Android


Cupcake
Donut
Eclair
Froyo
Gingerbread
Honeycomb
ICE Cream-Sandwitch
Jelly Bean
Kitkat
lollipop
Lollipop
Marshmallow
Nougat

ACTIVITE

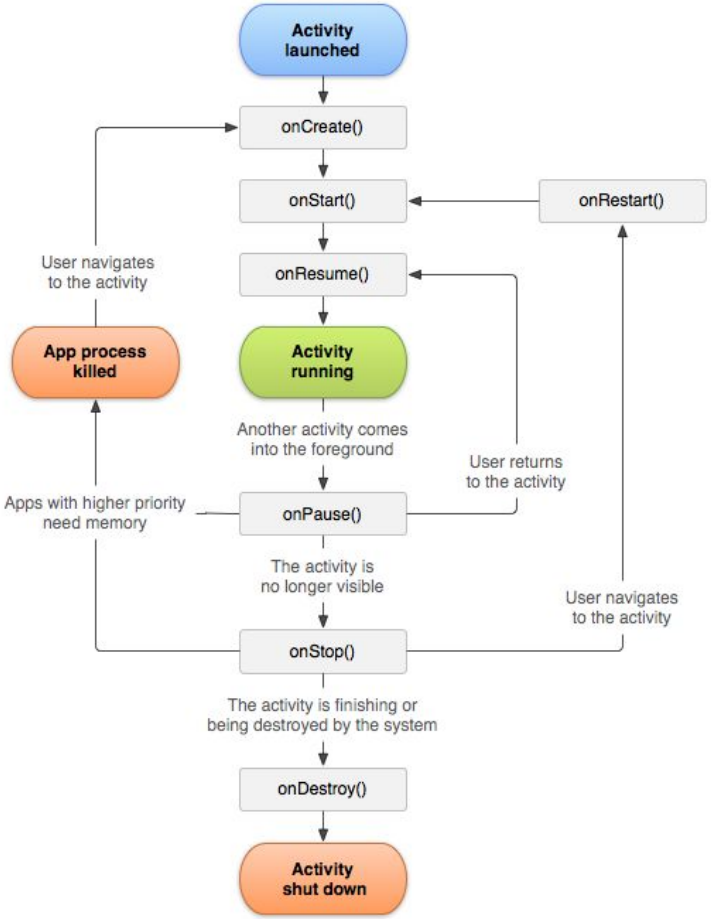
Cycle de vie d'une activité

Cycle de vie ***global***
onCreate() -> onDestroy()

Cycle de vie ***visible***
onStart() -> onStop()

Affichée à l'écran mais peut ne pas être utilisable
(en second plan)

- Cycle de vie ***en premier plan***
- onResume() -> onPause()

