Chapitre 5
Menus et boites de dialogues

5.1 Introduction

Dans ce chapitre nous allons apprendre la création des menus et nous présentons le mécanisme des toasts qui représentent une exception et une excellente alternative au système de boîtes de dialogue.

5.2 Créer un menu pour une activité

Tous les modèles Android possèdent un bouton Menu. Grâce à ce bouton, il vous est possible de proposer à vos utilisateurs des fonctionnalités supplémentaires n'apparaissant pas par défaut à l'écran et d'ainsi mieux gérer la taille limitée de l'écran d'un appareil mobile.

Chaque menu est propre à une activité, c'est pourquoi toutes les opérations que nous allons traiter dans cette partie se réfèrent à une activité. Pour proposer plusieurs menus vous aurez donc besoin de le faire dans chaque activité de votre application.

- ✓ Pour créer un menu il vous suffit de surcharger la méthode **onCreateOptionsMenu** de la classe Activity. Cette méthode est appelée la première fois que l'utilisateur appuie sur le bouton menu de son téléphone.
- ✓ Elle reçoit en paramètre **un objet de type Menu** dans lequel nous ajouterons nos éléments ultérieurement.
- ✓ Si l'utilisateur appuie une seconde fois sur le bouton Menu, cette méthode ne sera plus appelée tant que l'activité ne sera pas **détruite puis recréée.**
- ✓ Si vous avez besoin d'ajouter, de supprimer ou de modifier un élément du menu après coup, il vous faudra surcharger une autre méthode, la méthode **onPrepareOptionsMenu** que nous aborderons plus tard dans cette partie.
- ✓ Pour créer un menu, commencez par créer un fichier de définition d'interface nommé **main.xml**:

Code 5.1 : Définition de l'interface de l'exemple avec menu

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content">
   <TextView
      android:id="@+id/TextView01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Appuyez sur la touche menu">
   </TextView>
</LinearLayout>
```

- ✓ Créez une nouvelle classe d'activité, dans un fichier **main.java** par exemple, dans lequel vous placerez le contenu suivant :

Code 5.2 : Création d'un menu

```
import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;

public class Main extends Activity {

   // Pour faciliter la gestion des menus
   // nous créons des constantes
   private final static int MENU_PARAMETRE = 1;
   private final static int MENU_QUITTER = 2;

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

   @Override
   public boolean onCreateOptionsMenu(Menu menu) {
      // Nous créons un premier menu pour accéder aux paramètres.
      // Pour ce premier élément du menu, nous l'agrémentons
      // d'une image.
      menu.add(0, MENU_PARAMETRE, Menu.NONE, "Paramètres").setIcon(R.drawable.icon);
      // Nous créons un second élément.
      // Celui-ci ne comportera que du texte.
      menu.add(0, MENU_QUITTER, Menu.NONE, "Quitter");

      return true;
   }
```

menu.add (groupID, itemID, ordre, "nom du menu") ;

Cette méthode permet de créer un nouveau menu. Les paramètres nécessaires sont les suivants :

- **groupID :** identifiant du groupe. Il est possible de regrouper les éléments,
- **itemID :** identifiant de ce menu. Il nous sera utilisé pour identifier ce menu parmi les autres. On doit donner un identifiant différent à chaque menu (1, 2, 3… par exemple).
- **ordre :** associer un ordre d'affichage au menu. On donnera ici la valeur 0.
- **Nom du menu** : chaîne qui représente le titre du menu.

```
   @Override
   public boolean onOptionsItemSelected(MenuItem item) {
      switch (item.getItemId()) {
      case MENU_PARAMETRE:
         // Lorsque l'utilisateur cliquera sur le menu 'paramètres',
         // un message s'affichera.
         Toast.makeText(Main.this, "Ouverture des paramètres",
                        Toast.LENGTH_SHORT).show();
         // onOptionsItemSelected retourne un booléen,
         // nous lui envoyons la valeur "true" pour signaler
         // que tout s'est correctement déroulé.
         return true;
```

