

# Chapitre 7 : Communication sous Android

## Introduction

La communication interne d'Android est basée sur l'envoi et la réception de messages exprimant une opération à effectuer. Les messages peuvent être émis à destination d'un autre composant de la même application (une activité, un service...) ou vers n'importe quelle autre application. La classe `android.content.Intent` modélise un tel message.

### I. Les types d'Intentions

Android propose deux types d'Intentions : explicites et implicites. La méthode permettant de lancer une intention (`intent`) est `startActivity`.

`startActivity(Intent i)` est une méthode publique de la classe `Context`. Comme la classe `Activity` est une sous classe de `Context`, on peut appeler `startActivity(...)` directement dans une méthode de la classe `Activity` ainsi que toutes ses sous classes.

#### I.1. Les intentions explicites

Pour créer un Intent explicite il suffit de donner un Context qui appartient au package où se trouve la classe de destination et l'activité destinataire :

```
Intent intent = new Intent(Context context, Class<?> cls);  
startActivity(intent);
```

##### Exemple :

```
Intent i = new Intent(A.this, B.class);  
startActivity(i);
```

Lorsqu'une activité "A" du package "com.gest" lance une deuxième activité "B" avec `startActivity(...)`, l'activité "B" se lance et ne retourne aucune information à l'activité "A" lorsqu'elle se termine. Donc cette méthode s'utilise lorsque l'activité "A" n'a aucun besoin des données de l'activité "B".

##### Remarque :

Il faut déclarer la deuxième activité dans `AndroidManifest.xml` par l'ajout de la ligne suivante :

```
<activity android:name="com.gest.B"></activity> dans <application>...</application>
```

#### I.2. Les intents implicites

Pour les intents implicites, on fera en sorte d'envoyer une requête à un destinataire, sans savoir qui est et c'est à Android de déterminer et de lancer l'application qui exécute la requête. Par exemple, envoyer un SMS sans préciser l'application qui va être utilisée.

Les applications destinataires sont : soit fournies par Android, soit par d'autres applications téléchargées sur le Play Store par exemple.

Pour qu'Android puisse déterminer qui est capable de réceptionner un intent implicite. Il faut au moins fournir deux informations essentielles :

- **Une action** : ce qu'on désire que le destinataire fasse.
- **Un ensemble de données** : sur quelles données le destinataire doit effectuer son action.

## Cours : Développement Mobile

### 2.1. L'action

Une action est une constante qui se trouve dans la classe Intent et qui commence toujours par "ACTION\_" suivi d'un verbe (en anglais) de façon à bien faire comprendre qu'il s'agit d'une action. Si on veut voir quelque chose, on va utiliser l'action ACTION\_VIEW. Par exemple, si on utilise ACTION\_VIEW sur un numéro de téléphone, alors le numéro de téléphone s'affichera dans le composeur de numéros de téléphone.

Voici quelques actions natives parmi les plus utilisées :

Intitulé	Action	Entrée attendue
ACTION_MAIN	Pour indiquer qu'il s'agit du point d'entrée dans l'application	
ACTION_DIAL	Pour ouvrir le composeur de numéros téléphoniques	Un numéro de téléphone
ACTION_CALL	Composer un numéro téléphonique	Un numéro de téléphone
ACTION_SEARCH	Effectuer une recherche	Le texte à rechercher
ACTION_SENDTO	Envoyer un message à quelqu'un	Le numéro de la personne à qui envoyer le message
ACTION_VIEW	Visionner une donnée	Plusieurs utilisations possibles : Une adresse e-mail sera visionnée dans l'application pour les e-mails, un numéro de téléphone dans le composeur, une adresse internet dans le navigateur, etc.
ACTION_WEB_SEA RCH	Effectuer une recherche sur internet	S'il s'agit d'un texte qui commence par « http », le site s'affichera directement, sinon c'est une recherche dans Google qui se fera

### 2.2. Les données

Généralement les données sont envoyées sous forme d'URI. Les URI se comportent d'une manière un peu similaire. La syntaxe d'un URI peut être analysée de la manière suivante (les parties entre accolades {} sont optionnelles) :