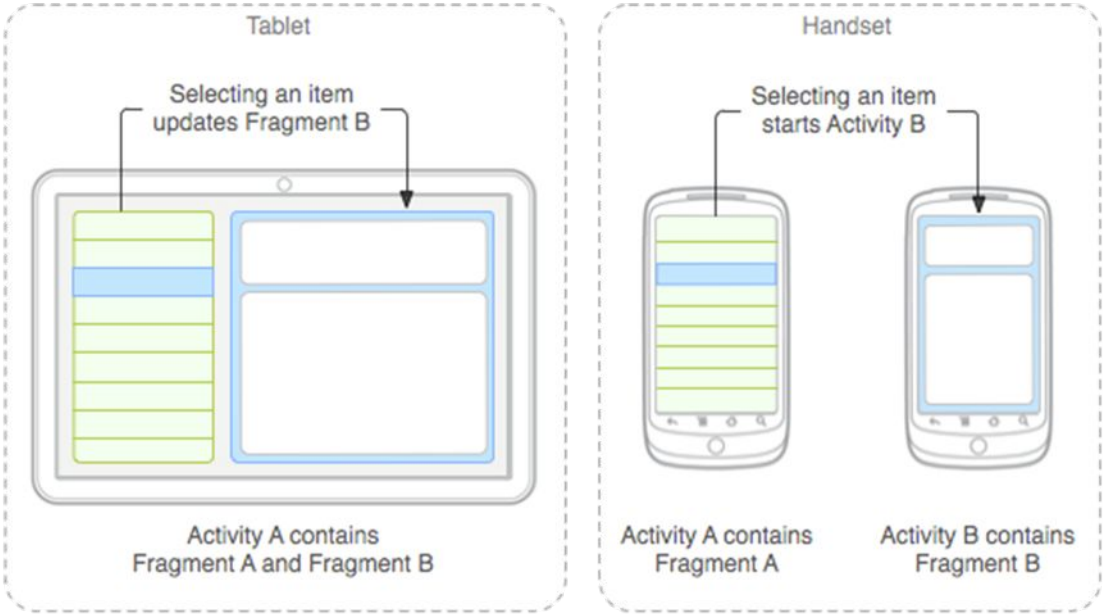


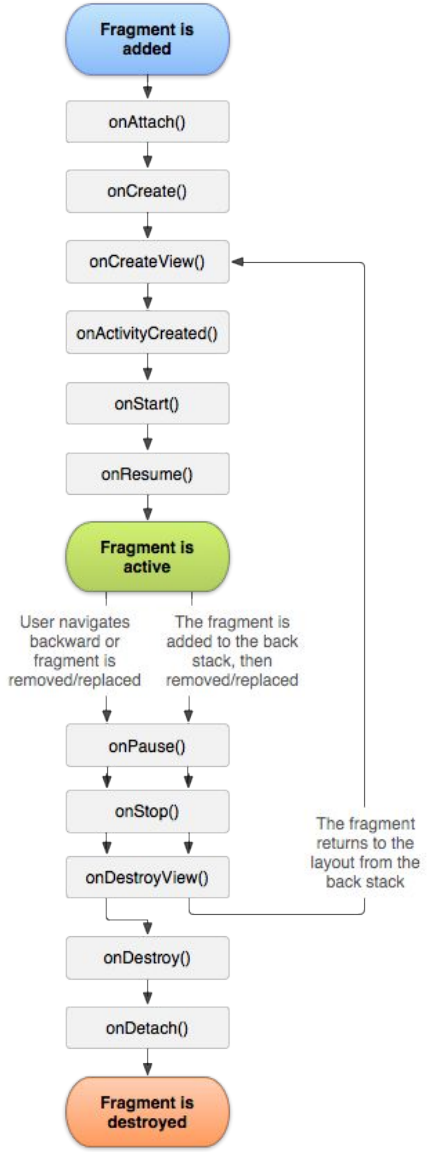
http://supertos.free.fr/supertos.php?page=1076

FRAGMENT

Un fragment pourquoi ?

En Android, les fragments sont des portions d'interface graphiques. On peut combiner plusieurs fragments dans une activité et réutiliser les fragments dans différentes activités. Ils permettent donc un meilleur découpage ainsi qu'une meilleure évolutivité du code. La combinaison de fragments permet par exemple de créer des vues différentes selon le type d'appareil (tablette ou téléphone), sans avoir à dupliquer du code.





Un fragment a :
Son propre cycle de vie,
ses propres événements en entrée

on peut rajouter ou retirer un fragment d'une activité en cours d'exécution

Le cycle de vie d'un fragment est directement impacté par le cycle de vie de l'activité dans laquelle il se trouve,

Si activité est paused/destroyed alors tous les fragments sont paused/destroyed

Pendant qu'une activité est utilisée on peut manipuler chaque fragment indépendamment (les ajouter ou les supprimer)

http://designthing.net/the-basic-components-in-an-android-app/

Anatomie d'une application

Activité ⟺ une page de site web
une vue (généralement en xml)

Service : traitements lourds

Broadcast Receiver : Intercepteur d'évènements

Content Provider : surcouche à une base de données

Intent : message envoyé au sein du système, communication inter composants
Peut contenir des données

On a vu !

L’essentiel pour l’IHM :

- Pourquoi Android
- Comment démarrer
- Les fondements Activity vs Fragment

?

Il reste à voir cette semaine :

- Layouts
- Manifest et Permissions
- Ressources
- Un plus sur les fragments

Il restera la semaine prochaine :

- Le passage de données entre composants avec les intents
- AsyncTask et appel de services en tâche de fond
- L’accès aux capteurs du dispositif
- L’accès aux Bases de données

