Les intents

Introduction

Les Intents sont des messages asynchrones qui permettent aux composants d'une application de demander des fonctionnalités à d'autres composants. Par exemple, une activité peut démarrer une autre activité. Android supporte deux types d'Intents, les Intents explicites et implicites.

- **Les intents explicites**

L'application définit les composants cibles directement dans l'Intent.
`Intent(Context, Class<?>) pour l'appels explicite`

Pour créer un intent explicite il suffit de donner un `Context` qui appartient au package où se trouve la classe de destination :
Intent i = new Intent(Context context, Class<?> cls);

Par exemple, si la classe de destination appartient au package du `Context` actuel :
Intent i = new Intent(Activite_de_depart.this, Activite_de_destination.class);

```
Intent i = new Intent(this, AutreActivity.class);
```

On peut utiliser la méthode `setClass` pour utiliser un intent:

```
Intent setClass(Context packageContext, Class<?> cls)
```

- `packageContext` : `Context` qui appartient au même package que le composant de destination
- `cls` : nom de la classe qui héberge cette activité.

```
Intent i = new Intent();
i.setClass(this, activity2.class);
```

Démarrer une activité

Pour démarrer une activité utilisez la méthode `startActivity(Intent).` Cette méthode est définie par l'objet Context hérité par l'activité.

Exemple

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

Dans certain cas, il ne faut pas mettre **this** mais faire appel à la méthode **getApplicationContext()** si l'objet manipulant l'*Intent* n'hérite pas de **Context**.

Déclaration des activités

On indique la classe de l'activité à lancer (par exemple dans une application multi interfaces)
Cette activité doit être déclarée dans `AndroidManifest.xml` de l'application par :

```
 <activity android:name=".classe ">
```

```
<application
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
```

```
<activity android:name=".MainActivity"> //declaration de l'activité
    <intent-filter>
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER"
    </intent-filter>
</activity>
<activity android:name=".classe1">
```

Transfert de données vers l'activité de destination

Les *Intent* peuvent transporter des informations vers l'activité cible. On appelle ces informations des *Extra* et sont manipulés par les méthodes **getExtra** et **putExtra**.

Types de données : types standards

Les Intents ont un champ "extra" qui leur permet de contenir des données à véhiculer entre les activités.
Un extra est une clé à laquelle on associe une valeur.
Pour insérer un extra, on utilise la méthode putExtra :
`Intent putExtra(String key, type value)` avec `key` la clé de l'extra et `value` la valeur associée.
Pour récupérer les extras d'un Intent on utilise la méthode `Bundle getExtras(),` les couples clé/valeurs sont contenus dans le `Bundle`.

L'objet Intent est récupéré avec la méthode getIntent().
Pour obtenir les données, on peut utiliser la méthode getIntent().getExtras().

Exemple

```
Intent i = new Intent(this, AutreActivity.class);
i.putExtra("myKey", Value);
startActivity(i);
```

Dans l'activité de destination :

```
Intent i = getIntent();
Bundle extras = i.getExtras();
if (extras == null)
return;
String valeur = extras.getString(key); // récupérer les données par la clé
if (valueur != null) {
….
}
```

Pour récupérer un extra précis à l'aide de sa clé et de son type on utilise la méthode
`getxxxExtra(String key, X defaultValue),`

`xxx` le type de l'extra et `defaultValue` la valeur qui sera retournée si la clé passée ne correspond à aucun extra de l'intent.

Exemple

```
   Intent i = getIntent();
 int a = i.getIntExtra("key", 0);
```

En revanche, pour les types plus complexes tels que les tableaux, on ne peut préciser des valeurs par défaut. Par exemple, pour un tableau de `float` on peut utiliser la méthode:
`float[] getFloatArrayExtra(String key).`

Exemple

```
Intent i = new Intent();
String[] noms = new String[] {"ali", "aziz"};
i.putExtra("cle1", noms); //On envoie les données

// on récupère les données
String[] T = i.getStringArrayExtra("cle1");
```

Remarque

Il existe deux façons de lancer l'Intent, selon que le composant de destination renvoie ou non une réponse.

1- Intent Sans valeur de retour

Dans une première activité, on met un bouton. Un clic sur ce bouton lance une seconde activité et lui envoie une donnée.

