4

Création d'un squelette d'application

Tous les livres consacrés à un langage ou à un environnement de programmation commencent par présenter un programme de démonstration de type "Bonjour à tous !" : il permet de montrer que l'on peut construire quelque chose tout en restant suffisamment concis pour ne pas noyer le lecteur dans les détails. Cependant, le programme "Bonjour à tous !" classique n'est absolument pas interactif (il se contente d'écrire ces mots sur la console) et est donc assez peu stimulant. Ce chapitre présentera donc un projet qui utilisera malgré tout un bouton et l'heure courante pour montrer le fonctionnement d'une activité Android simple.

Terminaux virtuels et cibles

Pour construire les projets, nous supposons que vous avez téléchargé le SDK (et, éventuellement, le plugin ADT d'Eclipse). Avant de commencer, nous devons présenter la notion de "cible" car elle risque de vous surprendre et elle est relativement importante dans le processus de développement.

Comme son nom l'indique, un AVD (*Android Virtual Device*), est un terminal virtuel – par opposition aux vrais terminaux Android comme le G1 ou le Magic de HTC. Les AVD sont utilisés par l'émulateur fourni avec le SDK et vous permettent de tester vos programmes avant de les déployer sur les véritables terminaux. Vous devez indiquer à l'émulateur un terminal virtuel afin qu'il puisse prétendre qu'il est bien le terminal décrit par cet AVD.

Création d'un AVD

Pour créer un AVD, vous pouvez lancer la commande `android create avd` ou utiliser Eclipse ; dans les deux cas, vous devez indiquer une cible qui précise la classe de terminaux qui sera simulée par l'AVD. À l'heure où cette édition est publiée, il existe trois cibles :

- une qui désigne un terminal Android 1.1, comme un G1 HTC de base qui n'aurait pas été mis à jour ;
- une deuxième qui désigne un terminal Android 1.5, sans le support de Google Maps ;
- une troisième qui désigne un terminal Android 1.5 disposant du support de Google Maps, ce qui est le cas de la majorité des terminaux Android actuels.

Si vous développez des applications utilisant Google Maps, vous devez donc utiliser un ADV ayant la cible 3. Sinon la cible 2 conviendra parfaitement. Actuellement, la plupart des G1 ayant été mis à jour avec Android 1.5, la cible 1 n'est plus très utile.

Vous pouvez créer autant d'AVD que vous le souhaitez du moment que vous avez assez d'espace disque disponible sur votre environnement de développement : si vous avez besoin d'un pour chacune des trois cibles, libre à vous ! N'oubliez pas, cependant, que l'installation d'une application sur un AVD n'affecte pas les autres AVD que vous avez créés.

Choix d'une cible

Lorsque vous créez un projet (avec `android create project` ou à partir d'Eclipse), vous devez également indiquer la classe de terminal visée par celui-ci. Les valeurs possibles étant les mêmes que ci-dessus, créer un projet avec une cible 3 donne les indications suivantes :

- Vous avez besoin d'Android 1.5.
- Vous avez besoin de Google Maps.

L'application finale ne s'installera donc pas sur les terminaux qui ne correspondent pas à ces critères.

Voici quelques règles pour vous aider à gérer les cibles :

- Demandez-vous ce dont vous avez réellement besoin. Ne créez un projet avec une cible 3 que si vous utilisez Google Maps, notamment. Sinon une cible 2 est préférable – vous exigerez toujours Android 1.5, mais votre application pourra s'exécuter sur des terminaux qui ne disposent pas de Google Maps.
- Testez autant de cibles que possible. Vous pourriez être tenté par la cible 1 pour viser le plus grand nombre de terminaux Android ; cependant, vous devrez alors tester votre application sur un AVD ayant une cible 1 et un AVD ayant une cible 2 (et il serait également souhaitable de la tester avec un AVD ayant une cible 3, au cas où).
- Vérifiez qu'une nouvelle cible n'a pas été ajoutée par une nouvelle version d'Android. Il devrait y avoir quelques nouvelles valeurs avec chaque version majeure (2.0 ou 1.6, par exemple), voire pour les versions intermédiaires (1.5r1 ou 1.5r2). Assurez-vous de tester votre application sur ces nouvelles cibles à chaque fois que cela est possible car certains peuvent utiliser ces nouvelles versions de terminaux dès qu'elles sortent.
- Le fait de tester avec des AVD, quelle que soit la cible, ne peut pas se substituer aux tests sur du vrai matériel. Les AVD sont conçus pour vous fournir des "environnements jetables", permettant de tester un grand nombre d'environnements, même ceux qui n'existent pas encore réellement. Cependant, vous devez mettre votre application à l'épreuve d'au moins un terminal Android. En outre, la vitesse de votre émulateur peut ne pas correspondre à celle du terminal – selon votre système, elle peut être plus rapide ou plus lente.

