



# *Développement d'applications pour Android*



## NOTE

Ce manuel n'a pas la prétention d'être exhaustif, il ne présente que quelques classes de l'API Android (les plus utilisées).

De même pour les classes décrites seules les méthodes les plus utilisées le sont.

## SOURCE DU CODE ANDROID

Il est accessible sur : <http://source.android.com/source/index.html>

## DOCUMENTATION

Pour une documentation complète d'Android on se reportera au site :

<http://developer.android.com>

L'onglet "Reference" contient la documentation en ligne des classes de l'API. La case à cocher et la liste déroulante en haut à droite permettent de limiter cette documentation à une version précise de l'API.

L'onglet "Dev Guide" propose des aides pour le développement.

L'onglet "Resources" propose des aides et des exemples d'utilisation des classes de l'API.

D'autres tutoriels sont disponibles sur :

<http://www.ace-art.fr/wordpress/tag/android/>

et sur bien d'autres sites (attention : certains se contentent de donner des "trucs et astuces" sans explications et parfois même pas très bien maîtrisés par leurs auteurs !).

Un forum très actif est celui de : <http://stackoverflow.com>

Un manuel intitulé "**Développer des applications mobiles pour les Google Phones**" écrit par F. Garin et publié chez Dunod (EAN13 : 9782100531813) peut être téléchargé en pdf sur : <http://kslimi.files.wordpress.com/2011/01/dunod-android.pdf>

Le livre "**Programmation Android – De la conception au déploiement avec le SDK Google Android 2**" de D. Guignard, J Chable, E. Robles aux éditions Eyrolles (ISBN : 978-2-212-12587-0) a également été utilisé pour rédiger ce document.

# Sommaire

Notion d'activité .....	1
Cycle de vie d'une application Android (activité).....	1
Méthodes de la classe Activity .....	2
Méthodes correspondant au cycle de vie.....	2
Méthodes relatives à l'interface .....	2
Méthodes relatives aux menus et menus contextuels.....	2
Méthodes relatives à l'environnement et aux ressources.....	3
Méthodes de lancement d'activités et de services .....	3
Manifeste .....	3
Activités .....	4
Permissions.....	5
Ressources .....	6
Répertoires des ressources .....	6
Classe de référencement des ressources (classe R).....	6
Ressources de type valeurs.....	6
Utilisation de ressources.....	7
Référencement d'une ressource dans un fichier XML.....	7
Récupération d'une ressource dans le code (la classe Resources) .....	7
Uri.....	7
Ressource sur Internet .....	7
Ressource locale .....	8
Fichiers et répertoires sur Android.....	8
Répertoires d'Android .....	8
La classe File .....	8
Méthodes de la classe File.....	8
Lecture/écriture dans un fichier.....	9
Interfaces .....	10
Mise en place d'une interface .....	10
Hiérarchie (partielle) des classes pour les interfaces.....	11
Propriétés et classes de base des interfaces .....	12
Les unités.....	12
Les couleurs.....	12
La classe View .....	12
La classe ViewGroup .....	14
Evénements traités par les classes de base des interfaces .....	15
L'interface OnClickListener .....	15
L'interface OnLongClickListener.....	15
L'interface OnKeyListener .....	15
La classe KeyEvent .....	16
L'interface onTouchListener.....	16
La classe MotionEvent .....	16
Les conteneurs.....	16
La classe FrameLayout.....	16
La classe LinearLayout .....	17
La classe AbsoluteLayout .....	17
Les classe ScrollView et HorizontalScrollView .....	18
La classe TableLayout.....	18
La classe RelativeLayout .....	19

Les groupes .....	20
RadioGroup .....	20
ListView .....	21
GridView .....	23
Gallery .....	25
Les composants d'interface.....	28
ImageView .....	29
TextView .....	30
EditText.....	33
AutoCompleteTextView .....	34
Button .....	34
ImageButton .....	35
ToggleButton.....	35
CheckBox .....	36
RadioButton .....	36
Spinner .....	36
DatePicker .....	37
TimePicker .....	38
ProgressBar .....	39
SeekBar .....	40
RatingBar .....	40
AnalogClock.....	41
DigitalClock .....	41
Chronometer.....	42
Notifications .....	42
La classe Toast .....	42
Textes Formatés .....	43
Création d'un texte formaté : .....	43
Application de formats : .....	43
Couleurs et Images .....	44
La classe Color d'Android .....	44
Méthodes de la classe Color.....	44
Couleurs prédéfinies.....	44
La classe Drawable.....	44
Les menus.....	44
Description en XML d'un menu .....	44
Menu général .....	45
Création du menu .....	45
Réactions aux choix .....	45
Menus contextuels.....	45
Association .....	45
Création du menu contextuel.....	46
Réactions aux choix .....	46
Communication entre activités.....	46
Filtres d'intensions.....	46
Forme générale .....	46
Lancer une activité .....	47
Sans attente de réponse .....	47
Avec attente de réponse.....	47
Passage de paramètres et de valeurs de retour .....	48

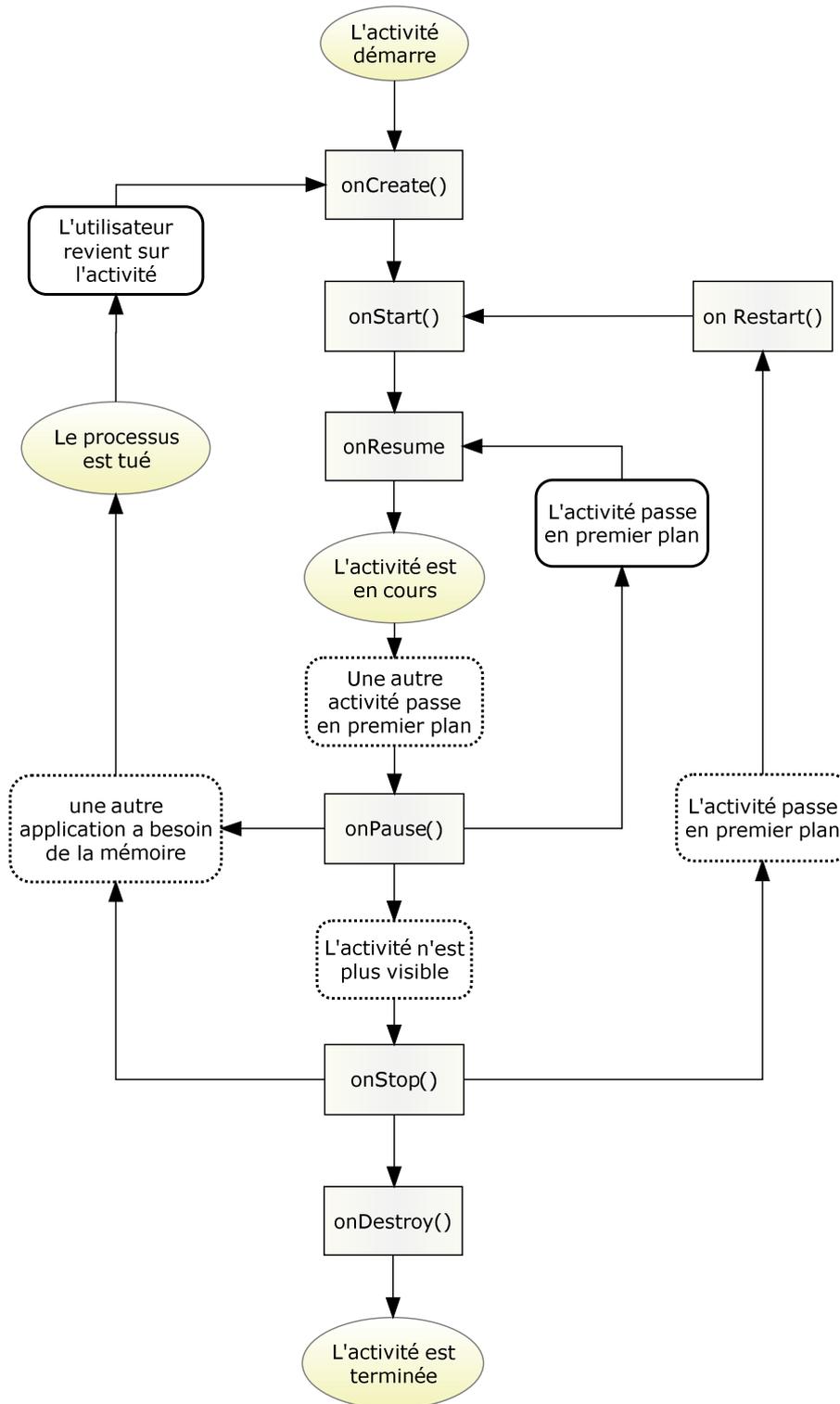
Le matériel et les capteurs .....	48
Envoi d'un SMS.....	48
Utilisation du GPS.....	48
Appareil photo.....	49
Accès à l'appareil photo.....	49
Prévisualisation .....	49
Prise de photo .....	50
Affichage de photo .....	50
Vibreur .....	51
Capteurs.....	51
Le multimédia .....	52
Jouer un son.....	52
La classe MediaPlayer.....	53
Afficher une video.....	53
La classe VideoView.....	54
Synthèse de parole.....	54
Internet .....	55
Récupérer un fichier texte depuis une URL.....	55
Récupérer une image depuis une URL.....	56
Jouer un son depuis une URL .....	56
Jouer une vidéo depuis une URL .....	56
Utiliser des services WEB.....	56
Trouver les coordonnées géographiques de lieux .....	56
La classe Geocoder.....	57
La classe Address .....	57
Exemple :.....	57
GoogleMaps .....	58
Préparation du projet .....	58
Clé d'utilisation.....	58
Afficher des cartes dans une application.....	58



# Notion d'activité

## Cycle de vie d'une application Android (activité).

A chaque étape du cycle de vie correspond l'appel d'une méthode que l'on peut surcharger pour définir le comportement correspondant à cette étape.





## Méthodes relatives à l'environnement et aux ressources

- `getResources()` renvoie un objet de classe `Resources` permettant l'accès aux ressources de l'application (voir plus loin la classe `Resources`)
- `onUserInteraction()` appelée à chaque interaction de l'utilisateur (clavier, trackball, touché)
- `onBackPressed()` appelée lorsque l'utilisateur appuie la touche de retour arrière du téléphone
- `onLowMemory()` appelée lorsque le niveau de batterie devient faible
- `getFilesDir()` renvoie un objet de classe `File` représentant le chemin où sont placés les fichiers propres à l'application
- `openFileInput(String)` renvoie un objet de classe `FileInputStream` correspondant à un flux de lecture connecté au fichier dont le nom est passé en paramètre.
- `openFileOutput(String, int)` renvoie un objet de classe `FileOutputStream` correspondant à un flux d'écriture connecté au fichier dont le nom est passé en premier paramètre. Le second paramètre est le mode d'ouverture qui peut être : `MODE_PRIVATE` (mode normal), `MODE_APPEND` (ajout à un fichier existant), `MODE_WORLD_READABLE` et `MODE_WORLD_WRITEABLE` (pour les droits d'accès). On peut cumuler plusieurs attributs par |

## Méthodes de lancement d'activités et de services

- `startActivity(Intent)` Lance une autre activité, le paramètre désigne l'activité ou les propriétés permettant à Android de l'identifier.
- `startActivityForResult(Intent, int)` Lance une autre activité dont on attend un retour, le 1<sup>er</sup> paramètre désigne l'activité ou les propriétés permettant à Android de l'identifier, le second paramètre est un numéro associé à cette activité qui permettra d'en identifier la réponse.
- `startNextMatchingActivity(Intent)` Utilisé pour demander à Android de lancer une autre activité lorsque celle proposée précédemment n'a pas donné satisfaction. Le paramètre correspond à celui qui a donné un résultat infructueux.
- `finishActivity(int)` Termine une activité dont on attendait un retour, le paramètre est le numéro associé à cette activité lors de son lancement
- `startService(Intent)` Lance un service, le paramètre désigne le service ou les propriétés permettant à Android de l'identifier
- `stopService(Intent)` Arrête un service, le paramètre désigne le service ou les propriétés permettant à Android de l'identifier
- `getIntent()` renvoie l'objet de classe `Intent` associé à l'activité lors de son lancement
- `setResult(int)` Utilisé par l'activité appelée pour renvoyer un code de résultat à l'activité appelante
- `setResult(int, Intent)` comme la précédente, le paramètre supplémentaire est un `Intent` contenant les valeurs de retour qui sera également renvoyé.
- `onActivityResult(int, int, Intent)` cette méthode de l'activité appelante est exécutée lorsqu'une activité lancée se termine, le 1<sup>er</sup> paramètre est le numéro associé à cette activité lors de son lancement, le deuxième est le code de retour de cette activité, le dernier est l'`Intent` de retour de cette activité

## Manifeste

Une application est décrite par un fichier XML appelé **AndroidManifest**. Ce fichier permet d'indiquer :

- Le SDK utilisé par une balise `uses-sdk`
- Les permissions : une balise `uses-permission` par autorisation demandée
- Les activités de l'application : une balise `activity` pour chaque activité contenant chacune une ou plusieurs balises `intent-filter`
- Les services de l'application : une balise `service` pour chaque service contenant chacune une ou plusieurs balises `intent-filter`
- les écouteurs d'intentions diffusées : une balise `receiver` pour chaque écouteur contenant chacune une ou plusieurs balises `intent-filter`
- Les fournisseurs de contenu : une balise `provider` pour chaque activité
- Les bibliothèques utilisées : une balise `uses-library` pour chaque bibliothèque

La structure générale de ce fichier est la suivante :

```
<?xml version="1.0" encoding="utf-8"?>
<manifest>
  <uses-sdk />
  <uses-permission />
  <application>
    <activity>
      <intent-filter>
        <action />
        <category />
        <data />
      </intent-filter>
    </activity>
    <service>
      <intent-filter> . . . </intent-filter>
    </service>
    <receiver>
      <intent-filter> . . . </intent-filter>
    </receiver>
    <provider>
      <grant-uri-permission />
    </provider>
    <uses-library />
  </application>
</manifest>
```

## Activités

Chaque activité constituant une application doit être décrite par une balise `<activity>`. La balise minimale de l'activité principale est de la forme :

```
<activity android:name=".ClasseDeLActivite"
  android:label="nom_de_l_activite"
  android:icon="@drawable/nom_du_fichier_icone"
>
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
```

La balise `<activity>` peut contenir de nombreux paramètres. Les paramètres généralement utilisés sont :

- name qui désigne la classe de l'activité
- label qui désigne le nom sous lequel elle apparaîtra sur le terminal
- icon qui désigne l'icône à afficher sur le terminal

Une activité secondaire (lancée par l'activité principale) n'a pas de propriété `android:icon` ni `android:label`. Elle peut ne pas avoir de balise `<intent-filter>` et si elle en a une on n'y rencontrera pas la propriété `<action android:name="android.intent.action.MAIN" />` puisque ce n'est pas un programme principal.

La balise `<intent-filter>` permet d'indiquer ce que fait l'activité (on parle d'intension). Elle sera utilisée par Android pour rechercher une activité pouvant répondre à une spécification donnée (cette spécification lui sera fournie dans un objet de classe `Intent`). La forme générale de la balise `<intent-filter>` est la suivante:

```
<intent-filter>
  <action android:name="nom_d_action_1" />
  ...
  <action android:name="nom_d_action_N" />
  <category android:name="nom_de_categorie_1" />
  ...
  <category android:name="nom_de_categorie_N" />
```

```

    <data android:mimeType="nom_de_type_mime"
        android:scheme="protocole://hote:port/chemin" />
  />
</intent-filter>

```

On y trouve les rubriques suivantes :

- Action qui indique le type d'action effectuée par l'activité (par exemple affichage, édition ...)
- Category qui indique la catégorie d'action (par exemple CATEGORY\_BROWSABLE indique une activité qui peut être appelée par un navigateur)
- Data qui indique le type de données transmises à l'activité lancée ou le type de réponse attendu ainsi que le protocole (http, content, file ...)

Bien que le concepteur puisse définir ses propres valeurs de paramètres, il existe des valeurs prédéfinies pour ces diverses rubriques dont les principales sont :

- Actions
  - o android.intent.action.VIEW affichage de données
  - o android.intent.action.EDIT affichage de données pour édition par l'utilisateur
  - o android.intent.action.MAIN activité principale d'une application
  - o android.intent.action.CALL appel téléphonique
  - o android.intent.action.WEB\_SEARCH recherche sur le WEB
- Catégories
  - o android.intent.category.LAUNCHER activité proposée au lancement par Android
  - o android.intent.category.DEFAULT activité pouvant être lancée explicitement
  - o android.intent.category.BROWSABLE peut afficher une information désignée par un lien
  - o android.intent.category.TAB activité associée dans un onglet d'interface (TabHost)

## Permissions

Pour autoriser une application à accéder à certaines ressources il faut lui en donner l'autorisation par une balise <uses-permission>. Les principales permissions sont :

### Géolocalisation (GPS)

```

<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_MOCK_LOCATION" />

```

### Accès aux données personnelles

```

<uses-permission android:name="android.permission.READ_CONTACTS" />
<uses-permission android:name="android.permission.WRITE_CONTACTS" />
<uses-permission android:name="android.permission.READ_CALENDAR" />
<uses-permission android:name="android.permission.WRITE_CALENDAR" />
<uses-permission android:name="android.permission.READ_HISTORY_BOOKMARKS" />
<uses-permission android:name="android.permission.WRITE_HISTORY_BOOKMARKS" />

```

### Modification de paramètres

```

<uses-permission android:name="android.permission.SET_ORIENTATION" />
<uses-permission android:name="android.permission.SET_TIME_ZONE" />
<uses-permission android:name="android.permission.SET_WALLPAPER" />

```

### Téléphonie

```

<uses-permission android:name="android.permission.CALL_PHONE" />
<uses-permission android:name="android.permission.CALL_PRIVILEGED" />
<uses-permission android:name="android.permission.MODIFY_PHONE_STATE" />

```

### Envoi et réception de SMS/MMS

```

<uses-permission android:name="android.permission.READ_SMS" />
<uses-permission android:name="android.permission.RECEIVE_MMS" />
<uses-permission android:name="android.permission.RECEIVE_SMS" />
<uses-permission android:name="android.permission.RECEIVE_WAP_PUSH" />
<uses-permission android:name="android.permission.SEND_SMS" />
<uses-permission android:name="android.permission.WRITE_SMS" />

```

## Audio

```
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
```

## Réseau

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
```

## Access au matériel

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.FLASHLIGHT" />
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.BATTERY_STATS" />
```

# Ressources

## Répertoires des ressources

Le répertoire `res` contient toutes les ressources qui seront mises dans le fichier application (`apk`). Il est constitué de sous répertoires :

- `drawable-hdpi` (images en haute définition)
- `drawable-ldpi` (images en basse définition)
- `drawable-mdpi` (images en moyenne définition)
- `layout` (description en XML des interfaces)
- `values` (définitions en XML de valeurs : chaînes, tableaux, valeurs numériques ...)
- `anim` (description en XML d'animations)
- `menu` (description en XML de menus pour l'application)
- `xml` (fichiers XML utilisés directement par l'application)
- `raw` (tous les autres types de ressources : fichiers texte, vidéo, son ...)

Remarque : Il est possible de créer d'autres répertoires que ceux indiqués ci-dessus pour des cas particuliers.

## Classe de référencement des ressources (classe *R*)

Eclipse explore les fichiers contenus dans les sous-répertoires de `res` et génère une classe (appelée `R`) dans laquelle un identifiant est créé pour chacun des éléments trouvés :

- Fichiers (images, textes ..) placés dans `drawable-xxxx`, `raw` et `xml`
- Éléments d'interface définis dans les fichiers placés dans `layout` et `menu`
- Valeurs définies dans les fichiers placés dans `values`
- Animations définies dans les fichiers placés dans `anim`

On pourra ensuite, dans la plupart des méthodes, désigner ces ressources par leur identifiant dans cette classe `R` sous la forme : `R.type.nom`.