L'activité principale

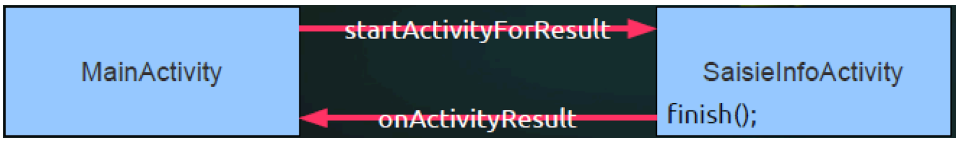
```
public class MainActivity extends Activity {
 private Button b = null;
 public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);

  b = (Button) findViewById(R.id.bouton1);
  b.setOnClickListener(new View.OnClickListener() {
   public void onClick(View v) {
    Intent i = new Intent(MainActivity.this, IntentActivity.class);

    i.putExtra("AGE", 31);
    startActivity(i);
   }  });
 }}
```

La seconde activité

```
public class IntentActivity extends Activity {
 public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.layout_example);
  // On récupère l'intent qui a lancé cette activité
  Intent i = getIntent();
  // Puis on récupère l'âge donné dans l'autre activité, ou 0 si cet extra n'est pas dans l'intent
  int age = i.getIntExtra("AGE", 0);     // 0 par défaut
  if(age != 0)
   // Traiter l'âge
 }}
```

2- Avec retour : Récupérer le résultat d'une sous-activité

Cette fois, on veut qu'au retour de l'activité qui vient d'être appelée cette dernière renvoie un *feedback* à l'activité source.



Pour cela, on utilise la méthode
`void startActivityForResult(Intent i, int requestCode),`
avec `requestCode` un code passé qui permet d'identifier de manière unique un intent et doit être supérieur ou égal à 0.
Quand l'activité appelée (destination) s'arrête, la première méthode de *callback* appelée se trouve dans l'activité source. Cette méthode s'appelle `onActivityResult` :

`void onActivityResult(int requestCode, int resultCode, Intent i).`

- **`requestCode`** : le même code que celui de la méthode `startActivityForResult` qui permet de repérer quel intent a provoqué l'appel de l'activité dont le cycle vient de s'interrompre.
- **`resultCode`** : code renvoyé par l'activité destination, indique comment elle s'est terminée (typiquement `Activity.RESULT_OK` si l'activité s'est terminée normalement, ou `Activity.RESULT_CANCELED` s'il y a eu un problème ou qu'aucun code de retour n'a été précisé).
- **`i`** : un intent qui contient éventuellement des données.

Quand la sous-activité se termine, elle retourne des données dans la méthode finish() à l'appelant avec un intent.
Dans l'activité destination, on peut définir un résultat (à envoyer par putExtra à l'activité principale) avec la méthode `void setResult(int resultCode, Intent data)` ou
`void setResult (int resultCode)`

```
public void finish() {
  Intent i = new Intent();
  i.putExtra("returnKey1", "Swinging on a star. ");
  i.putExtra("returnKey2", "You could be better then you are. ");
  setResult(RESULT_OK, i);
  super.finish();
}
```

Exemple
Activité principale

```
public class MainActivity extends Activity {
  private Button mPasserelle = null;
  // L'identifiant de notre requête
  public final static int CHOOSE_BUTTON_REQUEST = 0;

 // L'identifiant de la chaîne de caractères qui contient le résultat de l'intent
  public final static String BUTTONS = " com.example.projet1.Boutons";

public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
   mPasserelle = (Button) findViewById(R.id.passerelle);

   mPasserelle.setOnClickListener(new View.OnClickListener() {
     public void onClick(View v) {
```

```
      Intent i = new Intent(MainActivity.this, IntentExample.class);
          i.putExtra("Valeur1", " Valeur un pour ActivityTwo ");
          i.putExtra("Valeur2", " Valeur deux pour ActivityTwo");
    startActivityForResult(i, CHOOSE_BUTTON_REQUEST);
    }});
 }
 protected void onActivityResult(int requestCode, int resultCode, Intent data) {
  // On vérifie à quel Intent on fait référence ici à l'aide de notre identifiant
  if (requestCode == CHOOSE_BUTTON_REQUEST) {
   // On vérifie aussi que l'opération s'est bien déroulée
   if (resultCode == RESULT_OK) {
   Toast.makeText(this, "Vous avez choisi le bouton " + data.getStringExtra(BUTTONS),
Toast.LENGTH_SHORT).show();
//Toast.makeText(this, data.getExtras().getString("BUTTONS "), Toast.LENGTH_SHORT).show();
    } }  }}
```

La seconde activité

```
public class IntentExample extends Activity {
 private Button mButton1 = null;
Intent I;
 public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.layout_example);
  i = getIntent();
Toast.makeText(this, i.getExtras().getString("valeur1"), Toast.LENGTH_SHORT).show();

mButton1 = (Button) findViewById(R.id.button1);
  mButton1.setOnClickListener(new View.OnClickListener() {
   public void onClick(View v) {
    i = new Intent();
    i.putExtra(MainActivity.BUTTONS, "1");
    setResult(RESULT_OK, i);
    finish();
   }   });
 }}
```

Exercice : réaliser deux taches différentes selon le résultat envoyé par la seconde activité.

```
public void onClick(View v) {
setResult(1);
finish();
}
protected void onActivityResult(int requestCode, int resultCode, Intent
data){
if (resultCode == 1)
Toast.makeText(this, "tache 1", Toast.LENGTH_LONG).show();

if (resultCode == 2)
Toast.makeText(this, "tache 2", Toast.LENGTH_LONG).show();
}
```

Les parcelables

Pour passer un objet complexe entre activités, en utilisant le **Bundle**, il suffit que la classe représentant l'objet implémente l'interface **Parcelable.**

C'est l'équivalent de la sérialisation pour le **passage d'objet entre différentes activités ou différents processus.**
`Bundle` ne peut prendre que les objets sérialisables. Un objet est sérialisable s'il implémente l'interface `Parcelable`.