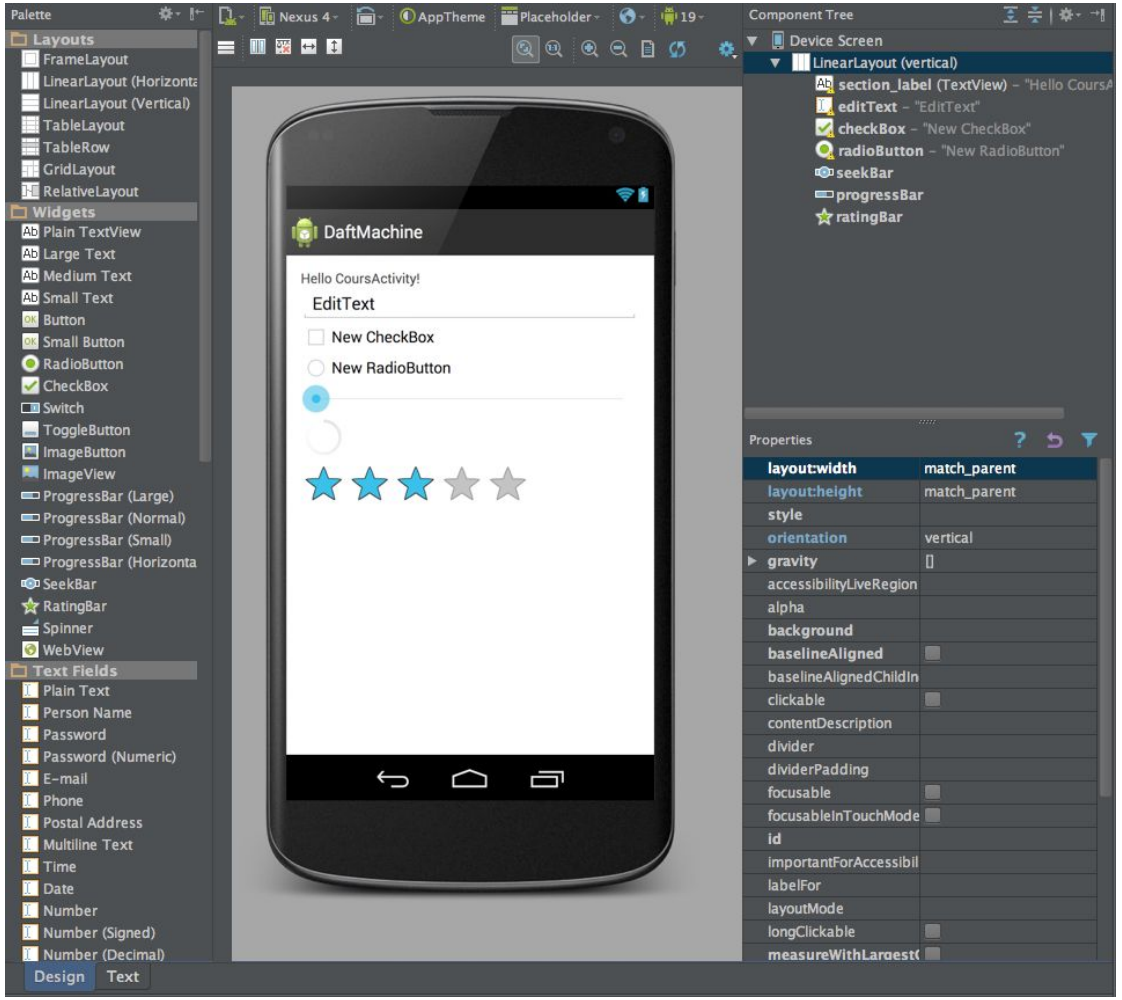
LAYOUT

Découvrez et créez vos vues !

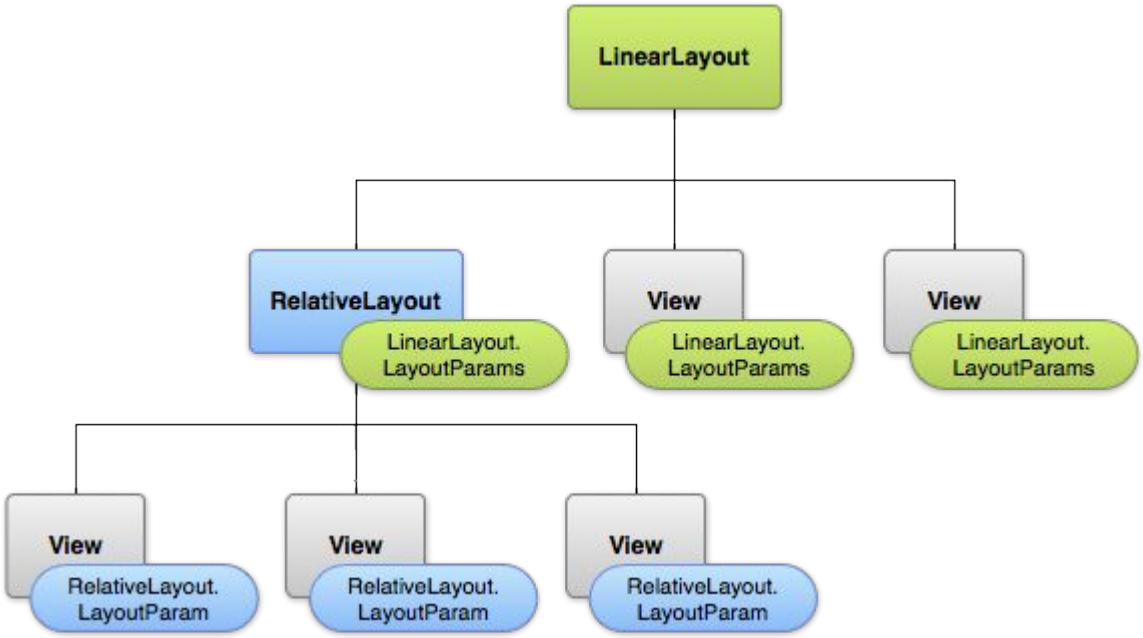
2 modes d'édition
- placement graphique (drag'n'drop)
- écriture d'XML

- Produit des fichiers XML pouvant Être associés à une Activity

- Spécifie les caractéristiques du téléphone cible
- Langue
- Orientation
- Type de téléphone
- Taille d'écran …



Organisation générale d'une vue



Définition

Groupes de vues dérivés de *ViewGroup*
fournissent un modèle de présentation de leurs vues “filles”.

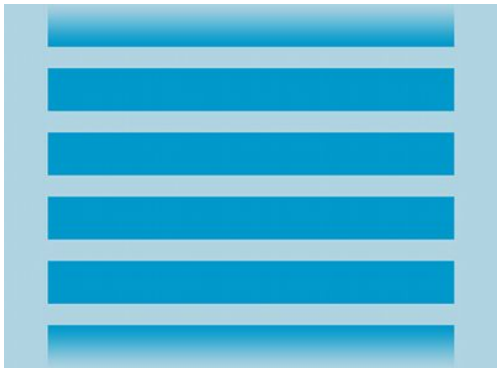
Vous pouvez aussi hériter de classes existantes pour créer votre propre layout et l’appliquer ensuite à votre activité.

