35

Gestion des appels téléphoniques

La plupart des terminaux Android, si ce n'est tous, sont des téléphones. Leurs utilisateurs s'attendent donc à pouvoir téléphoner et recevoir des appels et, si vous le souhaitez, vous pouvez les y aider :

- Vous pourriez créer une interface pour une application de gestion des ventes (à la Salesforce.com) en offrant la possibilité d'appeler les vendeurs d'un simple clic, sans que l'utilisateur soit obligé de mémoriser ces contacts à la fois dans l'application et dans son répertoire téléphonique.
- Vous pourriez développer une application de réseau social avec une liste de numéros de téléphone qui évolue constamment : au lieu de "synchroniser" ces contacts avec ceux du téléphone, l'utilisateur pourrait les appeler directement à partir de cette application.
- Vous pourriez créer une interface personnalisée pour le système de contacts existants, éventuellement pour que les utilisateurs à mobilité réduite (telles les personnes âgées) puissent disposer de gros boutons pour faciliter la composition des appels.

Quoi qu'il en soit, Android vous permet de manipuler le téléphone comme n'importe quelle autre composante du système.

Le Manager

Pour tirer le meilleur parti de l'API de téléphonie, utilisez la classe `TelephonyManager`, qui permet notamment de :

- déterminer si le téléphone est en cours d'utilisation, *via* sa méthode `getCallState()`, qui renvoie les valeurs `CALL_STATE_IDLE` (téléphone non utilisé), `CALL_STATE_RINGING` (appel en cours de connexion) et `CALL_STATE_OFFHOOK` (appel en cours) ;
- trouver l'identifiant de la carte SIM avec `getSubscriberId()`;
- connaître le type du téléphone (GSM, par exemple) avec `getPhoneType()` ou celui de la connexion (comme GPRS, EDGE) avec `getNetworkType()`.

Appeler

Pour effectuer un appel à partir d'une application, en utilisant par exemple un numéro que vous avez obtenu par votre propre service web, créez une intention `ACTION_DIAL` avec une `Uri` de la forme `tel:`*`NNNNN`* (où *NNNNN* est le numéro de téléphone à appeler) et utilisez cette intention avec `startActivity()`. Cela ne lancera pas l'appel, mais activera l'activité du combiné, à partir duquel l'utilisateur pourra alors appuyer sur un bouton pour effectuer l'appel.

Voici, par exemple, un fichier de disposition simple mais efficace, extrait du projet `Phone/Dialer` :

```
<?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"
    >
  <LinearLayout
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    >
    <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Numero : "
      />
    <EditText android:id="@+id/number"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
```

```
      android:cursorVisible="true"
      android:editable="true"
      android:singleLine="true"
    />
  </LinearLayout>
  <Button android:id="@+id/dial"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:text="Appeler !"
  />
</LinearLayout>
```

Nous utilisons simplement un champ de saisie pour entrer un numéro de téléphone et un bouton pour appeler ce numéro.

Le code Java se contente de lancer le combiné en utilisant le numéro saisi dans le champ :

```
package com.commonsware.android.dialer;
import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
public class DialerDemo extends Activity {
  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.main);

    final EditText number=(EditText)findViewById(R.id.number);
    Button dial=(Button)findViewById(R.id.dial);

    dial.setOnClickListener(new Button.OnClickListener() {
      public void onClick(View v) {
        String toDial="tel:" + number.getText().toString();

        startActivity(new Intent(Intent.ACTION_DIAL,
                                 Uri.parse(toDial)));
      }
    });
  }
}
```

Comme le montre la Figure 35.1, l'interface de cette activité n'est pas très impressionnante.

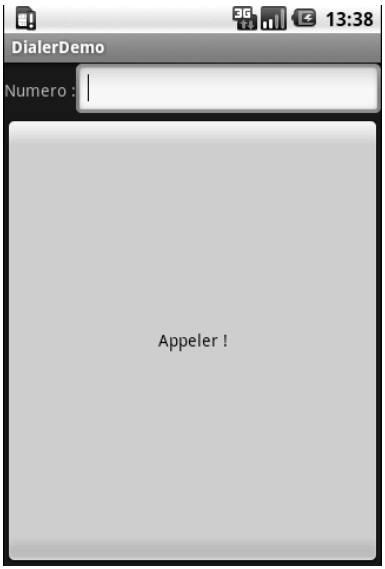

Figure 35.1
L'application `DialerDemo` *lors de son lancement.*

Cependant, le combiné téléphonique que l'on obtient en cliquant sur le bouton "Appeler !" est plus joli, comme le montre la Figure 35.2.