Commencer par le début

Avec Android, tout commence par la création d'un projet. En Java classique, vous pouvez, si vous le souhaitez, vous contenter d'écrire un programme sous la forme d'un unique fichier, le compiler avec `javac` puis l'exécuter avec Java. Android est plus complexe mais, pour faciliter les choses, Google a fourni des outils d'aide à la création d'un projet. Si vous utilisez un IDE compatible avec Android, comme Eclipse et le plugin Android (fourni avec le SDK), vous pouvez créer un projet directement à partir de cet IDE (menu Fichier > Nouveau > Projet, puis choisissez Android > Android Project).

Si vous vous servez d'outils non Android, vous pouvez utiliser le script `android`, qui se trouve dans le répertoire `tools/` de l'installation du SDK en lui indiquant que vous souhaitez créer un projet (`create project`). Il faut alors lui indiquer la version de la cible, le répertoire racine du projet, le nom de l'activité et celui du paquetage où tout devra se trouver :

```
android create project \
       --target 2 \
       --path chemin/vers/mon/projet \
       --activity Now \
       --package com.commonsware.android.skeleton
```

Cette commande crée automatiquement les fichiers que nous avons décrits au Chapitre 2 et que nous utiliserons dans le reste de ce chapitre.

Vous pouvez également télécharger les répertoires des projets exemples de ce livre sous la forme de fichiers ZIP à partir du site web de Pearson[1]. Ces projets sont prêts à être utilisés : vous n'avez donc pas besoin d'utiliser `android create project` lorsque vous aurez décompressé ces exemples.

L'activité

Le répertoire `src/` de votre projet contient une arborescence de répertoires Java classique, créée d'après le paquetage Java que vous avez utilisé pour créer le projet (`com.commonsware.android.skeleton` produit donc `src/com/commonsware/android/skeleton`). Dans le répertoire le plus bas, vous trouverez un fichier source nommé `Now.java`, dans lequel sera stocké le code de votre première activité. Celle-ci sera constituée d'un unique bouton qui affichera l'heure à laquelle on a appuyé dessus pour le dernière fois (ou l'heure de lancement de l'application si on n'a pas encore appuyé sur ce bouton).

Ouvrez `Now.java` dans un éditeur de texte et copiez-y le code suivant :

```
package com.commonsware.android.skeleton;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import java.util.Date;
public class Now extends Activity implements View.OnClickListener {
  Button btn;
  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    btn = new Button(this);
    btn.setOnClickListener(this);
    updateTime();
    setContentView(btn);
  }
```

1. http://pearson.fr.

```
  public void onClick(View view) {
    updateTime();
  }
  private void updateTime() {
    btn.setText(new Date().toString());
  }
}
```

Si vous avez téléchargé les fichiers à partir du site web de Pearson, vous pouvez vous contenter d'utiliser directement le projet `Skeleton/Now`.

Examinons maintenant chacune des parties de ce code.

Dissection de l'activité

La déclaration de paquetage doit être identique à celle que vous avez utilisée pour créer le projet.

Comme pour tout projet Java, vous devez importer les classes auxquelles vous faites référence. La plupart de celles qui sont spécifiques à Android se trouvent dans le paquetage `android` :

```
package com.commonsware.android.skeleton;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import java.util.Date;
```

Notez bien que toutes les classes de Java SE ne sont pas utilisables par les programmes Android. Consultez le guide de référence des classes Android[1] pour savoir celles qui sont disponibles et celles qui ne le sont pas.