Exemples de layout

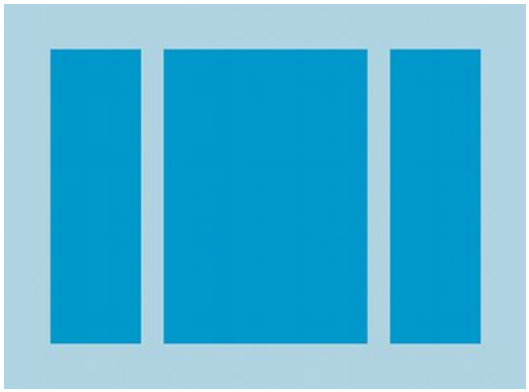
Conçus pour le Responsive design

- En lignes
- En colonnes
- En grille
- Relatif

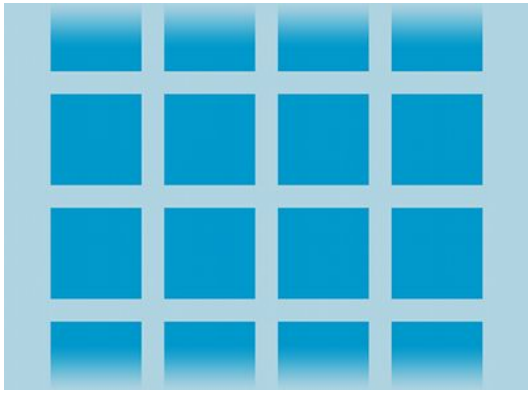
ListView



LinearLayout



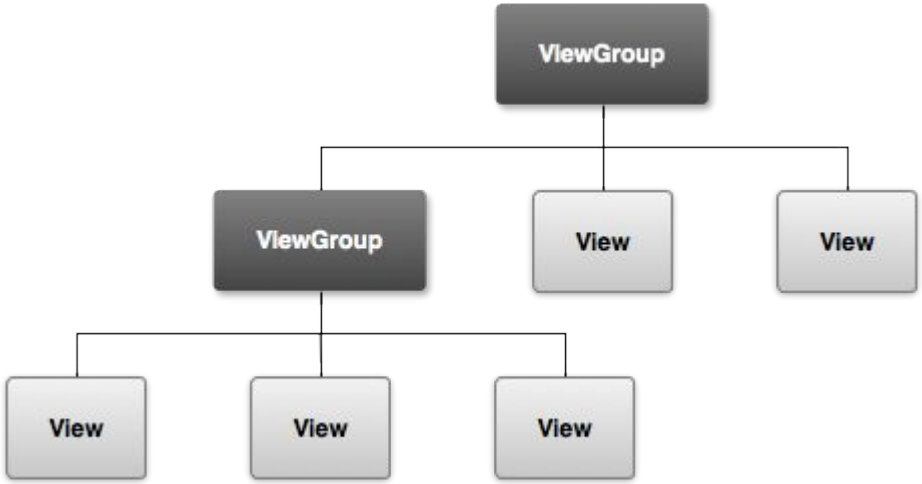
GridView



RelativeLayout

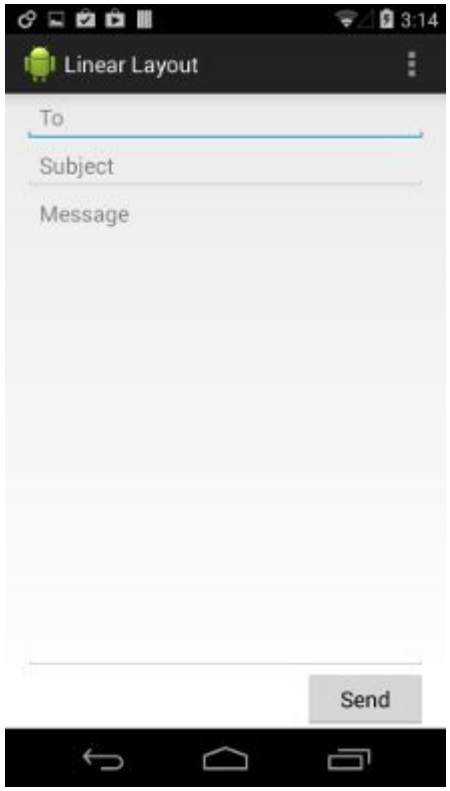
https://developer.android.com/guide/topics/ui/declaring-layout.html

Arborescences de vues



NE PAS CONFONDRE
HIERARCHIE DE COMPOSANTS
ET DE CLASSES

Linear Layout



Taille : hauteur / largeur
match_parent/-1 taille du parent - padding
wrap_content/-2 Taille suffisante pour le contenu + padding
Taille fixe en px, dp, …

Espacement entre les éléments android:padding (padding top, left, right et bottom) : par défaut proches sinon à déterminer
Difference avec Margin Espacement interne vs externe