Un `Parcelable` est un objet qui sera transmis à un `Parcel`.
L'objectif des `Parcel` est de transmettre des messages entre différents processus du système.

Pour implémenter l'interface `Parcelable`, il faut redéfinir deux méthodes **`describeContents`**`()`, et **`writeToParcel`**`(Parcel dest, int flags)`,

- `int` **`describeContents`**`()`:Sert à décrire le contenu du Parcel et plus précisément le nombre d'objet spéciaux contenus dans le Parcel. Elle définit s'il y a des paramètres spéciaux dans le `Parcelable`.
Parmi les objets spéciaux on trouve les `FileDescriptor`. Ainsi, s'il n'y a pas d'objet de type `FileDescriptor`, elle renvoie 0, sinon elle renvoie `Parcelable.CONTENT_FILE_DESCRIPTOR`.

- `void` **`writeToParcel`**`(Parcel dest, int flags)` : Sert à écrire l'objet dans un Parcel.
  - `dest` est le `Parcel` dans lequel les attributs de l'objet seront insérés
  - `flags` un entier.

Les attributs sont à insérer dans le `Parcel` dans l'ordre de déclaration dans la classe.

Etapes de réalisation d'un parcelable

```
public class Contact implements Parcelable{
  private String mNom;
  private String mPrenom;
  private int mAge;

  public Contact(String pNom, String pPrenom, int pAge) {
    mNom = pNom;
    mPrenom = pPrenom;
    mAge = pAge;
}

  public int describeContents() {
    //On renvoie 0, car notre classe ne contient pas de FileDescriptor
    return 0;
  }
  public void writeToParcel(Parcel dest, int flags) {
    // On ajoute les objets dans l'ordre dans lequel on les a déclarés dans la classe contact
    dest.writeString(mNom);
    dest.writeString(mPrenom);
    dest.writeInt(mAge);
  }

public Contact(Parcel   in) {
  mNom = in.readString();
```

```
  mPrenom = in.readString();
  mAge = in.readInt();
}
}
```

Une dernière étape est nécessaire, il s'agit de créer un objet **CREATOR** de la classe **Parcelable** ainsi qu'un constructeur prenant comme argument un **Parcel** pour la classe Contact.

Creator

Il faut ajouter un champ statique de type `Parcelable.Creator` et qui s'appellera impérativement "CREATOR", sinon on ne peut reconstruire un objet qui est passé par un `Parcel`.

Exemple

```
public static final Parcelable.Creator<Contact> CREATOR = new Parcelable.Creator<Contact>() {
public Contact createFromParcel(Parcel source) {
    return new Contact(source);
  }
 public Contact[] newArray(int size)  {
    return new Contact[size];
  }};
```

Le CREATOR permet d'indiquer comment l'objet de type Parcelable sera créé.
L'ordre de lecture dans le constructeur est le même que dans la méthode writeToParcel.

Utilisation

Un `parcelable` peut être ajouté dans un intent avec `putExtra` et on peut le récupérer avec `getParcelableExtra ou getParcelable`.

```
Intent i = new Intent(this, IntentExample.class);
Contact c = new Contact("benali", "aziz", 20);
i.putExtra("user", c);
```

Ensuite créer la deuxième activité.

```
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.result);
  Contact c = (Contact) getIntent().getExtras().getParcelable("user");
    // Contact c = i.getParcelableExtra("user");
Utiliser l'objet  c
}
```

Les intents implicites

Dans le cas des Intents implicites, la requête est envoyée à un destinataire sans le connaitre. Les applications destinataires sont soit fournies par Android, soit par d'autres applications organisées selon une certaine syntaxe.
Un Intent implicite spécifie ***l'action*** à exécuter et une ***URI* (Uniform Resource Identifier)** optionnelle qui sera utilisée par cette action pour spécifier pour les données.

- Pour les intents explicites, le composant cible est nommé, on active alors de tel composant.
- Pour les intents implicites le composant cible n'est pas nommé, c'est au système de trouver la ou les activités qui peuvent convenir à travers l'action et data.

Exemple : afficher une page Web
Deux composantes sont intéressantes : les actions et les données.

```
Intent intent=new Intent(Intent.ACTION_VIEW);  //action
intent.setData(Uri.parse("http://www.google.com"));   //data
startActivity(intent);
```

Lorsque `startActivity()` est appelé, le système examine toutes les applications installées pour déterminer celles qui peuvent gérer ce type d'intention (une intention avec l'action spécifiée et qui contient des données spécifiées. Si une seule application peut la gérer, cette application s'ouvre immédiatement et son intention est spécifiée.
Si plusieurs activités acceptent l'intention, le système affiche une boîte de dialogue et l'utilisateur peut choisir quelle application utiliser.