```
        case MENU_QUITTER:
           // Lorsque l'utilisateur cliquera sur le menu 'Quitter',
           // nous fermerons l'activité avec la méthode finish().
           finish();
           return true;
        default:
           return true;
        }
     }
}
```

- Dans le code précédent, nous avons surchargé deux méthodes, l'une responsable de la création du menu de l'activité, l'autre de la logique lors de la sélection par l'utilisateur d'un élément du menu.
- La méthode **onCreateOptionsMenu** sera appelée uniquement la première fois lorsque l'utilisateur appuiera sur le bouton **Menu.**
- Dans cette méthode, nous créons deux éléments de menu **Paramètres** et **Quitter.**
- La méthode **onOptionsItemSelected** est appelée lorsque l'utilisateur clique sur l'un des éléments du menu. Dans notre cas, si l'utilisateur clique sur l'élément **Paramètres**
- L'application affiche **un message**, alors qu'un clic sur **Quitter** ferme l'activité grâce à la méthode **finish.**

  Le résultat d'exécution est affiché sur la Figure 5.1.

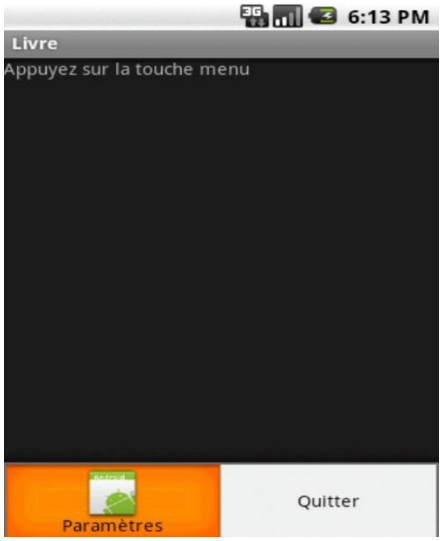


Figure 5.1. Résultat d'exécution

5.3 Mettre à jour dynamiquement un menu

- Il vous sera peut-être nécessaire de changer l'intitulé, de supprimer ou de rajouter un élément du menu en cours de fonctionnement.
- La méthode **onCreateOptionsMenu** n'étant appelée qu'une fois, la première fois où l'utilisateur clique sur le bouton Menu, vous ne pourrez pas mettre à jour votre menu dans cette méthode.
- Pour modifier le menu après coup, par exemple dans le cas où votre menu propose de s'authentifier et qu'une fois authentifié, vous souhaitez proposer à l'utilisateur de se déconnecter, vous devrez surcharger la méthode **onPrepareOptionsMenu.**

- Un objet de type **Menu** est envoyé à cette méthode qui est appelée à chaque fois que l'utilisateur cliquera sur le bouton **Menu**.

Modifiez le code 5.2 pour rajouter la redéfinition de la méthode **onPrepareOptionsMenu :**

Code 5.3 : Modification dynamique du menu

```
@Override
public boolean onPrepareOptionsMenu(Menu menu) {
    // Nous modifions l'intitulé de notre premier menu
    // Pour l'exemple nous lui donnerons un intitulé
    // différent à chaque fois que l'utilisateur cliquera
    // sur le bouton menu à l'aide de l'heure du système
    menu.getItem(0).setTitle(SystemClock.elapsedRealtime()+"");

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

- Dans cet exemple, lorsque l'utilisateur appuie sur le bouton Menu, l'intitulé du premier menu affiche une valeur différente à chaque fois.
- Vous pourrez placer à cet endroit le code nécessaire pour adapter l'intitulé que vous souhaitez afficher à l'utilisateur s'il est nécessaire de changer le menu.
- Si l'utilisateur appuie à nouveau, l'intitulé changera de concert. S'il appuie encore, l'intitulé changera de même.

5.4 Création des sous-menus

Créer des sous-menus peut se révéler utile pour proposer plus d'options sans encombrer l'interface utilisateur. Pour ajouter un sous-menu, vous devrez ajouter un menu de type **SubMenu** et des éléments le composant. Le paramétrage est le même que pour un menu, à l'exception de l'image, vous ne pourrez pas ajouter d'image sur les éléments de vos sous-menus. Néanmoins, il sera possible d'ajouter une image dans l'entête du sous-menu.

- Remplacez la méthode **onCreateOptionsMenu** du code 5.2 par celle-ci :

Code 5.4 : Création de sous-menus

```
private final static int SOUSMENU_VIDEO= 1000;
private final static int SOUSMENU_AUDIO= 1001;

...

@Override
public boolean onCreateOptionsMenu(Menu menu) {
   // Nous créons un premier menu sous forme de sous-menu
   // Les sous menu associés à ce menu seront ajoutés à ce sous-menu
   SubMenu sousMenu = menu.addSubMenu(0, MENU_PARAMETRE, Menu.NONE,
                                      "Paramètres").setIcon(R.drawable.icon);
   // Nous ajoutons notre premier sous-menu
   sousMenu.add(0, SOUSMENU_AUDIO, Menu.NONE, "Audio").setIcon(R.drawable.icon);
   // Nous ajoutons notre deuxième sous-menu
   sousMenu.add(0, SOUSMENU_VIDEO, Menu.NONE, "Vidéo");

   // Il est possible de placer une
   // image dans l'entête du sous-menu
   // Il suffit de le paramétrer avec la méthode
   // setHeaderIcon de notre objet SubMenu
   sousMenu.setHeaderIcon(R.drawable.icon);

   // Nous créons notre deuxième menu
   menu.add(0, MENU_QUITTER, Menu.NONE, "Quitter");

   return true;
}
```

- Lorsque l'utilisateur clique sur le bouton Menu, le menu qui s'affiche est identique à l'exemple précédent. Néanmoins en cliquant sur le menu **Paramètres** un sous-menu s'affiche sous forme de liste (Figure 5.2).