Orientation : Vertical ou horizontal

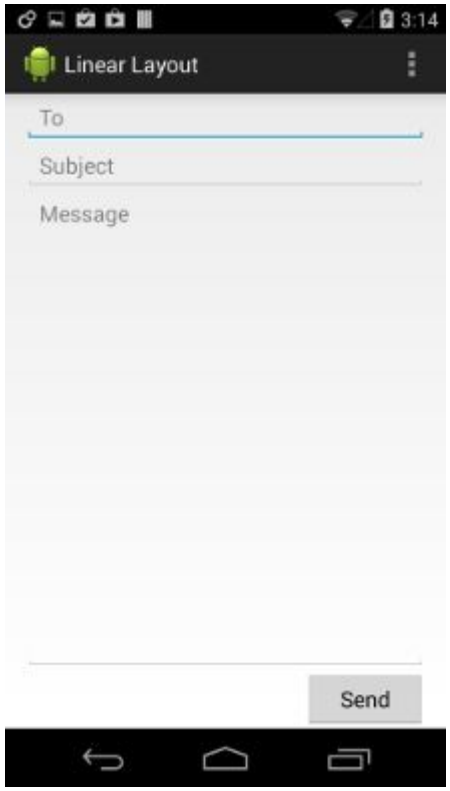
Gravité : positionnement dans la page
left, center_horizontal, center_vertical ,top, bottom, right

Poids des éléments : pour leur accorder plus ou moins de place

Mesures

Pour le développement web, il existe 3 unités principales : les px (pixels), les em et les %.
Pour le développement d'application Android, il y en a 6 :
dp : Density independen Pixel (Densité de pixels indépendant) - Unité abstraite qui est basés sur la densité physique de l'écran. Cette unité est égale à 160 DPI (Points par pouce) par écran. Cette dimension est utilisé pour la mise en page des éléments.
sp : Scale independent Pixel (Echelle de pixels indépendant) - Utilisé pour les tailles de polices. On pourrait comparer cette unité aux em du développement web. La police peut être plus ou moins grosse suivant les préférences utilisateurs
pt : Point - 72 points par pouces. basé sur la taille physique de l'écran.
px : Pixels - Corresponds aux pixels réels de l'écran. Cette unité de mesure n'est pas recommandées car le rendu sur les différents types d'écran peut être différents. Le nombre de pixels par pouce peut varier suivant les appareils.
mm : Millimètre - basée sur la taille physique de l'écran
in : Inches (Pouces) - basée sur la taille physique de l'écran
Ces dimensions se définissent dans les layout (les fichiers XML).

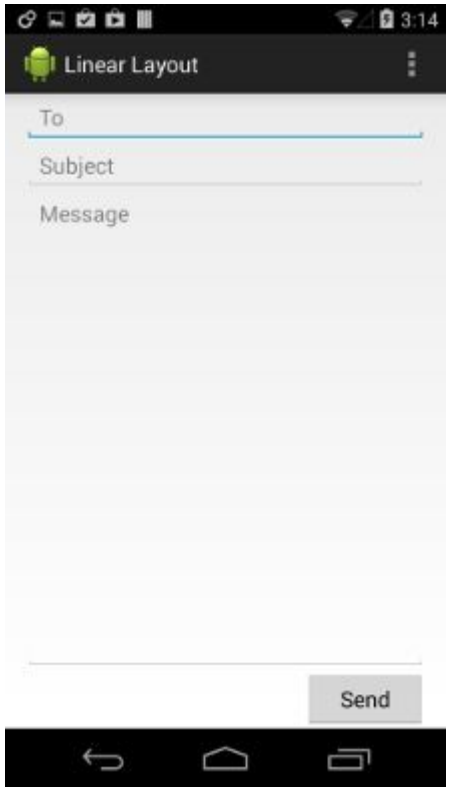
Linear Layout




```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingLeft="16dp"
   android:paddingRight="16dp"
   android:orientation="vertical" >
   <EditText
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:hint="@string/to" />
   <EditText
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:hint="@string/subject" />
   <EditText
      android:layout_width="match_parent"
      android:layout_height="0dp"
      android:layout_weight="1"
      android:gravity="top"
      android:hint="@string/message" />
   <Button
      android:layout_width="100dp"
      android:layout_height="wrap_content"
      android:layout_gravity="right"
      android:text="@string/send" />
</LinearLayout>
```

match_parent/-1 taille du parent - padding
wrap_content/-2 Taille suffisante pour le contenu + padding

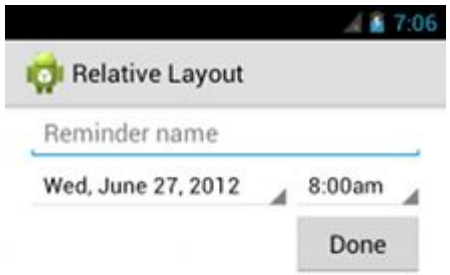
Linear Layout




```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingLeft="16dp"
   android:paddingRight="16dp"
   android:orientation="vertical" >
   <EditText
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:hint="@string/to" />
   <EditText
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:hint="@string/subject" />
   <EditText
      android:layout_width="match_parent"
      android:layout_height="0dp"
      android:layout_weight="1"
      android:gravity="top"
      android:hint="@string/message" />
   <Button
      android:layout_width="100dp"
      android:layout_height="wrap_content"
      android:layout_gravity="right"
      android:text="@string/send" />
</LinearLayout>
```

match_parent/-1 taille du parent - padding
wrap_content/-2 Taille suffisante pour le contenu + padding

