Création d'interfaces utilisateur avancées

Interfaces avancées

- *Android offre des possibilités en ergonomie pour capter **l'attention de vos utilisateurs.***

I. Ce cours prolonge le cours sur les interfaces utilisateur, en montrant comment créer des **interfaces beaucoup plus riches**, notamment en créant **des vues personnalisées**.

II. Les **adaptateurs** permettront la réalisation d'interfaces orientées données en associant une source de données à des contrôles d'interface, par exemple pour **créer une liste de contacts ou de pays.**

Interfaces avancées

III. Ce cours traitera également de **l'animation des vues**, pour plus de **dynamisme visuel**, ainsi que de la création de **gadgets.**

IV. L'internationalisation des applications est également un point clé de la réussite de vos applications auprès des utilisateurs – tant pour leur diffusion que leur adoption. C'est ce que permet Android, en facilitant **l'adaptation de l'application à la langue de l'utilisateur.**

I. Créer des composants d'interface personnalisés

Les widgets, ou vues standards

- Si les différentes **vues standards** ne suffisent plus à réaliser tous vos projets, si vous avez des besoins plus complexes en matière d'interface utilisateur ou avec un style graphique radicalement différent, vous n'aurez d'autre choix que de les **créer vous-même.**

Les widgets, ou vues standards

- Le mot **widget** désigne l'ensemble des vues standards incluses dans la plate-forme Android. Elles font partie du paquetage **android.widget**.

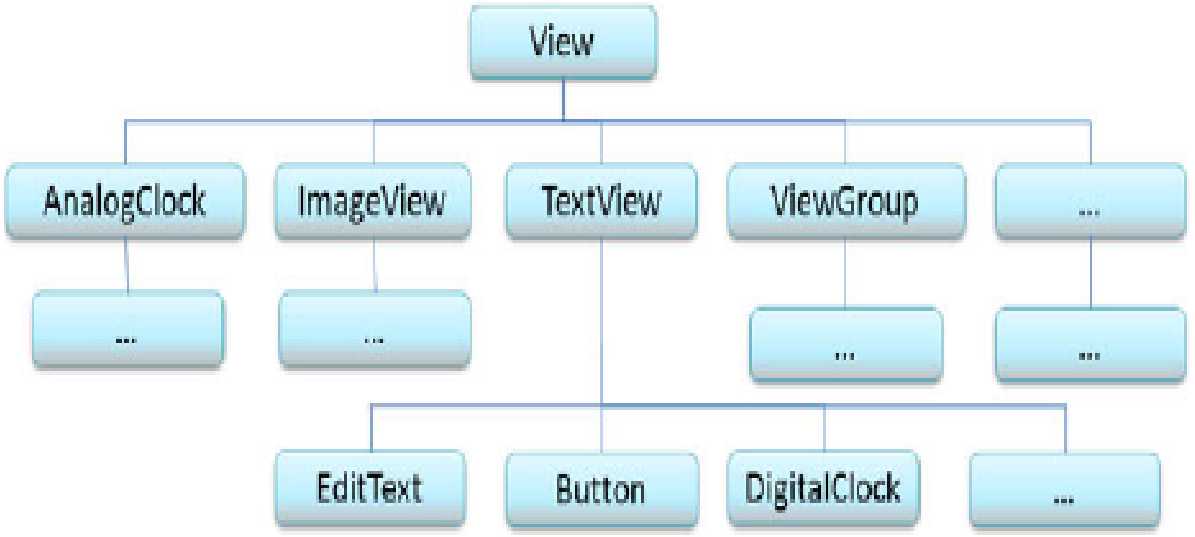
- Les vues héritant toutes de la classe **View**, chaque widget hérite aussi de cette classe.

- Ainsi l'élément **Button** hérite-t-il de **TextView**, qui lui-même hérite de classe **View**. L'élément **CheckBox**, quant à lui, hérite de la vue **Button.**

- Android tire profit **des méthodes de chaque vue** et de chaque widget pour former une plate-forme **paramétrable et modulaire.**

Les widgets, ou vues standards

Figure 5–1
Structure de l'héritage des contrôles



Les widgets, ou vues standards

Voici une liste non exhaustive des contrôles actuellement disponibles :

- `Button` : un simple bouton poussoir ;
- `CheckBox` : contrôle à deux états, coché et décoché ;
- `EditText` : boîte d'édition permettant de saisir du texte ;
- `TextView` : contrôle de base pour afficher un texte ;
- `ListView` : conteneur permettant d'afficher des données sous forme de liste ;
- `ProgressBar` : affiche une barre de progression ou une animation ;
- `RadioButton` : bouton à deux états s'utilisant en groupe.

- Vous pouvez décrire une interface de deux façons :
- **soit via une définition XML,**
- **Soit directement depuis le code en instanciant directement les objets adéquats.**

- La construction d'une interface au sein du code d'une activité, plutôt qu'en utilisant une définition XML, permet par exemple de créer des interfaces dynamiquement.

Créer un contrôle personnalisé

Code 5-1 : Instanciation d'un Widget au sein d'une activité

```
Button button = new Button(this);
button.setText("Cliquez ici");
button.setOnClickListener(this);
...
```

- Il peut arriver qu'une vue ne réponde pas totalement aux besoins du développeur.
- Dans ce cas il est possible, au même titre que les autres widgets d'hériter d'une vue existante pour former un contrôle personnalisé.

Créer un contrôle personnalisé

- Pour créer un contrôle personnalisé, **la première étape** est de déterminer quelle est **la classe la plus adaptée** pour former la base de votre contrôle.

- Dans l'exemple suivant, nous allons créer un contrôle affichant **une grille**.

Nous décidons d'étendre la vue de base **TextView** pour afficher cette grille :