<schéma> : <information> { ? <requête> } { # <fragment> }

- Le **schéma** décrit quelle est la nature de l'information. S'il s'agit d'un numéro de téléphone, alors le schéma sera « **tel** », s'il s'agit d'un site internet, alors le schéma sera « **http** », etc.
- **L'information** est la donnée en tant que telle. Cette information respecte aussi une syntaxe qui dépend du schéma. Ainsi, pour un numéro de téléphone, on peut insérer le numéro tel:0606060606, mais pour des coordonnées GPS il faudra séparer la latitude de la longitude à l'aide d'une virgule geo:123.456789,-12.345678. Pour un site internet, il s'agit d'un chemin hiérarchique.
- **La requête** permet de fournir une précision par rapport à l'information.
- **Le fragment** permet enfin d'accéder à une sous-partie de l'information.

#### Exemple 1:

Pour créer un intent qui va ouvrir le composeur téléphonique avec le numéro de téléphone 0606060606, on peut utiliser le code suivant :

```
Uri telephone = Uri.parse("tel:0606060606");
Intent secondeActivite = new Intent(Intent.ACTION_DIAL, telephone);
startActivity(secondeActivite);
```



### Remarque :

En cas de l'existence de plusieurs applications, une boîte de dialogue s'affiche offrant à l'utilisateur le choix de l'application à utiliser pour exécuter l'action (Viber, Skype,... pour l'action ACTION\_DIAL)

### Exemple 2:

Pour créer un objet URI, utiliser la méthode statique Uri.parse(String uri). Par exemple, pour envoyer un SMS à une personne, on utilise l'URI :

```
Uri sms = Uri.parse("sms:0606060606");
```

Il est également possible d'indiquer plusieurs destinataires et un contenu pour ce message :

```
Uri sms = Uri.parse("sms:060606606,060606067?body=Exemple%20de%20diffusion%20de%20message");
```

### Remarque :

Le contenu de la chaîne doit être encodé (%20 au lieu d'espace), sinon on rencontrera des problèmes.

## II. Passage et récupération de paramètres

Les paramètres transférés d'une activité à une autre peuvent être des données de types simples ou bien de type complexe.

### II.1. Paramètres simples et tableaux

La classe Intent propose des méthodes pour passer et récupérer des paramètres de type simple (int, byte, char, CharSequence, float, short, boolean, double, long, String) et des tableaux.

### Exemple :

```
int i = 10 ;
float f = 12.897f ;
String s= "TexteS" ;
boolean b= true ;
CharSequence cs= "TexteCS" ;
int[] Tab = new int[]{15,145,14,225,659} ;
```

### II.2. Utilisation de la classe Intent

Pour le passage d'un paramètre simple ou tableau, la classe Intent possède un ensemble de méthodes qui permettent de le faire passer. Le choix de la méthode à utiliser dépend du type du paramètre à passer.

Pour la lecture de ce paramètre passé, la classe Intent possède un ensemble de méthodes qui permettent de lire ce paramètre. Le choix de la méthode à utiliser dépend du type du paramètre passé.

Le tableau suivant contient ces méthodes :

Type	Méthode de passage du paramètre	Méthode de lecture du paramètre passé
Int	Intent.putExtra(String name, int value)	int getIntExtra(String name, int defaultValue)
Byte	Intent.putExtra(String name, byte value)	byte getByteExtra(String name, byte defaultValue)
Char	Intent.putExtra(String name, char value)	char getCharExtra(String name, char defaultValue)

## Cours : Développement Mobile

CharSequence	Intent putExtra(String name, CharSequence value)	CharSequence getCharsequenceExtra(String name)
Float	Intent putExtra (String name, float value)	float getFloatExtra(String name, float defaultValue)
Short	Intent putExtra(String name, byte value)	short getShortExtra(String name, short defaultValue)
boolean	Intent putExtra(String name, boolean value)	boolean getBooleanExtra(String name, boolean defaultValue)
double	Intent putExtra(String name, double value)	double getDoubleExtra(String name, double defaultValue)
Long	Intent putExtra(String name, long value)	long getLongExtra(String name, long defaultValue)
String	Intent putExtra(String name, String value)	String getStringExtra(String name)
Array	Intent putExtra(String name, Type[] t)	Type[] getTypeArrayExtra(String name)

### Remarque :

Pour les tableaux, il faut remplacer le terme « Type » dans le nom de la méthode de lecture du paramètre passé par le type correspondant. Par exemple, getIntArrayExtra pour un tableau d'entier et getStringtArrayExtra pour un tableau de chaînes de caractères...