Relative Layout



android:layout_alignParentTop
le coin haut de la vue est aligné au coin haut du parent

android:layout_centerVertical
la vue est centrée verticalement par rapport au parent

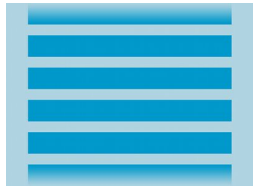
android:layout_below
le coin haut de la vue est au dessous de la vue identifiée par l'ID

android:layout_toRightOf
le coin gauche de la vue est à la droite de la vue identifiée par l'ID

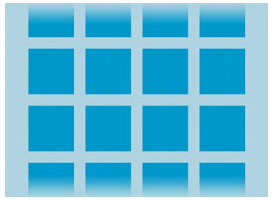
Adaptateurs et Layout

Pour un contenu dynamique non prédéfini,
AdapterView (par spécialisation…) permet de placer les vues dans le layout à l'éxécution.
Il faut un Adaptateur pour relier les données au layout. .

Par exemple **ListView** permet d'avoir des items scrollables. Les items sont automatiquement insérés dans la liste via un **Adaptateur** qui met les items d'un tableau ou d'une base de données.



Ce n'est pas le seul Layout qui se construit avec des Adaptateurs regardez aussi GridLayout



ListView layout : exemple

Les items sont des listes de noms et de numéros de téléphone.
Ce qui implique des requêtes au Contacts Provider et donc
L'ajout de permission au Manifest

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

ListActivity :
ListView layout par défaut.

```
public class ListViewLoader extends ListActivity
      implements LoaderManager.LoaderCallbacks<Cursor> {

   // This is the Adapter being used to display the list's data
   SimpleCursorAdapter mAdapter;

   // These are the Contacts rows that we will retrieve
   static final String[] PROJECTION = new String[] {ContactsContract.Data._ID,
         ContactsContract.Data.DISPLAY_NAME};

   // This is the select criteria
   static final String SELECTION = "((" +
         ContactsContract.Data.DISPLAY_NAME + " NOTNULL) AND (" +
         ContactsContract.Data.DISPLAY_NAME + " != " ))";
```

LoaderCallbacks pour
le CursorLoader qui charge
dynamiquement les données.

Un peu plus sur les adaptateurs

ArrayAdapter si source de données tableau : utilise toString() pour chaque item et met le résultat dans un TextView.

ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, myStringArray)
ListView listView = (ListView) findViewById(R.id.listview);
listView.setAdapter(adapter);

parma 1 : contexte de l'appli
param 2 : layout pour 1 item
param 3 : le tableau de chaîne

Pour faire votre propre visualisation de chaque item

- surcharger le toString
- ou spécialiser ArrayAdapter et surcharge de *getView()* pour remplacer le TextView (par exemple, par un ImageView)

Adaptateurs : suite

SimpleCursorAdapter données issues de Cursor (read-write access to the result set returned by a database query).
Implique de : spécifier un layout pour chaque ligne
et quelles colonnes insérer.
Dans notre cas, le résultat de la requête peut retourner une ligne pour chaque personne
Et 2 colonnes : une pour le nom et l'autre pour les numéros.

Créer un tableau de chaînes pour identifier les colonnes et un tableau d'entiers spécifiant chaque vue correspondant

```
String[] fromColumns =    {ContactsContract.Data.DISPLAY_NAME,
ContactsContract.CommonDataKinds.Phone.NUMBER};
int[] toViews = {R.id.display_name, R.id.phone_number};

SimpleCursorAdapter adapter = new SimpleCursorAdapter(this,
            R.layout.person_name_and_number, cursor, fromColumns, toViews, 0);
ListView listView = getListView(); listView.setAdapter(adapter);
```

Implique la création d'une vue pour chaque rangée en utilisant le layout fourni et les deux tableaux.

