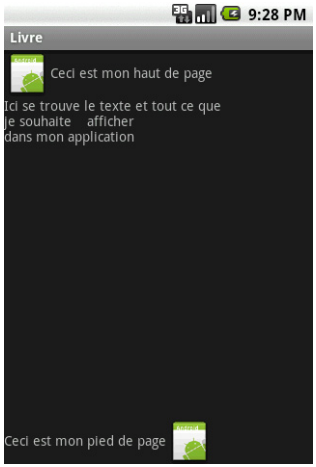


Figure 3–14
Résultat du *code* 3-18, après insertion de l'interface

Comme dans n'importe quelle interface, vous pouvez accéder aux éléments insérés.

Code 3-19 : Récupérer une vue d'un gabarit importé

```
// Nous récupérons notre gabarit d'inclusion.
LinearLayout monInclude = (LinearLayout)findViewById(R.id.include01);

// Nous récupérons le composant TextView qui se trouve
// dans la partie importée de l'interface.
TextView monText = (TextView)monInclude.findViewById(R.id.TexteHaut);

// Nous changeons le texte de ce composant.
monText.setText("Nouveau texte dans le haut de page");
```

Par sa simplicité de réalisation, ce mécanisme d'importation d'interface permet aux développeurs d'être souvent plus efficaces dans la réalisation et la maintenance des applications Android, dans la mesure où l'interface (que l'on inclut) n'a besoin d'être écrite qu'à un seul endroit pour être présente partout.

Ajouter des onglets

La taille de l'écran d'un téléphone étant limitée, vous serez souvent amené à utiliser des onglets pour afficher tout ce que votre application peut mettre à disposition de l'utilisateur.

Pour créer des onglets dans votre interface, vous devez d'abord ajouter un élément `TabHost` dans le gabarit de l'interface. À ce `TabHost`, vous devez ajouter un `TabWidget` comportant impérativement un identifiant `android:id="@android:id/tabs"`. Enfin, vous devrez ajouter un `ViewGroup` comportant les vues à afficher pour chaque onglet. Pour mettre en pratique ce que nous venons d'écrire, créez une interface dans un fichier `main.xml`, comme dans le code suivant.

Code 3-20 : Intégrer des onglets à l'interface utilisateur

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
<!-- Le TabHost qui contient tous les éléments de nos onglets
-->
<TabHost
   android:id="@+id/TabHost01"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent">
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
        <!-- TabWidget qui sert à afficher les onglets -->
      <TabWidget android:id="@android:id/tabs"
         android:layout_width="fill_parent"
         android:layout_height="wrap_content">
      </TabWidget>
      <!-- contenu de nos onglets. -->
      <FrameLayout
         android:id="@android:id/tabcontent"
         android:layout_width="fill_parent"
         android:layout_height="fill_parent">
```

```
        <!-- Contenu de l'onglet N°1 -->
        <LinearLayout
           android:orientation="vertical"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:scrollbars="vertical"
            android:id="@+id/Onglet1">
           <TextView
              android:text="Ceci est un texte dans l'onglet N°1"
              android:id="@+id/TextViewOnglet1"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content">
           </TextView>
        </LinearLayout>
        <!-- Contenu de l'onglet N°2 -->
        <LinearLayout
            android:orientation="vertical"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:id="@+id/Onglet2">
           <TextView
              android:text="Ceci est un texte dans l'onglet N°2"
              android:id="@+id/TextViewOnglet2"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content">
           </TextView>
           <ImageView
              android:id="@+id/ImageView01"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:src="@drawable/icon">
           </ImageView>
        </LinearLayout>
        <!-- Contenu de l'onglet N°3 -->
        <LinearLayout
            android:orientation="vertical"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:id="@+id/Onglet3">
           <TextView
              android:text="Ceci est un texte dans l'onglet N°3"
              android:id="@+id/TextViewOnglet3"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content">
           </TextView>
        </LinearLayout>
      </FrameLayout>
   </LinearLayout>
</TabHost>
</LinearLayout>
```

Dans le code de l'activité associée à l'interface, insérez les lignes suivantes.

Code 3-21 : Intégrer des onglets à l'interface utilisateur

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.TabHost;
import android.widget.Toast;
import android.widget.TabHost.TabSpec;

public class Main extends Activity {

   private TabHost monTabHost;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main7);

      // Récupération du TabHost
      monTabHost =(TabHost) findViewById(R.id.TabHost01);
      // Avant d'ajouter des onglets, il faut impérativement appeler la méthode
      // setup() du TabHost
      monTabHost.setup();

      // Nous ajoutons les 3 onglets dans notre TabHost

      // Nous paramétrons le 1er Onglet
      TabSpec spec = monTabHost.newTabSpec("onglet_1");
      // Nous paramétrons le texte qui s'affichera dans l'onglet
      // ainsi que l'image qui se positionnera
      // au dessus du texte.
     spec.setIndicator("Onglet 1",getResources().getDrawable(R.drawable.icon));
      // On spécifie le Layout qui s'affichera lorsque l'onglet sera sélectionné
      spec.setContent(R.id.Onglet1);
      // On ajoute l'onglet dans notre TabHost
      monTabHost.addTab(spec);

      // Vous pouvez ajouter des onglets comme ceci :
      monTabHost.addTab(monTabHost.newTabSpec("onglet_2").setIndicator(
            ➥ "Onglet 2").setContent(R.id.Onglet2));
      monTabHost.addTab(monTabHost.newTabSpec("onglet_3").setIndicator(
            ➥ "Onglet 3").setContent(R.id.Onglet3));

      // Nous paramétrons un écouteur onTabChangedListner pour récupérer
      // le changement d'onglet.
      monTabHost.setOnTabChangedListener(
            new TabHost.OnTabChangeListener (){
               public void onTabChanged(String tabId){
```

```
                    // Vous pourrez exécuter du code lorsqu'un
                    // onglet est cliqué. Pour déterminer
                    // quel onglet a été cliqué, il
                    // vous suffira de vérifier le tabId envoyé lors
                    // du clic et d'exécuter votre code en
                    // conséquence.
                    Toast.makeText(Main.this, "L'onglet avec l'identifiant : "
                        ➥ + tabId + " a été cliqué", Toast.LENGTH_SHORT).show();
                }
            }
        );
    }
}
```

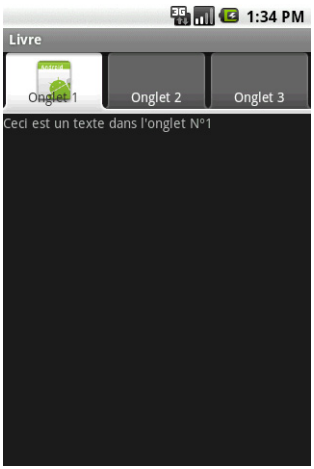


Figure 3–15 Résultat des codes 3-20 et 3-21

Figure 3–16 Résultat des codes 3-20 et 3-21 : sélection du deuxième onglet

Comme vous pouvez le constater, le contenu du deuxième onglet est bien différent du premier et comporte bien tous les éléments que nous avons paramétrés dans la déclaration d'interface en XML. Lorsque que vous cliquez sur un autre onglet, le message associé au clic apparaît et vous pouvez exécuter du code en conséquence si nécessaire.