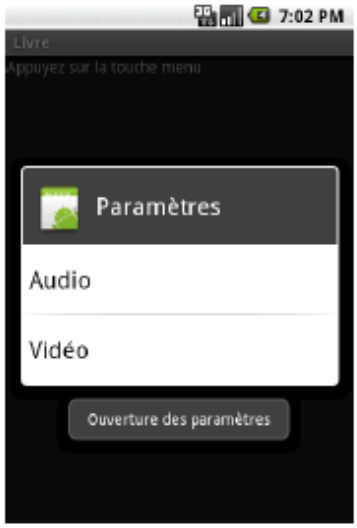


Figure 5.2. Création de sous menus

5.5 Création d'un menu contextuel

La plate-forme Android autorise également la création des menus contextuels, autrement dit de menus dont le contenu change en fonction du contexte. Sous Android, l'appel d'un menu contextuel se fait lorsque l'utilisateur effectue un clic de quelques secondes sur un élément d'interface graphique. Les menus contextuels fonctionnent d'ailleurs de façon similaire aux sous-menus.

- ✓ La création d'un menu contextuel se fait en deux étapes :
  - ➢ tout d'abord la création du menu proprement dit, puis
  - ➢ son enregistrement auprès de la vue, avec la méthode **registerForContextMenu**; en fournissant en paramètre la **vue concernée.**
- ✓ Concrètement, redéfinissez la méthode **onCreateContextMenu** et ajoutez-y les menus à afficher en fonction de la vue sélectionnée par l'utilisateur.
- ✓ Pour sélectionner les éléments qui s'y trouveront, redéfinissez la méthode **onContextItemSelected**.

Créez un nouveau fichier de définition d'une interface comme ceci :

Code 5.5 : Définition de l'interface de l'exemple avec menu contextuel

```
<?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"
   android:gravity="center_vertical|center_horizontal">
   <TextView
      android:id="@+id/monTexteContextMenu"
      android:layout_height="wrap_content"
      android:text="Cliquez ici 3 sec"
      android:textSize="30dip"
      android:layout_width="fill_parent"
      android:gravity="center_horizontal">
   </TextView>
</LinearLayout>
```

Effectuez les modifications du code 5.2 suivantes :

Code 5.6 : Création d'un menu contextuel

```
...
import android.view.ContextMenu;
import android.view.MenuItem;
import android.view.ContextMenu.ContextMenuInfo;
...

   // Pour faciliter notre gestion des menus
   // nous créons des variables de type int
   // avec des noms explicites qui nous aideront
   // à ne pas nous tromper de menu
   private final static int MENU_CONTEXT_1= 1;
   private final static int MENU_CONTEXT_2= 2;


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

      // Nous enregistrons notre TextView pour qu'il réagisse
      // au menu contextuel
      registerForContextMenu((TextView)findViewById(R.id.monTexteContextMenu));
   }


   @Override
   public void onCreateContextMenu(ContextMenu menu, View v,
         ContextMenuInfo menuInfo) {
      // Nous vérifions l'identifiant de la vue.
      // Si celle-ci correspond à une vue pour laquelle nous souhaitons
      // réagir au clic long, alors nous créons un menu.
      // Si vous avez plusieurs vues à traiter dans cette méthode,
      // il vous suffit d'ajouter le code nécessaire pour créer les
      // différents menus.
      switch (v.getId()) {
      case R.id.monTexteContextMenu:
         {
               menu.setHeaderTitle("Menu contextuel");
               menu.setHeaderIcon(R.drawable.icon);
               menu.add(0, MENU_CONTEXT_1, 0, "Mon menu contextuel 1");
               menu.add(0, MENU_CONTEXT_2, 0, "Mon menu contextuel 2");
               break;
            }
      }
      super.onCreateContextMenu(menu, v, menuInfo);
   }
```

```
   @Override
   public boolean onContextItemSelected(MenuItem item) {
      // Grâce à l'identifiant de l'élément
      // sélectionné dans notre menu nous
      // effectuons une action adaptée à ce choix
      switch (item.getItemId()) {
      case MENU_CONTEXT_1:
         {
               Toast.makeText(Main.this, "Menu contextuel 1 cliqué !",
                              Toast.LENGTH_SHORT).show();
               break;
         }
      case MENU_CONTEXT_2:
         {
               Toast.makeText(Main.this, "Menu contextuel 2 cliqué !",
                              Toast.LENGTH_SHORT).show();
               break;
         }
      }
      return super.onContextItemSelected(item);
   }
}
```

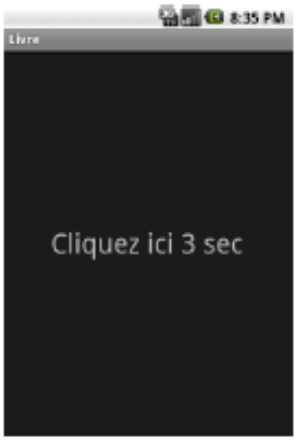


Figure 5.3. Vue possédant un menu contextuel

- Si l'utilisateur clique pendant trois secondes sur la vue **TextView** centrale, le menu contextuel s'affiche.

Figure 5.4. Affichage du menu contextuel

5.6 Notifier l'utilisateur par le mécanisme des « toasts »

Notifier un utilisateur ne signifie pas le déranger alors qu'il utilise une autre application que la vôtre ou qu'il rédige un SMS. À cette fin, la plate-forme Android propose le concept de toast, ou message non modal s'affichant quelques secondes tout au plus. De cette façon, ni l'utilisateur ni l'application active à ce moment ne sont interrompus.