Code 5-2 : Création d'un composant personnalisé : classe CustomView

```
import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Paint;
import android.util.AttributeSet;
import android.widget.TextView;

public class CustomView
   extends TextView {

   private Paint mPaint;
   private int mColor;
   private int mDivider;

   // Constructeurs
   public CustomView(Context context, AttributeSet attrs, int defStyle) {
      super(context, attrs, defStyle);
      init(context);
   }

   public CustomView(Context context, AttributeSet attrs) {
      super(context, attrs);
      init(context);
   }

   public CustomView(Context context) {
      super(context);
       init(context);
    }

    // Initialisation du contrôle
    private void init(Context context){
       mDivider = 10;
       mColor = 0xff808080;
       mPaint = new Paint();
       mPaint.setColor(mColor);
    }
```

```
    // Écriture de la propriété "GridColor"
    public void setGridColor(int color){
       mColor = color;
       mPaint.setColor(mColor);
       invalidate();
    }

    // Lecture de la propriété "GridColor"
    public int getGridColor(){
       return mColor;
    }

    // Écriture de la propriété "Divider"
    public void setDivider(int divider){
       mDivider = divider;
       invalidate();
    }

    // Lecture de la propriété "Divider"
    public int getDivider(){
       return mDivider;
    }

    @Override
    protected void onDraw(Canvas canvas) {
       // On récupère la taille de la vue
       int x = getWidth();
       int y = getHeight();
       int n = x / mDivider;
       // On dessine des lignes verticales
       for(int i = 0; i<x ; i+=n){
          canvas.drawLine(i, 0, i, y, mPaint);
       }
       n = y / mDivider;
       // On dessine des lignes horizontales
       for(int j = 0; j<y ; j+=n){
          canvas.drawLine(0, j, x, j, mPaint);
       }

        // On dessine en dessous du texte
        super.onDraw(canvas);
    }
}
```

Créer un contrôle personnalisé

- Vous noterez la déclaration des trois constructeurs de base d'une vue. Ils servent notamment à créer une vue à partir d'un gabarit d'interface au format XML. De cette façon, la création du contrôle et son utilisation dans un fichier de définition XML sont semblables à tout autre contrôle de la plate-forme Android.

- Vous remarquerez également que le code 5-2 contient quelques méthodes supplémentaires **set/get** permettant respectivement de mettre à jour et de récupérer les propriétés du contrôle. Les deux propriétés ajoutées sont **la couleur de grille** et **le nombre de cellules**.

- Pour mettre à jour l'affichage du contrôle nous emploierons conjointement les méthodes **invalidate** qui à son tour déclenchera un appel à la méthode **onDraw**. Le système nous fournira un **Canvas** sur lequel nous pourrons dessiner la grille de notre contrôle.

Créer un contrôle personnalisé

- Afin d'éprouver le contrôle personnalisé que vous venez de créer, construisez une activité et définissez le contrôle comme contenu principal. Vous pourrez utiliser l'écran tactile pour changer la taille des cellules de la grille :

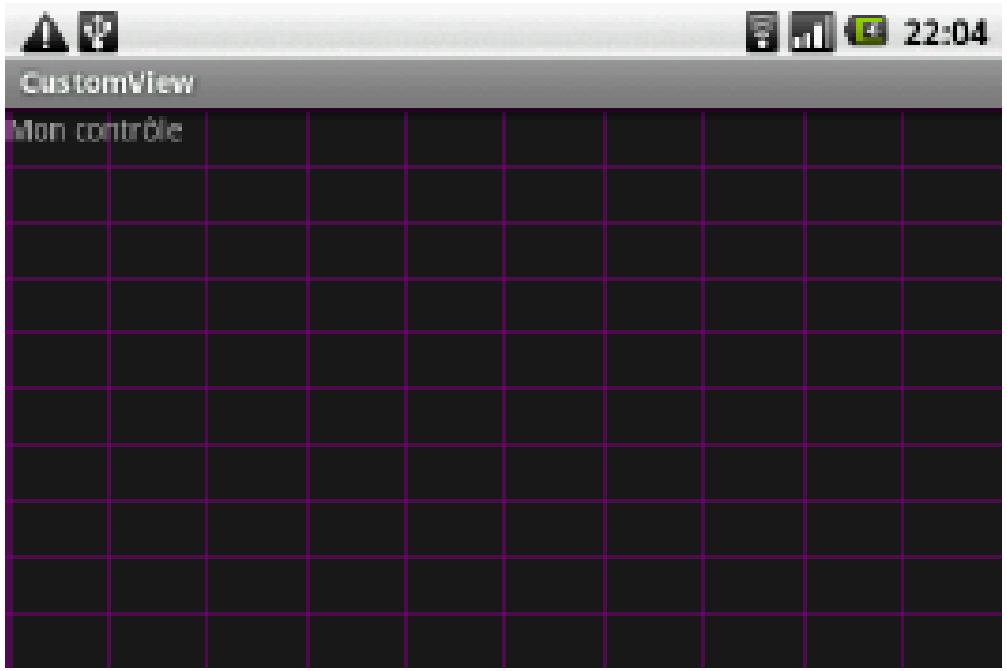
Crée alisé

Code 5-3 : Utilisation de notre contrôle dans du code

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

public class MainActivity
   extends Activity
{

   private CustomView view;

   @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        view = new CustomView(this);
        view.setText("Mon contrôle");
        view.setGridColor(0xff800080);
        setContentView(view);
    }

   @Override
   public boolean onTouchEvent(MotionEvent event) {
      switch(event.getAction()){
      case MotionEvent.ACTION_DOWN:
         view.setDivider(20);
         break;

        case MotionEvent.ACTION_UP:
           view.setDivider(10);
           break;
        }
        return super.onTouchEvent(event);
    }
}
```

Créer un contrôle personnalisé

Figure 5–2
Rendu de notre contrôle



Déclarer un contrôle personnalisé dans les définitions d'interface XML

- Créer une interface utilisateur directement dans le code d'une activité n'est pas une bonne pratique (même si cela peut se révéler nécessaire pour certains scénarios). Nous avons abordé l'utilisation des contrôles standards d'Android dans les définitions XML des interfaces des activités.

- Cette méthode a plusieurs avantages : séparation entre la logique fonctionnelle et la présentation, interface modifiable par un graphiste et non uniquement par un développeur, etc.

Déclarer un contrôle personnalisé dans les définitions d'interface XML

- Contrairement à l'utilisation des contrôles Android standards, l'utilisation d'un contrôle personnalisé dans un fichier de description d'interface XML nécessite un peu de préparation en amont.

- En effet, pour pouvoir renseigner les propriétés d'un contrôle personnalisé d'une interface de manière déclarative, vous devez tout d'abord exposer ses attributs dans un fichier **res/attrs.xml :**

Déclarer un contrôle personnalisé dans les définitions d'interface XML

Code 5-4 : Fichier attrs.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <declare-styleable name="CustomView">
      <attr name="GridColor" format="color"/>
      <attr name="Divider">
         <enum name="big" value="3" />
         <enum name="normal" value="10" />
         <enum name="small" value="20" />


      </attr>
   </declare-styleable>
</resources>
```

- Une fois les propriétés déclarées dans le fichier res/attrs.xml, celles-ci seront utilisables dans un fichier XML de description d'interface en ajoutant le nom du paquetage dans l'espace de nom comme indiqué ci-dessous :