Dans certains cas, votre contenu pourra comporter plus d'éléments qu'il ne peut en afficher à l'écran. Dans ce cas, vous pourrez faire en sorte de faire défiler le contenu avec l'aide d'un ascenseur défini par l'élément `ScrollView` (ou vue de défilement). La particularité du `Scrollview` est qu'il ne peut avoir qu'un seul enfant. Pour pallier cette limitation, insérez un autre gabarit, par exemple un `LinearLayout`, qui affichera alors tous les éléments du gabarit avec une barre de défilement sur l'écran.

Code 3-22 : Utilisation d'une vue de défilement ScrollView

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
   <ScrollView
      android:id="@+id/ScrollView01"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent">
      <LinearLayout
         android:id="@+id/LinearLayout01"
         android:layout_width="fill_parent"
         android:layout_height="fill_parent"
         android:orientation="vertical">
         <TextView
            android:text="Ceci est un texte\nComportant plusieurs\nlignes"
            android:id="@+id/TextView01"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content">
         </TextView>
         <Button
            android:text="Mon bouton"
            android:id="@+id/Button01"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content">
         </Button>
         <ImageView
            android:id="@+id/ImageView01"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:src="@drawable/icon">
         </ImageView>
         <TextView
            android:text="Ceci est un texte\nComportant plusieurs\nlignes"
            android:id="@+id/TextView01"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content">
         </TextView>
         <DatePicker android:id="@+id/DatePicker01"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content">
         </DatePicker>
         <TextView
            android:text="Ceci est un texte\nComportant plusieurs\nlignes"
            android:id="@+id/TextView01"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content">
         </TextView>
```

ANNEXE

Développer sur Android

Le développement sur Android n'est pas si difficile, du moins lorsque l'on sait utiliser les outils mis à notre disposition.

Utiliser les téléphones de développement

> **REMARQUE** **Acheter un téléphone de développement ou pas ?**
>
> Pour utiliser un appareil sur n'importe quel opérateur, il suffit désormais d'entrer le code de déverrouillage avant de taper le code PIN. Ce code peut être acheté auprès d'un revendeur spécialisé ou fourni directement par le constructeur. À noter que les appareils GeeksPhone, LG, Motorola et Samsung possèdent pour le moment des claviers. L'archos 5, lui, n'a pas de caméra, ni de GPS. Le Lenovo a un « dock » avec clavier. Pour pouvoir réellement tester une application Android, il faudrait pratiquement posséder tous les appareils, ce qui est rarement possible. Nous vous conseillons donc de bien organiser vos tests et de gérer un maximum de cas de figure. Si l'on devait ne retenir qu'un appareil au moment de la rédaction de ce livre, nous vous conseillerions le Nexus One.

Depuis l'origine d'Android, Google propose sur le marché Android deux téléphones de développement : les Android Dev Phone 1 et 2. Le premier est équivalent à un HTC Dream G1, le second est pour sa part un HTC Magic. Vous ne pourrez vous procurer ces téléphones qu'après vous être inscrit sur le site de l'Android Market en tant que développeur (la procédure d'inscription est décrite dans le chapitre 15 dédié à la publication des applications).

Ces téléphones de développement sont débridés, c'est-à-dire qu'ils sont utilisables avec tous les opérateurs et disposent des droits pour pouvoir être mis à jour avec n'importe quelle version de la plate-forme Android. Cependant concernant le G1, sa faible capacité mémoire ne lui permet pas d'installer Android2.0.

Tableau A–1 Tableau récapitulatif des caractéristiques techniques des deux téléphones de développement, telles qu'indiquées sur l'Android Market

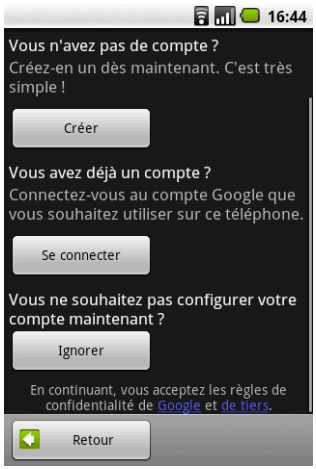
Dev Phone 1	Dev Phone 2
	
• Touch screen • Trackball • Appareil photo 3.2 Megapixel Auto focus • Wi-Fi • Bluetooth v2.0 • 3G WCDMA (1700/2100 MHz) • GSM (850/900/1800/1900 MHz) • GPS • Clavier coulissant QWERTY • Inclut une carte MicroSD 1GB (peut être remplacée par une carte allant jusqu'à 16Go)	• Android 1.6 • Dimension (mm) 113 x 55.56 x 14.65 • Taille écran 3.17 pouces • HVGA Resolution • Touch screen • Mémoire Flash 512MB • 192MB RAM • MSM7200A,528MHz chipset • GSM/GPRS/EDGE • 850/900/1800/1900 MHz • WCDMA 1700/2100 MHz : BC4 • 2100 MHz : BC1 • HSPA Speed HSDPA 7.2 Mbps • HSUPA 2 Mbps • Blutooth 2.0 avec EDR • Wi-Fi 802.11b/g • Appareil photo Auto Focus 3 Mégapixels • 1340 Battery(mAh) • Port mémoire microSD • USB 2.0 • GPS/AGPS • Mailing list
399€	399€

> **À NOTER Téléphones de développement versus téléphones du commerce**
>
> Il y a encore quelques mois, seuls les téléphones de développement pouvaient être débridés et modifiés complètement au niveau logiciel (« rooté » dans le jargon du développeur), y compris au niveau des accès système. Ils ont ainsi permis de promouvoir la plateforme Android et d'en accélérer l'adoption par les développeurs.
> De nos jours, les nouvelles générations d'appareils (tel que le Nexus One) ont dépassé leurs ainés puisqu'en plus de pouvoir être débridés (se renseigner auprès du constructeur ou du vendeur pour cela) et « rootés », ils offrent des caractéristiques techniques plus avancées pour un prix tout aussi intéressant. Un tel téléphone peut ainsi être utilisé comme téléphone de développement en lieu et place des téléphones historiques.
> Dans ce qui suit, nous utiliserons néanmoins un téléphone de développement, au cas où vous en auriez un.

Première utilisation du téléphone de développement

Après avoir déballé le téléphone, allumez-le. Une succession d'écrans vous invite à vous identifier grâce à un compte Google si vous en possédez un. Cette identification permet à Android de configurer la messagerie et le calendrier de votre compte Google. Vous pouvez ignorer cette identification et configurer ultérieurement votre téléphone si vous le souhaitez.

Figure A–1
Associer un compte Google au téléphone



Une fois votre compte Google associé à votre téléphone de développement, vous serez guidé pour choisir vos préférences concernant vos informations personnelles. Ces choix sont modifiables plus tard si vous voulez revenir sur ce que vous avez indiqué.

Ensuite, à vous de définir l'heure et les différentes options de format. Appuyez enfin sur *Terminer l'installation* et votre téléphone est prêt à l'utilisation !

ASTUCE Utiliser un téléphone de développement sans forfait données

Le premier écran, au démarrage de votre téléphone, vous invite à enregistrer un compte Google pour pouvoir utiliser l'appareil. Évidemment cette opération n'est possible que si votre forfait est activé pour transmettre des données.

Pour éviter l'utilisation d'un forfait données, vous pouvez passer l'étape, d'enregistrement (via le bouton *ignorer*), paramétrer une connexion Wi-Fi et reprendre le processus (en lançant le client de messagerie GMail par exemple).

Mais vous pouvez également faire immédiatement apparaître le panneau de configuration Wi-Fi et franchir normalement les différentes étapes. Pour cela vous devez connecter votre téléphone à votre ordinateur, installer les drivers comme expliqué plus loin dans ce chapitre, en clair prendre un peu d'avance sur le déroulement normal de la lecture de ce livre !

Une fois la connexion à l'ordinateur configurée, nous allons utiliser l'outil *adb* qui se trouve dans le répertoire *tools* du kit de développement que vous venez d'installer. Nous reviendrons plus tard sur ses utilisations, l'urgence est de pouvoir utiliser votre téléphone !