L'utilisation d'un toast est rendue aisée par les méthodes statiques de la classe **Toast** et notamment **makeText**. Il vous suffit de passer le **Context** de l'application, le **texte** et la **durée** d'affichage pour créer une instance de Toast que vous pourrez afficher avec la méthode **show** autant de fois que nécessaire.

Code 5.7 : Création d'un toast

```
// La chaîne représentant le message
String message = "Vous prendrez bien un toast ou deux ?";
// La durée d'affichage (LENGTH_SHORT ou LENGTH_LONG)
int duree= Toast.LENGTH_LONG;
```

```
// Création du Toast (le contexte est celui de l'activité ou du service)
Toast toast = Toast.makeText(this, message, duree);
// Affichage du Toast
toast.show();
```

La durée du toast ne peut être librement fixée : elle peut soit prendre la valeur **LENGTH_SHORT** (2 secondes) ou **LENGTH_LONG** (5 secondes).

L'extrait de code précédent affiche le résultat suivant (Figure 5.5) :

Figure 5.5. Affichage d'un toast

5.6.1 Positionner un toast

Par défaut, Android affiche le message en **bas de l'écran** de l'utilisateur. Ce comportement peut être redéfini pour en changer la position. Vous pouvez spécifier la disposition d'un toast en spécifiant son ancrage sur l'écran ainsi qu'un décalage sur l'axe horizontal et vertical.

Code 5.8 : Positionner un toast

```
// Création du Toast
Toast toast = Toast.makeText(this, "Vous prendrez bien un toast ou deux ;",
Toast.LENGTH_LONG);
// Spécifie la dispostion du Toast sur l'écran
toast.setGravity(Gravity.TOP, 0, 40);
// Affichage du Toast
toast.show();
```

Le code précédent affiche le message en haut de l'écran avec un décalage vertical (Figure 5.6).

Figure 5.6. Positionnement d'un toast

La position du toast est bien sûr fonction de l'importance du message. Affiché plus haut sur l'écran, le message aura plus de chance d'attirer l'attention de l'utilisateur que s'il se trouve tout en bas.

5.6.2 Personnaliser l'apparence d'un toast

Par défaut, Android utilise une fenêtre grisée pour afficher le texte du toast à l'utilisateur. Mais ce comportement n'est pas toujours le plus adapté pour communiquer des informations complexes à l'utilisateur. Dans certains cas, un affichage graphique sera plus approprié qu'un affichage textuel.

Sachez qu'Android est capable d'utiliser une vue personnalisée conçue pour l'occasion, en lieu et place du « carré grisâtre aux bords arrondis » par défaut.

Pour spécifier cette vue à afficher, utilisez la méthode **setView** de l'objet Toast. Même si vous ne souhaitez pas afficher autre chose que du texte, le simple fait de personnaliser le design de la vue vous permettra de différencier les messages émis par votre application des autres, et pour l'utilisateur de distinguer clairement l'origine des messages.

5.7 Conclusion

Nous avons vu dans ce chapitre les méthodes de création des deux types de menu Android (d'options et contextuel) ainsi que la notification des utilisateurs à travers le mécanisme des toasts.

Dans le chapitre suivant, nous allons voir le fichier de configuration AndroidManifest.xml ainsi que la communication entre composants d'applications Android.

Chapitre 6
AndroidManifest.xml et communication entre composants

6.1 Introduction

Nous allons découvrir dans ce chapitre le fichier de configuration qui représente la recette de chaque application Android. Nous allons voir par la suite comment se fait la communication entre les composants d'applications.

6.2 Le fichier de configuration Android

Chaque application Android nécessite un fichier de configuration : **AndroidManifest.xml**. Ce fichier est placé dans le répertoire de base du projet, à sa racine. Il décrit le contexte de l'application, les activités, les services, les récepteurs d'Intents (Broadcast receivers), les fournisseurs de contenu et les permissions.

6.2.1 Structure du fichier de configuration

Un fichier de configuration est composé d'une racine (le tag manifest **(1)**) et d'une suite de nœuds enfants qui définissent l'application.

Code 6.1 : Structure vide d'un fichier de configuration d'une application

```
<manifest (1)
      xmlns:android=http://schemas.android.com/apk/res/android (2)
      package="fr.domaine.application"> (3)
 </manifest>
```

La racine XML de la configuration est déclarée avec un espace de nom Android (xmlns:android **(2)**) qui sera utile plus loin dans le fichier ainsi qu'un paquetage **(3)** dont la valeur est celle du paquetage du projet.

Voici le rendu possible d'un manifeste qui donne une bonne idée de sa structure (code 6.2). Ce fichier est au format XML. Il doit donc toujours être :

- bien formé : c'est-à-dire respecter les règles d'édition d'un fichier XML en termes de nom des balises, de balises ouvrante et fermante, de non-imbrication des balises, etc. ;
- valide : il doit utiliser les éléments prévus par le système avec les valeurs prédéfinies.

Profitons-en pour étudier les éléments les plus importants de ce fichier de configuration :

<uses-permission> (1)

Les permissions qui seront déclarées ici seront un prérequis pour l'application. À l'installation, l'utilisateur se verra demander l'autorisation d'utiliser l'ensemble des fonctions liées à ces permissions comme la connexion réseau, la localisation de l'appareil, les droits d'écriture sur la carte mémoire...