Les activités sont des classes publiques héritées de la classe de base `android.app.Activity`. Ici, l'activité contient un bouton (`btn`) :

```
public class Now extends Activity implements View.OnClickListener {
  Button btn;
```

Un bouton, comme vous pouvez le constater d'après le nom du paquetage, est un widget Android. Les widgets sont des éléments d'interface graphique que vous pouvez utiliser dans une application.

1. **http://code.google.com/android/reference/packages.html**.

Pour rester simple, nous voulons capturer tous les clics de bouton dans l'activité elle-même : c'est la raison pour laquelle la classe de notre activité implémente également `OnClickListener`.

```
@Override
public void onCreate(Bundle icicle) {
  super.onCreate(icicle);
  btn = new Button(this);
  btn.setOnClickListener(this);
  updateTime();
  setContentView(btn);
}
```

La méthode `onCreate()` est appelée lorsque l'activité est lancée. La première chose à faire est d'établir un chaînage vers la superclasse afin de réaliser l'initialisation de l'activité Android de base.

Nous créons ensuite l'instance du bouton (avec `new Button(this)`) et demandons d'envoyer tous les clics sur ce bouton à l'instance de l'activité (avec `setOnClickListener()`). Nous appelons ensuite la méthode privée `updateTime()` (qui sera présentée plus loin), puis nous configurons la vue du contenu de l'activité pour que ce soit le bouton lui-même (avec `setContentView()`).

Tous les widgets dérivent de la classe de base `View`. Bien que l'on construise généralement l'interface graphique à partir d'une hiérarchie de vues, nous n'utiliserons ici qu'une seule vue.

Nous présenterons ce `Bundle icicle` magique au Chapitre 16. Pour l'instant, considérons-le comme un gestionnaire opaque, que toutes les activités reçoivent lors de leur création.

```
public void onClick(View view) {
  updateTime();
}
```

Avec Swing, un clic sur un `JButton` déclenche un `ActionEvent` qui est transmis à l'`ActionListener` configuré pour ce bouton. Avec Android, en revanche, un clic sur un bouton provoque l'appel de la méthode `onClick()` sur l'instance `OnClickListener` configurée pour le bouton. L'écouteur reçoit la vue qui a déclenché le clic (ici, il s'agit du bouton). Dans notre cas, nous nous contentons d'appeler la méthode privée `updateTime()` :

```
private void updateTime() {
  btn.setText(new Date().toString());
}
```

L'ouverture de l'activité (`onCreate()`) ou un clic sur le bouton (`onClick()`) doit provoquer la mise à jour du label du bouton avec la date courante. On utilise pour cela la méthode `setText()`, qui fonctionne exactement comme avec les `JButton` de Swing.

Compiler et lancer l'activité

Pour compiler l'activité, utilisez l'outil de création de paquetage Android intégré à votre IDE ou lancez `ant debug` depuis le répertoire racine de votre projet. Puis lancez l'activité en suivant les étapes suivantes :

1. Lancez l'émulateur (avec `tools/emulator` dans votre installation du SDK Android), pour obtenir une figure analogue à celle de la Figure 4.1. N'oubliez pas de préciser un AVD avec l'option `-avd`.

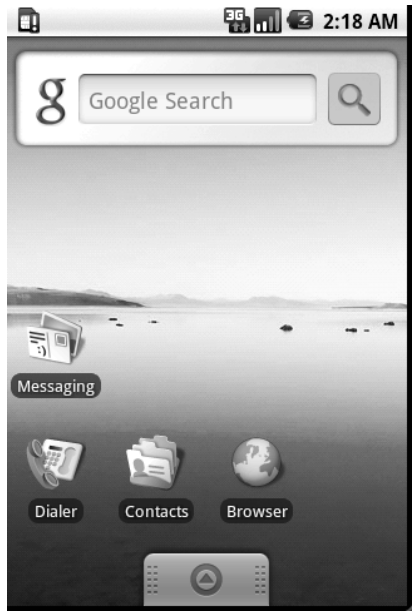

Figure 4.1
L'écran d'accueil d'Android.