```
adb shell
```

Une invite de commande commençant par $ apparaît. Saisir (sans retour à la ligne) :

```
am start -a android.intent.action.MAIN -n com.android.settings/
.Settings
```

Le panneau de configuration Wi-Fi apparaît, à vous de jouer !

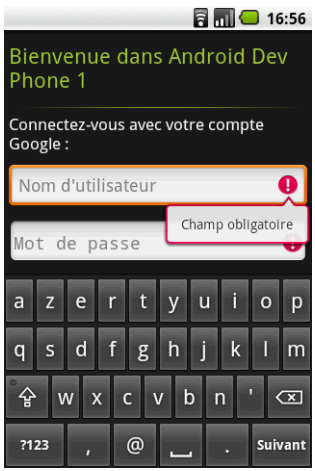


Figure A–2
Saisie des paramètres de votre compte Google

Connecter le téléphone à votre ordinateur

Maintenant que votre téléphone est fonctionnel, le temps est venu de le brancher à votre ordinateur. En effet pour pouvoir déboguer directement sur votre téléphone, vous devez le connecter à votre ordinateur. Cela se fait via le câble USB qui est fourni avec le téléphone. À noter, le téléphone est vu comme un périphérique USB spécifique qui nécessite des drivers particuliers (fournis dans le kit de développement). Nous allons aborder ce point maintenant.

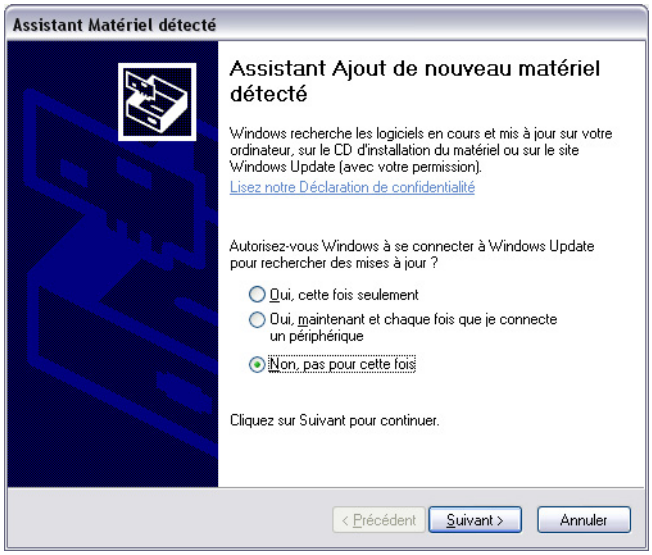
> **À SAVOIR Windows et les autres plates-formes**
>
> Ce chapitre ne traitera que de la plate-forme Windows. Pour déboguer sur les autres systèmes d'exploitation, reportez vous à la documentation Android disponible sur le site officiel.

Branchez le câble USB, fourni avec l'appareil, à votre ordinateur.

> **CONSEIL Choix d'un câble USB**
>
> Votre téléphone est vendu avec un câble USB. Utilisez-le en priorité. Nous avons pu lire sur Internet quelques mésaventures arrivées à des utilisateurs qui ont utilisé d'autres câbles USB de moins bonne qualité (notamment beaucoup plus fins). Le téléphone se mettait en charge mais n'était pas détecté par l'ordinateur. Penser à un problème de câble USB dans ce cas-là n'est pas totalement intuitif.

En prenant le cas de Windows (XP sur les captures d'écran), une fois le téléphone branché, un écran apparaît pour vous avertir qu'un nouveau matériel a été détecté. Il est tout à fait inutile de se connecter à *Windows Update :* choisissez donc *Non, pas pour cette fois* quand Windows vous propose une connexion pour rechercher les mises à jour.

Figure A–3
Nouveau matériel détecté



En sélectionnant *Suivant*, Windows vous demande un CD d'installation ou équivalent. Nous allons le faire pointer vers le répertoire d'installation du kit de développement où nous avons téléchargé la plate-forme 2.0 d'Android avec les drivers USB de l'appareil. Choisissez *Installer à partir d'un emplacement spécifié* dans la fenêtre.