Par exemple une image "photo.png" placée dans `drawable-hdpi` sera désignée par `R.drawable.photo`

## Ressources de type valeurs

On peut définir des ressources de type valeur (entiers, booléens, chaînes de caractères, etc. et des tableaux) en les décrivant dans des fichiers xml ayant la forme suivante :

```
<?xml version="1.0" encoding="utf-8" ?>
<resources>
    <color name="coulfond">#AA7B03</color>
    <integer name="limite">567</integer>
    <integer-array name="codes_postaux">
        <item>64100</item>
        ...
    </integer-array>
</resources>
```

```

        <item>33000</item>
    </integer-array>
    <string name="titre">Un titre</string>
    <string-array name="planetes">
        <item>Mercure</item>
        ...
        <item>Venus</item>
    </string-array>
    <bool name="actif">true</bool>
    <dimen name="taille">55px</dimen>
</resources>

```

## Utilisation de ressources

Les ressources peuvent être utilisées dans les fichiers XML (comme ceux de description d'interface) ou dans le code java.

## Référencement d'une ressource dans un fichier XML

On référence une ressource dans un fichier XML par :

```
"@[paquetage:]type/identificateur"
```

Par exemple : `@string/machaine` fait référence à une chaîne décrite dans un fichier XML placé dans le répertoire `res/values` (le nom de ce fichier n'a aucune importance) et définie par :

```
<string name="machaine">contenu de cette chaine</string>
```

## Récupération d'une ressource dans le code (la classe Resources)

Dans le code les ressources sont désignées par leur identifiant dans la classe `R` de la forme : `R.type.nom`. Certaines méthodes acceptent cette désignation comme paramètre pour accéder à la ressource. Toutefois lorsque l'on doit accéder au contenu de ressources on peut faire appel à la classe `Resources`.

```
import android.content.res.Resources
```

On obtient une instance de la classe `Resources` par la méthode `getResources()` de l'activité. On accède ensuite aux ressources grâce aux méthodes de cet objet qui acceptent en paramètre un identifiant de la forme `R.type.nom`, ces méthodes sont :

- `boolean getBoolean(int)` renvoie le booléen désigné
- `int getInteger(int)` renvoie l'entier désigné
- `int[] getIntArray(int)` renvoie le tableau d'entiers désigné
- `String getString(int)` renvoie la chaîne désignée
- `String[] getStringArray(int)` renvoie le tableau de chaînes désigné
- `int getColor(int)` renvoie le code de la couleur désignée
- `float getDimension(int)` renvoie la valeur de l'unité désignée
- `Drawable getDrawable(int)` renvoie l'image désignée (formats possibles du fichier placé dans le répertoire `res` : gif , jpeg , png et bmp)
- `InputStream openRawResource(int)` renvoie un flux de lecture sur la ressource désignée.

Par exemple : `String titre = getResources().getString(R.string.texte_titre);`

permet de récupérer la chaîne définie dans un fichier XML par : `<string name="texte_titre">.....</string>`

## Uri

Les Uri sont un moyen utilisé pour désigner des ressources que ce soit sur Internet ou localement

### Ressource sur Internet

Pour créer une Uri faisant référence à une ressource sur Internet :

```
Uri ressource =
Uri.parse("http://domaine.sous_domaine/chemin/nom_du_fichier");
```

Exemple : `Uri.parse("http://www.iutbayonne.univ-pau.fr/~dalmau/testTPs/ma_video.3gp");`  
Désigne le fichier vidéo `ma_video.3gp` accessible sur `http://www.iutbayonne.univ-pau.fr/~dalmau/testTPs/`

## Ressource locale

Pour créer une Uri faisant référence à une ressource embarquée :

```
Uri ressource =  
Uri.parse("android.resource://nom_du_paquetage_de_l_activité/" +  
R.chemin.mon_son);
```

Exemple : `Uri.parse("android.resource://iut.TP/" + R.raw.mon_son);`

Désigne le fichier `son_mon_son` placé dans le répertoire `res/raw` de l'activité dont le paquetage est `iut.TP`

# Fichiers et répertoires sur Android

## Répertoires d'Android

```
import android.os.Environment
```

Android permet la manipulation de fichiers de façon classique en java (classe `File`). Certains répertoires ont un rôle particulier dans le système. On peut les obtenir grâce l'activité ou à la classe `Environment` comme suit :

- Répertoire propre à l'application : on l'obtient par la méthode `getFilesDir()` de l'activité
- Répertoire de données : on l'obtient par `Environment.getDataDirectory()`
- Répertoire des téléchargements : on l'obtient par `Environment.getDownloadCacheDirectory()`
- Répertoire de stockage externe (en général une carte SD) : on l'obtient par `Environment.getExternalStorageDirectory()`
- Répertoire racine d'Android : on l'obtient par `Environment.getRootDirectory()`

Toutes ces méthodes renvoient un objet de classe `File`. Lorsqu'il s'agit d'un répertoire il peut être utilisé pour ouvrir un fichier par : `File monFichier = new File(repertoire, "nom_du_fichier");`

**ATTENTION** : certains répertoires peuvent être totalement fermés ou n'autoriser que la lecture.

## La classe File

```
import java.io.File
```

C'est une classe java qui n'est pas propre à Android, elle permet de manipuler des fichiers et des répertoires

## Méthodes de la classe File

### Ouverture

- `File(String, String)` ouvre un fichier, le premier paramètre est le nom du chemin, le second le nom du fichier
- `File(File, String)` ouvre un fichier, le premier paramètre est le répertoire, le second le nom du fichier
- `File(URI uri)` ouvre un fichier, le paramètre désigne ce fichier sous forme d'URI (de la forme `file:/chemin/nom`). Ce constructeur peut lever une exception de classe `IllegalArgumentException` si le paramètre n'est pas une URI correcte

### Création

- `createNewFile()` crée le fichier correspondant à l'objet `File`. Cette méthode peut lever une exception de classe `IOException` si le fichier ne peut pas être créé.

### Suppression

- `delete()` supprime le fichier, cette méthode renvoie un booléen indiquant si la destruction a pu se faire ou pas

- `deleteOnExit()` le fichier sera supprimé lorsque la machine virtuelle java se terminera si elle se termine normalement

### Etats

- `exists()` renvoie true si le fichier ou le répertoire existe
- `canRead()` renvoie true si le fichier ou le répertoire peut être lu
- `canWrite()` renvoie true si le fichier ou le répertoire peut être écrit
- `isDirectory()` renvoie true si c'est un répertoire
- `isFile()` renvoie true si c'est un fichier
- `isHidden()` renvoie true si le fichier ou le répertoire est caché
- `length()` renvoie la taille du fichier (ne fonctionne pas pour un répertoire)
- `setReadOnly()` positionne le fichier ou le répertoire en lecture seule

### Noms

- `getName()` renvoie la partie correspondant au nom du fichier ou du répertoire (String)
- `getPath()` renvoie la partie correspondant au chemin du fichier ou du répertoire (String)
- `getAbsolutePath()` renvoie le chemin absolu du fichier ou du répertoire (String)
- `getCanonicalPath()` renvoie le chemin relatif du fichier ou du répertoire (String). Cette méthode peut lever une exception de classe `IOException`.
- `toURI()` renvoie le fichier ou le répertoire sous forme d'URI (de la forme `file:/chemin/nom`)

### Répertoires

- `list()` renvoie la liste des noms des fichiers contenus dans le répertoire (String[])
- `listFiles()` renvoie la liste des fichiers contenus dans le répertoire (File[])
- `mkdirs()` créer le répertoire désigné par l'objet, si nécessaire les répertoires du chemin son également créés

### Lecture/écriture dans un fichier

On utilise des flux de lecture ou d'écriture de classe `FileInputStream` et `FileOutputStream` qui sont construits à partir de l'objet de classe `File` par :

```
new FileInputStream(File)
new FileOutputStream(File)
```

Puis, à partir de ces flux liés au fichier, on construit des flux adaptés aux types de données à lire ou écrire.

# Interfaces

## ***Mise en place d'une interface***

Elle peut se faire de deux façons :

- Par description de l'interface dans des fichiers XML
- Par programme

Les fichiers XML qui décrivent une interface sont placés dans le répertoire `res/layout`.  
Ils sont référencés par `R.layout.nom_du_fichierXML`.

Les activités peuvent utiliser la méthode `setContentView(R.layout.nom_du_fichierXML)` pour mettre en place l'interface décrite par un tel fichier.

Leur forme générale est :

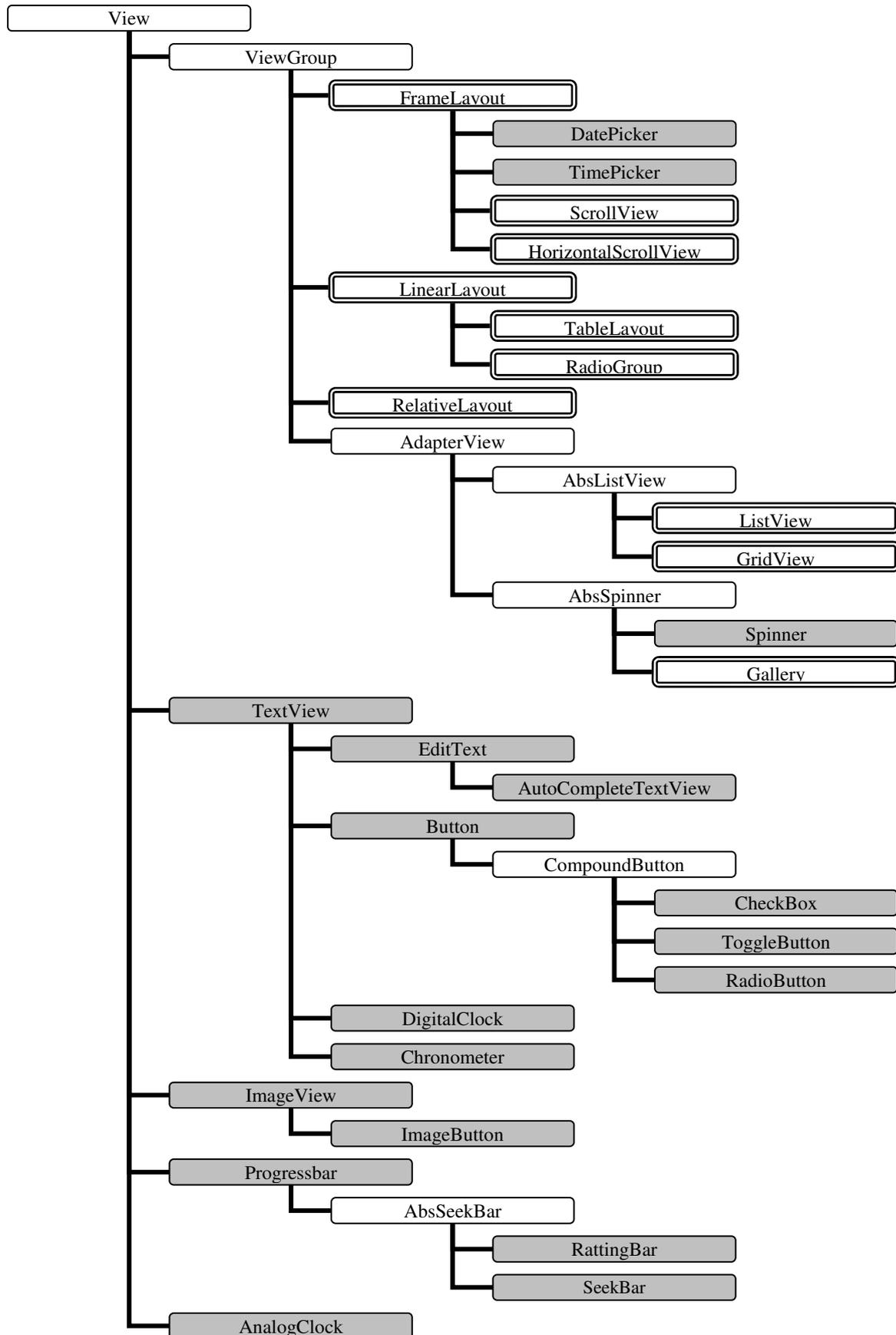
```
<?xml version="1.0" encoding="utf-8"?>
<!-- Commentaire -->

<Classe_du_conteneur_principal
    xmlns:android="http://schemas.android.com/apk/res/android"
    propriétés du conteneur principal
>
    <Classe de conteneur ou d'élément d'interface
        propriétés du conteneur ou de l'élément d'interface
    />
    ...
    <Classe de conteneur ou d'élément d'interface
        propriétés du conteneur ou de l'élément d'interface
    />
</Classe_du_conteneur_principal>
```

Lorsque l'un des éléments est un conteneur, il faut indiquer les éléments qu'il contient :

```
<Classe_de_conteneur
    propriétés du conteneur (voir ci-dessous)
>
    <Classe de conteneur ou d'élément d'interface
        propriétés du conteneur ou de l'élément d'interface
    />
    ...
    <Classe de conteneur ou d'élément d'interface
        propriétés du conteneur ou de l'élément d'interface
    />
</Classe_du_conteneur>
```

## Hierarchie (partielle) des classes pour les interfaces



## Propriétés et classes de base des interfaces

### Les unités

Lorsque l'on indique des unités dans les fichiers XML elles peuvent l'être de plusieurs façons :

- en pixels (faire suivre la valeur de **px**)
- relativement à la taille de la fonte de caractères (faire suivre la valeur de **sp**)
- en millimètres (faire suivre la valeur de **mm**)
- en pouces (faire suivre la valeur de **in**)
- en points = 1/72 pouce (faire suivre la valeur de **pt**)
- en pixels à densité indépendante valent 1 pixel pour un écran de 160 dpi (faire suivre la valeur de **dp**)

La valeur est exprimée par un réel de la forme "14.3mm" par exemple.

Dans ce qui suit on désignera les unités par "unité"

### Les couleurs

Elles sont définies par un code hexadécimal indiquant la transparence et les composantes (RVB) sous la forme "#AARRVVB" où :

- AA est la transparence (FF opaque, 00 totalement transparent). Si le code de couleur ne comporte que 3 composantes AA est considéré comme FF c'est-à-dire que #44FA8B est équivalent à #FF44FA8B
- RR VV et BB sont, respectivement, les composantes rouge, verte et bleue

Dans ce qui suit on désignera les couleurs par "couleur"

### La classe View

Les classes View et ViewGroup ne sont pas utilisées directement mais constituent les classes mères des autres. Elles sont décrites ici pour éviter de répéter leurs propriétés dans les autres classes d'interface.

View est la classe dont héritent toutes les classes utilisées pour réaliser des interfaces. Ses propriétés et ses méthodes se retrouvent donc dans tous les éléments d'interface.

### Propriétés positionnables en XLM

#### Identifiant

Un identifiant peut être associé à chaque élément décrit dans un fichier XML, cet identifiant permet d'accéder à l'objet créé dans le code. Les éléments ne devant pas être accédés dans le code peuvent ne pas avoir d'identifiant.

- `android:id="@+id/monident"` permettra de retrouver cet élément par `findViewById(R.id.monident)`.

#### Visibilité

- `android:visibility="x"` (les valeurs possibles pour x sont : visible, invisible ou gone) Avec invisible la place est conservée, avec gone la place n'est pas conservée.

#### Fond

- `android:background="couleur"` pour définir une couleur ou `android:background="@drawable/monimage"` pour utiliser une image. L'image est placée dans `res/drawable-xxxx/` et s'appelle `monimage.t` (où t est png, jpg, ou gif). ATTENTION : les noms des images ne doivent utiliser que des minuscules ou des chiffres.

#### Taille et marges internes

- `android:minHeight="unité"` définit la hauteur minimale si possible
- `android:minWidth="unité"` définit la largeur minimale si possible
- `android:paddingBottom="unité"` marge interne en bas
- `android:paddingLeft="unité"` marge interne à gauche
- `android:paddingRight="unité"` marge interne à droite
- `android:paddingTop="unité"` marge interne en haut

## ***Ascenseurs***

Ces paramètres ne permettent pas d'ajouter des ascenseurs mais seulement de gérer leur comportement lorsqu'ils existent.

- `android:fadeScrollbars="b"` où `b` vaut `true` ou `false` selon que l'on souhaite que les ascenseurs disparaissent ou pas lorsqu'ils ne sont pas utilisés.
- `android:scrollbarDefaultDelayBeforeFade="x"` où `x` est un entier qui définit le délai (en ms) avant que les ascenseurs non utilisés ne disparaissent
- `android:scrollbarFadeDuration="x"` où `x` est un entier qui définit la durée d'effacement des ascenseurs (en ms)

## ***Evénements***

- `android:clickable="b"` (où `b` vaut `true` ou `false`).
- `android:longClickable="b"` (où `b` vaut `true` ou `false`).

## **Méthodes de la classe View :**

```
import android.view.View
```

## ***Construction***

- `View(Context)` le paramètre est généralement l'activité elle-même

## ***Placement***

- `setLayoutParams(ViewGroup.LayoutParams)` permet de définir le placement des objets dans la vue. Le paramètre de cette méthode est généralement `new ViewGroup.LayoutParams(l, h)` où `l` définit la largeur et `h` la hauteur. Les valeurs pour `l` et `h` peuvent être exprimées en pixels ou prendre les valeurs `ViewGroup.LayoutParams.WRAP_CONTENT` pour adapter la taille au contenu ou `ViewGroup.LayoutParams.FILL_PARENT` pour adapter la taille à la place disponible.

## ***Position***

- `bringToFront()` met la vue en premier plan
- `getLeft()` renvoie la position en pixels de la gauche de la vue
- `getTop()` renvoie la position en pixels du haut de la vue
- `getRight()` renvoie la position en pixels de la droite de la vue
- `getBottom()` renvoie la position en pixels du bas de la vue

## ***Taille et marges internes***

- `getWidth()` renvoie la largeur de la vue en pixels
- `getHeight()` renvoie la hauteur de la vue en pixels
- `setPadding(int, int, int, int)` définit les marges (en pixels). Les paramètres sont, dans l'ordre : gauche, haut, droite, bas.
- `getPaddingLeft()` renvoie la marge gauche (en pixels).
- `getPaddingTop()` renvoie la marge haute (en pixels).
- `getPaddingRight()` renvoie la marge droite (en pixels).
- `getPaddingBottom()` renvoie la marge basse (en pixels).