<application> (2)

Un manifeste contient un seul et unique nœud application qui en revanche contient des nœuds concernant la définition d'activités, de services…

<activity> (3)

Déclare une activité présentée à l'utilisateur. Comme pour la plupart des déclarations que nous allons étudier, si vous oubliez ces lignes de configuration, vos éléments ne pourront pas être utilisés.

Code 6.2 : Structure du fichier AndroidManifest.xml extraite de la documentation

```
<?xml version="1.0" encoding="utf-8"?>

<manifest>

    <uses-permission /> (1)
    <permission />
    <permission-tree />
    <permission-group />
    <instrumentation />
    <uses-sdk />
    <uses-configuration />
    <uses-feature />
    <supports-screens />

    <application> (2)

        <activity> (3)
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
            <meta-data />
        </activity>

        <activity-alias>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </activity-alias>

        <service> (4)
            <intent-filter> . . . </intent-filter>
            <meta-data/>
        </service>

        <receiver> (5)
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </receiver>


        <provider> (6)
            <grant-uri-permission />
            <path-permission />
            <meta-data />
        </provider>

        <uses-library />

    </application>

</manifest>
```

<service> (4)
Déclare un composant de l'application en tant que service. Ici pas question d'interface graphique, tout se déroulera en tâche de fond de votre application.

<receiver> (5)
Déclare un récepteur d'objets Intent. Cet élément permet à l'application de recevoir ces objets alors qu'ils sont diffusés par d'autres applications ou par le système.

<provider> (6)
Déclare un fournisseur de contenu qui permettra d'accéder aux données gérées par l'application.

6.3 Communication entre composants (les Intents)

6.3.1 Définition et Utilisations

Une application Android peut contenir plusieurs activités :
• Une activité utilise la méthode **setContentView** pour s'associer avec une interface graphique.
• Les activités sont indépendantes les unes des autres, cependant, elles peuvent collaborer pour échanger des données et des actions.
• Typiquement, l'une des activités est désignée comme étant la première à être présentée à l'utilisateur quand l'application est lancée : on l'appelle l'activité de démarrage.
• Les activités interagissent en mode asynchrone.
• Le passage d'une activité à une autre est réalisé en demandant à l'activité en cours d'exécuter un **Intent.**
• Un Intent est un message qui peut être utilisé pour demander une action à partir d'un autre composant de l'application
• Un Intent permet invoquer des Activités, des Broadcast Receivers ou des Services. Les différentes méthodes utilisées pour appeler ces composantes sont les suivantes :
    • startActivity(intent) : lance une activité
    • sendBroadcast(intent) : envoie un intent à tous les composants Broadcast Receivers intéressés
    • startService(intent) ou bindService(intent, …) : communiquent avec un service en arrière plan.

6.3.2 Construction d'un Intent

Un Intent comporte des informations que le système Android utilise :

• **Nom du composant à démarrer**
• **Action à réaliser**
    • ACTION-VIEW, ACTION_SEND…
• **Donnée**
    • URI référençant la donnée sur laquelle l'action va agir
• **Catégorie**
    • Information supplémentaire sur le type de composants qui va gérer l'intent
    • CATEGORY-BROWSABLE, CATEGORY-LAUNCHER…
• **Extras**
    • Paires clef-valeur qui comportent des informations additionnelles pour réaliser l'action demandée
• **Drapeaux (Flags)**
    • Définissent la classe qui fonctionne comme métadonnée pour cet intent
    • Peuvent indiquer comment lancer une activité, comment la traiter une fois lancée

6.3.3 Types d'Intent

Il existe deux types d'Intents :

• **Intents Explicites:**
    • Spécifient le composant à démarrer par nom (nom complet de la classe)
    • Permettent de démarrer un composant de votre propre application, car le nom de la classe est connu
    • Exemple: démarrer une activité en réponse à l'action d'un utilisateur

• **Intents Implicites**
  • Ne nomment pas un composant spécifique, mais déclarent une action à réaliser
  • Permet à un composant d'une application d'appeler un composant d'une autre application
  • Exemple : montrer à l'utilisateur un emplacement sur une Map

6.4 Intents implicites

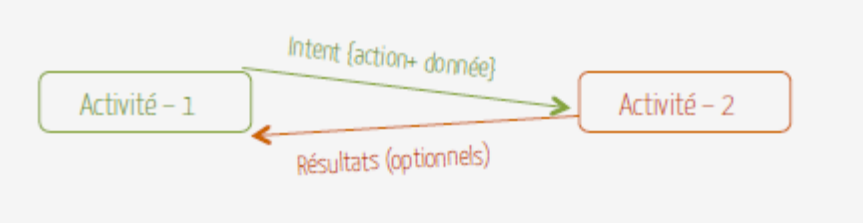
6.4.1 Arguments et Utilisation

Les principaux arguments d'un Intent implicite sont :

- Action : l'action à réaliser, peut être prédéfinie (ACTION_VIEW, ACTION_EDIT, ACTION_MAIN…) ou créée par l'utilisateur.
- Donnée : Les données principales sur lesquelles on va agir, tel que le numéro de téléphone à appeler.