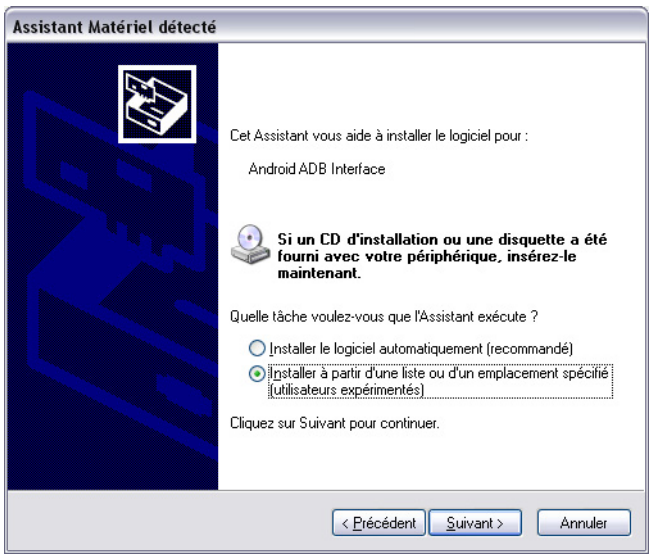


Figure A–4
Choisir un emplacement spécifique

Cliquez sur *Suivant*, l'installation démarre.

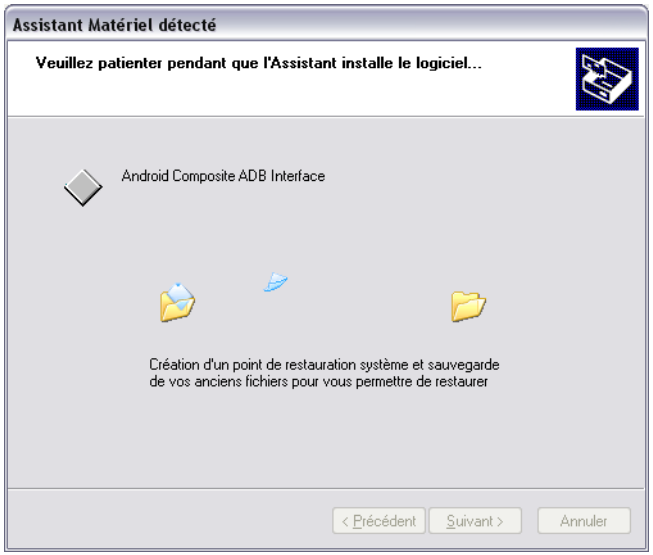


Figure A–5
Progression de l'installation

Une fois l'installation terminée, fermez la fenêtre. Windows reconnaîtra votre téléphone quand vous le brancherez grâce à un câble USB.

Pour pouvoir déboguer des programmes grâce à votre téléphone, il reste un réglage à réaliser. Déplacez-vous dans les interfaces de votre téléphone *Menu>Paramètres>Applications>Développement* et cochez débogage USB.

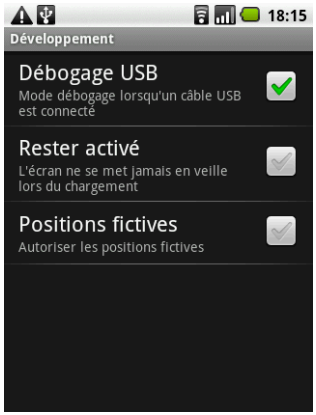


Figure A–6
Mode débogage configuré sur le téléphone

Côté téléphone, la barre de notification, en haut de l'écran, vous confirme la connexion USB et le mode de débogage.

En déroulant cette interface, vous pouvez même choisir de monter la carte mémoire de l'appareil, c'est à dire la transformer en stockage de masse type clé USB, où vous pourrez mettre de nouveaux fichiers (photos, sons, mises-à-jour du système, etc.). Pour cela, cliquez sur *USB connected*, puis sur le bouton *Monter*.

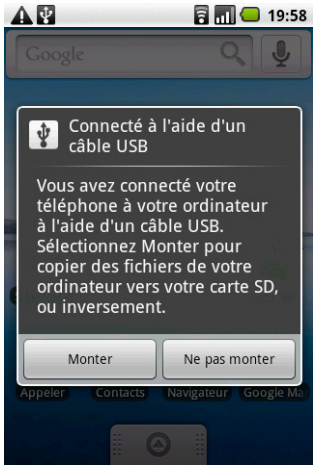


Figure A–7
Notification de connexion USB et utilisation de la carte du téléphone

Tout est prêt !

Utilisation de votre téléphone avec Eclipse

En fait, il n'y a rien de plus à faire pour lancer votre application sur le téléphone de développement. Il suffit que le câble USB soit branché entre votre téléphone et votre ordinateur et de lancer l'application, l'environnement de développement l'enverra automatiquement sur le téléphone.

Une fois plus avancé dans le livre et dans vos développements, vous pourrez modifier ce comportement choisissant l'émulateur à la place du téléphone. Pour cela il faut régler les paramètres dans le menu *Run>Run configurations...* Sélectionnez à gauche le projet concerné et allez dans l'onglet *Target*.

À noter que dans la vue *package explorer* d'Eclipse, un clic droit sur un projet Android vous permet de faire apparaître un *Run as ...>Android Application* sur lequel vous pouvez cliquer. Un écran apparaît alors vous invitant à choisir entre le téléphone si il est branché et l'émulateur (dans une des versions matérielles créées).

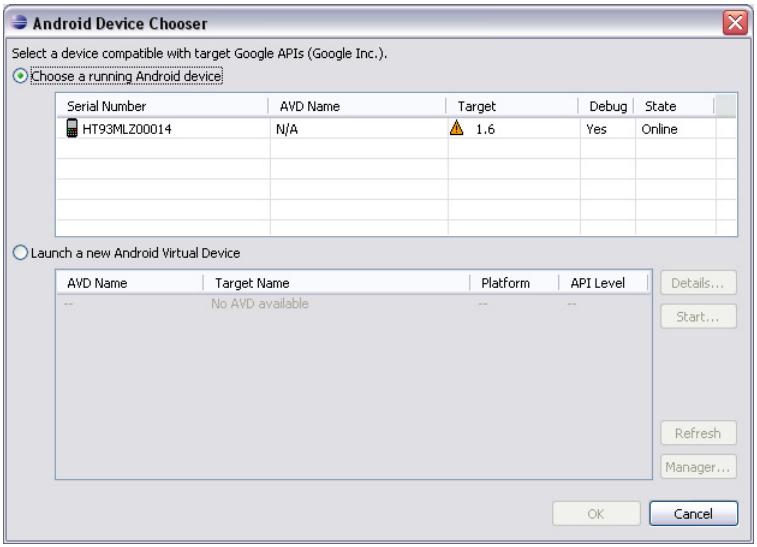


Figure A–8
Écran de sélection du moyen d'exécution à partir du menu Run as...