### Exemple :

```

public class MainActivity extends Activity {
    private final int[] Tab= new int[]{15,145,554};
    protected void passer() {
        Intent i = new Intent(this, Appl.class);
        i.putExtra("texte", "Param Texte");
        i.putExtra("bool", true);
        i.putExtra("val", 15);
        i.putExtra("t", Tab);
        startActivity(i);
    }
}
public class Appl extends Activity {
    private void recupererParam() {
        Intent i=getIntent();
        String t = i.getStringExtra("texte");
        Boolean b=i.getBooleanExtra("bool", false);
        int max=i.getIntExtra("val", 100);
        int[] tableau = i.getIntArrayExtra("t");
        for (int j : tableau) {
            Log.i("Tab",j) ;
        }
    }
}

```

## II.3. Utilisation de la classe Intent et de la classe Bundle

Dans cette solution, on utilise la méthode Intent putExtras(Bundle b) de la classe Intent pour passer un paramètre de type Bundle. On utilise Bundle getExtras() pour récupérer le Bundle passé en paramètre. Un Intent ne peut passer qu'un seul Bundle.

Un Bundle (paquet en français) est une classe qui permet de stocker un ensemble de paires Clé/Valeur avec la clé est de type String.

Pour le passage d'un paramètre simple/tableau, la classe Bundle possède un ensemble de méthodes qui permettent de le faire passer. Le choix de la méthode à utiliser dépend du type du paramètre à passer.

Pour la lecture de ce paramètre, la classe Bundle possède un ensemble de méthodes qui permettent de lire le paramètre, le choix de la méthode à utiliser dépend du type du paramètre passé :

Le tableau suivant contient ces méthodes :

Type	Méthode de passage du paramètre	Méthode de lecture du paramètre passé
Int	void putInt(String key, int value)	int getInt(String key) int getInt(String key, int defaultValue)
Byte	void putByte(String key, byte value)	byte getByte(String key) byte getByte(String key, byte defaultValue)
Char	void putChar(String key, char value)	char getChar(String key) char getChar(String key, char defaultValue)
CharSequence	void putCharSequence(String key, CharSequence value)	CharSequence getCharSequence(String key) CharSequence getCharSequence(String key, CharSequence defaultValue)
float	void putFloat(String key, float value)	float getFloat(String key) float getFloat(String key, float defaultValue)
short	void putShort(String key, short value)	short getShort(String key) short getShort(String key, short defaultValue)
boolean	void putBoolean(String key, boolean value)	boolean getBoolean(String key) boolean getBoolean(String key, boolean defaultValue)
double	void putDouble(String key, double value)	double getDouble(String key) double getDouble(String key, double defaultValue)
long	void putLong(String key, long value)	long getLong(String key) long getLong(String key, long defaultValue)
String	void putString(String key, String value)	String getString(String key) String getString(String key, String defaultValue)
Array	Intent putTypeArray(String key, Type[] t)	Type[] getTypeArray(String key)

### Remarque :

Pour les tableaux, il faut remplacer le terme « Type » dans le nom de la méthode de lecture du paramètre passé par le type correspondant. Par exemple, getIntArray pour un tableau d'entier et getStringtArray pour un tableau de chaînes de caractères...

### Exemple :

```
public class MainActivity extends Activity {  
    private final int[] Tab= new int[]{15,145,554};  
    protected void passer() {  
        Intent i = new Intent(this, App2.class);  
        Bundle b = new Bundle();  
        b.putString("message", "Param Texte");  
        b.putBoolean("bool", true); b.putInt("val", 100);  
        b.putIntArray("t", Tab);  
        i.putExtras(b);  
        startActivity(i);  
    } }  
public class App2 extends Activity {  
    private void recupererParam() {  
        Intent i = getIntent();  
        Bundle bundle = i.getExtras();  
        String m = bundle.getString("message");  
        Boolean b= bundle.getBoolean("bool");  
        int max=bundle.getInt("val");  
        Array tableau = bundle.getIntArray("t");  
    } }
```

## II.4. Paramètre Complexé

Les types complexes sont les types qui ne sont pas simples. Généralement, ils sont définis par le programmeur. Par exemple : Personne, Matière, Voiture, Pays...