## ***Fond***

- `setBackgroundColor(int)` définit la couleur du fond
- `setBackgroundDrawable(Drawable)` définit une image de fond
- `setBackgroundResource(int)` définit le fond à partir d'un identificateur de ressource

## ***Rafraichissement***

- `requestLayout()` provoque un rafraichissement des positions des éléments de la vue

- invalidate() provoque un rafraîchissement de la vue

### **Evenements**

- setOnClickListener(OnClickListener) associe un écouteur d'événements aux clics sur la vue
- setOnLongClickListener(OnLongClickListener) associe un écouteur d'événements aux clics longs sur la vue
- setOnKeyListener(OnKeyListener) associe un écouteur d'événements aux actions clavier sur la vue
- setOnTouchListener(OnTouchListener) associe un écouteur d'événements aux touchés sur la vue

### **Focus**

- isFocusable() renvoie true si la vue peut prendre le focus
- setFocusable(boolean) fait ou pas prendre le focus à la vue
- isFocusableInTouchMode() renvoie true si la vue peut prendre le focus lorsqu'elle est touchée
- setFocusableInTouchMode(boolean) autorise ou non la vue à prendre le focus lorsqu'elle est touchée

## **La classe ViewGroup**

### **Propriétés positionnables en XLM pour les éléments placés à l'intérieur**

#### **Taille**

- android:layout\_height="t" (où t peut être une unité ou prendre les valeurs : FILL\_PARENT ou WRAP\_CONTENT) pour occuper tout l'espace en hauteur ou seulement ce qui est nécessaire
- android:layout\_width="t" (où t peut être une unité prendre les valeurs : FILL\_PARENT ou WRAP\_CONTENT) pour occuper tout l'espace en largeur ou seulement ce qui est nécessaire

#### **Marges externes**

- android:layout\_marginBottom="unité" marge externe en bas
  - android:layout\_marginLeft="unité" marge externe à gauche
  - android:layout\_marginRight="unité" marge externe à droite
  - android:layout\_marginTop="unité" marge externe en haut
- La méthode correspondant à ces 4 paramètres est : setPadding(int, int, int, int)

### **Méthodes de la classe ViewGroup :**

```
import android.view.ViewGroup
```

#### **Construction**

- ViewGroup(Context) le paramètre est généralement l'activité elle-même

#### **Eléments du groupe**

- addView(View) ajoute un élément au groupe
- getChildCount() renvoie le nombre d'éléments du groupe
- getFocusedChild() renvoie l'élément qui a actuellement le focus (objet de classe View)
- removeView(View) enlève un élément du groupe
- removeAllViews() enlève tous les éléments du groupe

#### **Marges**

- setPadding(int, int, int, int) définit les marges (en pixels). Les paramètres sont, dans l'ordre : gauche, haut, droite, bas.

## ***Événements traités par les classes de base des interfaces***

### **L'interface OnClickListener**

```
import android.view.View.OnClickListener
```

Définit les écouteurs d'événements de type clic. La méthode à surcharger pour traiter les événements est :

```
- onClick(View)
```

### **L'interface OnLongClickListener**

```
import android.view.View.OnLongClickListener
```

Définit les écouteurs d'événements de type clic long. La méthode à surcharger pour traiter les événements est :

```
- onLongClick(View)
```

### **L'interface OnKeyListener**

```
import android.view.View.OnKeyListener
```

Définit les écouteurs d'événements de type clavier. La méthode à surcharger pour traiter les événements est :

```
- onKey(View, int, KeyEvent)
```

Le deuxième paramètre est le code de la touche tapée, le troisième est l'événement clavier.

Le code de la touche tapée peut prendre les valeurs :

- KEYCODE\_0 à KEYCODE\_9 pour les chiffres
- KEYCODE\_A à KEYCODE\_Z pour les majuscules
- KEYCODE\_ALT\_LEFT , KEYCODE\_ALT\_RIGHT , KEYCODE\_SHIFT\_LEFT et KEYCODE\_SHIFT\_RIGHT pour les touches Alt et Shift
- KEYCODE\_NUM pour la touche Num
- KEYCODE\_APOSTROPHE , KEYCODE\_GRAVE pour l'apostrophe et l'anti apostrophe
- KEYCODE\_SLASH et KEYCODE\_BACKSLASH pour le / et l'antislash
- KEYCODE\_AT pour @
- KEYCODE\_STAR pour la touche \*
- KEYCODE\_TAB pour la touche Tab
- KEYCODE\_SPACE pour l'espace
- KEYCODE\_COMMA , KEYCODE\_PERIOD , KEYCODE\_SEMICOLON pour la virgule, le point et le point virgule
- KEYCODE\_BACK pour la touche de correction (backspace)
- KEYCODE\_CALL et KEYCODE\_ENDCALL pour les touches d'appel et de fin d'appel
- KEYCODE\_CAMERA , KEYCODE\_FOCUS pour la touche de déclenchement et de réglage de focus de l'appareil photo
- KEYCODE\_CLEAR pour la touche Clear
- KEYCODE\_DEL pour la touche Del
- KEYCODE\_HOME pour la touche Home
- KEYCODE\_MENU pour la touche Menu
- KEYCODE\_DPAD\_CENTER, KEYCODE\_DPAD\_DOWN, KEYCODE\_DPAD\_LEFT, KEYCODE\_DPAD\_RIGHT, KEYCODE\_DPAD\_UP pour le joystick
- KEYCODE\_ENTER pour la touche Entrée
- KEYCODE\_ENVELOPE pour la touche spéciale Enveloppe
- KEYCODE\_EXPLORER pour la touche spéciale Explorateur
- KEYCODE\_EQUALS, KEYCODE\_MINUS ,
- KEYCODE\_LEFT\_BRACKET , KEYCODE\_RIGHT\_BRACKET pour les crochets
- KEYCODE\_MEDIA\_FAST\_FORWARD , KEYCODE\_MEDIA\_NEXT , KEYCODE\_MEDIA\_PLAY\_PAUSE , KEYCODE\_MEDIA\_PREVIOUS , KEYCODE\_MEDIA\_REWIND , KEYCODE\_MEDIA\_STOP pour les touches de contrôle des médias

- KEYCODE\_MUTE , KEYCODE\_VOLUME\_DOWN, KEYCODE\_VOLUME\_UP pour les touches de contrôle du volume

## La classe KeyEvent

```
import android.view.KeyEvent
```

Associée aux événements clavier. Ses méthodes sont les suivantes :

- `getAction ()` qui renvoie un entier pouvant prendre les valeurs ACTION\_DOWN ou ACTION\_UP qui indique l'action faite sur la touche (appuyée, lâchée).
- `getRepeatCount ()` renvoie le nombre de répétitions lorsque la touche est maintenue
- `getKeyCode ()` renvoie le code de la touche (même valeur que le dernier paramètre de `onKey`)
- Les méthodes `isAltPressed ()` , `isShiftPressed ()` , `isCtrlPressed ()` , `isCapsLockOn ()` , `isNumLockOn ()` , `isScrollLockOn ()` permettent de tester l'état des touches de modification.

## L'interface onTouchListener

```
import android.view.View.OnTouchListener
```

Définit les écouteurs d'événements de type touché. La méthode à surcharger pour traiter les événements est :

- `onTouch (View, MotionEvent)`

Le second paramètre est l'événement de touché.

## La classe MotionEvent

```
import android.view.MotionEvent
```

Associée aux événements de déplacement (souris, écran tactile). Ses méthodes sont les suivantes :

- `getAction ()` qui renvoie un entier pouvant prendre les valeurs ACTION\_DOWN , ACTION\_UP , ACTION\_MOVE ou ACTION\_OUTSIDE
- `getPressure ()` renvoie un réel entre 0 et 1 indiquant la force de pression du touché sur l'écran
- `getX ()` et `getY ()` renvoient un réel entre -1 et 1 indiquant les coordonnées (resp en x et en y).
- `getXPrecision ()` et `getYPrecision ()` renvoient un réel indiquant la précision des coordonnées (resp en x et en y).

## Les conteneurs

Les conteneurs sont utilisés pour placer des éléments d'interface ou d'autres conteneurs. Ils héritent tous de ViewGroup. Leurs propriétés sont donc au moins celles décrites ci-dessus pour View et ViewGroup auxquelles viennent s'ajouter des propriétés spécifiques décrites dans cette partie.

## La classe FrameLayout

Placement en haut à gauche. Si l'on place plusieurs éléments ils se superposent, généralement FrameLayout est utilisé pour ne placer qu'un seul élément.

## Propriétés supplémentaires positionnables en XML

### Couleurs ou images

- `android:foreground="couleur"` pour définir une couleur ou `android:foreground="@drawable/monimage"` pour utiliser une image. L'image est placée dans `res/drawable/` et s'appelle `monimage.x` (où x est png, jpg, ou gif).
- `android:foregroundGravity="g"` (où g peut prendre les valeurs : top, bottom, left, right, center\_vertical, fill\_vertical, center\_horizontal, fill\_horizontal, center, fill) utilisé lorsque `android:foreground` est défini par une image pour donner sa position.

## Méthodes de la classe `FrameLayout`

`android.widget.FrameLayout`

### *Construction*

- `FrameLayout(Context)` le paramètre est généralement l'activité elle-même

### *Fond*

- `setForeground(Drawable)` définit une image comme fond
- `getForeground()` renvoie l'image de fond (objet de classe `Drawable`)
- `setForegroundGravity(int)` Le paramètre est l'une des constantes définies dans la classe `Gravity` on utilise donc `Gravity`. `TOP`, `BOTTOM`, `LEFT`, `RIGHT`, `CENTER_VERTICAL`, `FILL_VERTICAL`, `CENTER_HORIZONTAL`, `FILL_HORIZONTAL`, `CENTER`, `FILL`

## La classe `LinearLayout`

Place les éléments les uns à côté des autres horizontalement ou verticalement.

### Propriétés supplémentaires positionnables en XML

- `android:orientation="o"` (où `o` peut prendre les valeurs : `vertical` ou `horizontal`)
- `android:weightSum="x"` (où `x` est un réel entre 0 et 1) désigne la somme des poids de tous les éléments mis dans le `LinearLayout`. Un paramètre de type `android:layout_weight="x"` (où `x` est un réel entre 0 et 1) peut être associé à chacun des éléments placés dans le `LinearLayout` pour indiquer leur poids de redimensionnement relatif à la valeur de `layout_weightSum`.
- `android:gravity="g"` (où `g` peut prendre les valeurs : `top`, `bottom`, `left`, `right`, `center_vertical`, `fill_vertical`, `center_horizontal`, `fill_horizontal`, `center`, `fill`) définit comment se placent les éléments contenus par ce conteneur.

## Méthodes de la classe `LinearLayout`

`android.widget.LinearLayout`

### *Construction*

- `LinearLayout(Context)` le paramètre est généralement l'activité elle-même

### *Position et disposition*

- `getOrientation()` renvoie un entier qui vaut `LinearLayout.HORIZONTAL` ou `LinearLayout.VERTICAL`
- `setOrientation(int)` définit l'orientation, le paramètre est un entier qui vaut `LinearLayout.HORIZONTAL` ou `LinearLayout.VERTICAL`
- `getWeightSum()` renvoie un réel correspondant à la somme des poids associée au `LinearLayout`
- `setWeightSum(float)` définit la somme des poids associée au `LinearLayout`
- `setGravity(int)` Le paramètre est l'une des constantes définies dans la classe `Gravity` on utilise donc `Gravity`. (`TOP`, `BOTTOM`, `LEFT`, `RIGHT`, `CENTER_VERTICAL`, `FILL_VERTICAL`, `CENTER_HORIZONTAL`, `FILL_HORIZONTAL`, `CENTER`, `FILL`)

## La classe `AbsoluteLayout`

Place les éléments selon des coordonnées.

### Propriétés positionnables en XML pour les éléments placés à l'intérieur

Chaque élément ajouté dans un `AbsoluteLayout` indique sa position en mettant dans ses propriétés :

- `android:layout_x="unité"`
- `android:layout_y="unité"`

## Méthodes de la classe `AbsoluteLayout`

`android.widget.AbsoluteLayout`

### Construction

- `AbsoluteLayout(Context)` le paramètre est généralement l'activité elle-même

## Les classe `ScrollView` et `HorizontalScrollView`

Un `ScrollView` est une zone à défilement vertical tandis qu'un `HorizontalScrollView` permet un défilement horizontal. Ils ne peuvent contenir qu'un seul élément. Généralement ils sont utilisés pour contenir un `LinearLayout` (dont l'orientation correspond à celle du `ScrollView`) et offrir des ascenseurs. Lorsque l'on veut avoir un défilement horizontal et vertical il suffit de mettre un `HorizontalScrollView` dans un `ScrollView`.

ATTENTION : En raison de l'écran tactile il n'est pas possible de mettre une zone avec ascenseurs dans une zone ayant elle-même des ascenseurs car le défilement se fera toujours sur la zone la plus externe (sur le contenant et non sur le contenu).

### Propriétés supplémentaires positionnables en XML

Mêmes paramètres XML que `FrameLayout` plus :

- `android:fillViewport="b"` (où b vaut true ou false) indique si le contenu doit être étiré pour occuper la place disponible ou pas.

## Méthodes des classes `ScrollView` et `HorizontalScrollView`

`android.widget.ScrollView`

`android.widget.HorizontalScrollView`

### Construction

- `ScrollView(Context)` le paramètre est généralement l'activité elle-même
- `HorizontalScrollView(Context)` le paramètre est généralement l'activité elle-même

### Défilement

- `smoothScrollBy(int, int)` permet de faire défiler la vue. Les paramètres indiquent le nombre de pixels de défilement (resp horizontalement et verticalement)
- `smoothScrollTo(int, int)` permet de faire défiler la vue. Les paramètres indiquent la nouvelle position de la vue en pixels (resp horizontalement et verticalement)
- `awakenScrollBars()` provoque l'affichage des ascenseurs (qui s'effaceront au bout d'un moment).
- `isHorizontalScrollBarEnabled()` renvoie true si l'ascenseur horizontal est autorisé.
- `setHorizontalScrollBarEnabled(boolean)` active ou pas l'ascenseur horizontal.
- `isVerticalScrollBarEnabled()` renvoie true si l'ascenseur vertical est autorisé.
- `setVerticalScrollBarEnabled(boolean)` active ou pas l'ascenseur vertical.

### Placement

- `setFillViewport(boolean)` permet de définir si le contenu doit être étiré pour occuper la place disponible (paramètre à true) ou pas (paramètre à false)

## La classe `TableLayout`

Dispose les éléments en lignes et colonnes. Chaque ligne est désignée dans le fichier XML par `<TableRow> ... </TableRow>`. Elle contient 1 ou plusieurs éléments. Le nombre de colonnes est déterminé par la ligne contenant le plus d'éléments. `TableLayout` n'a pas d'ascenseurs, on peut les ajouter en l'incluant dans un `ScrollView` ou `HorizontalScrollView`.

### Propriétés supplémentaires positionnables en XML

- `android:collapseColumns="x,y, ..."` (où x, y, ... sont des numéros de colonnes à cacher (démarrant à 0)).

- `android:shrinkColumns="x,y, ..."` (où x, y, ... sont des numéros de colonnes qui peuvent être rétrécies en fonction de la place disponible (démarrant à 0)).
- `android:stretchColumns="x,y, ..."` (où x, y, ... sont des numéros de colonnes qui peuvent être agrandies en fonction de leur contenu (démarrant à 0)).

### Propriétés positionnables en XLM pour les éléments placés à l'intérieur

- `android:layout_span="x"` (où x est un entier indiquant sur combien de colonnes s'étend l'élément (par défaut 1))

### Méthodes de la classe `TableLayout`

`android.widget.TableLayout`

#### *Construction*

- `TableLayout (Context)` le paramètre est généralement l'activité elle-même

#### *Aspect*

- `isColumnCollapsed(int)` indique si la colonne dont le numéro est donné en paramètre est cachée
- `setColumnCollapsed(int, boolean)` cache ou pas (second paramètre) la colonne dont le numéro est donné en premier paramètre
- `isColumnShrinkable(int)` indique si la colonne dont le numéro est donné en paramètre est rétrécissable
- `setColumnShrinkable(int, boolean)` rend ou pas rétrécissable (second paramètre) la colonne dont le numéro est donné en premier paramètre
- `isColumnStretchable(int)` indique si la colonne dont le numéro est donné en paramètre est étirable
- `setColumnStretchable(int, boolean)` rend ou pas étirable (second paramètre) la colonne dont le numéro est donné en premier paramètre
- `isShrinkAllColumns()` indique si toutes les colonnes sont rétrécissables
- `setShrinkAllColumns(boolean)` rend toutes les colonnes (ou aucune) rétrécissables
- `isStretchAllColumns()` indique si toutes les colonnes sont étirables
- `setStretchAllColumns(boolean)` rend toutes les colonnes (ou aucune) étirables

### La classe `RelativeLayout`

Permet de placer des éléments les uns relativement aux autres.

### Propriétés supplémentaires positionnables en XLM

- `android:gravity="g"` (où g peut prendre les valeurs : `top`, `bottom`, `left`, `right`, `center_vertical`, `fill_vertical`, `center_horizontal`, `fill_horizontal`, `center`, `fill`) définit comment se placent les éléments contenus par ce conteneur.

### Propriétés positionnables en XLM pour les éléments placés à l'intérieur

Pour définir la position d'un élément par rapport à un autre ou par rapport au `RelativeLayout` lui-même (parent) on ajoute dans la balise de cet élément :

Pour les positions par rapport au `RelativeLayout` :

- `android:layout_alignParentBottom="b"` (où b vaut true ou false)
- `android:layout_alignParentLeft="b"` (où b vaut true ou false)
- `android:layout_alignParentRight="b"` (où b vaut true ou false)
- `android:layout_alignParentTop="b"` (où b vaut true ou false)
- `android:layout_centerHorizontal="b"` (où b vaut true ou false)
- `android:layout_centerInParent="b"` (où b vaut true ou false)
- `android:layout_centerVertical="b"` (où b vaut true ou false)

Pour les positions par rapport à d'autres éléments :

- `android:layout_above="@+id/ident"`

- android:layout\_below="@+id/ident"/
- android:layout\_toLeftOf="@+id/ident"/
- android:layout\_toRightOf="@+id/ident"/
- android:layout\_alignLeft="@+id/ident"/
- android:layout\_alignRight="@+id/ident"/
- android:layout\_AlignTop="@+id/ident"/
- android:layout\_AlignBottom="@+id/ident"/

## Méthodes de la classe RelativeLayout

android.widget.RelativeLayout

### Construction

- RelativeLayout (Context) le paramètre est généralement l'activité elle-même

### Disposition

- setGravity(int) Le paramètre est l'une des constantes définies dans la classe Gravity on utilise donc Gravity. (TOP, BOTTOM, LEFT, RIGHT, CENTER\_VERTICAL, FILL\_VERTICAL, CENTER\_HORIZONTAL, FILL\_HORIZONTAL, CENTER, FILL)

## Les groupes

Les groupes permettent de regrouper des éléments en liste, tableaux, etc. Ils héritent tous de ViewGroup. Leurs propriétés sont donc au moins celles décrites ci-dessus pour View et ViewGroup auxquelles viennent s'ajouter des propriétés spécifiques décrites dans cette partie.

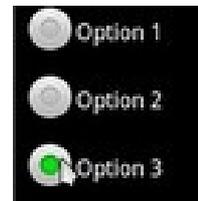
Le groupe RadioGroup ne permet que de regrouper des boutons radio et de faire en sorte qu'un seul soit sélectionné à la fois.

Les groupes ListView, GridView et Gallery permettent d'organiser des éléments quelconques.

Généralement ListView et GridView sont utilisés pour des listes ou des tableaux de chaînes de caractères tandis que Gallery est utilisé pour des images. Toutefois ces trois groupes peuvent gérer des vues quelconques (classe View ou dérivée) dans ce cas il faut écrire un gestionnaire de contenu (Adapter) et le leur associer.

## RadioGroup

Réservé au groupage de boutons radio (classe RadioButton). Le RadioGroup garantit qu'un seul bouton soit coché à la fois toutefois il ne fait aucune hypothèse sur l'état initial des boutons. Ceci signifie que si dans le fichier XML vous avez activé l'un des boutons du groupe il le restera même lorsque vous en activerez un autre. Il est donc préférable de ne mettre dans le fichier XML que des boutons non cochés puis d'initialiser le RadioGroup par sa méthode check (voir plus loin).



## Propriétés supplémentaires positionnables en XML

Mêmes paramètres XML que LinearLayout

Exemple de fichier XML :

```
<RadioGroup android:id="@+id/groupe"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:orientation="vertical">
    <RadioButton
    ...
    />
    ...
</RadioGroup>
```

## Méthodes de la classe RadioGroup

android.widget.RadioGroup

## **Construction**

- `RadioGroup(Context)` le paramètre est généralement l'activité elle-même

## **Selection**

- `check(int)` active le bouton radio dont l'identifiant est passé en paramètre
- `clearCheck()` désactive tous les boutons radio
- `getCheckedRadioButtonId()` renvoie l'identifiant du bouton radio actif

## **Evénements**

- `setOnCheckedChangeListener(RadioGroup.OnCheckedChangeListener)` associe un écouteur aux changements de sélection. La méthode `onCheckedChanged(RadioGroup, int)` de l'interface `RadioGroup.OnCheckedChangeListener` doit être surchargée pour traiter les événements. Le premier paramètre est la `RadioGroup` concerné, le second indique le rang du bouton radio activé.