Si vous choisissez le mode *debug*, un message vous invitera à attendre quelques instants au lancement de l'application, le temps que l'environnement de développement paramètre correctement l'exécution du programme lancé.

Tout se déroule ensuite exactement comme avec l'émulateur et vice versa.

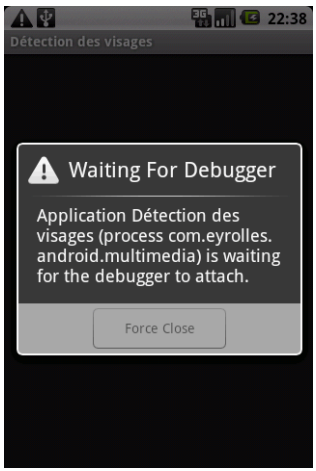


Figure A–9
Écran intermédiaire avant le lancement de l'application en mode debug

Si vous avez un doute, vous pouvez contrôler la présence de votre téléphone avec le menu *Windows>Show View>Other...>Android>Devices*. Dès que vous connecterez votre téléphone, l'affichage se mettra à jour et vous donnera en détails les processus qui s'exécutent sur votre téléphone.

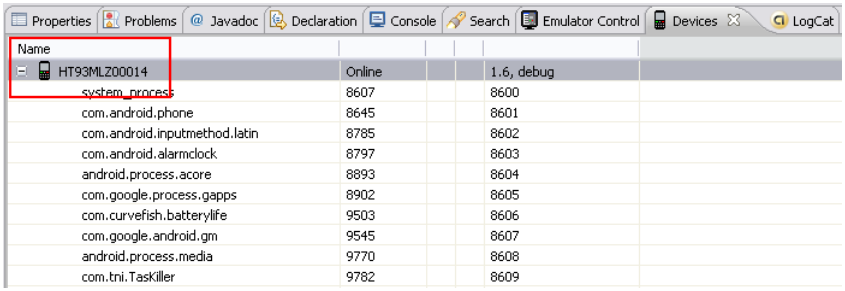


Figure A–10
Vue du plugin ADT concernant les appareils

> **ASTUCE** **Installer une application de source inconnue**
>
> Si vous souhaitez installer des applications dont la source est inconnue, c'est-à-dire ne provenant pas de l'Android Market, vous pouvez paramétrer votre téléphone pour les accepter. Naviguez dans *Menu>Settings>Applications* puis cochez la case *Unknown sources*.

Réglages cachés du navigateur

Afin de permettre aux développeurs d'être correctement outillé pour déboguer leurs applications, Android propose un menu de débogage qui est par défaut invisible des utilisateurs.

Pour activer ce menu caché, entrez l'adresse suivante dans la barre d'adresse du navigateur d'Android : about:debug. Une fois cette opération réalisée, naviguez dans *Menu>More>Settings* dans le navigateur et si vous descendez tout en bas, vous devriez voir de nouveaux réglages ou actions disponibles.

Prenons comme exemple le sous-menu *UAString* (User-Agent), qui correspond à la chaîne qui identifie le type d'appareil utilisant le navigateur. Les sites Internet peuvent récupérer cette information et savoir qu'un appareil mobile se connecte et ainsi adapter les pages qu'ils proposent au téléphone ou à la tablette. Ce paramètre sert beaucoup également en termes de statistiques pour calculer les parts de marché.

Figure A–11
Exemple de réglage caché

Ici, comme vous pouvez le voir sur la capture d'écran, vous pouvez vous faire passer pour un navigateur de système bureautique, c'est à dire un ordinateur de travail (pour éviter d'obtenir des pages adaptées) ou pour un iPhone !

Utiliser le concepteur graphique dans Eclipse pour réaliser des interfaces graphiques