2. Installez le paquetage (avec `tools/adb install` *`/racine/projet`*`/bin/Now-debug.apk`).
3. Consultez la liste des applications installées sur l'émulateur et recherchez celle qui s'appelle `Now` (voir Figure 4.2).
4. Ouvrez cette application.

Vous devriez voir apparaître un écran d'activité comme celui de la Figure 4.3.

Figure 4.2
Le lanceur d'applications d'Android.

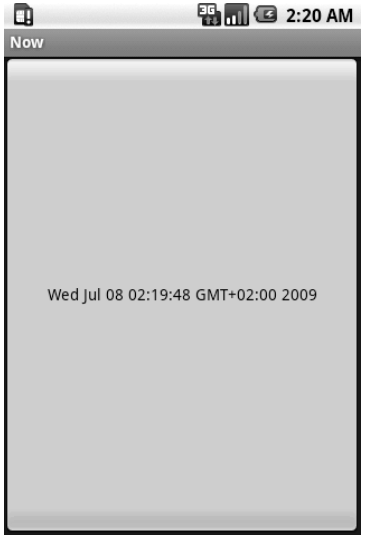

Figure 4.3
Démonstration de l'activité Now.

En cliquant sur le bouton – en d'autres termes, quasiment n'importe où sur l'écran du téléphone –, l'heure courante s'affichera sur le label du bouton.

Vous remarquerez que ce label est centré horizontalement et verticalement car c'est le style par défaut. Nous verrons au Chapitre 6 que nous pouvons évidemment contrôler ce formatage.

Une fois repu de cette technologie époustouflante des boutons, vous pouvez cliquer sur le bouton de retour en arrière de l'émulateur pour revenir au lanceur.

5

Utilisation des layouts XML

Bien qu'il soit techniquement possible de créer et d'attacher des composants widgets à une activité en utilisant uniquement du code Java comme nous l'avons fait au Chapitre 4, on préfère généralement employer un fichier de positionnement (*layout*) codé en XML. L'instanciation dynamique des widgets est réservée aux scénarios plus complexes, où les widgets ne sont pas connus au moment de la compilation (lorsqu'il faut, par exemple, remplir une colonne de boutons radio en fonction de données récupérées sur Internet).

Examinons maintenant ce code XML pour savoir comment sont agencées les vues des activités Android lorsque l'on utilise cette approche.

Qu'est-ce qu'un positionnement XML ?

Comme son nom l'indique, un positionnement XML est une spécification des relations existant entre les composants widgets – et avec leurs conteneurs (voir Chapitre 7) – exprimée sous la forme d'un document XML. Plus précisément, Android considère les layouts XML comme des ressources stockées dans le répertoire `res/layout` du projet.

Chaque fichier XML contient une arborescence d'éléments précisant le layout des widgets et les conteneurs qui composent une `View`. Les attributs de ces éléments sont des propriétés qui décrivent l'aspect d'un widget ou le comportement d'un conteneur. Un élément `Button` avec un attribut `android:textStyle = "bold"`, par exemple, signifie que le texte apparaissant sur ce bouton sera en gras.

L'outil `aapt` du SDK d'Android utilise ces layouts ; il est appelé automatiquement par la chaîne de production du projet (que ce soit *via* Eclipse ou par le traitement du fichier `build.xml` de `Ant`). C'est `aapt` qui produit le fichier source `R.java` du projet, qui vous permet d'accéder directement aux layouts et à leurs composants widgets depuis votre code Java, comme nous le verrons bientôt.

Pourquoi utiliser des layouts XML ?

La plupart de ce qui peut être fait avec des fichiers de positionnement XML peut également être réalisé avec du code Java. Vous pourriez, par exemple, utiliser `setTypeface()` pour qu'un bouton affiche son texte en gras au lieu d'utiliser une propriété dans un fichier XML. Ces fichiers s'ajoutant à tous ceux que vous devez déjà gérer, il faut donc qu'il y ait une bonne raison de les utiliser.