Déclarer un contrôle personnalisé dans les définitions d'interface XML

Code 5-5 : Fichier main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res/
              ➥ com.eyrolles.android.customview"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
   <com.eyrolles.android.customview.CustomView
      android:layout_height="fill_parent"
      android:layout_width="fill_parent"
      android:id="@+id/CustomViewId"
      android:text="Cliquez ici"
      app:GridColor="#ffff0000"
      app:Divider="big"
      />
</LinearLayout>
```

Déclarer un contrôle personnalisé dans les définitions d'interface XML

- Après avoir défini les propriétés dans un fichier d'attributs et avant de pouvoir profiter pleinement de votre contrôle personnalisé en mode déclaratif,

- il faut revoir les constructeurs contenus dans le code du contrôle personnalisé. En effet, chaque propriété personnalisée sera passée en paramètre du constructeur sous la forme d'un tableau d'objets.

- Vous devez donc manipuler ce tableau pour récupérer les valeurs de ces propriétés et les appliquer au contrôle personnalisé :

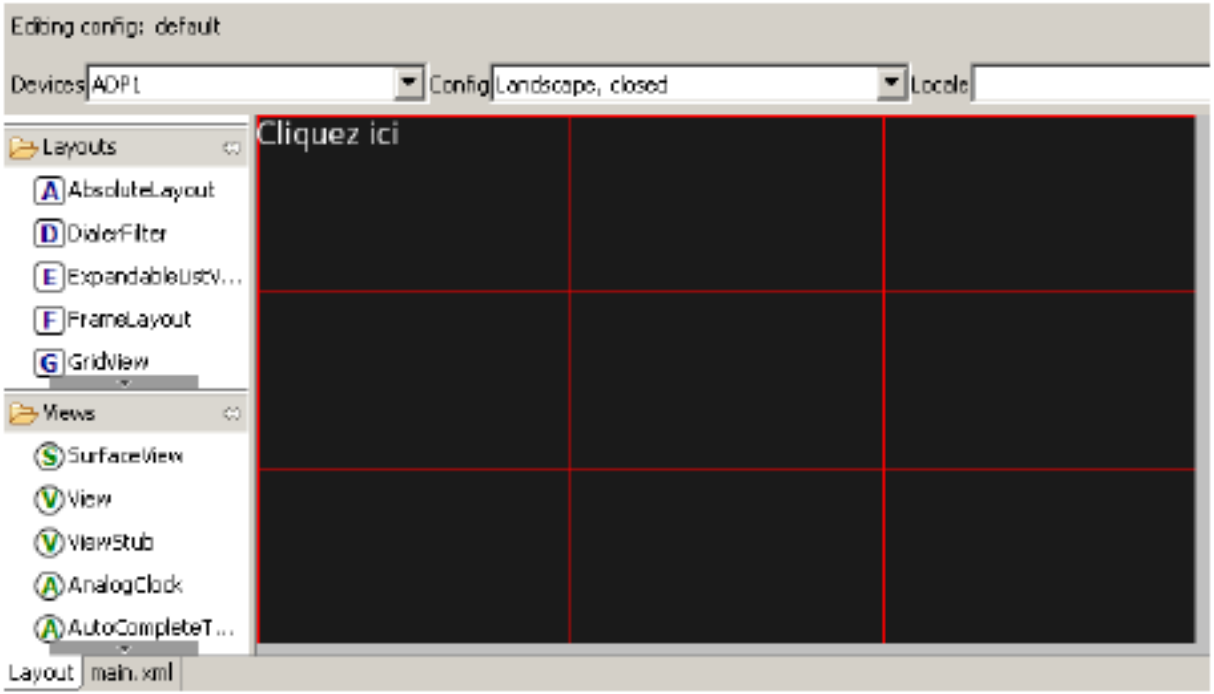
Code 5-6 : Chargement des propriétés depuis le fichier xml

```
public CustomView(Context context, AttributeSet attrs, int defStyle) {
   super(context, attrs, defStyle);
   mDivider = 10;
   mColor  = 0xff808080;
   mPaint   = new Paint();
   loadAttrFromXml(context, attrs, defStyle);
   mPaint.setColor(mColor);
}
```

```
public CustomView(Context context, AttributeSet attrs) {
   this(context, attrs, 0);
}

public CustomView(Context context) {
   this(context,null);
}

// Charge les propriétés à partir du fichier xml
private void loadAttrFromXml(Context context, AttributeSet attrs, int defStyle){
   TypedArray a = context.obtainStyledAttributes(attrs, R.styleable.CustomView,
                                                 defStyle, 0);
   int n = a.getIndexCount();
   for (int i = 0; i < n; i++) {
      int attr = a.getIndex(i);
      switch (attr) {
      case R.styleable.CustomView_Divider:
         mDivider = a.getInt(attr, mDivider);
         break;
      case R.styleable.CustomView_GridColor:
         mColor = a.getColor(attr, mColor);
         break;
      }
   }
   a.recycle();
}
```

Déclarer un contrôle personnalisé dans les définitions d'interface XML

Figure 5–3
Rendu du contrôle dans le concepteur d'interface d'ADT(voir annexes).



II. Les adaptateurs pour accéder aux données de l'interface.

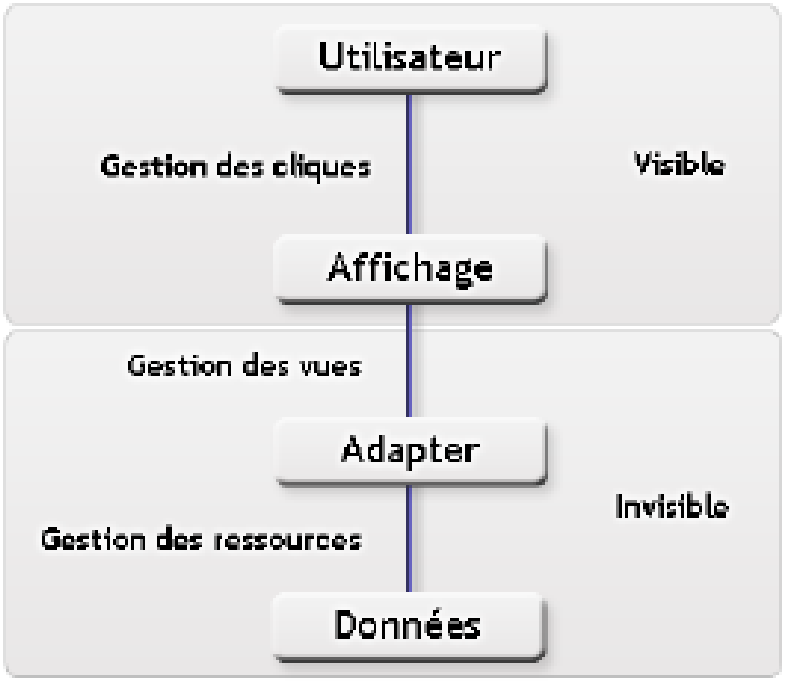
Les adaptateurs

- Les adaptateurs servent à gérer les sources de données qui seront notamment affichées dans les différentes listes de l'interface utilisateur.

- Ils réalisent la liaison entre vos sources de données (un simple tableau de chaînes de caractères, une base de données, un fournisseur de contenu, etc.) et les contrôles de votre interface utilisateur.

Les adaptateurs

Figure 5–4
L'objet Adapter dans son environnement



Les adaptateurs

- Prenons comme exemple un tableau de chaînes de caractères à afficher sous la forme d'une liste dans une activité.

- Commençons par déclarer **le tableau** puis un **ArrayAdapter.** Pour simplifier la gestion des vues, nous appliquerons cet adaptateur dans une activité de type **ListActivity.**

- Ce type d'activité embarque en effet une ListView et des méthodes simplifiant la gestion des adaptateurs.

Les adaptateurs

Code 5-8 : Création d'un adaptateur