Le module ADT (*Android Developement Tools*) d'Eclipse fournit un grand nombre d'outils, que ce soit au niveau de l'édition, de la structure du projet, de la compilation, du débogage ou des différentes fenêtres de génération. Parmi ces outils, il en existe un vous permettant de visualiser et d'éditer l'interface utilisateur de vos activités.

Bien qu'à l'origine cet outil était destiné aux débutants dans ses premières versions en raison de limites techniques, les dernières versions d'ADT prennent maintenant en compte les composants personnalisés dans le rendu visuel et en font maintenant un réel outil utilisable par les développeurs avancés.

Cet outil, nommé *concepteur* ci-après, peut vous faire gagner un temps non négligeable pour concevoir rapidement vos interfaces sans avoir à tester chaque modification dans l'émulateur ou sur un téléphone.

Le concepteur est disponible lorsque vous éditez une déclaration d'interface utilisateur, c'est à dire un fichier XML localisé dans `/res/layout` de votre projet. Sélectionnez l'onglet *Layout* et non `<nom_layout>.xml` en bas de la vue d'édition pour disposer de l'aperçu en temps réel.

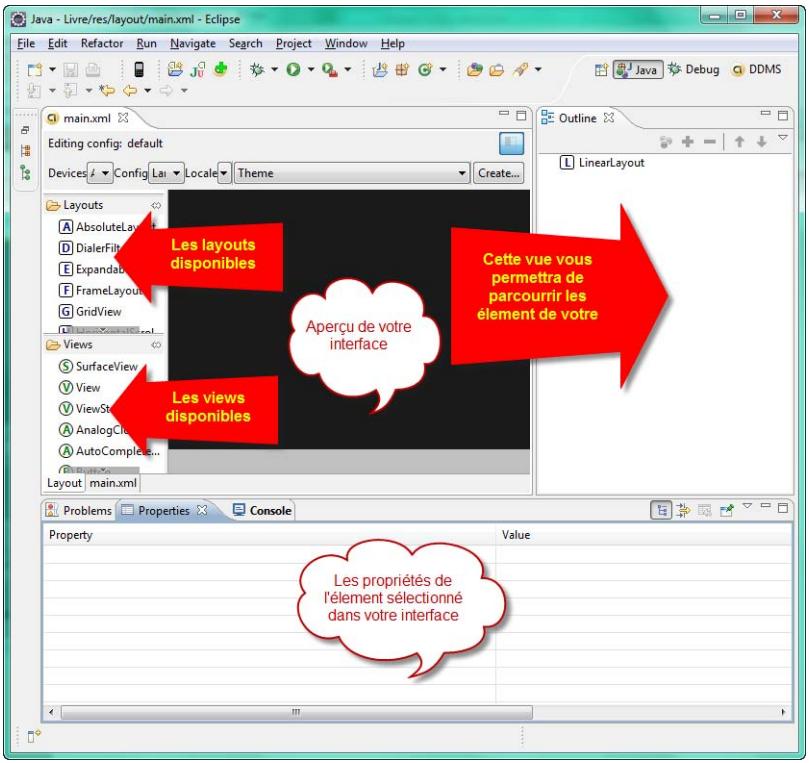


Figure A–12
Vue du concepteur d'interface utilisateur dans Eclipse

Dans la partie supérieure gauche, vous trouverez tous les types de gabarits de mise en page disponibles que vous pourrez intégrer dans votre interface. Dans la partie inférieure gauche, vous trouverez tous les types de vues que vous pouvez intégrer dans votre interface.

Naviguer dans le concepteur d'interfaces

Sur la droite se trouve la fenêtre *Outline* qui est un aperçu de notre schéma XML. Cette fenêtre fournie par Eclipse est très importante car dans certains cas elle vous

permettra d'accéder à des éléments de votre interface qui ne le seront pas via la zone centrale, qui comporte l'aperçu de l'interface. En bas de la fenêtre, vous trouverez la liste des propriétés de l'élément sélectionné dans votre interface. Cet affichage est également très important car il vous permettra de voir toutes les propriétés disponibles pour l'élément sélectionné.

Par exemple, en sélectionnant le `LinearLayout` à droite, nous pouvons voir toutes les propriétés modifiables pour cet élément.

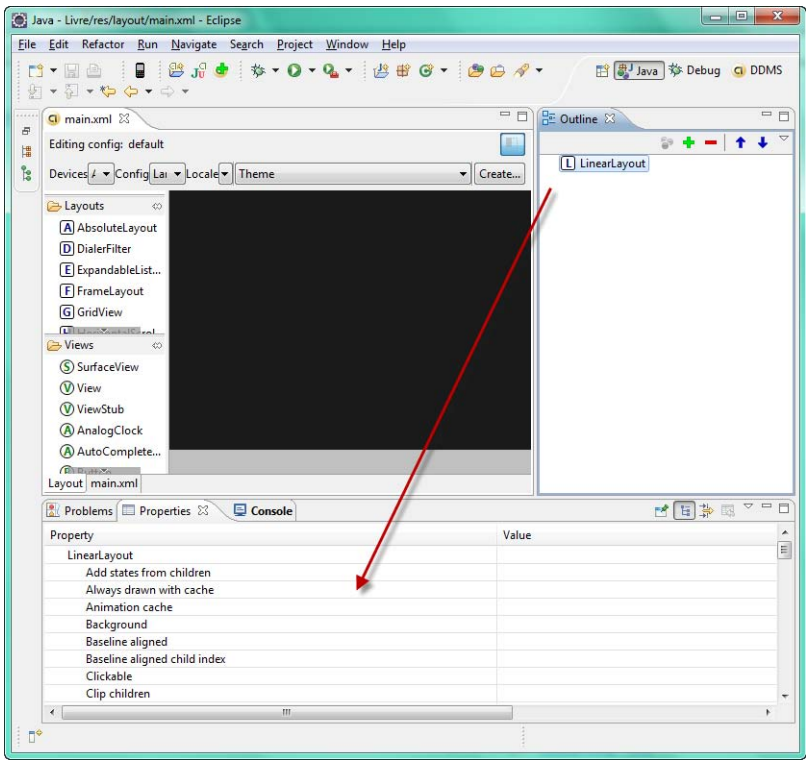


Figure A–13
Propriétés d'une vue dans Eclipse