## **ListView**

`ListView` place les éléments en liste avec un ascenseur vertical si nécessaire. `ListView` est normalement utilisé pour afficher des éléments textuels éventuellement accompagnés d'une case à cocher lorsqu'il s'agit d'une liste à choix multiples. Il est toutefois possible d'y afficher des éléments plus complexes en utilisant un gestionnaire de contenu.

## **Propriétés supplémentaires positionnables en XML**

### **Contenu de type texte**

- `android:entries="@array/maliste"` définit le contenu de la liste à partir d'une ressource de type tableau de chaînes de caractères définie dans un fichier xml placé dans `res/values/` sous la forme suivante :

```
<string-array name="maliste">
    <item>premier élément</item>
    <item>deuxième élément</item>
    ...
    <item>dernier élément</item>
</string-array>
```

### **Séparateurs**

- `android:divider="couleur"` définit la couleur des séparateurs ou `android:divider="@drawable/monimage"` pour utiliser une image. L'image est placée dans `res/drawable/` et s'appelle `monimage.x` (où x est png, jpg, ou gif).
- `android:dividerHeight="unité"` définit la hauteur des séparateurs (si c'est une image elle sera déformée).

### **Type de choix**

- `android:choiceMode="c"` (où c peut prendre les valeurs : `none`, `singlechoice`, `multipleChoice`) pour indiquer le mode de choix dans la liste (aucun, un seul, plusieurs).

## **Méthodes de la classe ListView**

`android.widget.ListView`

### **Construction**

- `ListView(Context)` le paramètre est généralement l'activité elle-même

## Contenu

Le contenu d'une `ListView` peut être défini de façon statique par la propriété `android:entries` dans le fichier xml. Lorsque l'on souhaite avoir un contenu dynamique on utilise un `ArrayAdapter` (collection) que l'on remplit (méthodes `add` ou `insert` du `ArrayAdapter`) et que l'on associe à la `ListView` par la méthode `setAdapter(ArrayAdapter)`. La classe `ArrayAdapter` possède les méthodes suivantes :

### Construction

- `ArrayAdapter(Context, type)` le premier paramètre est généralement l'activité elle-même, le second paramètre peut être une valeur prédéfinie : `android.R.layout.simple_list_item_1` pour une liste à choix unique ou `android.R.layout.simple_list_item_multiple_choice` pour une liste à choix multiple (dans ce cas une case à cocher apparaît à coté de chaque élément de la liste). Le second paramètre peut également être l'identificateur d'un widget `TextView` personnalisé défini dans un fichier xml placé dans **res/layout** et désigné par `R.layout.nom_du_fichier_xml`

### Eléments

- `add(Object)` pour ajouter à la fin
- `insert(Object, int)` pour insérer un élément au rang donné en 2<sup>ème</sup> paramètre
- `clear()` pour enlever tous les éléments
- `remove(Object)` pour enlever un élément
- `getCount()` renvoie le nombre d'éléments
- `getItem(int)` renvoie l'élément (Object) dont le rang est donné en 2<sup>ème</sup> paramètre
- `getPosition(Object)` renvoie le rang de l'objet désigné en paramètre

Remarque : Pour gérer un contenu autre que du texte il existe un gestionnaire (classe `SimpleAdapter`) permettant que chaque élément de la liste soit une petite interface définie dans un fichier xml. On peut également écrire son propre gestionnaire de contenu en définissant une classe héritant de `BaseAdapter` (voir exemple fourni pour la classe `Gallery` plus loin).

### Aspect

- `setDivider(Drawable)` définit l'image qui sera dessinée entre chaque élément de la liste
- `getDivider()` retourne l'image dessinée entre chaque élément de la liste
- `setDividerHeight(int)` définit la hauteur de la séparation entre chaque élément de la liste
- `getDividerHeight()` renvoie la hauteur de la séparation entre chaque élément de la liste
- `addFooter(View)` ajoute une vue en bas de la liste
- `addHeader(View)` ajoute une vue en haut de la liste
- `setFooterDividersEnabled(boolean)` autorise ou interdit la dessin d'un séparateur pour le bas de liste
- `setHeaderDividersEnabled(boolean)` autorise ou interdit la dessin d'un séparateur pour le haut de liste
- `invalidateViews()` provoque le rafraichissement des tous le éléments internes

### Selection

- `setItemChecked(int, boolean)` sélectionne (true) ou désélectionne (false) un élément de liste par son rang (à partir de 0).
- `isItemChecked(int)` renvoie un booléen qui indique si l'élément de liste désigné par son rang (à partir de 0) est sélectionné.
- `clearChoices()` invalide les choix actuels
- `getChoiceMode()` renvoie le mode de choix peut prendre les valeurs : `none`, `singleChoice` ou `multipleChoice`
- `setChoiceMode(int)` définit le mode de choix le paramètre peut prendre les valeurs : `none`, `singleChoice` ou `multipleChoice`
- `clearTextFilter()` supprime le filtrage des éléments au fur et à mesure de la saisie

- `getTextFilter()` renvoie la chaîne de caractères utilisée pour le filtrage des éléments au fur et à mesure de la saisie
- `isTextFilterEnabled()` renvoie un booléen qui indique si le filtrage des éléments au fur et à mesure de la saisie est actif ou pas
- `setFilterText(String)` définit la chaîne de caractères utilisée pour le filtrage des éléments au fur et à mesure de la saisie
- `setTextFilterEnabled(boolean)` active ou désactive le filtrage des éléments au fur et à mesure de la saisie

### ***Evénements***

- `setOnClickListener(AdapterView.OnItemClickListener)` associe un écouteur d'événements au clic sur un élément. La méthode `onClick(AdapterView<?>, View, int, long)` de l'interface `AdapterView.OnItemClickListener` est surchargée pour traiter l'événement. Le 3<sup>ème</sup> paramètre indique le rang de l'élément et le dernier son identifiant.
- `setOnItemLongClickListener(AdapterView.OnItemLongClickListener)` associe un écouteur d'événements au clic long sur un élément. La méthode `onItemLongClick(AdapterView<?>, View, int, long)` de l'interface `AdapterView.OnItemLongClickListener` est surchargée pour traiter l'événement. Le 3<sup>ème</sup> paramètre indique le rang de l'élément et le dernier son identifiant.
- `setOnItemSelectedListener(AdapterView.OnItemSelectedListener)` associe un écouteur d'événements à la sélection d'un élément (pour les listes à choix multiples). La méthode `onItemSelected(AdapterView<?>, View, int, long)` de l'interface `AdapterView.OnItemSelectedListener` est surchargée pour traiter l'événement de sélection d'un élément. Le 3<sup>ème</sup> paramètre indique le rang de l'élément et le dernier son identifiant. La méthode `onNothingSelected(AdapterView<?>)` est surchargée pour traiter l'événement de non sélection.

### **GridView**

Fonctionne comme un `ListView` mais place les éléments en tableau avec ascenseur vertical seulement si nécessaire.

### **Propriétés supplémentaires positionnables en XML**

- `android:gravity="g"` (où `g` peut prendre les valeurs : `top`, `bottom`, `left`, `right`, `center_vertical`, `fill_vertical`, `center_horizontal`, `fill_horizontal`, `center`, `fill`) définit comment se placent les éléments contenus par ce conteneur.

### **Couleurs**

- `android:listSelector="couleur"` pour définir une couleur pour désigner l'objet sélectionné

### **Sélection**

- `android:choiceMode="m"` (où `m` peut prendre les valeurs : `none`, `singleChoice` ou `multipleChoice` définit le type de sélection
- `android:textFilterEnabled="b"` (où `b` vaut `true` ou `false`) mise en place d'un filtrage des éléments au fur et à mesure de la saisie
- `android:drawSelectorOnTop="b"` (où `b` vaut `true` ou `false`) `true` indique que le selecteur est dessiné sur l'élément choisi (par défaut c'est `false`)
- `android:fastScrollEnabled="b"` (où `b` vaut `true` ou `false`) autorise ou pas le parcours rapide

### **Dimensions et espacement des colonnes**

- `android:numColumns="x"` (où `x` est une valeur entière) définit le nombre de colonnes
- `android:columnWidth="unité"`
- `android:horizontalSpacing="unité"`
- `android:verticalSpacing="unité"`

- `android:stretchMode="s"` (où s peut prendre les valeurs : none, spacingWidth, columnWidth, spacingWidthUniform) définit comment les colonnes occupent la place disponible : none=non occupée, spacingWidth=étirement de l'espace entre les colonnes, columnWidth=étirement des colonnes, spacingWidthUniform=étirement uniforme de toutes les colonnes.

## Méthodes de la classe GridView

`android.widget.GridView`

### Construction

- `GridView(Context)` le paramètre est généralement l'activité elle-même

### Contenu

Pour remplir un GridView on utilise les mêmes mécanismes que ceux décrits plus haut pour la classe ListView.

### Position et disposition

- `setGravity(int)` Le paramètre est l'une des constantes définies dans la classe Gravity on utilise donc Gravity. (TOP, BOTTOM, LEFT, RIGHT, CENTER\_VERTICAL, FILL\_VERTICAL, CENTER\_HORIZONTAL, FILL\_HORIZONTAL, CENTER, FILL)
- `setStretchMode(int)` définit comment les colonnes occupent la place disponible le paramètre peut prendre les valeurs : none, spacingWidth, columnWidth, spacingWidthUniform) none=non occupée, spacingWidth=étirement de l'espace entre les colonnes, columnWidth=étirement des colonnes, spacingWidthUniform=étirement uniforme de toutes les colonnes
- `getStretchMode()` renvoie l'information de placement des colonnes (voir `setStretchMode` pour la valeur renvoyée).
- `setColumnWidth(int)` définit en pixels la largeur des colonnes
- `setHorizontalSpacing(int)` définit en pixels l'espacement entre les colonnes
- `setVerticalSpacing(int)` définit en pixels l'espacement entre les lignes
- `setNumColumns(int numColumns)` définit le nombre de colonnes
- `invalidateViews()` provoque le rafraichissement des tous le éléments internes

### Selection

- `setSelection(int)` sélectionne un éléments de liste par son rang (à partir de 0).
- `clearChoices()` invalide les choix actuels
- `getChoiceMode()` renvoie le mode de choix peut prendre les valeurs : none, singleChoice ou multipleChoice
- `setChoiceMode(int)` définit le mode de choix le paramètre peut prendre les valeurs : none, singleChoice ou multipleChoice
- `clearTextFilter()` supprime le filtrage des éléments au fur et à mesure de la saisie
- `getTextFilter()` renvoie la chaîne de caractères utilisée pour le filtrage des éléments au fur et à mesure de la saisie
- `isTextFilterEnabled()` renvoie un booléen qui indique si le filtrage des éléments au fur et à mesure de la saisie est actif ou pas
- `setFilterText(String)` définit la chaîne de caractères utilisée pour le filtrage des éléments au fur et à mesure de la saisie
- `setTextFilterEnabled(boolean)` active ou désactive le filtrage des éléments au fur et à mesure de la saisie

### Evénements

Mêmes méthodes que pour ListView

## Gallery

Cette classe est généralement utilisée pour faire une galerie d'images mais elle peut contenir d'autres types d'éléments (voir plus loin). Le contenu d'une galerie est défini dans le code java par un gestionnaire de contenu.



### Propriétés supplémentaires positionnables en XLM

- `android:gravity="g"` (où `g` peut prendre les valeurs : `top`, `bottom`, `left`, `right`, `center_vertical`, `fill_vertical`, `center_horizontal`, `fill_horizontal`, `center`, `fill`) définit comment se placent les éléments dans la galerie.
- `android:animationDuration="x"` (où `x` est une valeur entière) définit la durée de la transition (en ms) lorsque l'on passe d'un élément à l'autre.
- `android:unselectedAlpha="x"` (où `x` est une valeur réelle) définit la transparence des éléments non sélectionnés.

Exemple de fichier XML :

```
<Gallery android:id="@+id/galerie"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
/>
```

### Méthodes de la classe Gallery

`android.widget.Gallery`

### Construction

- `Gallery(Context)` le paramètre est généralement l'activité elle-même

### Contenu

- `setAdapter(Adapter)` permet de définir le contenu de la galerie. Lorsqu'il s'agit d'images, la façon de procéder est la suivante :
  1. Récupérer l'objet de classe `Gallery` défini dans le fichier xml de description de l'interface par :  
`galerie = (Gallery) findViewById(R.id.identificateur_de_votre_galerie_dans_le_xml);`
  2. Lui associer un gestionnaire de contenu par :  
`galerie.setAdapter(new GestionnaireDeContenuImages(mon_activité, largeur, hauteur));`  
Le premier paramètre est l'activité elle-même. Les 2 derniers paramètres définissent la hauteur et la largeur des images affichées dans la galerie. Celles-ci seront éventuellement déformées pour obtenir ces dimensions.
  3. Définir une classe pour ce gestionnaire de contenu par :

```
public class MonGestDeContenu extends BaseAdapter {
    private Context contexte;
    // Tableau des identificateurs d'images.
    // Les images ont été placées dans res/drawable/hdmi sous les noms a_1 ... a_N
    private Integer[] identificateurDIImages = { R.drawable.a_1, R.drawable.a_2, ... , R.drawable.a_N };
    private int largeur, hauteur;
    private int marges;
    private int couleurFond;

    public GestionnaireDeContenuImages(Context ctx, int l, int h) {
        contexte = ctx;
        largeur = l; hauteur = h; // dimensions des images
        marges = 6; // marges à droite et à gauche de chaque image (éventuellement à passer en paramètre)
        couleurFond = Color.BLACK; // couleur de fond des images (éventuellement à passer en paramètre)
    }

    public int getCount() { return identificateurDIImages.length; }
}
```

```

    public Object getItem(int position) {
        return contexte.getResources().getDrawable(imgId[position]);
    }

    public long getItemId(int position) {
        return imgId[position];
    }

    public View getView(int position, View ancienneVue, ViewGroup vueParente) {
        ImageView vueDImage = new ImageView(contexte);
        vueDImage.setImageResource(identificateurDImages[position]);
        vueDImage.setLayoutParams(new Gallery.LayoutParams(largeur, hauteur));
        vueDImage.setPadding(marges, 0, marges, 0);
        vueDImage.setScaleType(ImageView.ScaleType.FIT_XY);
        vueDImage.setBackgroundColor(couleurFond);
        return vueDImage;
    }
}

```

### ***Disposition et aspect***

- `setGravity(int)` Le paramètre est l'une des constantes définies dans la classe `Gravity` on utilise donc `Gravity`. (`TOP`, `BOTTOM`, `LEFT`, `RIGHT`, `CENTER_VERTICAL`, `FILL_VERTICAL`, `CENTER_HORIZONTAL`, `FILL_HORIZONTAL`, `CENTER`, `FILL`)
- `setSpacing(int)` définit l'espace en pixels entre les éléments de la galerie
- `setUnselectedAlpha(float)` définit la transparence pour les éléments non sélectionnés
- `showContextMenu()` rend visible le menu contextuel

### ***Sélection***

- `getCount()` renvoie le nombre d'éléments de la galerie
- `setSelection(int, boolean)` sélectionne un élément de la galerie par son rang (à partir de 0), le second paramètre active ou pas l'animation liée à cette sélection
- `getSelectedItemId()` renvoie l'identificateur l'élément sélectionné
- `getSelectedItemPosition()` renvoie le rang de l'élément sélectionné (à partir de 0)

### ***Evénements***

Mêmes méthodes que pour `ListView`

### ***Animation***

- `setAnimationDuration(int)` définit la durée (en ms) de l'animation de changement d'élément

### **Utilisation de la classe `Gallery` pour faire une interface organisée en pages**

La classe `Gallery` est normalement utilisée pour faire défiler des images. Toutefois elle ne se restreint pas à ce seul usage, en effet il est possible d'utiliser une galerie contenant des interfaces. Le contenu d'une galerie est géré par un gestionnaire de contenu (`Adapter`) si l'on écrit un gestionnaire de contenu qui gère des contenants (`ViewGroup`) on peut réaliser une interface dont on fait défiler les pages (comme le fait le bureau d'Android).

Fichier XML de l'interface de base de l'application (par exemple "main.xml") placé dans **res/layout** :

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<Gallery
    android:id="@+id/magalerie"

```

```

        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:unselectedAlpha="0.5"
        android:animationDuration="1000"
    />
</LinearLayout>

```

Il ne contient qu'un `LinearLayout` ne contenant qu'une galerie.

Il faut ensuite définir autant de fichiers xml que de pages chacun décrivant l'interface d'une page de façon tout à fait classique. Supposons que ces fichiers s'appellent **page1.xml** , **page2.xml** ... Ils sont placés dans **res/layout**.

On va associer à la galerie un gestionnaire de contenu qui crée les pages à partir de ces fichiers :

```

public class GaleriePagesActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        ...
        Gallery galerie = (Gallery) findViewById(R.id.magalerie);
        galerie.setAdapter(new MonGestDeContenu());
        ...
    }
    public class MonGestDeContenu extends BaseAdapter {
        private static final int nombreDePages = 2; // l'exemple a 2 pages

        public MonGestDeContenu() {
        }
        @Override
        public View getView(int position, View old, ViewGroup parent) {
            View vue = null;
            LayoutInflater decode =
                LayoutInflater.from(GaleriePagesActivity.this);
            switch (position) {
                case 0:
                    vue = decode.inflate(R.layout.page1, parent, false);
                    break;
                case 1 :
                    vue = decode.inflate(R.layout.page2, parent, false);
                    break;
            }
            return vue;
        }
        @Override
        public int getCount() {
            return nombreDePages;
        }
        @Override
        public Object getItem(int position) {
            return position;
        }
        @Override
        public long getItemId(int position) {
            return position;
        }
    }
}

```

Si l'on veut associer des écouteurs d'événements aux éléments d'interface qui sont dans les pages, il faudra le faire lorsque la page est affichée c'est-à-dire dans chaque branche du switch du gestionnaire de contenu de la galerie. Ceci pourra se faire de la façon suivante (on suppose pour cet exemple avoir dans la première page un bouton dont l'identifiant dans le fichier "page1.xml" est "mon\_bouton") :