```
String[] tableau = new String[]{
    "Un" ,"Deux" ,"Trois" ,"Quatre"
   ,"Cinq" ,"Six" ,"Sept" ,"Huit"
   ,"Neuf" ,"Dix"};

ArrayAdapter<String> adapter =
new ArrayAdapter<String>(
   this
   ,android.R.layout.simple_list_item_1, tableau);
this.setListAdapter(adapter);
```

Figure 5–5
Exemple de liste

Les adaptateurs

Le constructeur utilisé pour instancier un nouvel objet ArrayAdapter prend trois paramètres :

- **Context context :** grâce à ce paramètre l'adaptateur sera capable de créer seul les objets nécessaires à la transcription des fichiers XML en ressources ;

- **int textViewResourceId** : l'identifiant du fichier XML à utiliser comme modèle pour la liste (en général contenu dans le dossier /res/layout du projet). Le modèle est la vue ou le groupe de vue qui sera utilisé par chaque entrée pour s'afficher dans la liste associée avec l'ArrayAdapter ;

- **String[] objects :** les données sous forme d'un tableau de chaînes de caractères.

III. Animation des vues

Animation des vues

- Les animations font aujourd'hui partie de notre quotidien que ce soit sur le Web (via les technologies XHTML/CSS/JavaScript, Adobe Flash/Flex, Microsoft Silverlight, etc.) ou dans les applications bureautiques. Face à cela, la plate-forme Android n'est pas en reste puisqu'elle propose deux mécanismes pour animer vos interfaces :

• **les animations image par image** (dédiées comme son nom l'indique, aux images) ;

• **les animations de positionnement** (dédiées à l'animation des vues).

Animation des vues

- La plate-forme Android permet là encore de définir des animations des deux façons habituelles : par le code ou via l'utilisation d'un fichier XML.

- Définir les animations dans des fichiers de ressources externes offre deux atouts : d'une part, une personne de l'art tel qu'un designer peut modifier le fichier indépendamment du code et d'autre part, Android peut alors appliquer automatiquement l'animation la plus appropriée selon le matériel (taille et orientation de l'écran).

Les animations d'interpolation

- Les animations d'interpolation permettent d'effectuer une rotation et/ou de changer la position, la taille, l'opacité d'une vue.
- Avec ce type d'animation, la plate-forme s'occupera pour vous de définir les étapes intermédiaires par interpolation en fonction du temps et des états d'origine et de fin que vous aurez spécifiés.

- Ce type d'animation est souvent utilisé pour réaliser des transitions entre activités ou pour mettre en exergue un élément de l'interface à l'utilisateur (saisie incorrecte, options à la disposition de l'utilisateur, etc).

Les animations d'interpolation

➢ Les interpolations possibles sont les suivantes :

• **opacité (Alpha)** : permet de jouer sur la transparence/opacité de la vue ;

• **échelle (Scale)** : permet de spécifier l'agrandissement/réduction sur les axes et X et Y à appliquer à la vue ;

•**translation (Translate) :** permet de spécifier la translation/déplacement à effectuer par la vue ;

• **rotation (Rotate) :** permet d'effectuer une rotation selon un angle en degré et un point de pivot.

Animer par le code

- Chaque type d'animation par interpolation possède une sous-classe dérivant de Animation : **ScaleAnimation**, **AlphaAnimation,** **TranslateAnimation** et **RotateAnimation.** Chacune de ces animations peut être créée, paramétrée, associée et exécutée directement depuis le code :

Code 5-31 : Animer une vue directement depuis le code

```
TextView vueTexte = ...;

// Animation d'une alpha
Animation animationAlpha = new AlphaAnimation(0.0f, 1.0f);
animationAlpha.setDuration(100);
vueText.startAnimation(animationAlpha);
```

```
// Animation de translation d'une vue
TranslateAnimation animationTranslation = new TranslateAnimation(
   Animation.RELATIVE_TO_SELF, 0.0f, Animation.RELATIVE_TO_SELF, 0.0f,
Animation.RELATIVE_TO_SELF, -1.0f, Animation.RELATIVE_TO_SELF, 0.0f);
animationTranslation.setDuration(500);
vueText.startAnimation(animationTranslation);
```

- Chaque type d'animation possède des constructeurs propres prenant des paramètres qui leur sont spécifiques. D'une manière globale, la plupart des méthodes sont communes.

- Par exemple, pour spécifier la durée de l'animation, utilisez la méthode **setDuration** spécifiant le nombre de millisecondes en paramètre.

- Enfin, pour démarrer une animation, appelez la méthode **startAnimation** de la vue cible et spécifiez l'objet Animation que vous souhaitez utiliser.
- Nous verrons plus loin comment créer une série d'animation par le code, c'est-à-dire un enchaînement d'animations élémentaires.

Animer par le XML

- Le tableau suivant présente les attributs XML des paramètres de chaque type d'animation(vous remarquerez que ces propriétés XML retrouvent toutes un équivalent sous la forme de méthodes d'accès **get/set** dans les classes dérivées d'Animation).

Tableau 5–3 Les propriétés des animations par interpolation

Type d'animation	Propriété	Description
ScaleAnimation	`fromXScale`	`float`
	`toXScale`	`float`
	`fromYScale`	`float`
	`toYScale`	`float`
	`pivotX / pivotY`	Valeur en pourcentage du centre de rotation, ou pivot, relatif à la vue sur l'axe des ordonnées. Chaîne dont la valeur est comprise entre '0 %' et '100 %'.
AlphaAnimation	`fromAlpha`	Valeur de l'opacité de départ. Valeur comprise entre 0.0 et 1.0 (0.0 est transparent).
	`toAlpha`	Valeur de l'opacité d'arrivée. Valeur comprise entre 0.0 et 1.0 (0.0 est transparent).
TranslateAnimation	`fromX`	`float`
	`toX`	`float`

Type d'animation	Propriété	Description
RotateAnimation	`fromDegrees`	Orientation de départ en degrés. Valeur comprise entre 0 et 360.
	`toDegrees`	Orientation d'arrivée en degrés. Valeur comprise entre 0 et 360.
	`PivotX / pivotY`	Valeur en pourcentage du centre de rotation, ou pivot, relatif à la vue sur l'axe des abscisses. Chaine dont la valeur est comprise entre '0 %' et '100 %'.

- Les fichiers XML de description d'animation doivent être placés dans le répertoire **res/anim** afin que le système génère un identifiant qui sera ensuite utilisé pour y faire référence dans le code.

- L'extrait de fichier XML suivant présente la définition d'animations de chaque type:

Code 5-32 : Ensemble d'animations