ListView layout

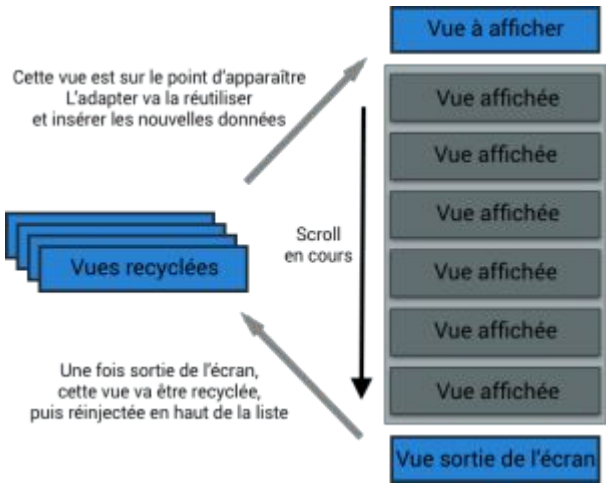
```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Create a progress bar to display while the list loads
    ProgressBar progressBar = new ProgressBar(this);
    progressBar.setLayoutParams(new LayoutParams(LayoutParams.WRAP_CONTENT,
            LayoutParams.WRAP_CONTENT, Gravity.CENTER));
    progressBar.setIndeterminate(true);
    getListView().setEmptyView(progressBar);

    // Must add the progress bar to the root of the layout
    ViewGroup root = (ViewGroup) findViewById(android.R.id.content);
    root.addView(progressBar);

    // For the cursor adapter, specify which columns go into which views
    String[] fromColumns = {ContactsContract.Data.DISPLAY_NAME};
    int[] toViews = {android.R.id.text1}; // The TextView in simple_list_item_1

    // Create an empty adapter we will use to display the loaded data.
    // We pass null for the cursor, then update it in onLoadFinished()
    mAdapter = new SimpleCursorAdapter(this,
            android.R.layout.simple_list_item_1, null,
            fromColumns, toViews, 0);
    setListAdapter(mAdapter);

    // Prepare the loader.  Either re-connect with an existing one,
    // or start a new one.
    getLoaderManager().initLoader(0, null, this);
}
```

ListView Layout : accès aux contacts

```
    // Called when a new Loader needs to be created
    public Loader<Cursor> onCreateLoader(int id, Bundle args) {
        // Now create and return a CursorLoader that will take care of
        // creating a Cursor for the data being displayed.
        return new CursorLoader(this, ContactsContract.Data.CONTENT_URI,
                PROJECTION, SELECTION, null, null);
    }

    // Called when a previously created loader has finished loading
    public void onLoadFinished(Loader<Cursor> loader, Cursor data) {
        // Swap the new cursor in.  (The framework will take care of closing the
        // old cursor once we return.)
        mAdapter.swapCursor(data);
    }

    // Called when a previously created loader is reset, making the data unavailable
    public void onLoaderReset(Loader<Cursor> loader) {
        // This is called when the last Cursor provided to onLoadFinished()
        // above is about to be closed.  We need to make sure we are no
        // longer using it.
        mAdapter.swapCursor(null);
    }

    @Override
    public void onListItemClick(ListView l, View v, int position, long id) {
        // Do something when a list item is clicked
    }
}
```

" />

Fonctionnement de ListView : recyclage et fluidité



Pour réduire la consommation en mémoire la ListView stocke seulement les vues qu'elle a la capacité d'afficher,
Lorsqu'une vue sort de l'écran (scroll) elle est réutilisée pour la nouvelle vue à apparaître.

Afin d'éviter d'appeler les méthodes findViewById à chaque réutilisation des vues, Android a rajouté un concept, le ViewHolder (gardien/protecteur de vue) : mini controleur, associé a chaque cellule, et qui va stocker les références vers les sous vues.
C'est une propriété de la vue (dans l'attribut tag) : une vue n'a qu'un seul ViewHolder, et inversement.

http://tutos-android-france.com/listview-afficher-une-liste-delements/

```
class TweetViewHolder{
    public TextView pseudo;
    public TextView text;
    public ImageView avatar;
}
View cellule = ...;
TweetViewHolder viewHolder = (TweetViewHolder) cellule.getTag();
if(viewHolder == null){
viewHolder = new TweetViewHolder();

    //récupérer nos sous vues
    viewHolder.pseudo = (TextView)  cellule.findViewById(R.id.pseudo);
    viewHolder.text   = (TextView)  cellule.findViewById(R.id.text);
    viewHolder.avatar = (ImageView) cellule.findViewById(R.id.avatar);

    //puis on sauvegarde le mini-controlleur dans la vue
    cellule.setTag(viewHolder);
}
```

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {
 if(convertView == null){
       //Nous récupérons notre row_tweet via un LayoutInflater,
       //qui va charger un layout xml dans un objet View
       convertView = LayoutInflater.from(getContext()).inflate(R.layout.row_tweet,parent, false);
   }

   TweetViewHolder viewHolder = (TweetViewHolder) convertView.getTag();
   if(viewHolder == null){
       viewHolder = new TweetViewHolder();
       viewHolder.pseudo = (TextView) convertView.findViewById(R.id.pseudo);
       viewHolder.text = (TextView) convertView.findViewById(R.id.text);
       viewHolder.avatar = (ImageView) convertView.findViewById(R.id.avatar);
       convertView.setTag(viewHolder);
   }

   //nous renvoyons notre vue à l'adapter, afin qu'il l'affiche
   //et qu'il puisse la mettre à recycler lorsqu'elle sera sortie de l'écran
   return convertView;
}….
```

Remplir les cellules avec les données d’un Tweet

Grid View

http://developer.android.com/guide/topics/ui/layout/gridview.html

COMPOSANTS : WIDGETS

Concrètement en Android

Une interface utilisateur d’une activité correspond à une hiérarchie de vues (dérivées de la classe View)

Chaque vue correspond à un espace de la fenêtre d’activité et peut correspondre à une interaction utilisateur : ex. un bouton.

Vous pouvez utiliser des widgets "Widgets" existant pour concevoir votre IHM

Vous pouvez aussi créér en sous classant des classes existantes vos propres vues.

Boutons