Les solutions précédentes ne permettent pas de passer un tel paramètre. Cependant, il existe deux solutions possibles :

- Utiliser l'interface `Serializable` de Java
- Utiliser l'interface `Parcelable` d'Android

### 4.1. Implémenter l'interface Serializable

Dans cette solution, il faut :

- Que la classe de l'objet à passer en paramètre implémente l'interface `Serializable`,
- Utiliser la méthode `Intent.putExtra(String name, Serializable value)` de la classe `Intent` ou la méthode `void putSerializable(String key, Serializable value)` de la classe `Bundle` pour passer un objet serialisable,
- Utiliser la méthode `Serializable getSerializableExtra (String name)` de la classe `Intent` ou la méthode `Serializable getSerializable(String key)` de la classe `Bundle` pour récupérer un objet serialisable.

### Exemple :

```
public class Personne implements Serializable {  
    private String nom;  
    public Personne(String nom) {  
        this.nom = nom;  
    }
```

```
public String getNom() {  
    return nom;  
}  
}  
  
public class MainActivity extends Activity {  
private void passer() {  
Intent i = new Intent(this, AffichagePersonne.class);  
i.putExtra("p",new Personne("Mohamed"));  
startActivity(i);  
}  
}  
  
public class AffichagePersonne extends Activity {  
private void afficher() {  
Intent i = getIntent();  
Personne p = (Personne) i.getSerializableExtra("p");  
String message = "Nom: " + p.getNom()+"\n";  
Log.i("Personne",message);  
}  
}
```

### 4.2. Implémenter l'interface Parcelable

Dans cette solution, il faut :

- Que la classe de l'objet à passer en paramètre implémente l'interface Parcelable, et redéfinit les deux méthodes abstraites : int describeContents() et void writeToParcel(Parcel dest, int flags) :
  - int describeContents() qui permet de définir les paramètres spéciaux dans le Parcelable
  - void writeToParcel(Parcel dest, int flags), avec dest représente le Parcel dans lequel les attributs de l'objet seront insérées et flags indique un entier qui vaut la plupart du temps 0.

C'est dans cette classe que nous allons écrire dans le Parcel qui transmettra le message. Il faut que :

- Que la classe de l'objet à passer en paramètre ajoute un constructeur qui prend en paramètre un Parcel.
- Que la classe de l'objet à passer en paramètre ajoute une constante :  
public static final Parcelable.Creator<Personne> CREATOR
- Utiliser la méthode Intent.putExtra (String name, Parcelable value) de la classe Intent ou la méthode void putParcelable(String key, Parcelable value) de la classe Bundle pour passer un objet parcelable,
- Utiliser la méthode Parcelable.getParcelableExtra(String name) de la classe Intent ou la méthode Parcelable.getParcelable (String key) de la classe Bundle pour récupérer un objet parcelable,
- Utiliser la méthode Intent.putExtra(String name, Parcelable[] value) de la classe Intent ou la méthode putParcelableArray(String key, Parcelable[] value) de la classe Bundle pour passer un tableau d'objets parcelables,

## Cours : Développement Mobile

- Utiliser la méthode `Parcelable[] getParcelableArrayExtra(String name)` de la classe `Intent` ou la méthode `Parcelable[] getParcelableArray(String key)` de la classe `Bundle` pour récupérer un tableau d'objets parcelables,

### Exemple :

```
public class Personne implements Parcelable {  
    private String nom;  
    public Personne(String nom) {  
        this.nom = nom;  
    }  
    public Personne(Parcel in) {  
        this.nom = in.readString();  
    }  
  
    public String getNom() {  
        return nom;  
    }  
    @Override  
    public int describeContents() {  
        return 0;  
    }  
  
    @Override  
    public void writeToParcel(Parcel dest, int flags) {  
        dest.writeString(nom);  
    }  
    public static final Parcelable.Creator<Personne> CREATOR = new  
    Parcelable.Creator<Personne>() {  
        @Override  
        public Personne createFromParcel(Parcel source) {  
            return new Personne(source);  
        }  
        @Override  
        public Personne[] newArray(int size) {  
            return new Personne[size];  
        }  
    };  
}  
  
-----  
public class MainActivity extends Activity {  
    private void passer() {  
        Intent i = new Intent(this, AffichagePersonne.class);  
        i.putExtra("p", new Personne("Mohamed"));  
        startActivity(i);  
    }  
    public class AffichagePersonne extends Activity {  
        private void afficher() {  
            Intent i = getIntent();  
            Personne p = (Personne) i.getParcelableExtra("p");  
            String message = "Nom: " + p.getNom() + "\n";  
            Log.i("Personne", message);  
        }  
    }  
}
```