```
// Une animation sur l'opacité de la vue.
<alpha
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:interpolator="@android:anim/accelerate_interpolator"
   android:fromAlpha="0.0"
   android:toAlpha="1.0"
   android:duration="1000" />

// Animation de translation
<translate android:fromXDelta="100%p"
   android:toXDelta="0"
   android:duration="1000"/>

// Animation d'échelle
<scale android:fromXScale="0.0"
   android:toXScale="1.0"
   android:fromYScale="0.0"
   android:toYScale="1.0"
   android:pivotX="50%"
   android:pivotY="50%"
   android:duration="1000" >

// Animation de rotation
<rotate android:fromDegrees="0"
   android:toDegrees="360"
   android:pivotX="50%"
   android:pivotY="50%"
   android:duration="1000" >
```

- Une fois votre animation décrite, vous devrez charger et associer l'animation à une vue – ou un groupe de vues – afin d'animer celle-ci. Le chargement s'effectue à l'aide de la méthode statique **loadAnimation** de la classe **AnimationUtils** en spécifiant l'identifiant de la ressource d'animation.

- Pour terminer, associez l'animation ainsi chargée à la vue en appelant sa méthode **setAnimation** et en spécifiant l'objet Animation précédemment chargé à partir de la définition XML de l'animation.

Code 5-33 : Charger et exécuter une animation décrite en XML

```
Animation animation = AnimationUtils.loadAnimation(this, R.anim.animation1);
animation.setRepeatMode(Animation.RESTART);
animation.setRepeatCount(Animation.INFINITE);
maVueAAnimer.startAnimation(animation);
```

Créer une série d'animations

- Vous pouvez créer une série d'animations en utilisant la classe **AnimationSet** ou l'attribut set dans la définition XML. Vous allez pouvoir exécuter ces animations soit simultanément soit en décalé dans le temps.

- Pour vous aider à planifier l'exécution des différentes animations vous pouvez utiliser les attributs XML (ou leur méthode équivalente au niveau des classes) communs à tous les types d'animations par interpolation.

Créer une série d'animations

Tableau 5–4 Propriétés communes des animations par interpolation pour l'exécution planifiée

Nom de la propriété	Description
`duration`	La durée en millisecondes de l'exécution totale de l'animation.
`fillAfter`	Une valeur booléenne pour spécifier si la transformation doit s'effectuer avant l'animation.
`fillBefore`	Une valeur booléenne pour spécifier si la transformation doit s'effectuer après l'animation.
`interpolator`	L'interpolateur utilisé pour gérer l'animation (cf. ci-après dans ce chapitre).
`startOffset`	Le délai en millisecondes avant le démarrage de l'animation. La référence à l'origine de ce délai est le début de la série d'animation. Si vous ne spécifiez pas cette propriété, l'animation sera exécutée immédiatement, de la même façon que vous auriez spécifié la valeur 0.
`repeatCount`	Définit le nombre de fois où l'animation doit être jouée.
`repeatMode`	Définit le mode de répétitions si le nombre de répétitions est supérieur à 0 ou infini. Les valeurs possibles sont `restart` (valeur 1) pour recommencer l'animation, et `reverse` (valeur 2) pour jouer l'animation à l'envers.

Code 5-34 : Réalisation d'une série d'animation en code Java

```
// Création d'une série d'animations. La valeur true spécifie que
// chaque animation utilisera l'interpolateur de l'AnimationSet.
// Si false, chaque animation utilisera l'interpolateur
// définit dans chacune d'elle.
AnimationSet serieAnimations = new AnimationSet(true);

AlphaAnimation animationAlpha = new AlphaAnimation(0.0f, 1.0f);
animationAlpha.setDuration(500);
// Ajout de l'animation à la série d'animation
serieAnimation.addAnimation(animation);

TranslateAnimation animationTranslation = new TranslateAnimation(
   Animation.RELATIVE_TO_SELF, 0.0f,
   Animation.RELATIVE_TO_SELF, 0.0f,
   Animation.RELATIVE_TO_SELF, -1.0f,
   Animation.RELATIVE_TO_SELF, 0.0f
);
animationTranslation.setDuration(300);
animationTranslation.setStartOffset(500);
// Ajout de l'animation à la série d'animations
serieAnimation.addAnimation(animation);

// Exécution de la série d'animations
vueAAnimer.startAnimation(serieAnimation);
```

Code 5-35 : Réalisation d'une série d'animation en XML

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
   android:interpolator="@android:anim/accelerate_interpolator">
   <alpha
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:interpolator="@android:anim/accelerate_interpolator"
      android:fromAlpha="0.0" android:toAlpha="1.0" android:duration="500" />
   <translate android:fromXDelta="0"
      android:toXDelta="100%p"
      android:startOffset="500"
      android:duration="300" />
</set>
```

Animer un groupe de vues

- Animer une vue permet de créer des effets très saisissants mais limités à leur périmètre.

- Si vous souhaitez animer la page complète, par exemple pour réaliser une page qui se tourne ou un effet d'apparition digne des transitions des meilleurs logiciels de présentation, Android permet d'appliquer une transformation à une mise en page complète.

- Pour animer un groupe de vues (toute mise en page dérivant de la classe **ViewGroup**) vous devrez spécifier l'animation à la méthode **setLayoutAnimation** du groupe de vues.