Un Intent implicite doit spécifier les données et ***l'action***.

Les données

Les données sont formatées à l'aide des URI (une chaîne de caractères qui permet d'identifier un endroit).

- **L'URI (Uniform Resource Identifier)**

Les données sont formatées à l'aide des URI. Un URI est une chaîne de caractères qui permet d'identifier un endroit. Par exemple sur internet, ou dans le cas d'Android sur le périphérique ou une ressource.
Afin d'étudier les URI, on peut faire l'analogie avec les adresses URL (Uniform Resource Locator, ou localisateur uniforme de ressource) qui permettent d'accéder à des sites internet.

La forme générale d'un URI est analogue aux URL :
https://www.google.com/webhp?sourceid=chrome-instant&ion=1&espv=2&ie=UTF-8#q=astronomie

Uri.parse() crée un objet Uri en utilisant le paramètre String. Cet objet permet de faire référence à des ressources.
Pour créer un objet URI, on utilise la méthode statique Uri.parse(String uri).
Par exemple, pour envoyer un SMS à une personne, l'URI sera :

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

Mais on peut aussi indiquer plusieurs destinataires et un corps pour ce message :

```
sms=Uri.parse("sms:0606060606,0606060607?body=Salut%20les%20amis");
```

- **Type MIME**

Multipurpose Internet Mail Extensions (*MIME*) ou **Extensions multifonctions du courrier Internet** : pour supporter des textes sous différents codage de caractères autres que l'ASCII, des contenus non textuels.

Si une donnée est accompagnée du type MIME `text`, alors les données sont du texte. Mais on trouve aussi `audio` et `video`...
Pour affiner les informations sur les données, on peut préciser un sous-type, par exemple `audio/mp3` et `audio/wav` sont deux types MIME.

Intent.setType(String type)
Ex : Intent.setType(image/jpeg)
Intent.setDataAndType(Uri data, String type)

Exemple

```
String s=editText1.getText().toString();
Intent intent=new Intent(Intent.ACTION_VIEW, Uri.parse(s));
startActivity(intent);
```

```
Intent i =new Intent(Intent.ACTION_VIEW, Uri.parse("tel:+15555555555"));
ou
Intent i = new Intent(Intent.ACTION_VIEW);
i.setData(Uri.parse("tel:+15555555555"));
```

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 comprendre qu'il s'agit d'une action.
Par exemple, pour visualiser une donnée, on utilise l'action `ACTION_VIEW`.
Si on utilise une `ACTION_VIEW` sur un numéro de téléphone, alors le numéro de téléphone s'affichera dans le composeur de numéros de téléphone.

```
Intent i = new Intent(Intent.ACTION_VIEW);
i.setData(Uri.parse("tel:+15555555555"));
```

Exemple : visualiser une page web :

```
Intent i = new Intent(android.content.Intent.ACTION_SENDTO, Uri.parse(uri));
startActivity(i);
```

Si un Intent implicite est envoyé au système Android, il recherche tous les composants qui sont enregistrés avec l'action spécifique et le type de données approprié.
Si un seul composant est trouvé, Android démarre directement ce composant. Si plusieurs composants sont identifiés par le système Android, une boîte de dialogue de sélection permet à l'utilisateur d'indiquer quel composant utiliser.

Syntaxe des Actions

```
//New Intent(Intent.ACTION_VIEW);
Ou
Intent i=new Intent();
i.setAction(Intent.ACTION_VIEW);
```

Exemple : Créer un intent qui ouvre le composeur téléphonique dans une nouvelle activité :

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

Quelques exemples d'actions parmi les plus utilisées

Intitulé	Action
ACTION_MAIN	Pour indiquer qu'il s'agit du point d'entrée dans l'application
ACTION_DIAL	Pour ouvrir le composeur de numéros téléphoniques
ACTION_SENDTO	Envoyer un message à quelqu'un
ACTION_VIEW	Permet de visionner une donnée
ACTION_WEB_SEARCH	Effectuer une recherche sur internet

Résolution des intents : Recevoir et filtrer les Intents (filtres d'Intents)

Pour annoncer les intentions implicites que votre application peut recevoir, déclarez un ou plusieurs filtres d'intention pour chacun de ses composants avec un élément `<intent-filter>` dans le fichier manifeste.

Chaque filtre d'intention spécifie le type d'intentions qu'il accepte en fonction de l'action, des données et de la catégorie de l'intention.

Le système fournit une intention implicite à votre composant d'application uniquement si l'intention peut passer par l'un de vos filtres d'intention.

Chaque filtre d'intention est défini par un élément `<intent-filter>` du fichier manifeste de l'application, imbriqué dans le composant d'application correspondant (tel qu'un élément `<activity>`).

La résolution d'intents est un processus d'association des intents avec les activités voulant les recevoir.

Les filtres d'intents permettent de savoir quels intents seront traités par une activité.
Si un composant n'a aucun filtre, il ne peut recevoir que des intents explicites.