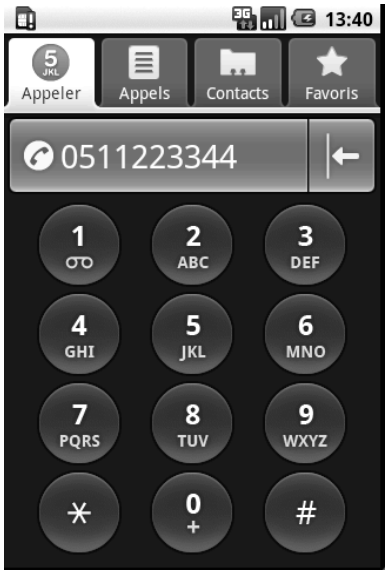

Figure 35.2
L'activité `Dialer` *d'Android lancée à partir de* `DialerDemo`.

36

Recherches avec SearchManager

L'une des sociétés à l'origine de l'alliance *Open Handset* – Google – dispose d'un petit moteur de recherche dont vous avez sans doute entendu parler. Il n'est donc pas étonnant qu'Android intègre quelques fonctionnalités de recherche.

Plus précisément, les recherches avec Android ne s'appliquent pas seulement aux données qui se trouvent sur l'appareil, mais également aux sources de données disponibles sur Internet.

Vos applications peuvent participer à ce processus en déclenchant elles-mêmes des recherches ou en autorisant que l'on fouille dans leurs données.

Cette fonctionnalité étant assez récente dans Android, les API risquent d'être modifiées : surveillez les mises à jour.

La chasse est ouverte

Android dispose de deux types de recherches : locales et globales. Les premières effectuent la recherche dans l'application en cours tandis que les secondes utilisent le moteur de Google pour faire une recherche sur le Web. Chacune d'elles peut être lancée de différentes façons :

- Vous pouvez appeler `onSearchRequested()` à partir d'un bouton ou d'un choix de menu afin de lancer une recherche locale (sauf si vous avez redéfini cette méthode dans votre activité).
- Vous pouvez appeler directement `startSearch()` pour lancer une recherche locale ou globale en fournissant éventuellement une chaîne de caractères comme point de départ.
- Vous pouvez faire en sorte qu'une saisie au clavier déclenche une recherche locale avec `setDefaultKeyMode(DEFAULT_KEYS_SEARCH_LOCAL)` ou globale avec `setDefaultKeyMode(DEFAULT_KEYS_SEARCH_GLOBAL)`.

Dans tous les cas, la recherche apparaît comme un ensemble de composants graphiques disposés en haut de l'écran, votre activité apparaissant en flou derrière eux (voir Figures 36.1 et 36.2).

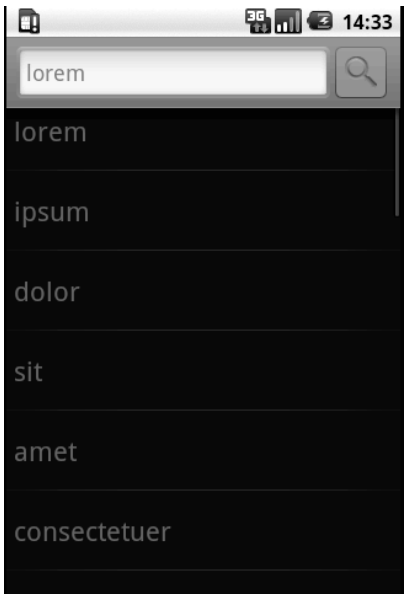

Figure 36.1
Les composants de la recherche locale d'Android.

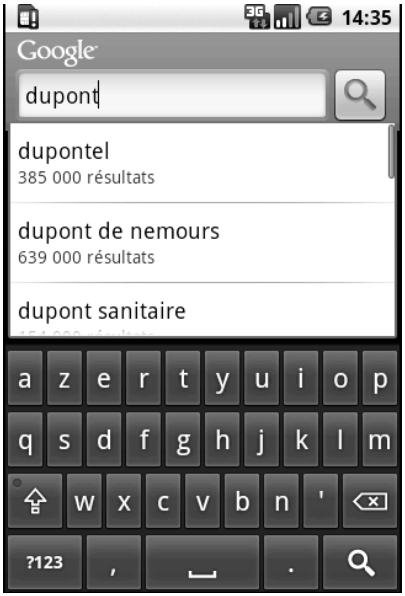

Figure 36.2
Les composants de la recherche globale d'Android avec une liste déroulante montrant les recherches précédentes.

Recherches personnelles

À terme, il existera deux variantes de recherches disponibles :

- les recherches de type requête, où la chaîne recherchée par l'utilisateur est passée à une activité qui est responsable de la recherche et de l'affichage des résultats ;
- les recherches de type filtre, où la chaîne recherchée par l'utilisateur est passée à une activité à chaque pression de touche et où l'activité est chargée de mettre à jour une liste des correspondances.

Cette dernière approche étant encore en cours de développement, intéressons-nous à la première.