Animer un groupe de vues

Code 5-36 : Chargement et exécution d'une animation d'un groupe de vues

```
AnimationSet serieAnimations = new AnimationSet(true);
...
serieAnimation.addAnimation(animation);

TranslateAnimation animationTranslation = ...
...
serieAnimation.addAnimation(animation);

// Crée un gestionnaire d'animation qui appliquera l'animation à
// l'ensemble des vues avec un délai - ici nul - entre chaque vue.
LayoutAnimationController controller =
                    new LayoutAnimationController(serieAnimation, 0.f);
// Spécifie l'animation de groupe de vues à utiliser.
groupeVues.setLayoutAnimation(controller);
// Démarre l'animation du groupe de vues.
groupeVues.startLayoutAnimation();
```

Les animations image par image

- Les animations image par image permettent de spécifier une séquence d'images qui seront affichées les unes à la suite des autres selon un délai spécifié, un peu à la manière d'un dessin animé.

- Comme les animations par interpolation, une séquence d'animation image par image peut être définie soit dans le code soit dans une ressource XML externe. La séquence d'images est décrite dans une liste d'éléments item contenant dans un élément parent animation-list. Pour chaque élément, vous spécifiez l'identifiant de la ressource et sa durée d'affichage :

Les animations image par image

Code 5-41 : Séquence d'une animation image par image

```
<?xml version="1.0" encoding="utf-8"?>
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
   <!-- Ce paramètre est utilisé pour spécifier si l'animation doit être
jouée une ou plusieurs fois -->
   android:oneshot="false">
   <!-- Chaque image est déclarée dans un élément item -->
   <itemandroid:drawable="@drawable/android1" android:duration="500" />
   <item android:drawable="@drawable/android2" android:duration="500" />
   <item android:drawable="@drawable/android3" android:duration="500" />
</animation-list>
```

Les animations image par image

- Au niveau du code vous devez spécifier l'animation à la vue via sa méthode setBackgroundResource en précisant l'identifiant de l'animation séquentielle d'images.

- Pour démarrer l'animation, récupérez l'objet AnimationDrawable en appelant la méthode getBackground de la vue, puis appelez la méthode start de l'animation.

À SAVOIR **Ne pas démarrer l'animation dans la méthode onCreate**

Attention à l'endroit où vous appelez la méthode `start`. En effet, si vous appelez cette méthode pour lancer votre animation depuis la méthode `onCreate`, l'animation ne démarrera pas !

Les animations image par image

Code 5-42 : Animation d'une vue avec une animation image par image

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

   ImageView image = (ImageView) findViewById(R.id.image);
   // Spécifie l'animation comme fond de l'image
   image.setBackgroundResource(R.drawable.animation_images);
   image.setOnClickListener(new View.OnClickListener() {

      @Override
      public void onClick(View v) {
         ImageView image = (ImageView) AnimationImageParImage.this
            .findViewById(R.id.image);
         // Récupère l'animation à animer
         AnimationDrawable animation = (AnimationDrawable) image
            .getBackground();
         // Démarre l'animation
         animation.start();
      }
   });
}
```

Les animations image par image

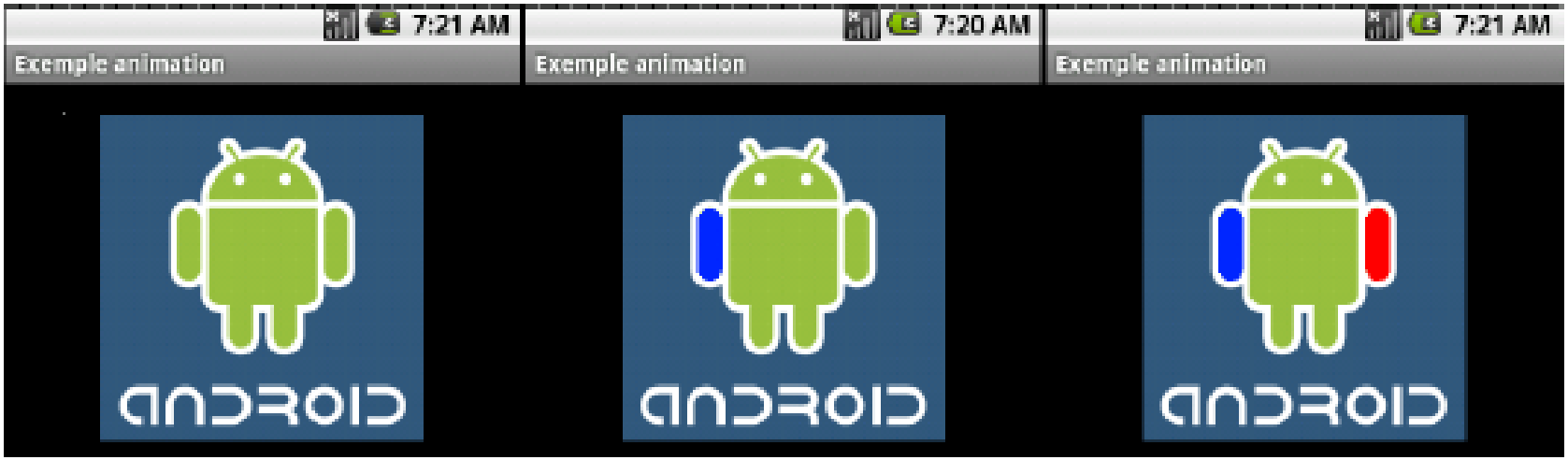


Figure 5–25 Résultat de l'animation du code 5-42

Les AppWidgets

- Hormis les icônes des applications, le bureau (ou écran d'accueil) peut également contenir ce que l'on appelle les ***AppWidgets***. Leur utilisation consiste à exporter votre application sous la forme d'un contrôle personnalisable disponible directement sur le bureau de l'utilisateur.

- Les gadgets sont une très bonne manière d'étendre vos applications. Ils offrent un accès direct aux fonctionnalités de celles-ci, tout en gardant un format discret. Nous pouvons les comparer à de petits programmes embarqués, tels que des extensions, comme le proposent aujourd'hui les systèmes d'exploitation de dernière génération.

Les AppWidgets

- Les avantages sont nombreux :

- il est inutile de lancer l'application à proprement parler ;
- l'application est toujours disponible sur le bureau ;
- plusieurs gadgets peuvent se partager un même écran ;
- les ressources de fonctionnement ne sont allouées que lorsque cela est nécessaire ;
- chaque gadget peut sauvegarder ses propres paramètres.

- À titre d'exemple l'horloge analogique fait partie des gadgets, ainsi que le lecteur média, l'album photo et le panneau de contrôle comme le montre l'écran ci-joint.

Les AppWidgets

Figure 5–26
Exemples de gadgets
(source Google)

Création d'un gadget

- Pour concevoir un gadget nous allons avoir besoin de réaliser plusieurs tâches :

1. créer un objet hérité de la classe AppWidgetProvider ;

2. définir l'aspect visuel du gadget dans un autre fichier XML ;

3.définir les paramètres du AppWidgetProvider dans un fichier XML

4 .ajouter une activité pour personnaliser le gadget ;

5 .déclarer le gadget dans le fichier de configuration de l'application.

IV. Internationalisation des applications

Internationalisation des applications

- Pour que vos applications rencontrent le plus grand succès, il faut les rendre compréhensibles par le plus grand nombre d'utilisateurs, et donc les traduire. Les auteurs de la plate-forme Android ont bien pris en compte cette nécessité et offrent un mécanisme assez facile à prendre en main.

Étape 0 : créer une application à manipuler

Code 5-28 : Mise en page d'une application avec une image et une case à cocher

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout (1) xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<ImageButton (2)
    android:id="@+id/android_button"
    android:layout_width="100dip"
    android:layout_height="wrap_content"
    android:src="@drawable/android" />

<CheckBox android:id="@+id/checkbox" (3)
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/checkboxtext" />
</LinearLayout>
```