Il est typiquement appelé comme suit:

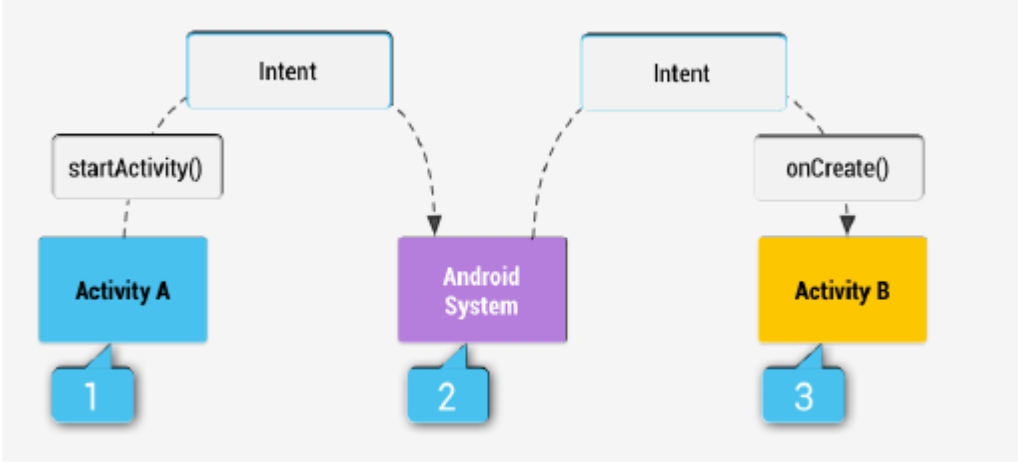
```
Intent myActivityIntent = new Intent (<action>, <donnee>) ;
startActivity (myActivityIntent) ;
```



6.4.2 Comportement d'un Intent Implicite

Un Intent implicite se comporte comme suit:

1. Activité **A** crée un Intent avec une action et le passe en paramètre à **startActivity**
2. Le système Android cherchent toutes les applications pour trouver un **Intent Filter** qui correspond à cet Intent
3. Quand une correspondance est trouvée, le système démarrent l'activité (Activity **B**) en invoquant sa méthode **onCreate** et en lui passant l'intent



➢ **Intent Filter**

• Un Intent Filter est une expression dans le fichier Manifest d'une application qui spécifie le type d'intents que le composant veut recevoir
• Permet aux autres activités de lancer directement votre activité en utilisant un certain Intent
• Si vous ne déclarez pas d'Intent Filters à votre activité, elle ne pourra être déclenchée que par un Intent Explicite