Création de l'activité de recherche

Pour qu'une application puisse proposer des recherches de type requête, la première chose à faire consiste à créer une activité de recherche. Bien qu'il soit possible qu'une même activité puisse être ouverte à partir du lanceur et à partir d'une recherche, il s'avère que cela trouble un peu les utilisateurs. En outre, utiliser une activité séparée est plus propre d'un point de vue technique.

L'activité de recherche peut avoir l'aspect que vous souhaitez. En fait, à part examiner les requêtes, elle ressemble, se comporte et répond comme toutes les autres activités du système. La seule différence est qu'une activité de recherche doit vérifier les intentions

fournies à `onCreate()` (*via* `getIntent()`) et à `onNewIntent()` pour savoir si l'une d'elles est une recherche, auquel cas elle effectue la recherche et affiche le résultat.

L'application `Search/Lorem`, par exemple, commence comme un clone de l'application du Chapitre 8, qui affichait une liste des mots pour démontrer l'utilisation du conteneur `ListView`. Ici, nous la modifions pour pouvoir rechercher les mots qui contiennent une chaîne donnée.

L'activité principale et l'activité de recherche partagent un layout formé d'une `ListView` et d'un `TextView` montrant l'entrée sélectionnée :

```
<?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:id="@+id/selection"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
  />
  <ListView
    android:id="@android:id/list"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:drawSelectorOnTop="false"
  />
</LinearLayout>
```

L'essentiel du code des activités se trouve dans une classe abstraite `LoremBase` :

```
abstract public class LoremBase extends ListActivity {
  abstract ListAdapter makeMeAnAdapter(Intent intent);

  private static final int LOCAL_SEARCH_ID = Menu.FIRST+1;
  private static final int GLOBAL_SEARCH_ID = Menu.FIRST+2;
  private static final int CLOSE_ID = Menu.FIRST+3;
  TextView selection;
  ArrayList<String> items=new ArrayList<String>();

  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.main);
    selection=(TextView)findViewById(R.id.selection);

    try {
      XmlPullParser xpp=getResources().getXml(R.xml.words);

      while (xpp.getEventType()!=XmlPullParser.END_DOCUMENT) {
```

```
      if (xpp.getEventType()==XmlPullParser.START_TAG) {
        if (xpp.getName().equals("word")) {
          items.add(xpp.getAttributeValue(0));
        }
      }

      xpp.next();
    }
  }
  catch (Throwable t) {
    Toast
      .makeText(this, "Echec de la requete : " + t.toString(), 4000)
      .show();
  }

  setDefaultKeyMode(DEFAULT_KEYS_SEARCH_LOCAL);

  onNewIntent(getIntent());
}

@Override
public void onNewIntent(Intent intent) {
  ListAdapter adapter=makeMeAnAdapter(intent);

  if (adapter==null) {
    finish();
  }
  else {
    setListAdapter(adapter);
  }
}

public void onListItemClick(ListView parent, View v, int position,
                long id) {
  selection.setText(items.get(position).toString());
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
  menu.add(Menu.NONE, LOCAL_SEARCH_ID, Menu.NONE, "Recherche locale")
          .setIcon(android.R.drawable.ic_search_category_default);
  menu.add(Menu.NONE, GLOBAL_SEARCH_ID, Menu.NONE, "Recherche globale")
          .setIcon(R.drawable.search)
          .setAlphabeticShortcut(SearchManager.MENU_KEY);
  menu.add(Menu.NONE, CLOSE_ID, Menu.NONE, "Fermeture")
          .setIcon(R.drawable.eject)
          .setAlphabeticShortcut('f');

  return(super.onCreateOptionsMenu(menu));
}
```

```
  @Override
  public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
      case LOCAL_SEARCH_ID:
        onSearchRequested();
        return(true);

      case GLOBAL_SEARCH_ID:
        startSearch(null, false, null, true);
        return(true);

      case CLOSE_ID:
        finish();
        return(true);
    }
    return(super.onOptionsItemSelected(item));
  }
}
```

Cette activité prend en charge tout ce qui est lié à l'affichage d'une liste de mots, y compris l'extraction des mots à partir du fichier XML. En revanche, elle ne fournit pas le `ListAdapter` à placer dans la `ListView` – cette tâche est déléguée aux sous-classes.