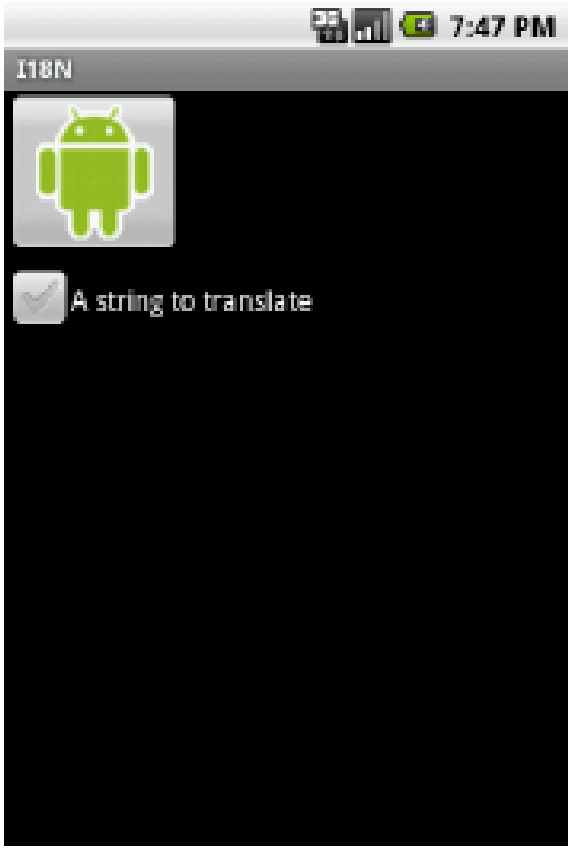
Cette mise en page est simple avec une disposition linéaire (1), une image du bonhomme Android (2) et une case à cocher (3) contenant un texte à traduire.

Il faut également renseigner le fichier ressource qui contient les chaînes de caractères.