```
public class MonGestDeContenu extends BaseAdapter {
    private static final nombreDePages = 2;
    private ActionBouton actionBouton = null;

    public MonGestDeContenu() {
        actionBouton = new ActionBouton();
    }
    @Override
    public View getView(int position, View old, ViewGroup parent) {
        View vue = null;
        LayoutInflater decode =
            LayoutInflater.from(GalleriePagesActivity.this);
        switch (position) {
            case 0:
                vue = decode.inflate(R.layout.page1, parent, false);
                Button bouton = (Button) vue.findViewById(R.id.mon_bouton);
                bouton.setOnClickListener(actionBouton);
                break;
            case 1 :
                ...
        }
    }
}
```

Il ne reste plus qu'à écrire la classe ActionBouton qui réagit aux événements du bouton "mon\_bouton"

## Les composants d'interface

Ce sont les composants finals d'interface qui permettent l'interaction de l'utilisateur.

Leur position est définie :

- Lorsqu'ils sont placés dans un `AbsoluteLayout` par :  
`android:layout_x="unité"`  
`android:layout_y="unité"`
- Lorsqu'ils sont placés dans un `RelativeLayout` par rapport à ce contenant et/ou aux autres composants (voir `RelativeLayout`)
- Lorsqu'ils sont placés dans un autre contenant par ce contenant (voir ci-dessous)

Leur taille est définie par :

- `android:layout_height="t"` (où t peut être une unité ou prendre les valeurs : `FILL_PARENT` ou `WRAP_CONTENT`) pour occuper tout l'espace en hauteur ou seulement ce qui est nécessaire
- `android:layout_width="t"` (où t peut être une unité prendre les valeurs : `FILL_PARENT` ou `WRAP_CONTENT`) pour occuper tout l'espace en largeur ou seulement ce qui est nécessaire

Sauf pour `AbsoluteLayout` et `RelativeLayout`, leur occupation du conteneur est définie par :

- `android:layout_gravity="g"` (où g peut prendre les valeurs : `top`, `bottom`, `left`, `right`, `center_vertical`, `fill_vertical`, `center_horizontal`, `fill_horizontal`, `center`, `fill`). Il est possible de composer ces valeurs par | (par exemple : `top|right`)

**Remarque :** le fonctionnement de `layout_gravity` est plutôt étrange dans un `LinearLayout`. En effet ce contenant place les éléments les uns à côté des autres ou les uns sous les autres selon son orientation, et `layout_gravity` n'a d'effet que sur l'autre direction c'est-à-dire que pour un `LinearLayout` horizontal seules les valeurs `top`, `center` et `bottom` ont un sens alors que pour un `LinearLayout` vertical seules les valeurs `left`, `center` et `right` ont un sens. Si on veut définir un placement de certains éléments dans le sens du `LinearLayout` lui-même il faut les inclure dans un `FrameLayout` et les placer dans ce `FrameLayout`.

Exemple :

La description ci-dessous ne placera pas, comme on pourrait s'y attendre, la case à cocher à droite de l'écran

Essai placement



```
<LinearLayout
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical"
        android:text="Essai placement"
    />
    <CheckBox
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="top|right"
    />
</LinearLayout>
```

Pour obtenir le résultat souhaité il faut encapsuler la case à cocher dans un `FrameLayout` :

```
<FrameLayout
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
>
    <CheckBox
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="top|right"
    />
</FrameLayout>
```

Essai placement



Pour un `TableLayout` le nombre de colonnes occupées par l'élément est défini par :

- `android:layout_span="s"` (où `s` indique le nombre de colonnes (par défaut 1))

Les marges externes sont définies par :

- `android:layout_marginBottom="unité"` marge externe en bas
- `android:layout_marginLeft="unité"` marge externe à gauche
- `android:layout_marginRight="unité"` marge externe à droite
- `android:layout_marginTop="unité"` marge externe en haut

## ImageView

Il s'agit simplement d'une zone dans laquelle s'affiche une image.

## Propriétés supplémentaires positionnables en XML :

### Contenu

- `android:src="couleur"` pour définir une couleur ou `="@drawable/monimage"` pour définir une image. L'image est placée dans `res/drawable/` et s'appelle `monimage.x` (où `x` est `png`, `jpg`, ou `gif`). Méthodes correspondantes : `setImageBitmap`, `setImageDrawable`, `setImageResource`

### Dimensions

- `android:tint="couleur"` pour définir une couleur qui teinte l'image
- `android:adjustViewBounds="b"` (où `b` vaut `true` ou `false`) `false` indique que la taille de la zone d'affichage correspond à celle de l'image

Dans le cas où ce paramètre est à true on précise le comportement de l'image

- `android:baselineAlignBottom="b"` (où b vaut true ou false) indique si l'image est placée en bas de la zone
- `android:cropToPadding="b"` (où b vaut true ou false) true indique que l'image sera coupée si elle est plus grande que la taille de la zone d'affichage
- `android:scaleType="s"` (où s peut prendre les valeurs :
  - o `matrix` l'image n'est ni centrée ni redimensionnée
  - o `center` l'image est centrée mais non redimensionnée
  - o `fitXY` l'image est déformée pour occuper tout l'espace
  - o `fitStart` l'image est cadrée en haut à gauche et redimensionnée
  - o `fitCenter` l'image est centrée et redimensionnée
  - o `fitEnd` l'image est cadrée en bas à droite et redimensionnée
  - o `centerCrop` l'image est centrée et redimensionnée selon sa dimension minimale et non déformée, l'autre dimension pourra donc être coupée
  - o `centerInside` l'image est centrée et redimensionnée selon sa dimension maximale et non déformée; elle ne sera donc pas coupée

Exemple de fichier XML :

```
<ImageView android:id="@+id/image"
  android:src="@drawable/photo"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:adjustViewBounds="true"
/>
```

## Méthodes de la classe **ImageView**

`android.widget.ImageView`

### **Construction**

- `ImageView(Context)` le paramètre est généralement l'activité elle-même

### **Aspect**

- `getDrawable()` renvoie l'image du bouton (objet de classe `Drawable`)
- `setAlpha(int)` définit la transparence (le paramètre est un réel entre 0 et 1)
- `setImageBitmap(Bitmap)` définit l'image (objet de classe `Bitmap`)
- `setImageDrawable(Drawable)` définit l'image (objet de classe `Drawable`)
- `setImageResource(int)` définit l'image par son identificateur
- `setMaxHeight(int)` définit la hauteur maximale en pixels
- `setMaxWidth(int)` définit la largeur maximale en pixels
- `setScaleType(int)` définit le positionnement de l'image. les valeurs possibles du paramètre sont : `ImageView.ScaleType.MATRIX`, `CENTER`, `FIT_XY`, `FIT_START`, `FIT_CENTER`, `FIT_END`, `CENTER_CROP` ou `CENTER_INSIDE`. Après avoir modifié le positionnement si on n'a pas changé l'image il faut appeler la méthode `invalidate()` de `ImageView` pour que ce soit pris en compte.

## **TextView**

Affichage de texte a priori non éditable (pour du texte éditable utiliser `EditView`).

## **Propriétés supplémentaires positionnables en XLM**

### **Contenu**

- `android:text="texte à afficher"`
- `android:hint="initial"` définit le texte à afficher quand la zone est vide

## ***Taille et aspect du texte***

- `android:gravity="g"` (où `g` peut prendre les valeurs : `top`, `bottom`, `left`, `right`, `center_vertical`, `fill_vertical`, `center_horizontal`, `fill_horizontal`, `center`, `fill`) définit comment se place le texte
- `android:textSize="unité"` utiliser de préférence l'unité **sp** qui est liée aux polices
- `android:textScaleX="x"` (où `x` est une valeur réelle) définit l'échelle horizontale du texte
- `android:textStyle="g"` (où `s` peut être `normal`, `bold`, `italic`) ces styles peuvent être combinés par |
- `android:typeface="s"` (où `s` peut être `normal`, `sans`, `serif`, `monospace`)
- `android:singleLine="b"` (où `b` vaut `true` ou `false`) limite le texte à une seule ligne
- `android:lines="x"` (où `x` est une valeur entière) définit le nombre de lignes du texte
- `android:maxLines="x"` (où `x` est une valeur entière) définit le nombre maximal de lignes du texte
- `android:minLines="x"` (où `x` est une valeur entière) définit le nombre minimal de lignes du texte
- `android:lineSpacingExtra="unité"` espace supplémentaire entre les lignes
- `android:scrollHorizontally="b"` (où `b` vaut `true` ou `false`) autorise ou pas le défilement horizontal du texte

## ***Dimensions***

- `android:ems="x"` (où `x` est une valeur entière) définit la taille du texte en caractères
- `android:maxEms="x"` (où `x` est une valeur entière) définit le nombre maximal de caractères des lignes du texte
- `android:height="unité"`
- `android:maxHeight="unité"`
- `android:minHeight="unité"`
- `android:width="unité"`
- `android:maxLength="unité"`
- `android:minWidth="unité"`

## ***Comportement***

- `android:autoLink="a"` (où `a` peut être : `none`, `web`, `email`, `phone`, `map` ou `all`) indique si les liens de ce type apparaissant dans le texte sont automatiquement rendus cliquables.
- `android:autoText="b"` (où `b` vaut `true` ou `false`) valide ou pas le mode correction du texte
- `android:capitalize="c"` (où `c` peut être : `none`, `sentences`, `words`, `characters`) indique le type de saisies que le texte mémorise et peut re-proposer.
- `android:digits="b"` (où `b` vaut `true` ou `false`) `true` indique que la saisie n'accepte que du numérique
- `android:numeric="x"` (où `x` peut être `integer`, `signed`, `decimal`) définit le mode de saisie numérique
- `android:password="b"` (où `b` vaut `true` ou `false`) si `true` lors de la saisie des points sont affichés
- `android:phoneNumber="b"` (où `b` vaut `true` ou `false`) `true` indique que la saisie n'accepte que des numéros de téléphone
- `android:inputType="t"` (où `t` peut être : `none`, `text`, `textCapCharacters`, `textCapWords`, `textCapSentences`, `textAutoCorrect`, `textAutoComplete`, `textMultiLine`, `textUri`, `textEmailAddress`, `textEmailSubject`, `textShortMessage`, `textLongMessage`, `textPersonName`, `textPostalAddress`, `textPassword`, `textVisiblePassword`, `textWebEditText`, `textFilter`, `textPhonetic`, `textWebEmailAddress`, `textWebPassword`, `number`, `numberDecimal`, `numberPassword`, `phone`, `datetime`, `date` ou `time`) définit le mode de saisie.

## ***Affichage***

- `android:cursorVisible="b"` (où `b` vaut `true` ou `false`) rend visible ou non le curseur
- `android:editable="b"` (où `b` vaut `true` ou `false`) autorise ou pas la modification du texte
- `android:ellipsize="e"` (où `e` peut être : `none`, `start`, `middle`, `end`, `marquee`) définit le mode de césure du texte

- `android:linksClickable="b"` (où b vaut true ou false) rend ou pas les liens cliquables
- `android:textIsSelectable="b"` (où b vaut true ou false) autorise ou interdit le sélection dans le texte

### ***Couleurs et images***

- `android:textColor="couleur"` pour définir la couleur du texte
- `android:textColorHighlight="couleur"` pour définir la couleur de surlignage du texte
- `android:textColorHint="couleur"` pour définir la couleur du texte par défaut
- `android:textColorLink="couleur"` pour définir la couleur des liens dans le texte
- `android:drawableBottom="couleur"` pour définir la couleur de fond du texte ou `="@drawable/monimage"` pour définir une image de fond au texte. L'image est placée dans `res/drawable/` et s'appelle `monimage.x` (où x est `png`, `jpg`, ou `gif`)
- `android:shadowColor="couleur"` pour définir la couleur d'ombrage du texte
- `android:shadowDx="x"` (où est une valeur réelle) définit le décalage en X de l'ombre (voir `shadowRadius`)
- `android:shadowDy="x"` (où est une valeur réelle) définit le décalage en Y de l'ombre (voir `shadowRadius`)
- `android:shadowRadius="x"` (où est une valeur réelle) définit l'épaisseur de l'ombre (ATTENTION : par défaut c'est 0 donc il n'y a pas d'ombre)

Exemple de fichier XML :

```
<TextView android:id="@+id/untexte"
  android:text="Contenu du texte"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
/>
```

### **Méthodes de la classe `TextView`**

`android.widget.TextView`

#### ***Construction***

- `TextView(Context)` le paramètre est généralement l'activité elle-même

#### ***Manipulation du texte***

- `getText()` renvoie le texte contenu dans le `TextView` sous forme d'un objet de classe `Editable` que l'on peut transformer en `String` par sa méthode `toString()` c'àd par : `getText().toString()`
- `setText(CharSequence)` définit le texte du `TextView`
- `append(CharSequence)` ajoute du texte au `TextView`
- `getLineCount()` renvoie le nombre de lignes du texte contenu dans le `TextView`
- `length()` renvoie la longueur du texte contenu dans le `TextView`
- `getSelectionStart()` renvoie la position de début du texte sélectionné
- `getSelectionEnd()` renvoie la position de fin du texte sélectionné
- `hasSelection()` indique s'il y a ou pas du texte sélectionné (true si oui, false si non)

#### ***Aspect***

- `setCursorVisible(boolean)` rend le curseur visible (true) ou invisible (false)
- `setEnabled(boolean)` rend le `TextView` actif ou pas
- `getCurrentTextColor()` renvoie la couleur actuelle du texte normal
- `setTextColor(int)` définit la couleur du texte
- `setHighlightColor(int)` définit la couleur du texte surligné
- `setHint(CharSequence)` définit le texte à afficher quand la zone est vide

## Événements

- `addTextChangedListener(TextWatcher)` pour associer un écouteur d'événements aux modifications du texte. L'interface `TextWatcher` possède les méthodes à surcharger suivantes :
  - o `beforeTextChanged(CharSequence, int, int, int)` appelée avant la modification du texte
  - o `onTextChanged(CharSequence, int, int, int)` appelée après la modification du textePour ces 2 méthodes les paramètres sont :
  - o Le texte (avant ou après sa modification)
  - o Le point de départ de la modification (rang du caractère)
  - o Le nombre de caractères remplacés
  - o Le nombre de caractères de remplacement
  
- `setOnEditorActionListener(TextView.OnEditorActionListener)` pour associer un écouteur d'événements aux actions. L'interface `TextView.OnEditorActionListener` possède la méthode à surcharger suivante :
  - o `onEditorAction(TextView, int, KeyEvent)` appelée lorsqu'une action est effectuée (en particulier quand on tape la touche Entrée). Cette méthode doit retourner un booléen (true si l'événement a été traité)Les paramètres sont :
  - o Le `TextView` lui-même
  - o Un identificateur d'action ou la valeur `EditorInfo.IME_NULL` lorsque l'action provient de la touche Entrée
  - o L'événement clavier lorsque l'action provient de la touche Entrée
  
- `setKeyListener(KeyListener)` pour associer un écouteur d'événements au clavier. L'interface `KeyListener` possède les méthodes à surcharger suivantes :
  - o `onKeyDown(View, Editable, int, KeyEvent)` appelée lorsqu'une touche est appuyée
  - o `onKeyUp(View, Editable, int, KeyEvent)` appelée lorsqu'une touche est lâchéePour ces 2 méthodes les paramètres sont :
  - o Le `TextView` lui-même
  - o Le texte
  - o Le code de la touche (voir `OnKeyListener`)
  - o L'événement clavier

## EditText

C'est la même chose que `TextView` mais pour du texte éditable

Exemple de fichier XML :

```
<EditText android:id="@+id/texte"
  android:text="contenu"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:typeface="serif"
  android:phoneNumber="true"
/>
```

## Méthodes de la classe EditText

`android.widget.EditText`

### Construction

- `EditText(Context)` le paramètre est généralement l'activité elle-même

### Manipulation du texte

- `getText()` renvoie le texte contenu dans le `EditText`

- `selectAll()` sélectionne tout le texte
- `setSelection(int start, int stop)` sélectionne le texte entre les positions passées en paramètre (début, fin).

## AutoCompleteTextView

Texte éditable proposant de la complétion. Possède les mêmes paramètres XML que `EditText` plus :

- `android:completionHint="texte"` texte affiché en titre du menu déroulant
- `android:completionThreshold="x"` (où `x` est une valeur entière) définit le nombre de caractères à taper avant que la complétion n'entre en action.
- `android:dropDownHeight="unité"` on peut aussi utiliser les constantes `fill_parent` et `wrap_content`, définit la hauteur du menu déroulant
- `android:dropDownWidth="unité"` on peut aussi utiliser les constantes `fill_parent` et `wrap_content`, définit la largeur du menu déroulant
- `android:dropDownHorizontalOffset="unité"` décalage horizontal du menu déroulant
- `android:dropDownVerticalOffset="unité"` décalage vertical du menu déroulant



## Méthodes de la classe AutoCompleteTextView

`android.widget.AutoCompleteTextView`

### Construction

- `AutoCompleteTextView(Context)` le paramètre est généralement l'activité elle-même

### Aspect et contenu

- `showDropDown()` montre la liste déroulante
- `dismissDropDown()` cache la liste déroulante
- `getListSelection()` renvoie le rang de la proposition choisie
- `setListSelection(int)` choisit une proposition par son rang
- `setAdapter(ArrayAdapter)` Permet de remplir la liste de propositions. La classe `ArrayAdapter` est décrite dans `ListView`.
- `setCompletionHint(CharSequence)` définit le texte affiché en titre du menu déroulant

### Evénements

- `setOnClickListener(AdapterView.OnItemClickListener)` associe un écouteur d'événements au clic sur un élément proposé. La méthode `onClick(AdapterView<?>, View, int, long)` de l'interface `AdapterView.OnItemClickListener` est surchargée pour traiter l'événement. Le 3<sup>ème</sup> paramètre indique le rang de l'élément et le dernier son identifiant.
- `setOnItemSelectedListener(AdapterView.OnItemSelectedListener)` associe un écouteur d'événements à la sélection d'un élément proposé. La méthode `onItemSelected(AdapterView<?>, View, int, long)` de l'interface `AdapterView.OnItemSelectedListener` est surchargée pour traiter l'événement de sélection d'un élément. Le 3<sup>ème</sup> paramètre indique le rang de l'élément et le dernier son identifiant. La méthode `onNothingSelected(AdapterView<?>)` est surchargée pour traiter l'événement de non sélection.

## Button

Mêmes paramètres XML que `TextView`



Exemple de fichier XML :

```
<Button android:id="@+id/compte"
```

```

android:text="Count "
android:layout_width="fill_parent "
android:layout_height="fill_parent "
/>

```

## Méthodes de la classe Button

android.widget.Button

### Construction

- Button(Context) le paramètre est généralement l'activité elle-même

## ImageButton

Mêmes paramètres XML que ImageView



Exemple de fichier XML :

```

<ImageButton android:id="@+id/boutonloupe "
  android:src="@drawable/loupe "
  android:layout_width="wrap_content "
  android:layout_height="wrap_content "
/>

```

## Méthodes de la classe ImageButton

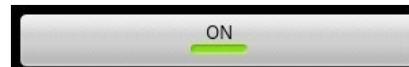
android.widget.ImageButton

ImageButton hérite de ImageView il a donc les mêmes sauf le constructeur :

### Construction

- ImageButton(Context) le paramètre est généralement l'activité elle-même

## ToggleButton



Mêmes paramètres XML que TextView plus :

- android:disabledAlpha="x" (où x est une valeur réelle entre 0 et 1) définit la transparence appliquée lorsque le bouton est inactif
- android:textOff="txt" définit le texte quand le bouton n'est pas allumé
- android:textOn="txt" définit le texte quand le bouton n'est pas allumé
- android:checked="b" (où b vaut true ou false) place le bouton en position allumé ou éteint

## Méthodes de la classe ToggleButton

android.widget.ToggleButton

### Construction

- ToggleButton(Context) le paramètre est généralement l'activité elle-même

### Etat

- isChecked() indique si le bouton est allumé ou éteint (true/false)
- setChecked(boolean) positionne le bouton (true = allumé, false = éteint)
- toggle() change l'état du bouton

### Aspect

- getTextOff() renvoie le texte associé au bouton quand il n'est pas allumé
- getTextOn() renvoie le texte associé au bouton quand il est allumé
- setTextOff(CharSequence) définit le texte associé au bouton quand il n'est pas allumé
- setTextOn(CharSequence) définit le texte associé au bouton quand il est allumé

- setBackgroundDrawable (Drawable) définit le fond du bouton

## CheckBox



Mêmes paramètres XML que TextView plus :

- android:checked="b" où b vaut true ou false coche ou décoche la case au départ

Exemple de fichier XML :

```
<CheckBox android:id="@+id/case"
  android:text="Checkbox"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:checked="true"
/>
```

## Méthodes de la classe CheckBox

android.widget.CheckBox

### Construction

- CheckBox (Context) le paramètre est généralement l'activité elle-même

### Etat

- isChecked() renvoie true si la case est cochée
- setChecked(boolean) coche (true) ou décoche (false) la case
- toggle() change l'état de la case

## RadioButton

Mêmes paramètres XML que TextView plus :

- android:checked="b" où b vaut true ou false coche ou décoche le bouton au départ



Exemple de fichier XML :

```
<RadioButton android:id="@+id/gauche"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Left"
  android:checked="false"
  android:layout_gravity="left"
/>
```

## Méthodes de la classe RadioButton

android.widget.RadioButton

### Construction

- RadioButton (Context) le paramètre est généralement l'activité elle-même

### Etat

- isChecked() renvoie true si la case est cochée
- setChecked(boolean) coche (true) ou décoche (false) la case
- toggle() change l'état de la case

## Spinner



Propose une liste de choix. Le choix actuellement sélectionné est affiché, la flèche permet de faire apparaître les autres possibilités sous la forme d'un RadioGroup.

### Propriétés supplémentaires positionnables en XML

- `android:gravity="g"` (où `g` peut prendre les valeurs : `top`, `bottom`, `left`, `right`, `center_vertical`, `fill_vertical`, `center_horizontal`, `fill_horizontal`, `center`, `fill`) définit comment se placent les éléments de choix.
- `android:prompt="texte"` définit le titre de la fenêtre qui s'ouvre lorsque l'on fait un choix
- `android:entries="@array/maliste"` définit le contenu de la liste à partir d'une ressource de type tableau de chaînes de caractères mise dans un fichier xml placé dans `res/values/` sous la forme :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string-array name="maliste">
    <item>Mercure</item>
    <item>Venus</item>
    <item>Terre</item>
    <item>Mars</item>
  </string-array>
</resources>
```

### Méthodes de la classe Spinner

`android.widget.Spinner`

#### Construction

- `Spinner(Context)` le paramètre est généralement l'activité elle-même

#### Contenu

- `setAdapter(ArrayAdapter)` Permet de remplir la liste de choix. La façon de procéder est décrite dans `ListView`.