L'activité principale – `LoremDemo` – utilise simplement un `ListAdapter` pour la liste de mots :

```
package com.commonsware.android.search;
import android.content.Intent;
import android.widget.ArrayAdapter;
import android.widget.ListAdapter;
public class LoremDemo extends LoremBase {
  @Override
  ListAdapter makeMeAnAdapter(Intent intent) {
    return(new ArrayAdapter<String>(this,
                    android.R.layout.simple_list_item_1,
                    items));
  }
}
```

L'activité de recherche, cependant, fonctionne un peu différemment. Elle commence par inspecter l'intention fournie à la méthode `makeMeAnAdapter()` ; cette intention provient soit d'`onCreate()`, soit d'`onNewIntent()`. S'il s'agit d'`ACTION_SEARCH`, on sait que c'est une recherche : on peut donc récupérer la requête et, dans le cas de notre exemple stupide, dérouler la liste des mots chargés pour ne conserver que ceux qui contiennent la chaîne recherchée. La liste ainsi obtenue est ensuite enveloppée dans un `ListAdapter` que l'on renvoie pour qu'il soit affiché :

```
package com.commonsware.android.search;
import android.app.SearchManager;
import android.content.Intent;
```

```
import android.widget.ArrayAdapter;
import android.widget.ListAdapter;
import java.util.ArrayList;
import java.util.List;
public class LoremSearch extends LoremBase {
  @Override
  ListAdapter makeMeAnAdapter(Intent intent) {
    ListAdapter adapter=null;

    if (intent.getAction().equals(Intent.ACTION_SEARCH)) {
      String query=intent.getStringExtra(SearchManager.QUERY);
      List<String> results=searchItems(query);

      adapter=new ArrayAdapter<String>(this,
                      android.R.layout.simple_list_item_1,
                      results);
      setTitle("LoremSearch de : " + query);
    }

    return(adapter);
  }

  private List<String> searchItems(String query) {
    List<String> results=new ArrayList<String>();

    for (String item : items) {
      if (item.indexOf(query)>-1) {
        results.add(item);
      }
    }

    return(results);
  }
}
```

Modification du manifeste

Bien que ce code implémente la recherche, il n'est pas intégré au système de recherche d'Android. Pour ce faire, vous devez modifier le fichier `AndroidManifest.xml` :

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.commonsware.android.search">
  <application>
    <activity android:name=".LoremDemo" android:label="LoremDemo">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
```

```
        <meta-data android:name="android.app.default_searchable"
              android:value=".LoremSearch" />
      </activity>
      <activity
        android:name=".LoremSearch"
        android:label="LoremSearch"
        android:launchMode="singleTop">
        <intent-filter>
          <action android:name="android.intent.action.SEARCH" />
          <category android:name="android.intent.category.DEFAULT" />
        </intent-filter>
        <meta-data android:name="android.app.searchable"
              android:resource="@xml/searchable" />
      </activity>
    </application>
</manifest>
```

Les modifications nécessaires sont les suivantes :

1. L'activité `LoremDemo` reçoit un élément `meta-data` avec un attribut `android:name` valant `android.app.default_searchable` et un attribut `android:value` contenant la classe qui implémente la recherche (`.LoremSearch`).
2. L'activité `LoremSearch` reçoit un filtre d'intention pour `android.intent.action.SEARCH`, afin que les intentions de recherche puissent être sélectionnées.
3. L'activité `LoremSearch` reçoit l'attribut `android:launchMode = "singleTop"`, ce qui signifie qu'une seule instance de cette activité sera ouverte à un instant donné, afin d'éviter que tout un lot de petites activités de recherche encombre la pile des activités.
4. L'activité `LoremSearch` reçoit un élément `meta-data` doté d'un attribut `android:name` valant `android.app.searchable` et d'un attribut `android:value` pointant vers une ressource XML contenant plus d'informations sur la fonctionnalité de recherche offerte par l'activité (`@xml/searchable`).

```
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
    android:label="@string/searchLabel"
    android:hint="@string/searchHint" />
```

Actuellement, cette ressource XML fournit deux informations :

- le nom qui doit apparaître dans le bouton du domaine de recherche à droite du champ de saisie, afin d'indiquer à l'utilisateur l'endroit où il recherche (`android:label`) ;
- le texte qui doit apparaître dans le champ de saisie, afin de donner à l'utilisateur un indice sur ce qu'il doit taper (`android:hint`).

Effectuer une recherche

Android sait désormais que votre application peut être consultée, connaît le domaine de recherche à utiliser lors d'une recherche à partir de l'activité principale et l'activité sait comment effectuer la recherche.

Le menu de cette application permet de choisir une recherche locale ou une recherche globale. Pour effectuer la première, on appelle simplement `onSearchRequested()` ; pour la seconde, on appelle `startSearch()` en lui passant `true` dans son dernier paramètre, afin d'indiquer que la portée de la recherche est globale.

En tapant une lettre ou deux, puis en cliquant sur le bouton, on lance l'activité de recherche et le sous-ensemble des mots contenant le texte recherché s'affiche. Le texte tapé apparaît dans la barre de titre de l'activité, comme le montre la Figure 36.3.

Figure 36.3
L'application `Lorem`, montrant une recherche locale.

Vous pouvez obtenir le même effet en commençant à taper dans l'activité principale car elle est configurée pour déclencher une recherche locale.