Dans le `<intent-filter>`, vous pouvez spécifier le type d'intention à accepter en utilisant un ou plusieurs de ces trois éléments: action, data et catégorie.

L'action

Indique le type d'action effectuée par l'activité (par exemple affichage, édition …).
Permet de filtrer les opérations en fonction du champ `Action` d'un intent dans le nœud <intent-filter>.
Pour réussir ce test, l'action spécifiée dans l'objet Intent doit correspondre à l'une des actions répertoriées dans le filtre.

- Un intent sera accepté si tout ce qui se trouve dans son champ Action se trouve parmi les actions du filtre, comme (`New Intent(Intent.ACTION_VIEW)`)
- Et si un intent ne précise pas d'action, alors il sera automatiquement accepté pour ce test.
- Si le filtre n'a aucune action, alors tous les intents échouent le test.

Remarque

Les intents explicites sont toujours acceptés puisqu'ils n'ont pas de champ `Action`,

Exemple

```
<activity>
 <intent-filter>
  <action android:name="android.intent.action.VIEW" />
  <action android:name="android.intent.action.SENDTO" />
 </intent-filter>
</activity>
```

Cette activité ne pourra intercepter que les intents qui ont dans leur champ action `ACTION_VIEW` et/ou `ACTION_SENDTO`, car *toutes* ses actions sont acceptées par le filtre.

Comme :

```
public void onClick(View v) {
Intent intent = new Intent("android.intent.action.VIEW ");
startActivity(intent);
};
```

Mais si un intent a pour action `ACTION_VIEW` et `ACTION_SEARCH`, alors il ne sera pas accepté, car une de ses actions n'est pas acceptée par le filtre.

La catégorie

Les catégories d'*Intent* permettent de grouper les applications par grands types de fonctionnalités (clients emails, navigateurs, players de musique, etc...). Elles permettent d'ajouter des informations supplémentaires. Par exemple CATEGORY_BROWSABLE indique une activité qui peut être appelée par un navigateur.

- Pour passer ce test il faut que *toutes* les catégories de l'Intent correspondent à des catégories du filtre.
- Le filtre peut contenir des catégories supplémentaires, mais il ne peut en omettre aucun qui soit dans l'intention.
- Les catégories du filtre doivent être le super-ensemble des catégories de l'objet Intent
- Un Intent sans catégories doit toujours réussir ce test, quel que soit le contenu du filtre

Si un Intent n'a pas de catégorie, alors il passe automatiquement ce test, mais dès qu'un Intent est utilisé avec la méthode `startActivity()`, alors on lui ajoute la catégorie `CATEGORY_DEFAULT`.
Android traite tous les intents implicites passés à startActivity () comme s'ils contenaient au moins une catégorie: `"android.intent.category.DEFAULT"`
Donc, pour qu'un composant accepte les Intents implicites, il faut rajouter cette catégorie au filtre.
Pour les actions et les catégories, la syntaxe est différente entre Java et XML.

Java

Action : `ACTION_VIEW`, on utilise `android.intent.action.VIEW`
Catégorie : `CATEGORY_DEFAULT` on utilise `android.intent.category.DEFAULT`.

Filtres d'Intent

```
<activity>
```

```
 <intent-filter>
  <action android:name="android.intent.action.VIEW" />
  <action android:name="android.intent.action.SEARCH" />
  <category android:name="android.intent.category.DEFAULT" />
  <category android:name="com.example.exe4.intent.category.categorie1"/>
 </intent-filter>
</activity>
```

Remarque
Ci-dessus, il faut que l'intent ait pour action ACTION_VIEW et/ou ACTION_SEARCH. Mais pour les catégories, il doit avoir CATEGORY_DEFAULT *et* CATEGORY_categorie1.

Quelques valeurs prédéfinies :
Actions
android.intent.action.VIEW affichage de données
android.intent.action.EDIT affichage de données pour édition par l'utilisateur
android.intent.action.MAIN activité principale d'une application
android.intent.action.CALL appel téléphonique
android.intent.action.WEB_SEARCH recherche sur le WEB

Catégories
android.intent.category.LAUNCHER activité proposée au lancement par Android
android.intent.category.DEFAULT activité pouvant être lancée explicitement
android.intent.category.BROWSABLE peut afficher une information désignée par un lien
android.intent.category.TAB activité associée dans un onglet d'interface (TabHost).

Par exemple pour répondre à un clic sur un lien http://google.com, il faut construire un filtre d'Intent utilisant la catégorie définissant ce qui est "navigable" tout en combinant cette catégorie avec un type d'action signifiant que l'on souhaite "voir" la ressource. Il faut en plus utiliser le tag data dans la construction du filtre:

❑ *Les données*

`Data` indique le type de données transmises à l'activité lancée ou le type de réponse attendue ainsi que le protocole (`http`, `content`, `file` …).
Il est possible de préciser plusieurs informations sur les données que cette activité peut traiter comme android:scheme ou android:mimeType.