Code 5-29 : Fichier strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">I18N</string>
     <string name="checkboxtext">A string to translate</string>
</resources>
```

Figure 5–19
Capture d'écran de l'application sans traduction



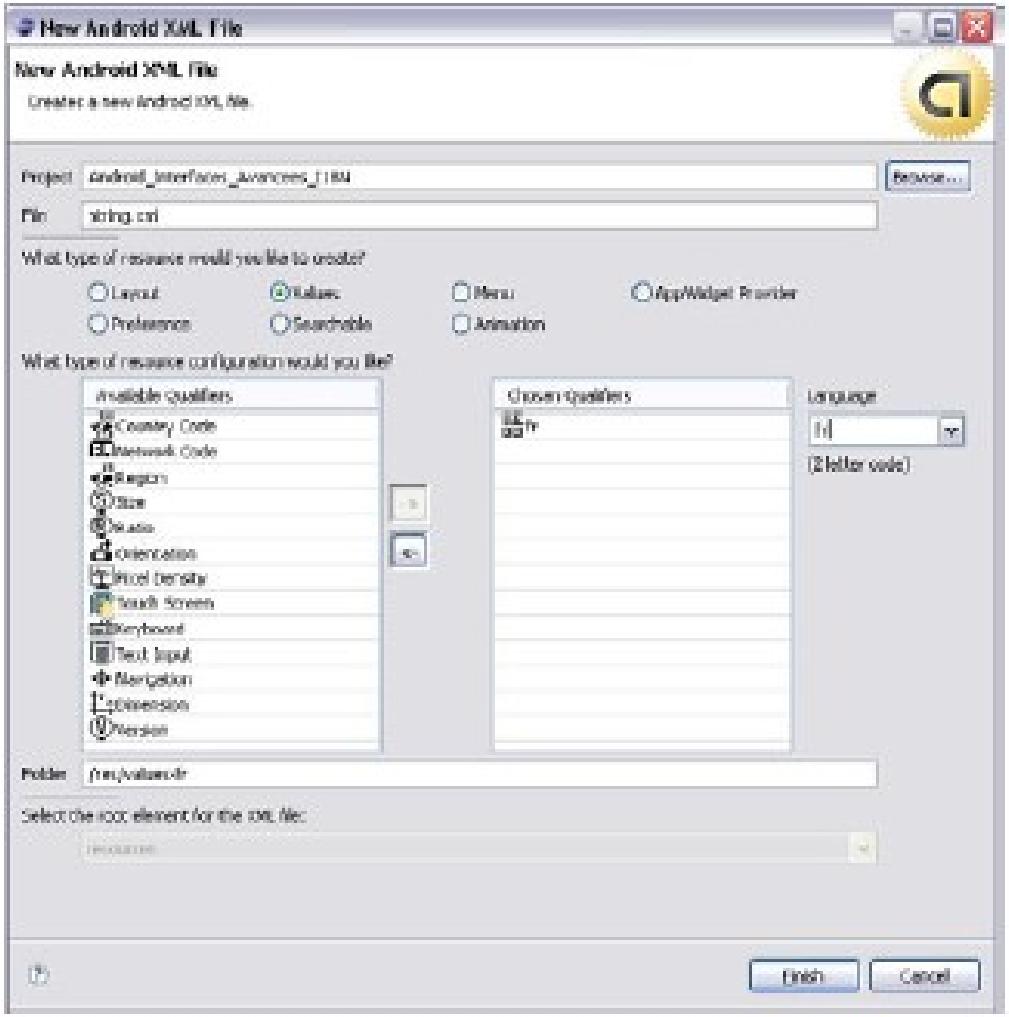
Internationaliser des chaînes de caractères

Maintenant, commençons à personnaliser l'application en fonction de la langue de l'appareil. Cliquez sur l'icône de l'assistant de création de fichiers XML ou parcourez les menus *File* > *New* > *Android XML File*.

Figure 5–20
Icône de l'assistant de création de fichiers XML

Grâce au formulaire qui apparaît, sélectionnez votre projet puis entrez un nom de fichier à traduire (`strings.xml` si vous n'avez pas modifié le projet). Spécifiez ensuite *Values* comme type de ressource à créer. Sélectionnez *Language* dans la liste gauche et appuyez sur la flèche. Ensuite dans la boîte *Language* qui doit apparaître, à droite de la fenêtre, saisissez le code à deux lettres qui correspond au langage que vous souhaitez utiliser. Prenons *fr* comme exemple.

Figure 5–21
Ajout d'un fichier de traduction en français



À CONSULTER **Liste des langues et des pays**

Vous pouvez consulter la liste des codes de langues et de pays aux adresses suivantes :

- http://www.loc.gov/standards/iso639-2/php/code_list.php
- http://www.iso.org/iso/country_codes/iso_3166_code_lists/english_country_names_and_code_elements.htm

Enfin, appuyez sur *Finish*. Votre projet devrait contenir un répertoire supplémentaire : `/res/values-fr`.

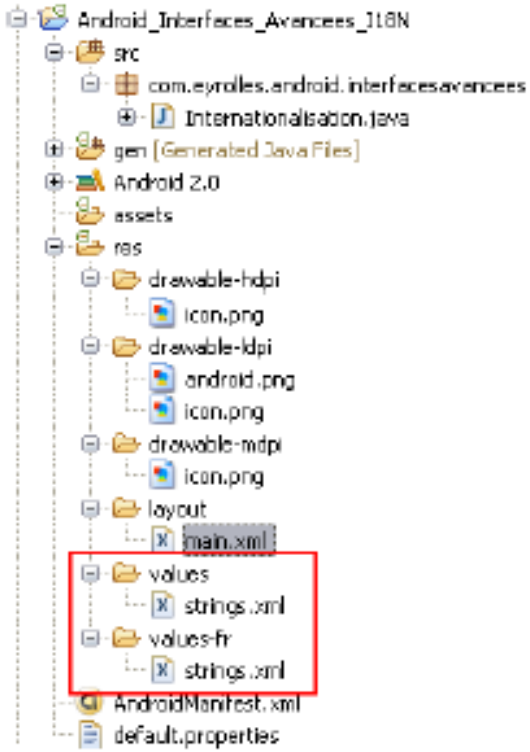


Figure 5–22
Impact sur le projet

Éditez le fichier `strings.xml` de ce nouveau répertoire et placez-y les chaînes traduites en français.

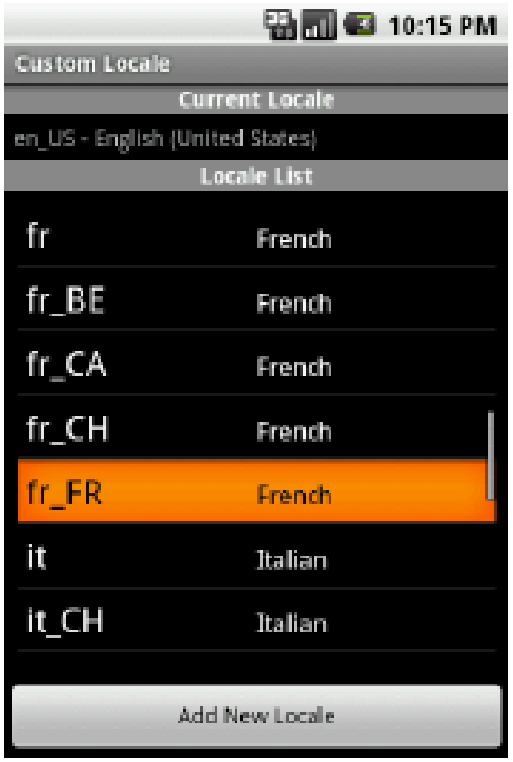
Code 5-30 : Chaînes traduites en français dans le répertoire res/values-fr

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">I18N France</string>
    <string name="checkboxtext">Une traduction pour la France</string>
</resources>
```

Ces modifications effectuées, relancez votre émulateur et vous devriez constater... que rien n'a changé ! En effet, cela est tout à fait normal étant donné que nous n'avons pas modifié la langue du système qui n'est pas le français par défaut.

Pour changer ce paramètre, ouvrez le volet des applications et sélectionnez *Custom Locale*. Le réglage affiché doit être *en_US*, sélectionnez *fr_FR* grâce à un appui long et à la validation demandée.

Figure 5–23
Réglage de la langue du système



Relancez l'application, elle est bien en français !

Voici un tableau qui met en évidence la logique entre le code pays et les fichiers et répertoires nécessaires. Tous les codes pays possibles sont consultables grâce à l'application *Custom Locale*.

Tableau 5–1 Les différents pays avec leurs codes et l'emplacement de leurs ressources

Code pays	Pays/Langue concernés	Répertoire du fichier string.xml
Défaut	Angleterre/Anglais	`res/values/`
en-rUS	Etats-Unis/Anglais	`res/values/`
fr-rFR	France/Français	`res/values-fr/`
Ja-rJP	Japon/Japonnais	`res/values-ja/`
...	...	...

Internationaliser les images

Comme nous l'avons laissé entendre tout au long du chapitre, il est également possible de choisir des images différentes en fonction de la langue du système (utile par exemple si l'on veut afficher le drapeau de la langue courante).

Pour cela, créons une seconde image de la mascotte Android francisée, par exemple en lui rajoutant quelques couleurs. Il ne reste qu'à la placer au bon endroit. La logique est d'ailleurs similaire à celle qui dicte la gestion des chaînes de caractères.

Tableau 5–2 Les différents pays avec leurs codes et l'emplacement de leurs ressources images

Code pays	Pays/Langue concernés	Répertoire des images
Défaut	Angleterre/Anglais	`res/drawable/`
en-rUS	Etats-Unis/Anglais	`res/drawable-en-rUS/`
fr-rFR	France/Français	`res/drawable-fr-rFR/`
Ja-rJP	Japon/Japonnais	`res/drawable-ja-rJP/`
...	...	...

En consultant rapidement le tableau présenté ci-dessus, il faut placer cette nouvelle image – mais avec le même nom de fichier `android.png` – dans le répertoire `res/drawable-fr-FR`.

Relancez l'application et voilà, le tour est joué !

Figure 5–24
Résultat de notre internationalisation