• Il est recommandé de ne pas déclarer d'Intent Filters pour les services, car cela peut causer des problèmes de sécurité

```
<activity
    android:name=".PopupActivity"
    android:label="Popup" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

➢ **Actions et Données Prédéfinies d'un Intent**

Voici des exemples d'actions prédéfinies communément utilisées

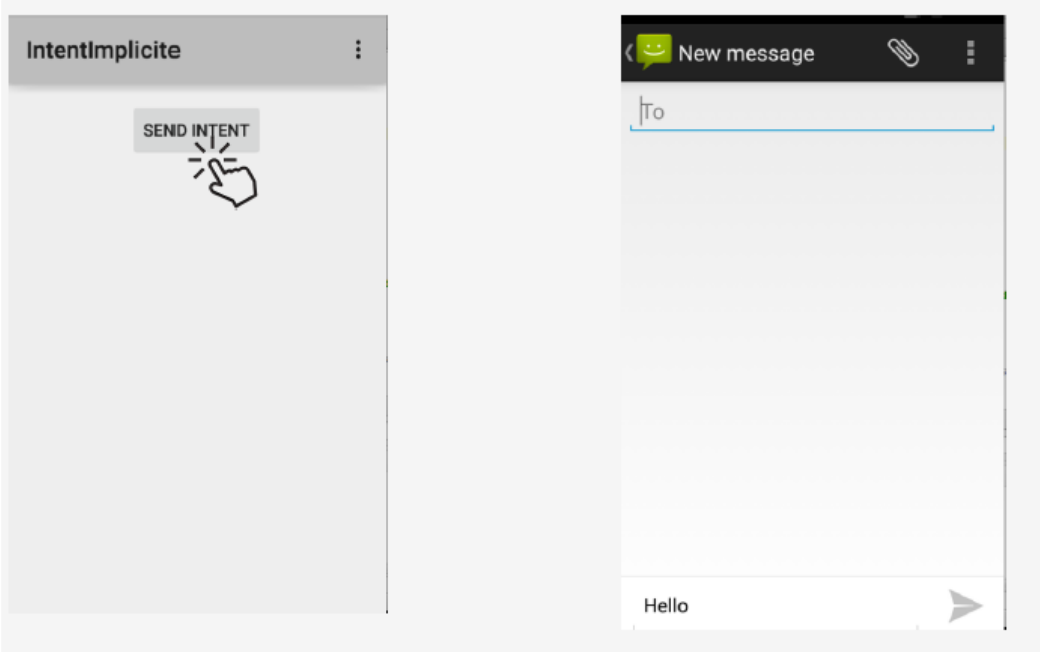
Action	Donnée	Description
ACTION_DIAL	tel:123	Affiche le numéroteur téléphonique avec le numéro (123) rempli
ACTION_VIEW	http://www.google.com	Affiche la page Google dans un navigateur.
ACTION_EDIT	content://contacts/people/2	Edite les informations sur la personne dont l'identifiant est 2 (de votre carnet d'adresse)
ACTION_VIEW	content://contacts/people/2	Utilisé pour démarrer une activité qui affiche les données du contact numéro 2
ACTION_VIEW	content://contacts/people	Affiche la liste des contacts, que l'utilisateur peut parcourir. La sélection d'un contact permet de visualiser ses détails dans un nouvel Intent.

➢ **Exemple d'Intent implicite**

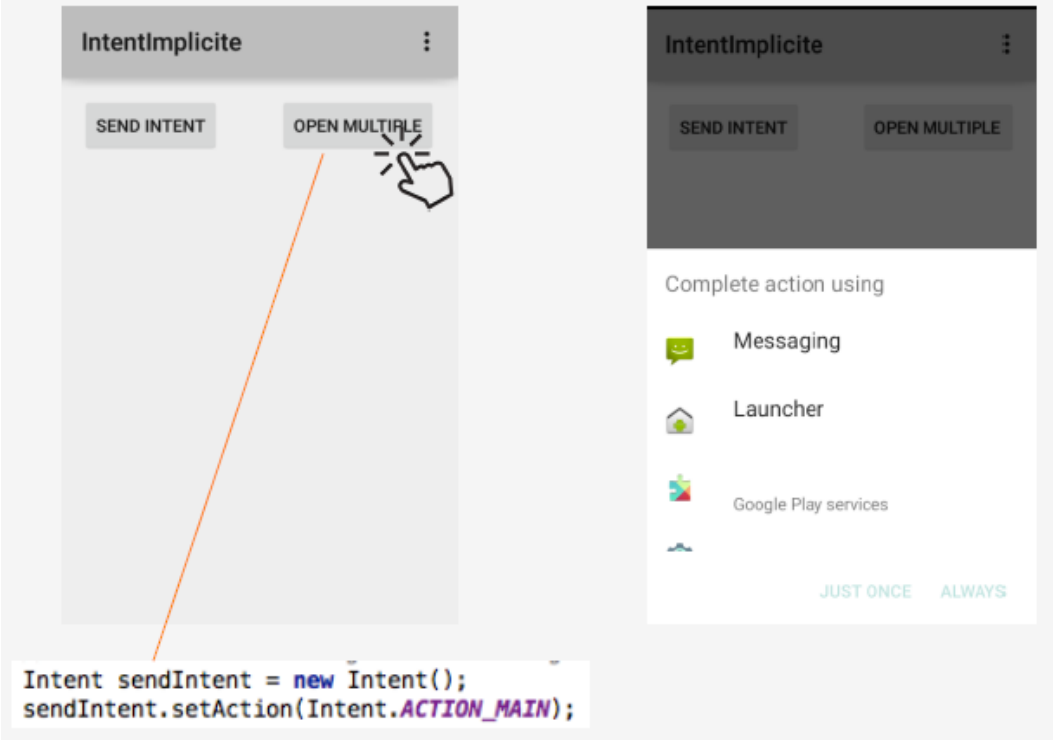
```
public void send(View v){
    // Create the text message with a string
    Intent sendIntent = new Intent();
    sendIntent.setAction(Intent.ACTION_SEND);
    sendIntent.putExtra(Intent.EXTRA_TEXT, "Hello");
    sendIntent.setType("text/plain");

    // Verify that the intent will resolve to an activity
    if (sendIntent.resolveActivity(getPackageManager()) != null) {
        startActivity(sendIntent);
    }else{
        Toast.makeText(this,"The send action could not be performed!",Toast.LENGTH_SHORT).show();
    }
}
```

Eviter que l'application crash si l'activité appelée n'existe pas



➢ **Intent Implicite: Plusieurs Activités Possibles**



6.5 Intents explicites

6.5.1 Arguments et Utilisation

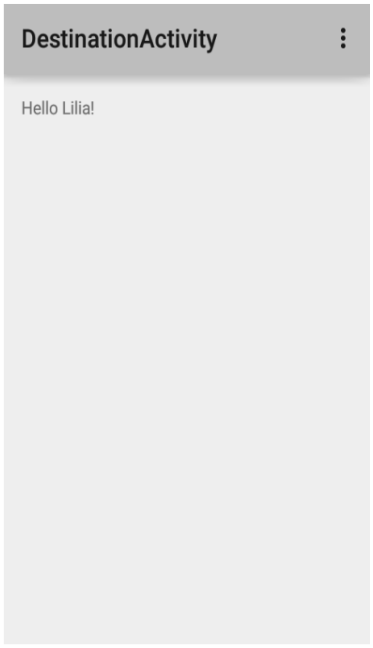
Les principaux arguments d'un Intent explicite sont :

- Le contexte déclenchant l'Intent (en général this, si on le lance à partir de l'activité de départ, ou bien <Activity_class_name>.this)
- La classe destination (en général <Activity_class_name>.class)

Il est typiquement appelé comme suit:
Intent myActivityIntent = new Intent (StartClass.this, EndClass.class) ;
startActivity (myActivityIntent) ;

➢ **Exemple d'Intent explicite :**