Chaque élément <data> peut spécifier un type de données (type de média MIME) et un URI.

Par exemple, si une application traite des fichiers textes qui proviennent d'Internet, on aura besoin du type "texte" et du schéma "Internet" :

```
<data android:mimeType="text/plain" android:scheme="http" />
<data android:mimeType="text/plain" android:scheme="https" />
```

- Un Intent qui contient à la fois un URI et un type de données (ou un type de données pouvant être déduit de l'URI) ne transmet la partie du type de données du test que si son type correspond à un type répertorié dans le filtre.

- L'Intent passe le test si son URI correspond à un URI dans le filtre ou s'il a un contenu alors le filtre ne spécifie pas d'URI.

Remarques

- Les activités pouvant démarrer des applications ont des filtres avec "`android.intent.action.MAIN`" spécifié comme action.

- S'ils doivent être représentés dans le lanceur d'application, ils spécifient également la catégorie "`android.intent.category.LAUNCHER`":

```
<intent-filter . . . >
<action android:name="code android.intent.action.MAIN" />
<category android:name="code android.intent.category.LAUNCHER" />
</intent-filter>
```

Le système Android remplit le lanceur d'applications (l'écran de niveau supérieur qui montre les applications qui sont disponibles pour l'utilisateur à lancer), en trouvant toutes les activités avec des filtres d'intention qui spécifient l'action "android.intent.action.MAIN" et de catégorie "android.intent.category.LAUNCHER". Il affiche ensuite les icônes et les libellés de ces activités dans le lanceur.

Une fois un intent lancé plusieurs cas de figure sont possibles :

- Si la recherche est infructueuse et on a 0 résultat alors une exception `ActivityNotFoundException` sera lancée. Il faut gérer ces erreurs.

- Si on n'a qu'un résultat, comme dans le cas des Intents explicites, alors ce résultat va directement se lancer.

- Si on est face à plusieurs réponses, alors le système demande à l'utilisateur de choisir à l'aide d'une boîte de dialogue :
  - Si l'utilisateur choisit une action par défaut pour un intent, alors à chaque fois que le même intent sera émis ce sera toujours le même composant qui sera sélectionné (Il se peut par exemple qu'avec ACTION_SEND, on cherche un jour à envoyer un SMS et un autre jour à envoyer un e-mail).

Exemple

```
public class MainActivity extends Activity {
    private EditText edit;

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

          edit = (EditText) findViewById(R.id.recipient);

          // Use ACTION_SENDTO action with correct data
          Button sms1 = (Button) findViewById(R.id.sendto_sms);
          sms1.setOnClickListener(new View.OnClickListener() {
             public void onClick(View view) {
                 String uri = "smsto:" +  edit.getText().toString();
                 Intent i = new Intent(android.content.Intent.ACTION_SENDTO,
```

```
                Uri.parse(uri));
                startActivity(i);
            }
        });

        // Use our custom SMS_INTENT intent with correct data
        Button sms2 = (Button) findViewById(R.id.smsintent_sms);
        sms2.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                String uri = "smsto:" + edit.getText().toString();
             Intent i = new Intent("com.example.probook.SMS_INTENT",
                        Uri.parse(uri));
                // put extra field
                i.putExtra("from", "javacode ");
                startActivity(i);
            }
        });

        // Use our custom SMS_INTENT intent with incorrect data
        Button exception = (Button) findViewById(R.id.exception);
        exception.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
               String uri = "mailto:" + edit.getText().toString();
               Intent i = new Intent("com.example.probook.SMS_INTENT",
                       Uri.parse(uri));
               i.putExtra("from", "javacode");
               startActivity(i);
            }
        });
    }
}
public class MySmsActivity extends Activity {
     @Override
       public void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);
          setContentView(R.layout.intent_view);

          String output = null;
          TextView dataIntent = (TextView) findViewById(R.id.output_intent);

          // take the data and the extras of the intent
          Uri url = getIntent().getData();
          Bundle extras = getIntent().getExtras();
          output = url.toString();
          // if there are extras, add them to the output string
          if(extras != null){
              output = output+ " from "+ extras.getString("from");
          }
          dataIntent.setText(output);
       }
}
```

Le fichier manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.javacodegeeks.android.intentfiltertest"
    android:versionCode="1"
    android:versionName="1.0" >
```

```
    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="19" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
    android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

          <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
      android:name=".MySmsActivity"
            android:label="@string/app_name">
           <intent-filter>
              <action android:name="android.intent.action.SENDTO" />
              <action android:name="com.example.probook.SMS_INTENT" />
              <category android:name="android.intent.category.DEFAULT" />
              <data android:scheme="smsto" />
           </intent-filter>
        </activity>
    </application>
</manifest>
```

Package Manager

il est possible d'obliger l'utilisateur à choisir parmi plusieurs éventualités à l'aide de la méthode createChooser(Intent target, CharSequence titre).

Dans tous les cas, on peut vérifier si un composant va réagir à un Intent à l'aide du `Package Manager`.

Le `Package Manager` est un objet qui permet d'obtenir des informations sur les packages qui sont installés sur l'appareil.
On fait appel au `Package Manager` avec la méthode `getPackageManager()`.

Puis on peut demander à l'Intent le nom de l'activité qui va le gérer à l'aide de la méthode `ComponentName resolveActivity (PackageManager pm)` :