La principale est probablement le fait qu'ils permettent de créer des outils de définition des vues : le constructeur d'interfaces graphiques d'un IDE comme Eclipse ou un assistant dédié à la création des interfaces graphiques d'Android, comme DroidDraw[1], par exemple. Ces logiciels pourraient, en principe, produire du code Java au lieu d'un document XML, mais il est bien plus simple de relire la définition d'une interface graphique afin de la modifier lorsque cette définition est exprimée dans un format structuré comme XML au lieu d'être codée dans un langage de programmation. En outre, séparer ces définitions XML du code Java réduit les risques qu'une modification du code source perturbe l'application. XML est un bon compromis entre les besoins des concepteurs d'outils et ceux des programmeurs.

En outre, l'utilisation de XML pour la définition des interfaces graphiques est devenue monnaie courante. XAML[2] de Microsoft, Flex[3] d'Adobe et XUL[4] de Mozilla utilisent toutes une approche équivalente de celle d'Android : placer les détails des positionnements dans un fichier XML en permettant de les manipuler à partir des codes sources (à l'aide de JavaScript pour XUL, par exemple). De nombreux frameworks graphiques moins connus, comme ZK[5], utilisent également XML pour la définition de leurs vues. Bien que

1. **http://droiddraw.org/**.
2. **http://windowssdk.msdn.microsoft.com/en-us/library/ms752059.aspx**.
3. **http://www.adobe.com/products/flex/**.
4. **http://www.mozilla.org/projects/xul/**.
5. **http://www.zkoss.org/**.

"suivre le troupeau" ne soit pas toujours le meilleur choix, cette politique a l'avantage de faciliter la transition vers Android à partir d'un autre langage de description des vues reposant sur XML.

Contenu d'un fichier layout

Voici le bouton de l'application du chapitre précédent, converti en un fichier XML que vous trouverez dans le répertoire `chap5/Layouts/NowRedux` de l'archive des codes sources, téléchargeable sur le site **www.pearson.fr**, sur la page dédiée à cet ouvrage. Pour faciliter la recherche des codes des exemples, cette archive est découpée selon les chapitres du livre.

```
<?xml version="1.0" encoding="utf-8"?>
<Button xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/button"
    android:text=""
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"/>
```

Ici, le nom de l'élément XML est celui de la classe du widget, `Button`. Ce dernier étant fourni par Android, il suffit d'utiliser simplement son nom de classe. Dans le cas d'un widget personnalisé, dérivé d'`android.view.View`, il faudrait utiliser un nom pleinement qualifié, contenant le nom du paquetage (`com.commonsware.android.MonWidget`, par exemple).

L'élément racine doit déclarer l'espace de noms XML d'Android :

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

Tous les autres éléments sont des fils de la racine et héritent de cette déclaration.

Comme l'on souhaite pouvoir faire référence à ce bouton à partir de notre code Java, il faut lui associer un identifiant avec l'attribut `android:id`. Nous expliquerons plus précisément ce mécanisme plus loin dans ce chapitre.

Les autres attributs sont les propriétés de cette instance de `Button` :

- `android:text` précise le texte qui sera affiché au départ sur le bouton (ici, il s'agit d'une chaîne vide).
- `android:layout_width` et `android:layout_height` précisent que la largeur et la hauteur du bouton rempliront le "parent", c'est-à-dire ici l'écran entier – ces attributs seront présentés plus en détail au Chapitre 7.

Ce widget étant le seul contenu de notre activité, nous n'avons besoin que de cet élément. Les vues plus complexes nécessitent une arborescence d'éléments, afin de représenter les widgets et les conteneurs qui contrôlent leur positionnement. Dans la suite de ce livre, nous utiliserons des positionnements XML à chaque fois que cela est nécessaire : vous

trouverez donc des dizaines d'exemples plus complexes que vous pourrez utiliser comme point de départ pour vos propres projets.

Identifiants des widgets

De nombreux widgets et conteneurs ne peuvent apparaître que dans le fichier de positionnement et ne seront pas utilisés par votre code Java. Le plus souvent, un label statique (`TextView`), par exemple, n'a besoin d'être dans le fichier XML que pour indiquer l'emplacement où il doit apparaître dans l'interface. Ce type d'élément n'a donc pas besoin d'un attribut `android:id` pour lui donner un nom.

En revanche, tous les éléments dont vous avez besoin dans votre source Java doivent posséder cet attribut.