En faisant défiler les propriétés, vous retrouverez des propriétés que nous avons abordées dans ce livre, comme *Layout width* et *Layout height* qui vous permettent de définir la taille de votre `LinearLayout`.

En passant la souris sur une des propriétés, une infobulle descriptive s'affiche.

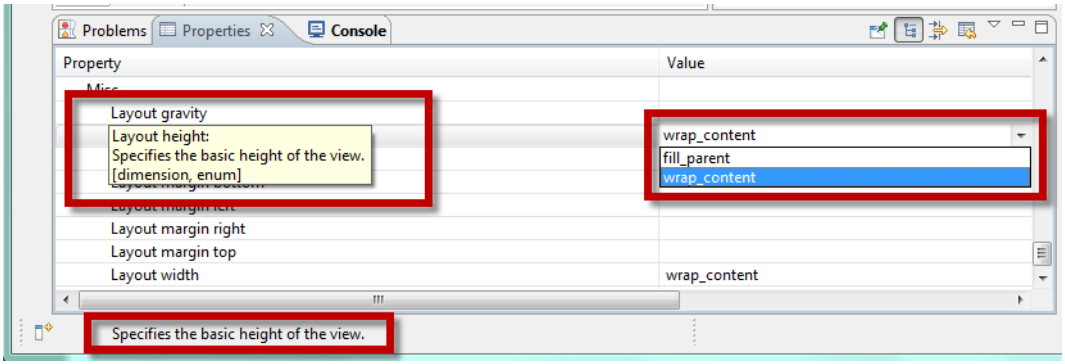


Figure A–14 Info bulle d'une propriété dans l'éditeur de propriétés d'une vue

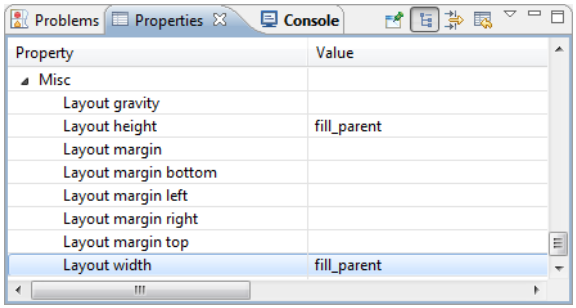
Si vous sélectionnez une des propriétés, la description de cette propriété s'affichera.

Modifier les propriétés des éléments graphiques

Certaines propriétés comme *Layout width* et *Layout height* peuvent avoir des choix prédéfinis et de ce fait proposer une liste de choix. Néanmoins dans ce cas précis, le choix n'est pas exclusif ; vous pourrez préciser une taille en pixel ou dpi. Nous vous recommandons de préférer les tailles en dip (dip) plutôt que px (pixel).

Mettez `fill_parent` sur les propriétés `Layout width` et `Layout height` pour occuper tout l'espace disponible sur l'écran.

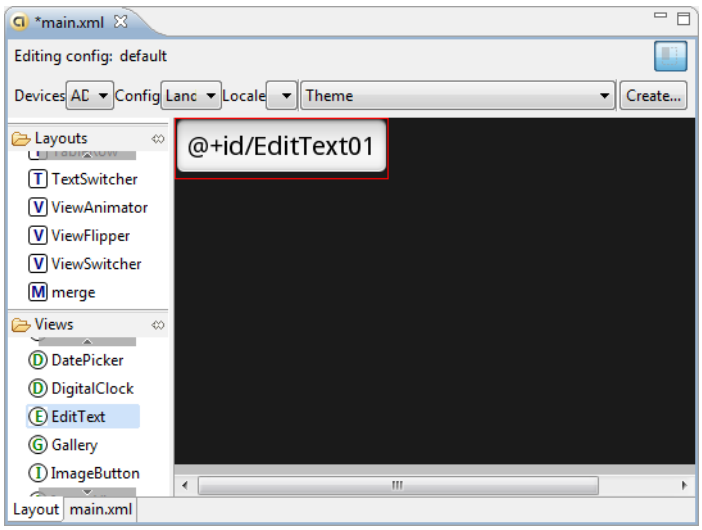
Figure A–15
Changer de propriétés dans l'éditeur de propriétés d'une vue



Créer une interface grâce au concepteur

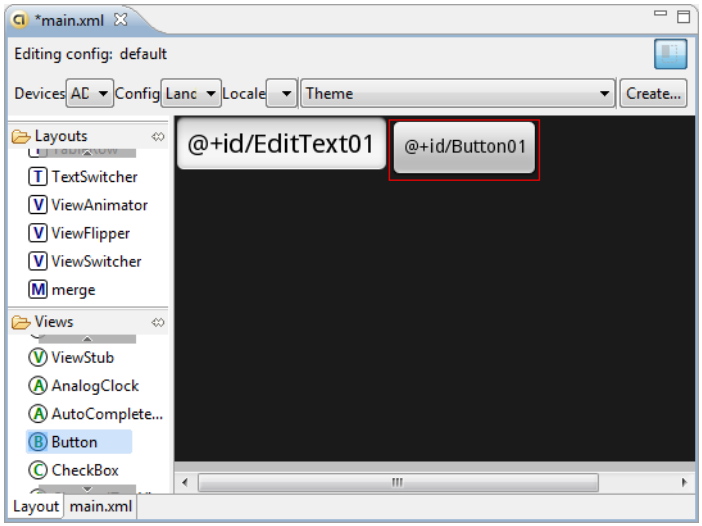
Pour ajouter un `EditText`, sélectionnez-le sur la gauche dans la liste des *Views* et faites le glisser sur la zone noire centrale.

Figure A–16
Ajout d'une vue de type éditeur de texte dans l'interface utilisateur



Ajoutez maintenant un bouton de la même façon. Sélectionnez *Button* dans la liste des vues et faite le glisser sur la zone centrale.

Figure A–17
Ajout d'une vue de type bouton dans l'éditeur d'interface utilisateur



Ces deux éléments s'affichent l'un à côté de l'autre. Si vous souhaitez les afficher l'un au dessus de l'autre, vous devrez changer l'orientation du `LinearLayout` parent de ces deux éléments.

Sélectionnez le fond noir ou bien sur la droite dans la fenêtre *Outline*, sélectionnez le `LinearLayout` qui se trouve à la base de notre arborescence.

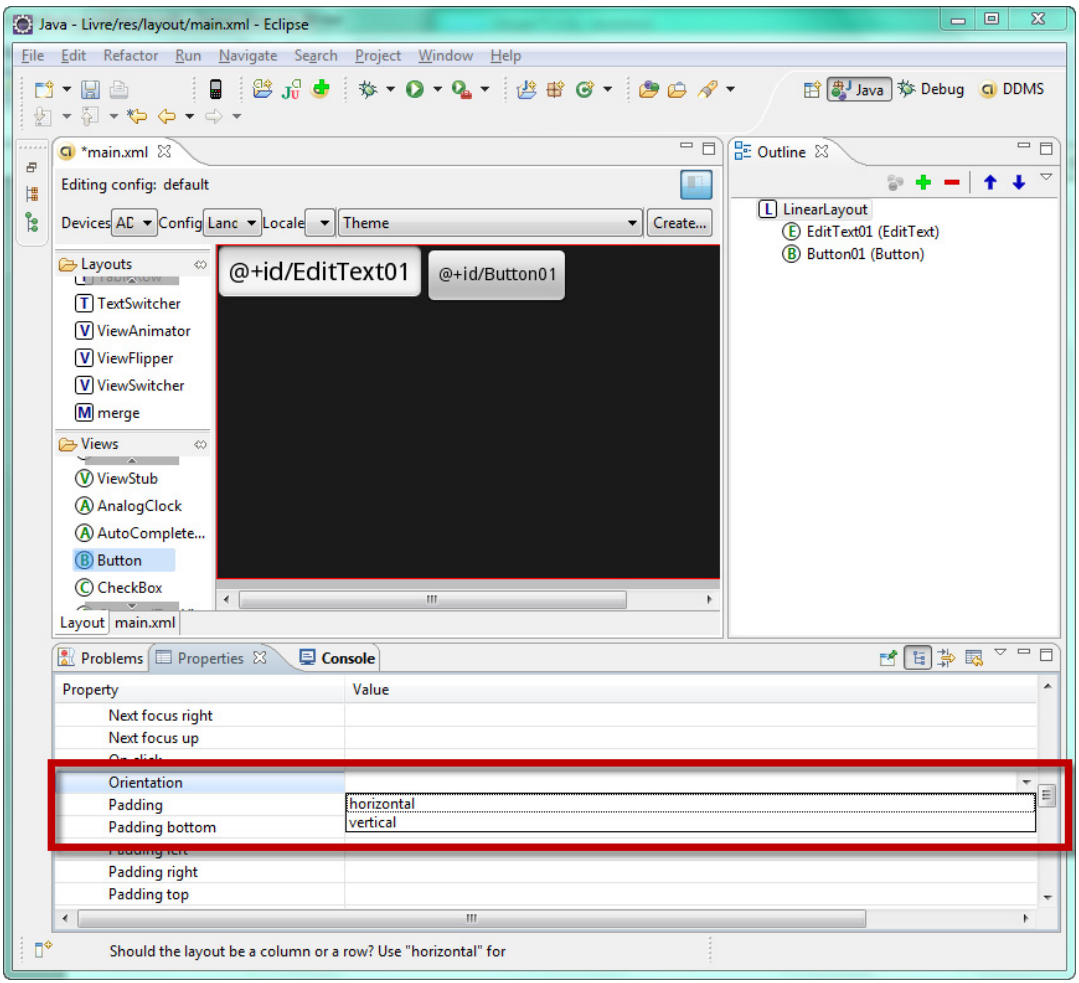


Figure A–18 Changement de l'orientation du LinearLayout

Deux choix sont alors à votre disposition : *Horizontal* et *Vertical*. Par défaut les `LinearLayout` dispose les éléments à l'horizontal, même si la propriété n'est pas spécifiée. Sélectionnez *Vertical* afin d'afficher les vues enfants les unes en dessous des autres. L'aperçu se met à jour immédiatement.