#### Sélection

- `getCount()` renvoie le nombre de choix
- `setSelection(int)` sélectionne un élément de la liste par son rang (à partir de 0)

#### Evénements

- `setOnItemSelectedListener(AdapterView.OnItemSelectedListener)` associe un écouteur d'événements à la sélection d'une choix.

La méthode `onItemSelected(AdapterView<?>, View, int, long)` de l'interface `AdapterView.OnItemSelectedListener` est surchargée pour traiter l'événement. Le 3<sup>ème</sup> paramètre indique le rang de l'élément et le dernier son identifiant.

La méthode `onNothingSelected(AdapterView<?>)` est surchargée pour traiter l'événement de non sélection.

### DatePicker

Mêmes paramètres XML que `FrameLayout` plus :

- `android:startYear="x"` où `x` est l'année de départ du calendrier affiché
- `android:endYear="x"` où `x` est l'année de fin du calendrier affiché
- `android:minDate="x"` où `x` est la date de départ du calendrier affiché sous la forme `mm/jj/aaaa`
- `android:maxDate="x"` où `x` est la date de fin du calendrier affiché sous la forme `mm/jj/aaaa`



Exemple de fichier XML :

```
<DatePicker android:id="@+id/date"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:startYear="2000"
  android:endYear="2020"
/>
```

## Méthodes de la classe DatePicker

android.widget.DatePicker

### Construction

- `DatePicker(Context)` le paramètre est généralement l'activité elle-même

### Contenu

- `getDayOfMonth()` renvoie le jour affiché (1 à 31)
- `getMonth()` renvoie le mois affiché (1 à 12)
- `getYear()` renvoie l'année affichée
- `updateDate(int, int, int)` met à jour la date affichée, les paramètres sont dans l'ordre : année, mois, jour
- `isEnabled()` indique si le DatePicker est actif ou pas
- `setEnabled(boolean)` rend le DatePicker actif ou pas

### Initialisation et prise en compte des événements

- `init(int, int, int, DatePicker.OnDateChangedListener)` initialise l'année (1er paramètre), le mois (2ème paramètre), le jour (3ème paramètre) et définit un écouteur d'événements lorsque une date est choisie (dernier paramètre). La méthode `onDateChanged(DatePicker, int, int, int)` de l'interface `DatePicker.OnDateChangedListener` doit être surchargée pour traiter l'événement. Ses paramètres sont : le DatePicker lui-même, l'année choisie, le mois choisi et le jour choisi.

## TimePicker

Mêmes paramètres XML que `FrameLayout`

Exemple de fichier XML :

```
<TimePicker android:id="@+id/temps"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
/>
```



## Méthodes de la classe TimePicker

android.widget.TimePicker

### Construction

- `TimePicker(Context)` le paramètre est généralement l'activité elle-même

### Contenu

- `getCurrentHour()` renvoie l'heure affichée
- `getCurrentMinute()` renvoie les minutes affichées
- `setCurrentHour(Integer)` définit l'heure affichée
- `setCurrentMinute()` définit les minutes affichées
- `is24HourView()` indique si l'affichage est fait sur 12 ou 24 heures
- `setIs24HourView(Boolean)` définit le mode d'affichage (true ⇒ en 24 heures, false ⇒ en 12 heures)
- `isEnabled()` indique si le TimePicker est actif ou pas
- `setEnabled(boolean)` rend le TimePicker actif ou pas

### Evénements

- `setOnTimeChangeListener(TimePicker.OnTimeChangeListener)` définit un écouteur d'événements lorsqu'une heure est choisie. La méthode `onTimeChanged(TimePicker,`

int, int) de l'interface `TimePicker.OnTimeChangeListener` doit être surchargée pour traiter l'événement. Ses paramètres sont : le `TimePicker` lui-même, l'heure choisie et la minute choisie.

## ProgressBar

Deux comportements possibles selon que l'on connaît le taux de progression ou pas. Dans le premier cas on peut faire évoluer la progression, dans le second on a une animation.

Deux formes sont possibles (barre verticale ou cercle).



## Propriétés supplémentaires positionnables en XML

### Mode

- `android:indeterminate="b"` (où `b` vaut `true` ou `false`) définit le type de `ProgressBar` (`true` = taux indéterminé, `false` = taux déterminé).

### Valeurs si on a choisi le mode déterminé

- `android:max="x"` (où `x` est une valeur entière) définit la valeur maximale
- `android:progress="x"` (où `x` est une valeur entière) définit la valeur initiale
- `android:secondaryProgress="x"` (où `x` est une valeur entière) définit une valeur secondaire (par exemple celle d'un buffer comme on le voit sur des vidéos en streaming)

### Animation si on a choisi le mode indéterminé

- `android:indeterminateBehavior="i"` (où `i` peut être : `repeat` ou `cycle`) définit le comportement de l'animation pour le type indéterminé (`repeat`=recommence l'animation, `cycle`=changer le sens de l'animation)
- `android:indeterminateDuration="x"` (où `x` est une valeur entière) définit la durée de l'animation (en ms)
- `android:animationResolution="x"` (où `x` est une valeur entière) définit le rythme de l'animation (en ms)

### Aspect

- `style="?android:attr/s"` où `s` peut être :
  - o `progressBarStyleHorizontal` une barre horizontale (cas déterminé)
  - o `progressBarStyleSmall` un cercle de petite taille (cas indéterminé)
  - o `progressBarStyleLarge` un cercle de grande taille (cas indéterminé)

Exemple de fichier XML :

```
<ProgressBar android:id="@+id/progres"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:minWidth="300px"
  android:minHeight="30px"
  android:max="100"
  android:progress="30"
  android:secondaryProgress="40"
  android:indeterminate="false"
  style="?android:attr/progressBarStyleHorizontal"
/>
```

## Méthodes de la classe ProgressBar

`android.widget.ProgressBar`

### Construction

- `ProgressBar(Context)` le paramètre est généralement l'activité elle-même

## Valeurs

- `getMax()` renvoie la valeur maximale
- `setMax(int)` définit valeur maximale
- `getProgress()` renvoie la valeur actuelle
- `setProgress(int)` définit la valeur actuelle
- `getSecondaryProgress()` renvoie la valeur secondaire actuelle
- `setSecondaryProgress(int)` définit la valeur secondaire actuelle
- `incrementProgressBy(int)` modifie le pas de progression par ajout du paramètre
- `incrementSecondaryProgressBy(int)` modifie le pas de progression de la valeurs secondaire par ajout du paramètre

## Aspect

- `isIndeterminate()` indique si le `ProgressBar` est en mode indéterminé (`true`) ou pas
- `setIndeterminate(boolean)` met le `ProgressBar` est en mode indéterminé (`true`) ou pas
- `setVisibility(int)` définit la visibilité. Le paramètre peut être : `VISIBLE`, `INVISIBLE`, ou `GONE`.

## SeekBar

C'est un `ProgressBar` sous forme de barre horizontale doté d'un curseur permettant de modifier la valeur.



Mêmes paramètres XML que `ProgressBar` mais l'aspect ne peut pas être modifié.

## Méthodes de la classe SeekBar

`android.widget.SeekBar`

## Construction

- `SeekBar(Context)` le paramètre est généralement l'activité elle-même

On trouve les mêmes méthodes que dans la classe `ProgressBar` puisque `SeekBar` en hérite mais on peut gérer les événements de modification de la valeur :

## Evénements

- `setOnSeekBarChangeListener(SeekBar.OnSeekBarChangeListener)` permet d'associer un écouteur d'événements aux modifications de la valeur. Les méthodes que l'on surcharge dans l'interface `SeekBar.OnSeekBarChangeListener` sont :
  - o `onProgressChanged(SeekBar, int, boolean)` le curseur a été déplacé. Le 2ème paramètre est la nouvelle valeur, le dernier paramètre vaut `true` si la valeur a été modifiée par l'utilisateur.
  - o `onStartTrackingTouch(SeekBar)` début du déplacement
  - o `onStopTrackingTouch(SeekBar)` fin du déplacement

Remarque : pour toutes ces méthodes le 1<sup>er</sup> paramètre est le `SeekBar` lui-même

## RatingBar

Mêmes paramètres XML que `ProgressBar` plus :



- `android:isIndicator="b"` (où `b` vaut `true` ou `false`) indique si l'utilisateur peut modifier la valeur ou pas (`true`= non modifiable)
- `android:numStars="x"` (où `x` est une valeur entière) définit le nombre d'étoiles affichées
- `android:rating="x"` (où `x` est une valeur réelle) définit la position initiale
- `android:stepSize="x"` (où `x` est une valeur réelle) définit le pas de progression (on peut cocher des 1/2 étoiles par exemple)

## Méthodes de la classe RatingBar

android.widget.RatingBar

### Construction

- RatingBar (Context) le paramètre est généralement l'activité elle-même

### Valeurs

- getNumStars () renvoie le nombre d'étoiles
- setNumStars (int) définit le nombre d'étoiles
- getRating () renvoie la valeur actuelle (réel)
- setRating (float) définit la valeur actuelle
- getStepSize () renvoie le pas de progression (réel)
- setStepSize (float) définit le pas de progression
- isIndicator () indique si l'utilisateur peut modifier la valeur ou pas (true= non modifiable)
- setIsIndicator (boolean) définit si l'utilisateur peut modifier la valeur ou pas (true= non modifiable)

### Evénements

- setOnRatingBarChangeListener (RatingBar.OnRatingBarChangeListener) associe un écouteur d'événements aux modifications. L'interface RatingBar.OnRatingBarChangeListener possède une méthode onRatingChanged (RatingBar, float, boolean) qui doit être surchargée pour traiter les événements. Le 1<sup>er</sup> paramètre est la barre, le 2<sup>ème</sup> est la valeur actuelle et le 3<sup>ème</sup> vaut true si la valeur a été modifiée par l'utilisateur (false si c'est par programme).

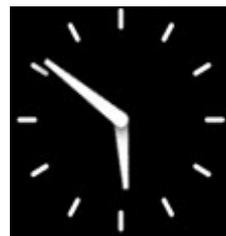
Exemple de fichier XML :

```
<RatingBar android:id="@+id/vote"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:isIndicator="false"
  android:numStars="5"
  android:stepSize="0.1"
  android:rating="3.5"
/>
```

## AnalogClock

Exemple de fichier XML :

```
<AnalogClock android:id="@+id/horloge"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
/>
```



## Méthodes de la classe AnalogClock

android.widget.AnalogClock

### Construction

- AnalogClock (Context) le paramètre est généralement l'activité elle-même

## DigitalClock

Mêmes paramètres XML que TextView

A rectangular box with a thin black border containing the text "9:26:00 pm" in a black, sans-serif font.

Exemple de fichier XML :

```
<DigitalClock android:id="@+id/digitalclock"
  android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"  
/>
```

## Méthodes de la classe **DigitalClock**

android.widget.DigitalClock

### **Construction**

- `DigitalClock(Context)` le paramètre est généralement l'activité elle-même

## **Chronometer**

C'est un compteur qui démarre à la valeur donnée par `setBase` et compte en incrémentant. Son apparence est celle d'un texte dans lequel apparaît le temps (voir format ci-dessous).

Mêmes paramètres XML que `TextView` plus :

- `android:format="f"` (où `f` est une chaîne dans laquelle la première occurrence de `%s` sera remplacée par la valeur du chronomètre sous la forme `MM:SS` ou `H:MM:SS`)

## Méthodes de la classe **Chronometer**

android.widget.Chronometer

### **Construction**

- `Chronometer(Context)` le paramètre est généralement l'activité elle-même

### **Utilisation**

- `getBase()` renvoie le point de départ du chronomètre (long)
- `setBase(long)` définit le point de départ du chronomètre
- `start()` lance le chronomètre
- `stop()` arrête le chronomètre

### **Aspect**

- `setFormat(String)` le paramètre est une chaîne dans laquelle la première occurrence de `%s` sera remplacée par la valeur du chronomètre sous la forme `MM:SS` ou `H:MM:SS`
- `getFormat()` renvoie le format utilisé (chaîne)

### **Evénements**

- `setOnChronometerTickListener(Chronometer.OnChronometerTickListener)` définit un écouteur d'événements pour chaque incrémentation du chronomètre. L'interface `Chronometer.OnChronometerTickListener` possède une méthode `onChronometerTick(Chronometer)` qui doit être surchargée pour traiter l'événement. Cette méthode reçoit le chronomètre lui-même en paramètre.

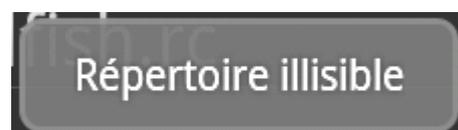
## **Notifications**

Android propose un mécanisme simple pour envoyer des informations visuelles à l'utilisateur : la classe `Toast`. Il s'agit d'une fenêtre qui apparaît en premier plan puis disparaît seule au bout d'un temps donné (2 ou 5 secondes).

## **La classe Toast**

### **Méthodes de la classe Toast**

android.widget.Toast



## Création d'un Toast

- `Toast.makeText(Context, String, int)` cette méthode renvoie l'objet de classe `Toast` créé. Le premier paramètre est généralement l'activité elle-même, le deuxième paramètre est le message à afficher, le dernier paramètre indique la durée d'affichage les seules valeurs possibles sont : `Toast.LENGTH_SHORT` (2 secondes) ou `Toast.LENGTH_LONG` (5 secondes).

## Positionnement d'un Toast

- `setGravity(int, int, int)` cette méthode doit être appelée avant l'affichage par `show` pour indiquer où s'affichera le message. Le premier paramètre sert à placer le message par rapport à l'écran. Il peut prendre l'une des valeurs définies dans la classe `Gravity` soit : `Gravity.TOP`, `Gravity.BOTTOM`, `Gravity.LEFT`, `Gravity.RIGHT`, `Gravity.CENTER_VERTICAL`, `Gravity.FILL_VERTICAL`, `Gravity.CENTER_HORIZONTAL`, `Gravity.FILL_HORIZONTAL`, `Gravity.CENTER`, `Gravity.FILL`). Les deux paramètres suivants indiquent le décalage horizontal et vertical (en pixels).

## Affichage d'un Toast

- `show()` cette méthode affiche le message pour la durée définie lors de sa création.

## Textes Formatés

Les widgets qui affichent du texte (`TextView`, `Button`, `CheckBox` ...) acceptent, pour définir ce texte, des objets de classe `CharSequence`. On peut utiliser la classe `String` qui hérite de `CharSequence` mais elle ne permet pas de formater le texte. Le format (taille, police, ..) défini dans le fichier XML s'applique donc à tout le texte. La classe `SpannableStringBuilder` qui hérite aussi de `CharSequence` (et peut donc être utilisée au lieu de `String`) permet d'apporter des formats à des parties de texte.

## Création d'un texte formaté :

```
SpannableStringBuilder texte = new SpannableStringBuilder("chaîne à formater");
```

## Application de formats :

Cela se fait par la méthode `setSpan` de `SpannableStringBuilder` :

```
texte.setSpan(CharacterStyle, int, int, Spannable.SPAN_EXCLUSIVE_EXCLUSIVE);
```

- Le premier paramètre permet de définir :

- La couleur du texte en y mettant : `new ForegroundColorSpan(int)` qui reçoit en paramètre un code de couleur
- La couleur du fond en y mettant : `new BackgroundColorSpan(int)` qui reçoit en paramètre un code de couleur
- La police en y mettant : `new TypeSpaceSpan(String)` qui reçoit en paramètre le nom de police qui peut être : `normal`, `sans`, `serif` ou `monospace`
- La taille en y mettant l'un des objets suivants :
  - `new AbsoluteSizeSpan(int, boolean)` qui reçoit en paramètre la taille en pixels (px) si le 2<sup>ème</sup> paramètre est `false` ou est omis et en pixels à densité indépendante (dp) sinon
  - `new RelativeSizeSpan(float)` qui reçoit en paramètre l'échelle de taille (1.5F signifie 1 fois et demi plus gros)
- L'étirement horizontal en y mettant : `new ScaleXSpan(float)` qui reçoit en paramètre l'échelle de taille (1.5F signifie étiré 1 fois et demi en largeur)
- Le style du texte en y mettant : `new StyleSpan(Typeface)` qui reçoit en paramètre le style qui est l'une des constantes :
  - `android.graphics.Typeface.BOLD`
  - `android.graphics.Typeface.BOLD_ITALIC`
  - `android.graphics.Typeface.ITALIC`
  - `android.graphics.Typeface.NORMAL`
- Le mode souligné en y mettant : `new UnderlineSpan()`

- Le deuxième paramètre permet de définir le point (rang du caractère) à partir duquel commence à s'appliquer ce format
- Le troisième paramètre permet de définir le point (rang du caractère) à partir duquel cesse de s'appliquer ce format

Bien entendu plusieurs formats peuvent s'appliquer sur la même partie de texte (par exemple : gras + souligné+ rouge sur fond vert)

## Couleurs et Images

Les couleurs sont représentées par un entier sur 32 bits (8 bits de transparence (FF=opaque, 0=totalement transparent), 8 bits rouge, 8 bits vert et 8 bits bleu). On peut manipuler ces codes à l'aide de la classe `Color`

### La classe `Color` d'Android

```
import android.graphics.Color
```

### Méthodes de la classe `Color`

- `argb(int, int, int, int)` renvoie le code de la couleur définie par les 4 paramètres (transparence, rouge, vert, bleu). Le 1<sup>er</sup> paramètre peut être omis pour obtenir une couleur opaque.
- `alpha(int)` renvoie la transparence de la couleur dont le code est passé en paramètre
- `red(int)` renvoie la composante rouge de la couleur dont le code est passé en paramètre
- `green(int)` renvoie la composante verte de la couleur dont le code est passé en paramètre
- `blue(int)` renvoie la composante bleue de la couleur dont le code est passé en paramètre

### Couleurs prédéfinies

Un certain nombre de couleurs sont prédéfinies en constantes dans la classe `Color` : `Color.BLACK`, `Color.WHITE`, `Color.LTGRAY`, `Color.GRAY`, `Color.DKGRAY`, `Color.RED`, `Color.GREEN`, `Color.BLUE`, `Color.CYAN`, `Color.MAGENTA`, `Color.YELLOW` et `Color.TRANSPARENT`

### La classe `Drawable`

```
import android.graphics.drawable.Drawable
```

Les images, quand elles sont récupérées dans les ressources, apparaissent sous la forme d'objets de la classe `Drawable`. C'est sous cette forme qu'elles peuvent être utilisées dans les divers éléments d'interface (images de fond, images affichées dans des boutons ou des `ImageView` séparateurs de liste, ...). Elles peuvent également être affichées dans un objet de classe `Canvas` par leur méthode `draw(Canvas)`.