```
Intent   i = new Intent(Intent.ACTION_VIEW, Uri.parse("tel:0606060606"));

// Check if an Activity exists to perform this action.
PackageManager manager = getPackageManager();
ComponentName component = i.resolveActivity(manager);
// On vérifie que component n'est pas null
if(component != null)
startActivity(i);
```

Intents de Diffusion

Les Intents peuvent aussi être utilisées pour envoyer des messages de diffusion au système Android.

Les récepteurs de messages de diffusion peuvent s'enregistrer à un événement et seront notifiés dès lors que cet événement est déclenché.

L'application peut s'enregistrer aux événements système, par exemple l'arrivée d'un nouveau mail, le redémarrage système terminé, ou un appel téléphonique reçu et réagir en conséquence.

Ici les intents seront anonymes et diffusés à tout le système.

Ce type d'intent est très utilisé au niveau du système pour transmettre un message à but informatif, comme par exemple l'état de la batterie ou du réseau. Ces Intents s'appellent Intents de diffusion ou *broadcast intents*.
Ainsi, toutes les applications pourront capturer ce message et récupérer l'information.

La création des *broadcast* utilise la méthode **`void sendBroadcast(Intent i).`**

1. Créer un intent

```
Intent   i = new Intent("nom.du.filtre");
```

2. Transporter des informations additionnelles

- `i.putExtra("clef", "valeur"); ( i.putExtra("unDouble", 3.14));`
- `i.putExtra(unBundle);`

3. Envoyer un broadcast

```
sendBroadcast( i );
```

Les Broadcast Receivers

L'intent ne sera reçu que par les *broadcast receivers*, qui sont des classes qui dérivent de l'interface **`BroadcastReceiver`** et qui redéfinie la méthode onReceive (Context context, Intent intent) où chaque message est reçu en tant que paramètre de l'Intent.

Cette méthode contient le code que réalisera le BroadcastReceiver à la réception des Intents.

```
public class MyReceiver extends BroadcastReceiver {
  public void onReceive(Context context, Intent intent) {
    Toast.makeText(context, "Intent Detected.", Toast.LENGTH_LONG).show();
  }
}
```

Un BroadcastReceiver ne contient aucune IHM.

Déroulement dans l'utilisation :
1. Le BroadcastReceiver est enregistré pour recevoir certains Intents
2. Des composants Broadcast émettent un Intent
3. Android identifie les bons récepteurs d'Intents et les active en appelant la méthode onReceive(...)
4. l'Intent est manipulé par la méthode onReceive(...).

Enregistrement d'un BroadcastReceiver :

Deux façons possibles :
- **Statique** : via AndroidManifest.xml
- **Dynamique** : via la méthode registerReceiver() de la classe Context :

public abstract Intent registerReceiver (BroadcastReceiver receiver, IntentFilter filter).

Enregistrement Statique :

Exemple

```
<receiver
  android:name=".bonjourReceiver">
 <intent-filter>
   <action android:name="com.example.exemple1.intent.action.action1" />
  <category android:name = "android.intent.category.DEFAULT" />
</intent-filter>
</receiver>
```

Exemple : intent qui transmet le nom de l'utilisateur à tout le système:

Le premier projet : BroadcastReceiver

```
<application
<receiver
android:name = ". MyReceiver"
android:enabled="true"
android:exported="true" >
<intent-filter>
<action android:name =" MON_INTENT" />
<category android:name = "android.intent.category.DEFAULT" />
</intent-filter>
</receiver>
```

```
public class MyReceiver extends BroadcastReceiver {
 public void onReceive(Context context, Intent intent) {
  if(intent.getAction().equals("MON_INTENT"))  {
    Toast.makeText(context, "diffusion détectée " !", Toast.LENGTH_LONG).show();
}}}
```

Le deuxième projet : émetteur

```
public class MainActivity extends Activity {
MyReceiver  receiver;
IntentFilter intentFilter;

protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
receiver = new MyReceiver();
intentFilter=new IntentFilter("MON_INTENT");
}
public void onClick (View view){
Intent intent = new Intent();
intent.setAction("MON_INTENT");
```

```
sendBroadcast(intent);
}
@Override
protected void onResume() {
    super.onResume();
    registerReceiver(receiver, intentFilter);
}