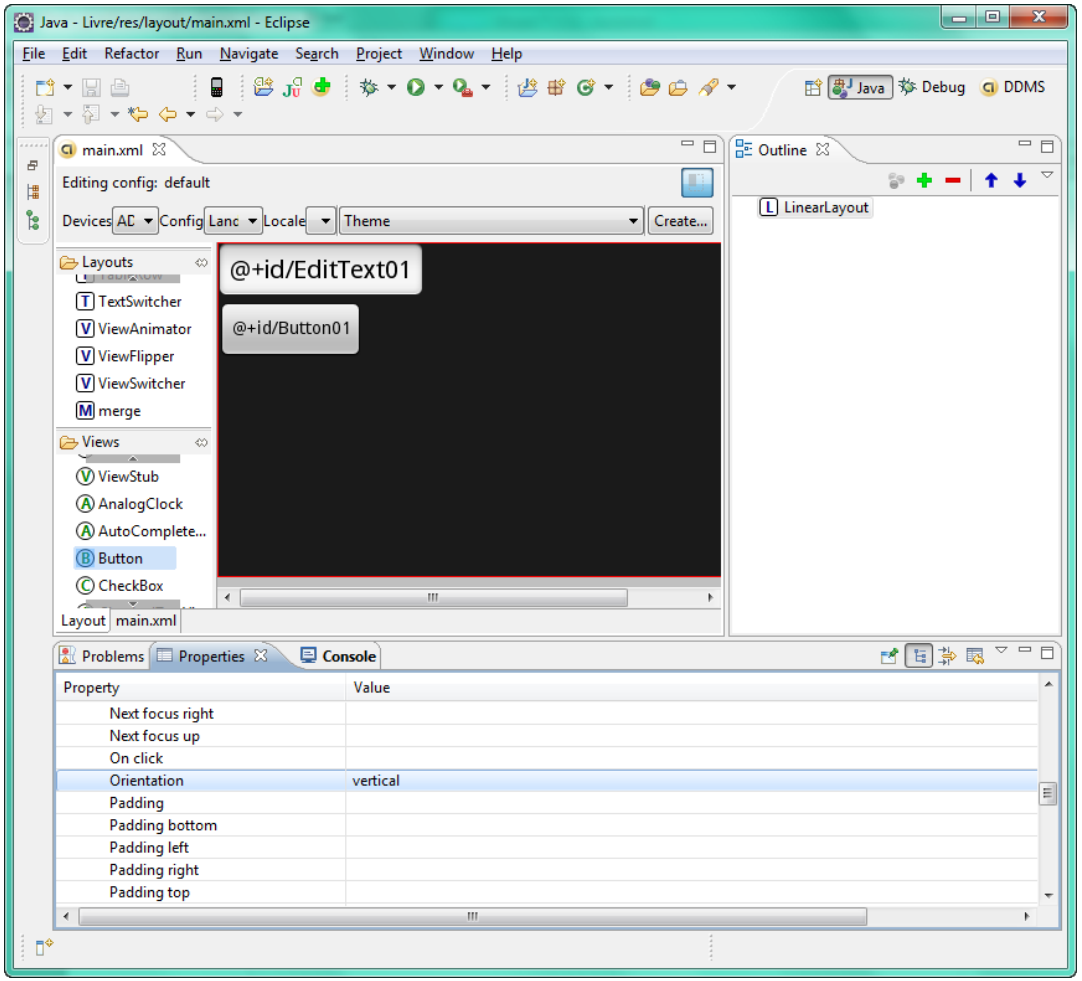


Figure A–19 Alignement vertical des vues du LinearLayout

Pour les interfaces complexes composées d'une multitude de `LinearLayout`, vous pourrez difficilement sélectionner les éléments directement dans l'aperçu au centre. Pour accéder aux éléments de l'interface, vous devrez alors utiliser la fenêtre *Outline* et naviguer dans l'arborescence pour sélectionner l'élément dont vous souhaitez changer les propriétés.

Vous pouvez à tout moment consulter et modifier manuellement le XML en mode texte en cliquant sur l'onglet en dessous de l'aperçu.

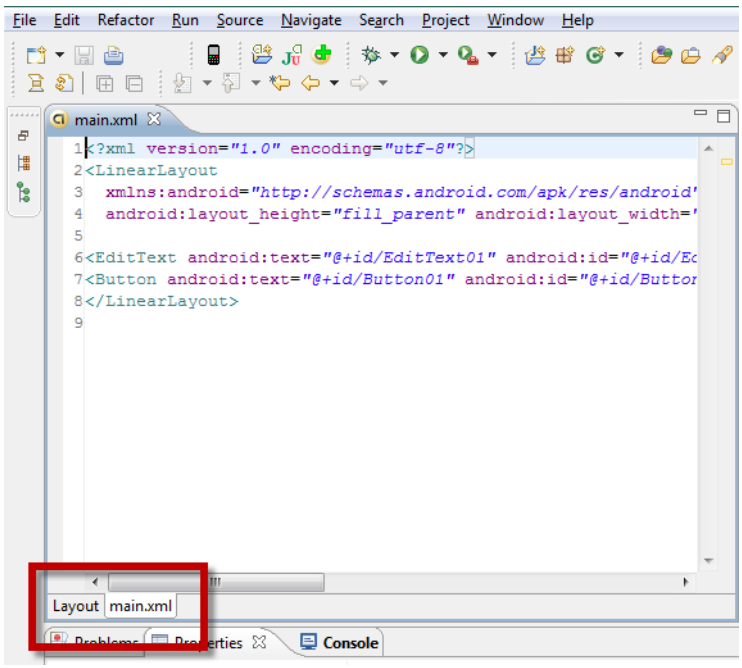


Figure A–20
Passer du concepteur graphique au source XML via les onglets

De cette façon, vous pourrez basculer à tout moment sur le concepteur graphique en cliquant sur *Layout* et voir l'aperçu des changements effectués dans le XML.

Vous voilà prêt à utiliser le concepteur d'interfaces graphique qui vous permettra de gagner encore plus en productivité grâce à ADT et Eclipse.

```
        <DatePicker
           android:id="@+id/DatePicker02"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content">
        </DatePicker>
     </LinearLayout>
  </ScrollView>
</LinearLayout>
```

Figure 3–17
Résultat du code 3-22 avec l'utilisation de la vue ScrollView

Comme vous pouvez le voir, une barre de défilement est visible sur la droite et est prise en compte pour l'ensemble du `LinearLayout` qui se trouve dans le `ScrollView`.

Différents outils permettant de réaliser une interface sont inclus dans le module ADT d'Eclipe. Ces outils sont présentés en annexe du présent ouvrage.

En résumé

La création d'interface est rendue simple et efficace sur la plate-forme Android. Ce chapitre n'est que la première partie sur ce sujet. Le chapitre 5, dédié également aux interfaces graphiques, vous montrera comment aller plus loin avec les vues et les gabarits, mais aussi comment créer vos propres composants.

L'interface d'une application est un élément essentiel : son succès reposera bien souvent sur une interface claire et intuitive. Sachez garder vos interfaces épurées et au plus proche du fonctionnement du système ou des applications standards. De cette façon l'utilisateur ne sera pas perturbé et aura l'impression d'utiliser votre application comme si elle faisait partie intégrante du système.