### III. Récupération d'un résultat

Le démarrage d'une autre activité, n'a pas besoin d'être une opération à sens unique. Il est possible de démarrer une autre activité et recevoir un résultat en retour. Par exemple, l'application peut démarrer une deuxième activité et recevoir le résultat d'une opération quelconque en retour ou bien démarrer l'appareil photo et recevoir la photo capturée. Pour aboutir à cette finalité, il faudrait utiliser la classe « `ActivityResultLauncher` » de la bibliothèque AndroidX.

Cette classe fournit des composants pour lancer une deuxième activité qui effectue un traitement donné et envoie par la suite un résultat qui sera récupéré par la première activité.

#### III.1. Lancer une activité

Il faut commencer par définir une variable membre. Il en faut une par activité qui sera lancée ultérieurement : la classe « `ActivityResultLauncher` » fournit la méthode `registerForActivityResult` pour enregistrer le rappel de résultat. Cette variable permettra de lancer la deuxième activité avec la méthode `launch()`.

#### >MainActivity.java

```
//Déclarer la variable DemarrerActivite
ActivityResultLauncher<Intent> DemarrerActivite =
registerForActivityResult(
    new ActivityResultContracts.StartActivityForResult(),
    new ActivityResultCallback<ActivityResult>() {
        @Override
        public void onActivityResult(ActivityResult result) {
            // cette fonction sera développée dans la section suivante
        }
    });
//Lancer la deuxième activité
Intent intent = new Intent(MainActivity.this, SecondActivity.class);
DemarrerActivite.launch(intent);
```

La variable `DemarrerActivite` doit être initialisée avec la méthode `registerForActivityResult()`. Cette méthode prend un `ActivityResultContracts` et un `ActivityResultCallback` en arguments et renvoie un `ActivityResultLauncher` en sortie qui est utilisé pour lancer la deuxième activité :

- **ActivityResultContracts** définit le type d'action ou d'interaction (Intent) nécessaire pour produire un résultat ainsi que le type de sortie du résultat. La classe « `ActivityResultLauncher` » fournit des contrats par défaut pour les actions d'intention de base telle que prendre une photo.
- **ActivityResultCallback** est une interface avec une méthode unique à redéfinir `onActivityResult()`.

#### III.2. Récupérer le résultat

La classe `Activity` contient les deux constantes `RESULT_OK` et `RESULT_CANCELED` qui permettent d'indiquer si l'activité s'est terminée normalement (OK) ou l'utilisateur l'a abandonnée (CANCELED).

Si l'activité appelée admet des données à transmettre à l'activité appelante alors elle utilise la méthode `setResult(int resultCode, Intent data)` avant d'appeler `finish()`.

### ❖ SecondActivity.java

```
Intent i = new Intent();
i.putExtra("nom", "Mohamed");
setResult(RESULT_OK, i);
finish();
```

### ❖ MainActivity.java

```
ActivityResultLauncher<Intent> ActivityLauncher = registerForActivityResult(
    new ActivityResultContracts.StartActivityForResult(),
    new ActivityResultCallback<ActivityResult>() {
        @Override
        public void onActivityResult(ActivityResult result) {
            if (result.getResultCode() == RESULT_OK) {
                Intent i = result.getData();
                String name = i.getStringExtra("nom");
                tvResult.setText(name);
            } else {
                Toast t = Toast.makeText(getApplicationContext(),
                    "L'utilisateur a annulé!", Toast.LENGTH_LONG);
                t.show();
            }
        }
    });
}
```

## Conclusion

Dans ce chapitre, nous avons introduit les intentions implicites et explicites et utilisé les différentes méthodes de passage et de récupération de paramètres entre 2 activités Android (paramètres simples, tableaux, paramètres complexes) ainsi que les étapes nécessaires pour la récupération du résultat d'une intention.

Le chapitre suivant sera consacré à la persistance des données sous Android.