@Override
protected void onDestroy() {
    super.onDestroy();
    unregisterReceiver(receiver);
}
}
```

Déclarer un *broadcast receiver* de manière dynamique

Un broascast receiver déclaré de cette manière sera disponible tout le temps, même quand l'application n'est pas lancée, mais ne sera actif que pendant la durée d'exécution de sa méthode `onReceive`.

Les étapes :
- Créer un BroadcastReceiver
- Enregistrer le BroadcastReceiver afin qu'il reçoive l'IntentFilter par la méthode registerReceiver de la classe Context.
- Au besoin "désenregistrer" le BroadcastReceiver par la méthode unRegisterReceiver(...) de la classe Context.

On crée une classe qui dérive de `BroadcastReceiver` sans l'enregistrer dans le Manifest. Ensuite, lui rajouter des lois de filtrage avec la classe `IntentFilter`,

L'enregistrement sera dans l'activité voulue avec la méthode

```
Intent registerReceiver(BroadcastReceiver receiver, IntentFilter filter)
```

Quand on n'en a plus besoin, il faut la désactiver avec la méthode

```
void unregisterReceiver(BroadcastReceiver receiver).
```

Exemple

Recevoir des broadcast intents et dire bonjour à l'utilisateur, mais uniquement quand l'application se lance et qu'elle n'est pas en pause.

```
public class BonjourActivity extends Activity {
  private static final String BONJOUR = "com.example.exemple1.intent.action.bonjour";
  private IntentFilter filtre = null;
  private BonjourReceiver receiver = null;

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

    filtre = new IntentFilter(BONJOUR);
    receiver = new BonjourReceiver();
  }
  public void onResume() {
```

```
    super.onResume();
    registerReceiver(receiver, filtre);
  }
//Si vous déclarez le receiver dans onResume, n'oubliez pas de l'arrêter dans onPause
   public void onPause() {
    super.onPause();
    unregisterReceiver(receiver);
  }}
```

Il existe quelques messages diffusés par le système de manière native. Comme par exemple `ACTION_CAMERA_BUTTON` qui est lancé dès que l'utilisateur appuie sur le bouton de l'appareil photo.

- **ACTION_BOOT_COMPLETED**: diffusé lorsque le système a fini son boot
- **ACTION_SHUTDOWN**: diffusé lorsque le système est en cours d'extinction
- **ACTION_SCREEN_ON / OFF**: allumage / extinction de l'écran
- **ACTION_POWER_CONNECTED / DISCONNECTED**: connexion / perte de l'alimentation
- **ACTION_TIME_TICK**: une notification envoyée toutes les minutes
- **ACTION_USER_PRESENT**: notification reçue lorsque l'utilisateur délock son téléphone

Sécurité

N'importe quelle application peut envoyer des `broadcast intents` à votre receiver, ce qui pose un problème au niveau sécurité.
On peut faire en sorte que le receiver déclaré dans le Manifest ne soit accessible qu'à l'intérieur d'une application donnée en lui ajoutant l'attribut `android:exported="false"`.

Exemple

```
<receiver android:name="BonjourReceiver"
 android:exported="false">
 <intent-filter>
  <action android:name="com.example.exemple1.intent.action.bonjour" />
 </intent-filter>
</receiver>
```

Afin de déterminer qui peut recevoir un *`broadcast intent`*, il suffit de lui ajouter une permission à l'aide de la méthode :
`void sendBroadcast (Intent intent, String receiverPermission)`, avec `receiverPermission` une permission à déterminer. Ainsi, seuls les receivers qui déclarent cette permission pourront recevoir ces *`broadcast intents`* :

```
private String c_broadcast = "com.example.exemple1.permission.c_broadcast";
…
sendBroadcast(i, c_broadcast);
```

Dans le Manifest, il faut rajouter :

```
<uses-permission android:name="package.permission.c_broadcast"/>
```

Exemples d'applications : Recevoir des messages

Pour recevoir les messages vous devez définir la permission ainsi que le Broadcast Receiver dans AndroidManifest.xml.

- Lecture et réception des SMS

```
<uses-permission android:name="android.permission.RECEIVE_SMS" />
<uses-permission android:name="android.permission.READ_SMS" />
```

- Déclaration du Broadcast Receiver

```
 <receiver class="com.tuto.android.SMSReceiver"
    android:name="com.tuto.android.SMSReceiver">
        android:enabled="true"
        android:exported="true">

    <intent-filter android:priority="100">
        <action android:name="android.provider.Telephony.SMS_RECEIVED" />
    </intent-filter>
</receiver>

public class SMSReceiver extends BroadcastReceiver {
public void onReceive(Context context, Intent intent){
if (intent.getAction().equals("android.provider.Telephony.SMS_RECEIVED")){
                          Bundle bundle = intent.getExtras();
 Bundle bundle = intent.getExtras();

if (bundle != null) {
                Object[] pdus = (Object[]) bundle.get("pdus");

for (int i = 0; i < pdus.length; i++)  {
SmsMessage currentMessage = SmsMessage.createFromPdu((byte[]) pdusObj[i]);
String phoneNumber = currentMessage.getDisplayOriginatingAddress();

String message =     currentMessage.getDisplayMessageBody();
Log.i("SmsReceiver", "senderNum: "+ senderNum + "; message: " + message);

Toast.makeText(context, phoneNumber + message,Toast.LENGTH_LONG). show();

}}}}}
```