La convention consiste à utiliser le format `@+id/`*`nom_unique`* comme valeur d'identifiant, où *`nom_unique`* représente le nom local du widget, qui doit être unique. Dans l'exemple de la section précédente, l'identifiant du widget `Button` était `@+id/button`.

Android utilise également quelques valeurs `android:id` spécifiques, de la forme `@android:id/...`. Nous les rencontrerons dans différents chapitres de ce livre, notamment aux Chapitres 8 et 10.

Utilisation des widgets dans le code Java

Une fois que vous avez douloureusement configuré les widgets et les conteneurs dans un fichier de positionnement XML nommé `main.xml` et que vous l'avez placé dans le répertoire `res/layout`, vous n'avez plus qu'à ajouter une seule instruction dans la méthode `onCreate()` de votre activité pour pouvoir utiliser ce positionnement :

```
setContentView(R.layout.main);
```

Il s'agit du même appel `setContentView()` que nous avions utilisé précédemment en lui passant une instance d'une sous-classe de `View` (un `Button`). La vue construite à partir de notre fichier XML est désormais accessible à partir de la classe `R`. Tous les positionnements définis se trouvent sous `R.layout`, indiqués par le nom de base du fichier – `R.layout.main` désigne donc `main.xml`.

Pour accéder à nos widgets, nous utilisons ensuite la méthode `findViewById()`, en lui passant l'identifiant numérique du widget concerné. Cet identifiant a été produit par Android dans la classe `R` et est de la forme `R.id.`*`qquechose`* (où *`qquechose`* est le widget que vous recherchez). Ces widgets sont des sous-classes de `View`, exactement comme l'instance de `Button` que nous avions créée au Chapitre 4.

Fin de l'histoire

Dans le premier exemple `Now`, le texte du bouton affichait l'heure à laquelle on avait appuyé dessus pour la dernière fois (ou l'heure à laquelle l'activité avait été lancée).

L'essentiel du code fonctionne encore, même dans cette nouvelle version (que nous appellerons `NowRedux`). Cependant, au lieu d'instancier le bouton dans la méthode `onCreate()`, nous faisons référence à celui qui est décrit dans l'agencement XML :

```
package com.commonsware.android.layouts;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import java.util.Date;
public class NowRedux extends Activity
  implements View.OnClickListener {
  Button btn;
  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);

    setContentView(R.layout.main);
    btn=(Button)findViewById(R.id.button);
    btn.setOnClickListener(this);
    updateTime();
  }
  public void onClick(View view) {
    updateTime();
  }
  private void updateTime() {
    btn.setText(new Date().toString());
  }
}
```

La première différence est qu'au lieu de créer la vue dans le code Java nous faisons référence au fichier XML de positionnement (avec `setContentView(R.layout.main)`). Le fichier source `R.java` sera mis à jour lorsque le projet sera recompilé, afin d'inclure une référence au fichier de positionnement (`main.xml` du répertoire `res/layout`).

La seconde différence est qu'il faut retrouver l'instance de notre bouton en appelant la méthode `findViewById()`. Comme l'identifiant de ce bouton est `@+id/button`, nous pouvons désigner son identifiant numérique par `R.id.button`. Il reste ensuite à mettre en place l'écouteur d'événement et à configurer son label.

La Figure 5.1 montre que le résultat est le même que celui de l'exemple `Now` précédent.

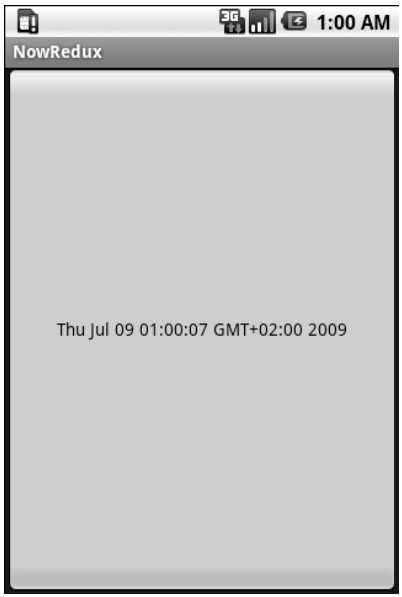

Figure 5.1
L'activité `NowRedux`.