## Les menus

```
import android.view.Menu
import android.view.MenuItem
import android.view.MenuInflater
import android.view.ContextMenu
import android.view.ContextMenuInfo
```

Android propose 2 types de menus:

- Menu général associé à l'interface d'une activité et qui apparaît lorsque l'on utilise la touche "menu" du téléphone
- Menu contextuel lié à un élément d'interface qui apparaît lorsque l'on touche longuement cet élément

### Description en XML d'un menu

Le fichier de description de menu doit être placé dans **res/menu** (ce répertoire n'est pas créé par Eclipse il faudra donc le faire).

Structure du fichier XML :

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
```

```

        <item android:id="@+id/nom_du_choix_1"
            android:icon="@drawable/image_du_choix_1"
            android:title="@string/texte_du_choix_1" />
        ...
    </item ... />
</menu>

```

A chaque élément de menu peut être associée une icône et un texte. On peut, bien entendu ne mettre que l'un des deux.

La création de sous menus dans un item se fait comme suit :

```

<item android:id="@+id/nom_du_choix_N"
    android:icon="@drawable/image_du_choix_N"
    android:title="@string/texte_du_choix_N">
    <menu>
        <item android:id="@+id/nom_du_sous_choix_1"
            android:title="texte_du_sous_choix_1" />
        ...
    </menu>
</item>

```

## ***Menu général***

### **Création du menu**

Elle se fait dans la méthode `onOptionsItemSelected(Menu)` de l'activité à partir du fichier xml de description sous la forme :

```

public boolean onOptionsItemSelected(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.nom_du_fichier_xml_du_menu, menu);
    return true;
}

```

### **Réactions aux choix**

Elle se fait dans la méthode `onOptionsItemSelected(MenuItem)` de l'activité qui est appelée lorsque l'utilisateur fait un choix dans un menu ou un sous menu. Cette méthode doit renvoyer un booléen à true si l'événement a été traité. Sa forme est celle d'un switch où l'on traite chaque item de menu n'ayant pas de sous menu et chaque item des sous menu :

```

public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.nom_du_choix_1:
            // traitement du choix 1
            return true;
        ...
        case R.id.nom_du_sous_choix_1:
            // traitement du sous choix 1
            return true;
        ...
        default:
            return super.onOptionsItemSelected(item);
    }
}

```

## ***Menus contextuels***

### **Association**

Le menu contextuel doit être associé à un élément d'interface. Ceci se fait lors de la création de l'interface (dans `onCreate`) de la façon suivante :

```

registerForContextMenu(element_associe_au_menu_contextuel);

```

## Création du menu contextuel

La création se fait dans la méthode `onCreateContextMenu` sous la forme :

```
public void onCreateContextMenu(ContextMenu menu, View element,
                               ContextMenuInfo info) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.nom_du_fichier_xml_du_menu, menu);
}
```

## Réactions aux choix

Elle se fait dans la méthode `onContextItemSelected(MenuItem)` de l'activité qui est appelée lorsque l'utilisateur fait un choix dans un menu contextuel. Cette méthode doit renvoyer un booléen à `true` si l'événement a été traité. Sa forme est celle d'un `switch` où l'on traite chaque item de menu n'ayant pas de sous menu et chaque item des sous menu :

```
public boolean onContextItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.nom_du_choix_1:
            // traitement du choix 1
            return true;
        ...
        case R.id.nom_du_sous_choix_1:
            // traitement du sous choix 1
            return true;
        ...
        default: return super.onContextItemSelected(item);
    }
}
```

## Communication entre activités

```
import android.os.Bundle
import android.content.Intent
import android.net.Uri
```

Une activité peut en lancer d'autres, leur passer des paramètres et en récupérer des valeurs de retour.

## Filtres d'intensions

Ils permettent à Android de savoir si une activité correspond à un type d'appel. On peut associer de tels filtres à toutes les activités d'une application.

## Forme générale

```
<activity android:name=".Nom_De_La_Classe_De_L_Activité"
...
>
    <intent-filter>
        <action android:name="nom_d_action_1" />
        ...
        <action android:name="nom_d_action_N" />
        <category android:name="nom_de_categorie_1" />
        ...
        <category android:name="nom_de_categorie_N" />
        <data
            android:mimeType="nom_de_type_mime"
            android:scheme="protocole://hote:port/chemin" />
        />
    </intent-filter>
    ...
</intent-filter>
```

```

...
    </intent-filter>
</activity>

```

## Lancer une activité

### Sans attente de réponse

Dans le cas où l'activité lancée est connue par le nom de sa classe :

```

Intent demarre = new Intent(this, NomDeLaClasseDeLActiviteALancer.class);
startActivity(demarre);

```

Dans le cas où on laisse Android choisir l'activité à lancer grâce aux filtres d'intensions :

```

Uri chemin = Uri.parse("http://www.google.fr");
Intent naviguer = new Intent(Intent.ACTION_VIEW, chemin);
startActivity(naviguer);

```

Remarque : on peut aussi créer un `Intent` sans préciser d'URI en omettant le second paramètre dans le constructeur de l'`Intent`.

Pour ajouter des informations à un `Intent` il existe les méthodes :

- `addCategory(String)` ajoute une catégorie
- `setData(URI)` définit une URI pour les données
- `setDataAndType(URI, String)` définit une URI et un type mime pour les données

### Avec attente de réponse

Le processus de lancement est le même que précédemment la seule différence porte sur la méthode utilisée pour le lancement : `startActivity` est remplacé par `startActivityForResult`.

Lancement de l'activité B dans l'activité A :

```

private static final int MON_CODE = 1; // code servant à identifier l'activité qui répond
Intent demarre = new Intent(this, NomDeLaClasseDeLActiviteB.class);
// ajouter les paramètres passés à B dans l'Intent demarre (voir plus loin)
startActivityForResult(demarre, MON_CODE); // lancement de l'activité B

```

Renvoi de réponse et terminaison de l'activité B :

```

Intent intent_retour = new Intent(); // Préparer un Intent pour les valeurs de retour
// Ajouter les valeurs de retour à l'Intent intent_retour (voir plus loin)
setResult(code, intent_retour); // renvoyer un code de retour (entier : on peut utiliser les constantes
RESULT_CANCELED et RESULT_OK) et l'Intent de retour
finish(); // terminer l'activité B

```

Récupération de la réponse de l'activité B dans l'activité A :

```

protected void onActivityResult(int ident, int code_retour, Intent retour) {
    switch(ident) {
        case MON_CODE : // c'est l'activité B qui a répondu
            // récupération des valeurs de retour contenues dans l'Intent retour (voir ci-dessous)
            // traitement selon le code de retour contenu dans l'entier code_retour
            return;
        ...
    }
}

```

Les valeurs de retour sont récupérées dans le dernier paramètre, selon leur type, par :

```

boolean x = retour.getBooleanExtra("nom_de_la_valeur_de_retour_x")
int y = retour.getIntExtra("nom_de_la_valeur_de_retour_y")
...
boolean[] tab = retour.getBooleanArrayExtra("nom_de_la_valeur_de_retour_tab")

```

...

ATTENTION : tester le cas où retour serait **null** pour éviter le plantage si on quitte l'activité par le bouton de retour en arrière du téléphone plutôt que normalement.

## ***Passage de paramètres et de valeurs de retour***

Le passage des paramètres de A à B ainsi que des valeurs de retour de B à A se font par un `Intent` en utilisant sa méthode :

```
putExtra(String, val)
```

Le 1<sup>er</sup> paramètre est un nom (clé) qui permet d'identifier la valeur associée

Le second paramètre est la valeur :

- De type simple (boolean, byte, int, short, long, float, double, char)
- Tableau de types simples
- String
- Tableau de Strings

La récupération des paramètres dans B se fait en récupérant le `Bundle` qui les contient dans l'`Intent` :

```
Bundle params = getIntent().getExtras()
```

puis en utilisant les méthodes `getBoolean`, `getInt`, ..., `getBooleanArray`, `getIntArray` ... de cet objet selon le type du paramètre à récupérer :

```
boolean x = params.getBoolean("nom_du_paramètre_x")
```

```
int taille = params.getInt("nom_du_paramètre_taille")
```

```
...
```

```
boolean[] tab = params.getBooleanArray("nom_du_paramètre_tab")
```

```
...
```

## **Le matériel et les capteurs**

### ***Envoi d'un SMS***

```
import android.telephony.SmsManager
```

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

On utilise la classe `SmsManager` dont une instance est obtenue par :

```
SmsManager gestionnaireSms = SmsManager.getDefault();
```

Pour envoyer un message on fait :

```
gestionnaireSms.sendTextMessage("numero", null, "message", null, null);
```

### ***Utilisation du GPS***

```
import android.location.LocationManager
```

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

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

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

On utilise la classe `LocationManager` dont une instance est obtenue par :

```
LocationManager geoloc =  
(LocationManager) getSystemService(Context.LOCATION_SERVICE);
```

On lui associe un écouteur d'événements qui recevra les informations de géolocalisation par la méthode :

```
geoloc.requestLocationUpdates(String, long, float, LocationListener)
```

- Le premier paramètre indique le moyen utilisé pour la géolocalisation, il peut en particulier prendre les valeurs :
  - `LocationManager.GPS_PROVIDER` pour utiliser le GPS
  - `LocationManager.NETWORK_PROVIDER` pour utiliser la triangulation des antennes réseau

- Le deuxième paramètre est un entier qui indique l'intervalle minimal de temps (en ms) entre les notifications. On peut le mettre à 0 pour obtenir l'intervalle minimal possible
- Le troisième paramètre est un réel qui indique la distance minimale (en mètres) de déplacement pour provoquer une nouvelle notification. On peut le mettre à 0 pour obtenir l'intervalle minimal possible.
- Le dernier paramètre est l'écouteur d'événements qui est un objet d'une classe à définir par héritage de `LocationListener` et par surcharge des méthodes :
  - o `public void onLocationChanged(Location)` qui est exécutée à chaque notification
  - o `public void onProviderDisabled(String)` qui est exécutée si le fournisseur de localisation est désactivé par l'utilisateur. Le paramètre est le nom de ce fournisseur.
  - o `public void onProviderEnabled(String)` qui est exécutée si le fournisseur de localisation est activé par l'utilisateur. Le paramètre est le nom de ce fournisseur.
  - o `public void onStatusChanged(String, int, Bundle)` qui est exécutée si l'état du fournisseur de localisation change (redevient actif après une interruption ...). Le premier paramètre est le nom de ce fournisseur, le deuxième indique son nouvel état (`OUT_OF_SERVICE`, `AVAILABLE` ou `TEMPORARILY_UNAVAILABLE`), le dernier paramètre contient des informations supplémentaires sur ce fournisseur.

C'est surtout la première méthode qui est utile (les autres peuvent être écrites vides). Le paramètre de cette méthode est un objet de classe `Localisation` qui contient la position. Ses principales méthodes sont :

- o `getLatitude()` qui renvoie un réel correspondant à la latitude
- o `getLongitude()` qui renvoie un réel correspondant à la longitude
- o `getAltitude()` qui renvoie un réel correspondant à l'altitude (en mètres)
- o `getAccuracy()` qui renvoie un réel correspondant à la précision de la localisation (en mètres)
- o `distanceTo(Location)` qui renvoie un réel représentant en mètres la distance entre la position actuelle et celle passée en paramètre.

## Appareil photo

```
import android.hardware.Camera
import android.view.SurfaceView
import android.view.SurfaceHolder
import android.view.SurfaceHolder.Callback
<uses-permission android:name="android.permission.CAMERA" />
```

## Accès à l'appareil photo

On déclare l'appareil photo comme une propriété de l'activité par :

```
private Camera photo;
```

## Prévisualisation

L'appareil photo ne peut fonctionner que s'il dispose d'une zone d'écran pour prévisualiser la photo qu'il va prendre. On va donc lui associer un élément d'interface de classe `SurfaceView` qui sera déclaré dans le fichier xml de description de l'interface par :

```
<SurfaceView
    android:name="@id/visualisation"
    ...
/>
```

Auquel on associera un écouteur d'événements pour gérer la prévisualisation par :

```
SurfaceView ecran = (SurfaceView)findViewById(R.id.visualisation);
SurfaceHolder surface = ecran.getHolder();
surface.addCallback(new GestionPrevisualisation());
surface.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
```

Et on créera cet écouteur d'événements par :

```
private class GestionPrevisualisation implements
```

```

        SurfaceHolder.Callback {
public void surfaceCreated(SurfaceHolder surf) {
    photo = Camera.open();
    try {
        photo.setPreviewDisplay(surf);
    }
    catch (IOException ioe) {
        // erreur lors de l'association de la prévisualisation
        photo.stopPreview();
        photo.release();
    }
}

public void surfaceChanged(SurfaceHolder surf, int format,
                           int largeur, int hauteur) {

}

public void surfaceDestroyed(SurfaceHolder holder) {
    if (photo != null) {
        photo.stopPreview();
        photo.release();
    }
}
}

```

## Prise de photo

A chaque prise de photo on doit lancer la prévisualisation et faire la photo en associant un écouteur d'événement qui récupèrera l'image par :

```

photo.startPreview();
photo.takePicture(Camera.ShutterCallback, Camera.PictureCallback,
                 Camera.PictureCallback);

```

Les paramètres de la méthode `takePicture` sont les suivants :

- Les deux premiers permettent d'associer des écouteurs d'événements pour l'instant de prise de vue et pour la capture d'image non compressée (raw). Si l'on souhaite seulement récupérer la photo en jpeg on met null dans ces paramètres.
- Le dernier est un objet d'une classe qui implémente l'interface `PictureCallback` et dont on surcharge la méthode `onPictureTaken` qui sera exécutée lorsque la photo est disponible en jpeg. Cette méthode reçoit deux paramètres :
  - o un tableau d'octets (`byte[]`) contenant l'image au format jpeg
  - o l'appareil photo lui-même (objet de classe `Camera`)

## Affichage de photo

```

import android.graphics.drawable.BitmapDrawable
import android.graphics.BitmapFactory
import android.graphics.BitmapFactory.Options

```

On peut transformer le tableau d'octets reçu en paramètre de `onPictureTaken` en objet de classe `BitmapDrawable` qui hérite de `Drawable` par le code suivant :

```

BitmapFactory.Options options = new BitmapFactory.Options();
options.inPurgeable = true; // pour libérer la mémoire prise par l'image
options.inSampleSize = x; // pour réduire l'image dans un rapport 1/x
BitmapDrawable image = new BitmapDrawable(
    BitmapFactory.decodeByteArray(photo, 0, photo.length, options));

```

## Vibreur

```
import android.os.Vibrator
<uses-permission android:name="android.permission.VIBRATE" />
```

Pour accéder au vibreur on utilise la classe `Vibrator` dont une instance est obtenue par :

```
Vibrator vibreur = (Vibrator) getSystemService(Context.VIBRATOR_SERVICE);
```

Puis on peut faire vibrer le téléphone pour une durée donnée par la méthode :

```
vibreur.vibrate(int) où le paramètre est la durée de vibration en ms.
```

## Capteurs

```
import android.hardware.Sensor
import android.hardware.SensorManager
import android.hardware.SensorEvent
```

Les types de capteurs disponibles sont les suivants (selon le modèle certains capteurs peuvent ne pas être disponibles) :

- `Sensor.TYPE_ACCELEROMETER`
- `Sensor.TYPE_GRAVITY`
- `Sensor.TYPE_GYROSCOPE`
- `Sensor.TYPE_LIGHT`
- `Sensor.TYPE_MAGNETIC_FIELD`
- `Sensor.TYPE_ORIENTATION`
- `Sensor.TYPE_PRESSURE`
- `Sensor.TYPE_PROXIMITY`
- `Sensor.TYPE_TEMPERATURE`

Pour accéder à un capteur on utilise la classe `SensorManager` dont une instance est obtenue par :

```
SensorManager gestionnaireCapteurs =
(SensorManager) getSystemService(Context.SENSOR_SERVICE);
```

Puis on récupère un capteur particulier par :

```
Sensor monCapteur = gestionnaireCapteurs.getDefaultSensor(type_de_capteur);
```

Le paramètre `type_de_capteur` est l'un des types indiqués ci-dessus.

On lui associe un écouteur d'événements par la méthode :

```
gestionnaireCapteurs.registerListener(SensorEventListener, Sensor, int)
```

- Le premier paramètre est l'écouteur d'événements qui est un objet d'une classe à définir par implémentation de l'interface `SensorEventListener` et par surcharge des méthodes :
  - o `void onSensorChanged(SensorEvent)` qui est exécutée chaque fois que le capteur effectue une nouvelle mesure. Le paramètre permet de récupérer les valeurs par `values.clone()` qui renvoie un tableau de réels contenant les mesures (voir ci-dessous).
  - o `void onAccuracyChanged(Sensor, int)` qui est exécutée si la précision du capteur change. Le premier paramètre est le capteur dont la précision a changé, le second indique la nouvelle précision.
- Le deuxième paramètre est le capteur auquel on associe l'écouteur d'événements.
- Le dernier paramètre est le rythme de mesure (en microsecondes) ou l'une des valeurs prédéfinies suivantes : `SENSOR_DELAY_NORMAL`, `SENSOR_DELAY_UI`, `SENSOR_DELAY_GAME`, ou `SENSOR_DELAY_FASTEST`. `SENSOR_DELAY_UI` correspond à la valeur la plus adéquate pour de l'interface tandis que `SENSOR_DELAY_GAME` correspond à la valeur la plus adéquate pour des jeux.

On peut supprimer un écouteur associé à un capteur par :

`gestionnaireCapteurs.unregisterListener(SensorEventListener, Sensor)` le premier paramètre est l'écouteur d'événements, le deuxième paramètre est le capteur auquel était associé cet écouteur d'événements.

Le tableau de valeurs récupéré par la méthode `values.clone()` contient 1 ou 3 valeurs qui doivent être interprétées selon le type de capteur (sa taille est toujours de 3 mais certaines valeurs ne sont pas significatives) :

Type de capteurs	valeurs	unité
SENSOR.TYPE_ACCELEROMETER	3 valeurs : accélération en X, en Y et en Z	m/s <sup>2</sup>
SENSOR.TYPE_GRAVITY	3 valeurs : gravitation en X, en Y et en Z	m/s <sup>2</sup>
SENSOR.TYPE_GYROSCOPE	3 valeurs : vitesse angulaire en X, en Y et en Z	radians/s
SENSOR.TYPE_LIGHT	1 valeur : luminosité	lux
SENSOR.TYPE_MAGNETIC_FIELD	3 valeurs : champ magnétique en X, en Y et en Z	microtesla
SENSOR.TYPE_ORIENTATION	3 valeurs : <ul style="list-style-type: none"><li>- angle par rapport au nord</li><li>- rotation par rapport à l'axe X</li><li>- rotation par rapport à l'axe X</li></ul>	degré
SENSOR.TYPE_PRESSURE	1 valeur : pression atmosphérique	psi $\equiv$ 6894,76 Pa
SENSOR.TYPE_PROXIMITY <sup>(1)</sup>	1 valeur : distance capteur/objet (mesuré par IR)	cm
SENSOR.TYPE_TEMPERATURE	1 valeur : température	°C

(1) : sur certains modèles le capteur de proximité se contente de détecter une présence. La valeur mesurée n'est que 0 ou la valeur maximale (généralement 10 cm).

## Le multimédia

Android peut utiliser de nombreux formats audio et vidéo.

- Pour le son : 3gp, mp3, mp4, ogg, mid et wav
- Pour la vidéo : 3gp et mp4

### *Jouer un son*