```
public void start(View v){
    Intent i = new Intent(this,DestinationActivity.class);
    i.putExtra("name",name.getText().toString());
    startActivity(i);
}
```




```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_destination);

    tv = (TextView)findViewById(R.id.show);

    String name = getIntent().getStringExtra("name");
    tv.setText("Hello "+name+"!");

}
```

6.5.2 Démarrer une Activité avec Résultat

Il est possible d'établir un lien bidirectionnel entre deux activités grâce à un Intent. Pour recevoir un résultat à partir d'une autre activité, appeler **startActivityForResult** au lieu de StartActivity. L'activité destination doit bien sûr être conçue pour renvoyer un résultat une fois l'opération réalisée. Le résultat est envoyé sous forme d'Intent et l'activité principale le recevra dans un **onActivityResult**.

➢ **Exemple d'Intent avec résultat :**

Activité 1

```
public void start(View v){
    Intent i = new Intent(this,DestinationActivity.class);
    i.putExtra("val",Integer.valueOf(texte.getText().toString()));
    startActivityForResult(i, REQUEST_CODE);
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent intent) {

    if (requestCode == REQUEST_CODE) {
        if(resultCode == RESULT_OK){
            Toast.makeText(this,"Resultat = "+intent.getIntExtra("resultat",0),Toast.LENGTH_LONG).show();
        }
        if (resultCode == RESULT_CANCELED) {
            Toast.makeText(this,"Pas de Resultat !",Toast.LENGTH_LONG).show();
        }
    }
}
```

Activité 2

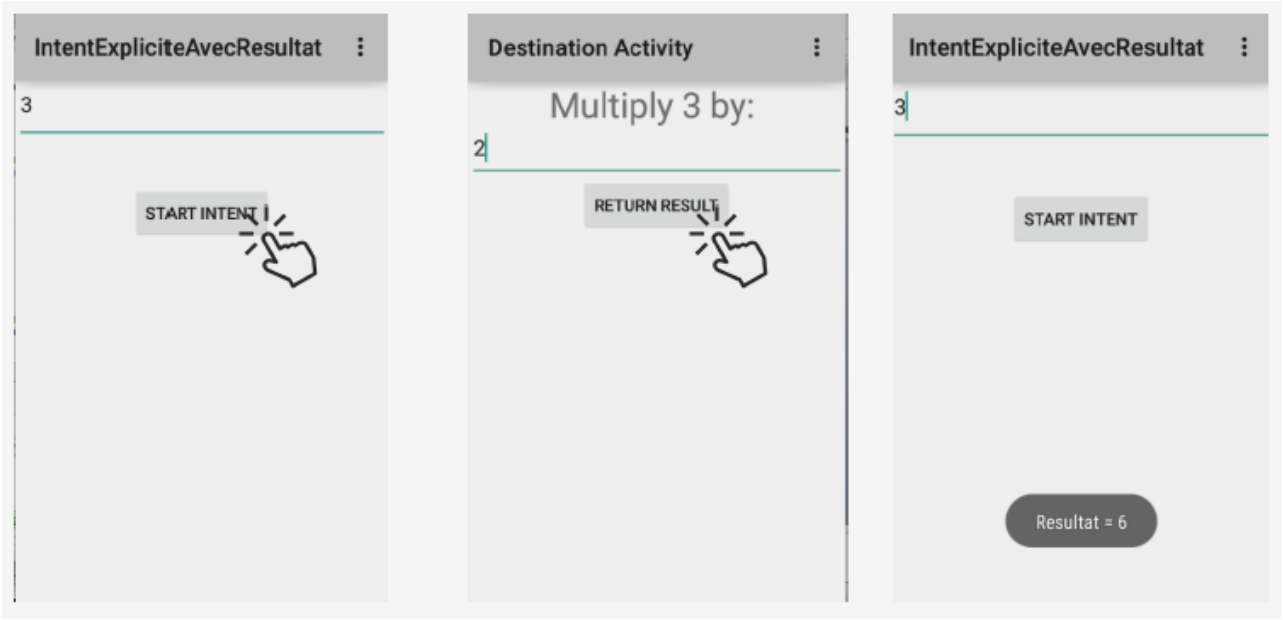
```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_destination);

    tv = (TextView)findViewById(R.id.show);
    mult = (EditText) findViewById(R.id.multEdit);

    valueReceived = getIntent().getIntExtra("val",0);
    tv.setText("Multiply "+valueReceived+" by: ");

}


public void retour (View v){
    Intent returnIntent = new Intent();
    if (mult.getText()!= null) {
        int result = valueReceived * Integer.valueOf(mult.getText().toString());
        returnIntent.putExtra("resultat", result);
        setResult(RESULT_OK, returnIntent);
    }else{
        setResult(RESULT_CANCELED, returnIntent);
    }
    finish();
}
```



6.6 Conclusion

Nous avons découvert tout au long de ce chapitre le fichier de configuration Android et comment se fait la communication entre composants d'applications. Dans le chapitre suivant nous allons découvrir les bases de données sous Android.