```
import android.media.MediaPlayer
```

Le fichier contenant le son doit être placé dans le répertoire **res/raw**. Il suffit alors de créer un lecteur audio par :

```
MediaPlayer lecteur = MediaPlayer.create(Context, int)
```

1. Le premier paramètre est l'activité elle-même
2. Le second paramètre est l'identificateur du fichier son obtenu par :  
`R.raw.nom_du_fichier_son`

Les opérations possibles sont alors :

```
lecteur.start() pour jouer le son  
lecteur.pause() pour suspendre le son, il sera repris par start()  
lecteur.stop() pour arrêter le son
```

Pour rejouer un son arrêté ou terminé il faut faire les 3 opérations suivantes :

```
lecteur.reset();  
lecteur.prepare();  
lecteur.start();
```

## La classe MediaPlayer

Les principales méthodes de MediaPlayer sont :

### Configuration

- `prepare()` initialisation du player
- `release()` libère les ressources du player (à faire dans la méthode `onDestroy` de l'activité)
- `reset()` réinitialisation du player
- `setDataSource(String)` définit le média par un chemin de fichier ou une URL
- `setDataSource(Context, Uri)` définit le média par une `Uri`, le premier paramètre est l'activité, le second l'`Uri` désignant le média.
- `setLooping(boolean)` met le player en mode boucle
- `setVolume(float, float)` définit le volume (le 1<sup>er</sup> paramètre est la voie gauche, le second la voie droite)

### Contrôle

- `pause()` met en pause
- `seekTo(int)` déplacement dans le média en ms (en + ou en -)
- `start()` lancement
- `stop()` arrêt

### Etat

- `getCurrentPosition()` renvoie la position actuelle dans le média (en ms)
- `getDuration()` renvoie la durée du média (en ms)
- `isPlaying()` renvoie true si le media est en cours
- `isLoopPlaying()` renvoie true si le media est en mode boucle

### Evénements

- `setOnCompletionListener(MediaPlayer.OnCompletionListener)` associe un écouteur d'événements dont la méthode `onCompletion(MediaPlayer)` sera appelée lorsque le média se termine
- `setOnBufferingUpdateListener(MediaPlayer.OnBufferingUpdateListener)` associe un écouteur d'événements dont la méthode `onBufferingUpdate(MediaPlayer, int)` est appelée lors de la mise à jour du buffer. Le second paramètre est le pourcentage de remplissage du buffer.
- `setOnPreparedListener(MediaPlayer.OnPreparedListener)` associe un écouteur d'événements dont la méthode `onPrepared(MediaPlayer)` est appelée lorsque le `MediaPlayer` est prêt.
- `setOnSeekCompleteListener(MediaPlayer.OnSeekCompleteListener)` associe un écouteur d'événements dont la méthode `onSeekComplete(MediaPlayer)` est appelée lorsque déplacement dans le média est terminé.

## Afficher une video

```
import android.widget.VideoView
import android.net.Uri
```

Remarque : en général l'affichage de vidéos ne fonctionne pas correctement sur l'émulateur (on a le son mais l'image est figée ou saccadée) pour des raisons d'émulation d'accélération graphique.

Le fichier contenant la vidéo doit être placé dans le répertoire **res/raw** ou accessible sur une URI. Ce dernier cas sera décrit plus loin.

L'interface doit comporter une zone d'affichage pour la vidéo qui sera déclarée dans le fichier xml de définition d'interface par :

```
<VideoView android:id="@+id/ecran_video"  
android:layout_width="fill-parent"  
android:layout_height="fill-parent" />
```

Le chemin d'accès à la vidéo est décrit par une Uri :

```
Uri chemin =  
Uri.parse("android.resource://paquetage_de_l_application/"  
+R.raw.nom_du_fichier_video);
```

On va ensuite associer ce média à la zone d'affichage par :

```
VideoView lecteur = (VideoView) findViewById(R.id.ecranVideo);  
lecteur.setVideoURI(chemin);  
lecteur.requestFocus();
```

## La classe VideoView

Les principales méthodes de VideoView sont :

### Configuration

- `setMediaController(MediaController)` associe un contrôleur de média qui apparaîtra par un clic long sur la vidéo et affiche les boutons de contrôle.
- `setVideoPath(String)` définit le média par un chemin de fichier
- `setVideoURI(Uri)` définit le média par une Uri

### Contrôle

- `start()` lancement
- `pause()` mise en pause, reprise par `start()`
- `seekTo(int)` déplacement dans le média, le paramètre est un temps en ms à partir du début
- `stopPlayback()` arrêt définitif ne sera pas relancé par `start()`

### Etat

- `canPause()` renvoie true si le media peut être mis en pause
- `canSeekBackward()` renvoie true si le media peut être reculé
- `canSeekForward()` renvoie true si le media peut être avancé
- `getBufferPercentage()` renvoie le pourcentage d'occupation du buffer de média
- `getCurrentPosition()` renvoie la position actuelle dans le média (en ms)
- `getDuration()` renvoie la durée du média (en ms)
- `isPlaying()` renvoie true si le media est en cours

### Evénements

- `setOnCompletionListener(MediaPlayer.OnCompletionListener)` associe un écouteur d'événements dont la méthode `onCompletion(MediaPlayer)` sera appelée lorsque le média se termine.

## Synthèse de parole

```
import android.speech.tts.TextToSpeech  
import android.speech.tts.TextToSpeech.OnInitListener
```

Android permet de lire à haute voix un texte par synthèse de parole. Il faut, bien entendu, que le synthétiseur connaisse la langue du texte. On peut installer des applications permettant d'ajouter des langues.

Pour créer et préparer le synthétiseur :

- `parle = new TextToSpeech(this, new SynthPret());`

Pour attendre que le synthétiseur soit prêt on utilise l'écouteur d'événements qui lui a été associé à la création. Cet écouteur peut se contenter de positionner la langue quand le synthétiseur est prêt. Son code est le suivant :

```

private class SynthPret implements TextToSpeech.OnInitListener {
    public void onInit(int status) {
        if (status == TextToSpeech.SUCCESS){
            parle.setLanguage(Locale.FRANCE); // langue
        }
    }
}

```

Autres réglages :

- `setSpeechRate(float)` permet de régler la vitesse de synthèse (1 normal, <1 plus lent, >1 plus rapide)
- `setPitch(float)` permet de régler la tonalité (1 normal, <1 plus grave, >1 plus aigu)

Pour synthétiser un texte :

```

String message = new String("texte a dire");
parle.speak(message, TextToSpeech.QUEUE_FLUSH, null);

```

Pour ajouter du texte à synthétiser pendant la synthèse :

```

String ajout = new String("texte a ajouter");
parle.speak(ajout, TextToSpeech.QUEUE_ADD, null);

```

Pour arrêter la synthèse de son :

```

parle.stop(); // arrêt de la synthèse

```

Pour arrêter le synthétiseur de son :

```

parle.shutdown(); // arrêt du synthétiseur

```

On met généralement ceci dans la méthode `onDestroy` de l'activité :

```

public void onDestroy() {
    if (parle != null) {
        parle.stop();
        parle.shutdown();
    }
    super.onDestroy();
}

```

## Internet

```

import java.net.URL
import java.io.InputStreamReader
import java.io.BufferedReader
import java.io.BufferedInputStream
import java.net.HttpURLConnection
<uses-permission android:name="android.permission.INTERNET" />

```

### **Récupérer un fichier texte depuis une URL**

Ce type d'accès se fait de façon classique par un flux ouvert sur l'URL :

```

try {
    URL url = new URL("mettre ici l'URL du fichier texte");
    BufferedReader in = new BufferedReader(new
        InputStreamReader(url.openStream()));

    String str;
    while ((str = in.readLine()) != null) {
        // traiter la ligne lue (qui est dans str)
    }
    in.close();
}
catch (MalformedURLException e) {}
catch (IOException e) {
}

```

## Récupérer une image depuis une URL

On peut fabriquer un `Drawable` qui pourra être mis dans un `ImageView` ou un `ImageButton` à partir d'une image prise sur une URL par :

```
BitmapDrawable img;
URLConnection connexion = (URLConnection)new URL("mettre ici
    l'url de l'image").openConnection();
connexion.connect();
BufferedInputStream lecture = new BufferedInputStream
    (connexion.getInputStream());
BitmapFactory.Options opts = new BitmapFactory.Options();
opts.inSampleSize = x; // pour réduire la taille en 1/x
img = BitmapFactory.decodeStream(lecture, null, opts);
```

## Jouer un son depuis une URL

Le procédé est exactement le même que pour un son placé dans le répertoire `res/raw` la seule différence porte sur la construction du `MediaPlayer`. Au lieu de :

```
MediaPlayer lecteur = MediaPlayer.create(Context, int)
```

Il suffit de faire :

```
MediaPlayer mp = new MediaPlayer();
try {
    mp.setDataSource("http://domaine.sous_domaine/chemin/nom_son.mp3");
    mp.prepare();
}
catch (IllegalArgumentException e) {
    // Le paramètre de setDataSource est incorrect
}
catch (IllegalStateException e) {
    // Le MediaPlayer n'est pas dans l'état initial
}
catch (IOException e) {
    // L'accès à l'URL provoque une erreur
}
```

## Jouer une vidéo depuis une URL

Le procédé est exactement le même que pour une vidéo placée dans le répertoire `res/raw` la seule différence porte sur l'Uri utilisée. Au lieu de :

```
Uri chemin =
    Uri.parse("android.resource://paquetage_de_l_application/"
        +R.raw.nom_du_fichier_video);
```

Il suffit de faire :

```
Uri chemin = Uri.parse("http://domaine.sous_domaine
    /repl/nom_video.3gp");
```

## Utiliser des services WEB

Pour accéder aux service WEB ,l'accès à internet doit être autorisé par l'ajout dans le fichier **AndroidManifest** d'une balise :

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

## Trouver les coordonnées géographiques de lieux

Android offre une classe permettant de trouver les coordonnées d'un lieu décrit par une adresse ou une adresse à partir de ses coordonnées géographiques.

## La classe Geocoder

```
import android.location.Geocoder
import java.util.Locale
import java.util.List
import android.location.Address
```

### Construction

- `Geocoder(Context, Locale)` le premier paramètre est généralement l'activité elle-même. Le second paramètre indique la zone géographique concernée, il peut prendre les valeurs (`Locale.FRANCE`, `Locale.CANADA`, `Locale.UK`, `Locale.US` ...). Ce dernier paramètre peut être omis si l'on ne souhaite pas limiter la localisation.

### Recherches

Toutes ces méthodes renvoient une liste (classe `List` de java) contenant des objets de classe `Address` (ATTENTION : cette liste peut être vide si la recherche ne donne aucun résultat) :

- `getFromLocation(double, double, int)` renvoie les adresses connues proches du point défini par ses coordonnées géographiques passées par les 2 premier paramètres (latitude et longitude exprimées en degrés). Le dernier paramètre permet de limiter la taille de cette liste.
- `getFromLocationName(String, int)` renvoie les adresses connues proches du point défini par le nom passé en premier paramètre (c'est une chaîne du type "3 Allée du parc Montaury, 64600, Anglet"). Le second paramètre permet de limiter la taille de cette liste.
- `getFromLocationName(String, int, double, double, double, double)` cette méthode fonctionne comme la précédente mais permet de limiter la zone de recherche à un rectangle. Les 3<sup>ème</sup> et 4<sup>ème</sup> paramètres indiquent la longitude et la latitude du coin inférieur gauche de ce rectangle tandis que les 5<sup>ème</sup> et 6<sup>ème</sup> paramètres en indiquent la longitude et la latitude du coin supérieur droit.

## La classe Address

```
import android.location.Address
```

Un objet de classe `Address` correspond à une localisation géographique et contient des chaînes de caractères correspondant aux lignes d'adresse de cette localisation (voir exemple ci-dessous).

### Construction

- `Address(Locale)` le paramètre indique la zone géographique concernée, il peut prendre les valeurs (`Locale.FRANCE`, `Locale.CANADA`, `Locale.UK`, `Locale.US` ...). Ce dernier paramètre peut être omis si l'on ne souhaite pas limiter la localisation.

### Contenu

- `getLatitude()` renvoie la latitude en degrés (réel type double)
- `getLongitude()` renvoie la longitude en degrés (réel type double)
- `getFeatureName()` renvoie le nom du lieu
- `getLocality()` renvoie le nom de la ville
- `getPostalCode()` renvoie le code postal
- `getCountryName()` renvoie le nom du pays
- `getAddressLine(int)` renvoie la ligne d'adresse désignée par son index passé en paramètre (en commençant à 0). Renvoie `null` s'il n'y a rien correspondant à l'index. On peut connaître le nombre de lignes d'adresse disponibles par la méthode : `getMaxAddressLineIndex()`

### Exemple :

Le code suivant :

```
Geocoder localisateur = new Geocoder(this, Locale.FRANCE);
List<Address> liste = localisateur.getFromLocationName("Parc Montaury, Anglet", 10);
```

Renvoie une liste ne contenant qu'un seul objet de classe `Address` dont le contenu est le suivant :

- latitude : 43,4800424

- longitude : -1,5093202
- nom de lieu : Allée du Parc Montauray
- nom de ville : Anglet
- code postal : 64600
- nom de pays : France
- Deux lignes d'adresse qui sont :
  - Allée du Parc Montauray
  - 64600 Anglet

## GoogleMaps

### Préparation du projet

Pour pouvoir utiliser GoogleMaps il faut que le projet ait été créé avec comme SDK cible le SDK "Google APIs" et non le SDK "Android x.x".

Si l'on utilise un émulateur de terminal il faudra également le configurer pour qu'il fonctionne avec ce SDK.

La bibliothèque GoogleMaps doit être intégrée à l'application par l'ajout dans le fichier **AndroidManifest** d'une balise : `<uses-library android:name="com.google.android.maps" />`

L'activité doit hériter de `MapActivity` et non de `Activity`. Elle doit implémenter la méthode `isRouteDisplayed` propre à `MapActivity` qui sert au serveur pour des statistiques concernant l'affichage d'itinéraires. On peut se contenter du code suivant :

```
protected boolean isRouteDisplayed() {
    return false;
}
```

### Clé d'utilisation

L'accès à Google maps nécessite une clé que l'on obtient comme suit :

- Avoir un compte Google ou en créer un en allant sur <https://www.google.com/accounts/>
- Obtenir l'empreinte md5 du certificat de l'application. Pour ce faire :
  - Ouvrir une fenêtre DOS et se mettre dans le répertoire du compilateur java (par exemple : C:\Program Files\Java\jdk1.6.0\_07\bin)
  - Taper la commande :
 

```
keytool -list -alias androiddebugkey -keystore chemin -storepass android
```

 dans laquelle *chemin* désigne le fichier **debug.keystore** qui se trouve dans le répertoire de l'utilisateur (pour trouver ce répertoire depuis Eclipse utiliser le menu Windows → Preferences : Android : Build rubrique Android debug keystore).  
 Cette commande affiche un certificat MD5 de la forme :  
 androiddebugkey, 2 oct. 2010, PrivateKeyEntry, Empreinte du certificat (MD5) :  
 CA:78:06:9E:1A:27:20:4D:28:78:6B:F3:A5:D2:BF:78
- Obtenir la clé en allant sur la page :
 

```
http://code.google.com/intl/fr/android/maps-api-signup.html
```

 Le code précédent est saisi dans "My certificate's Fingerprint" et on clique sur le bouton "Generate API Key". Après s'être authentifié on récupère une clé qui est une chaîne de caractères (faites un copier/coller dans un fichier texte pour ne pas la perdre).

Cette clé fonctionnera pour toutes vos applications.

### Afficher des cartes dans une application

Les cartes sont affichées dans un widget `MapView`

#### La classe `MapView`

Cette classe hérite de `ViewGroup`. Elle a donc les mêmes propriétés que `ViewGroup`

Exemple de fichier XML :

```
<com.google.android.maps.MapView android:id="@+id/carte"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
```

```
    android:apiKey="mettre ici la clé obtenue"
/>
```

## **Méthodes de la classe *MapView***

```
import com.google.android.maps.MapView
```

### **Construction**

- `mapView(Context, String)` le premier paramètre est généralement l'activité elle-même, le second paramètre est la clé.

### **Apparence**

- `displayZoomControls(boolean)` affiche le contrôle de zoom. Le paramètre permet de donner le focus à ce contrôle (`true`)
- `setBuiltInZoomControls(boolean)` autorise (`true`) ou invalide (`false`) le contrôle de zoom
- `setSatellite(boolean)` affichage en mode satellite ou plan
- `setStreetView(boolean)` affichage avec ou sans visualisation des rues
- `setTraffic(boolean)` affichage avec ou sans visualisation du trafic

### **Contenu**

- `getLatitudeSpan()` renvoie l'angle de latitude visible (entre le haut et le bas de la carte affichée) en millièmes de degrés
- `getLongitudeSpan()` renvoie l'angle de longitude visible (entre la gauche et la droite de la carte affichée) en millièmes de degrés
- `getMapCenter()` renvoie un objet de classe `GeoPoint` indiquant le centre de la carte. Cet objet possède les méthodes `getLatitudeE6()` et `getLongitudeE6()` qui renvoient ses coordonnées géographiques en millièmes de degrés.
- `getMaxZoomLevel()` renvoie un entier indiquant la valeur maximale de zoom possible
- `getZoomLevel()` renvoie un entier indiquant la valeur actuelle de zoom
- `isSatellite()` renvoie un booléen indiquant si la carte est en mode satellite (`true`)
- `isStreetView()` renvoie un booléen indiquant si la carte est en mode rues (`true`)
- `isTraffic()` renvoie un booléen indiquant si la carte est en mode trafic (`true`)

### **Projection**

Lorsqu'une carte est affichée il est possible de récupérer les coordonnées géographiques d'un point par rapport à sa position à l'écran (par exemple lorsque l'utilisateur touche la carte). Ceci peut se faire par la méthode `fromPixels(int, int)` de l'objet de classe `Projection` renvoyé par la méthode `getProjection()` de l'objet de classe `MapView` affiché. Cette méthode prend en paramètres les coordonnées (x et y) d'un pixel à l'écran et renvoie un objet de classe `GeoPoint` correspondant à la position. . Cet objet possède les méthodes `getLatitudeE6()` et `getLongitudeE6()` qui renvoient ses coordonnées géographiques en millièmes de degrés.

## **La classe *MapController***

Cette classe hérite sert à piloter la carte affichée. On l'obtient par la méthode `getController()` de l'objet `MapView`.

## **Méthodes de la classe *MapController***

```
import com.google.android.maps.MapController
```

### **Déplacements**

- `animateTo(GeoPoint)` positionne la carte sur le point passé en paramètre en faisant une animation. La classe `GeoPoint` possède un constructeur acceptant en paramètres ses coordonnées géographiques en millièmes de degrés; le 1<sup>er</sup> paramètre est la latitude, le second la longitude.

- `setCenter(GeoPoint)` positionne la carte sur le point passé en paramètre sans faire d'animation. La classe `GeoPoint` possède un constructeur acceptant en paramètres ses coordonnées géographiques en millièmes de degrés
- `scrollBy(int , int)` déplace la carte. Le premier paramètre exprime le déplacement horizontal (en pixels), le second le déplacement vertical.
- `setZoom(int)` définit le niveau de zoom
- `zoomIn()` zoome d'un niveau
- `zoomOut()` dé-zoome d'un niveau