```
<?xml version="1.0" encoding="utf-8"?>
<Button
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/button_send"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"
    android:onClick="sendMessage" />
```

Bouton sans bord

```
<Button
    android:id="@+id/button_send"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"
    android:onClick="sendMessage"
    style="?android:attr/borderlessButtonStyle" />
```

Gestion du click

```
/** Called when the user touches the button */
public void sendMessage(View view) {
    // Do something in response to button click
}
```

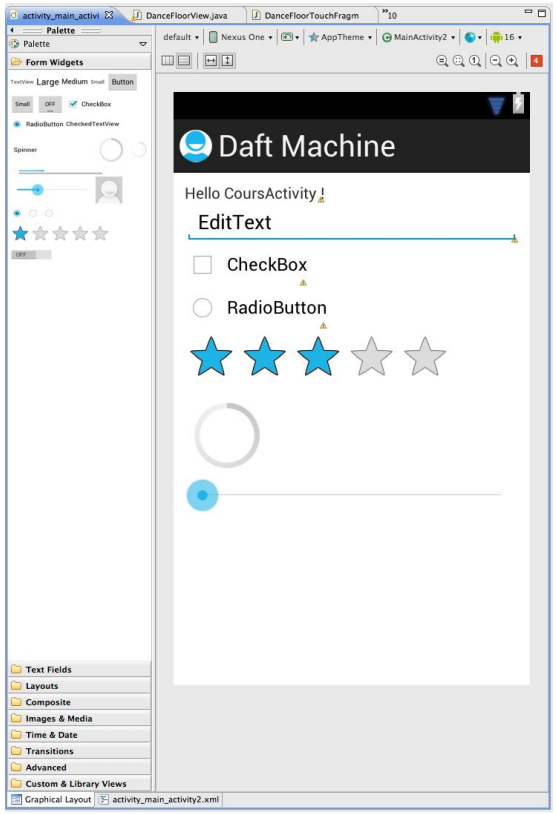
```
Button button = (Button) findViewById(R.id.button_send);
button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        // Do something in response to button click
    }
});
```

Pour personnaliser le fond

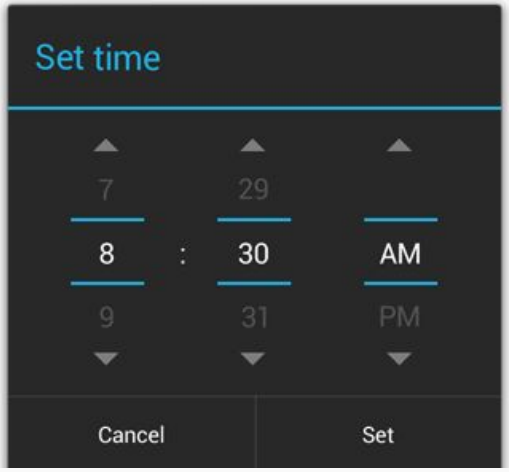
```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/button_pressed"
          android:state_pressed="true" />
    <item android:drawable="@drawable/button_focused"
          android:state_focused="true" />
    <item android:drawable="@drawable/button_default" />
</selector>
```

Exemples de WIDGETS

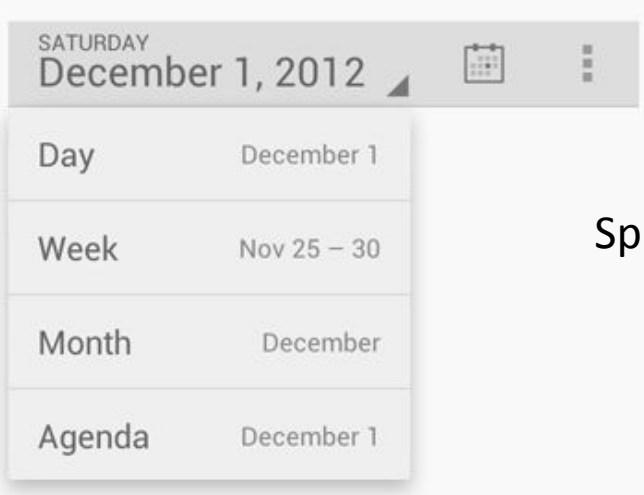
Formulaire



Gestion du temps

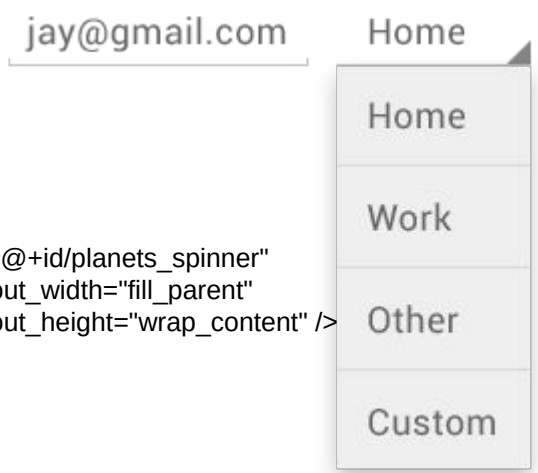


Data Time Picker



Spinner

Spinner




```
<Spinner
    android:id="@+id/planets_spinner"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />
```

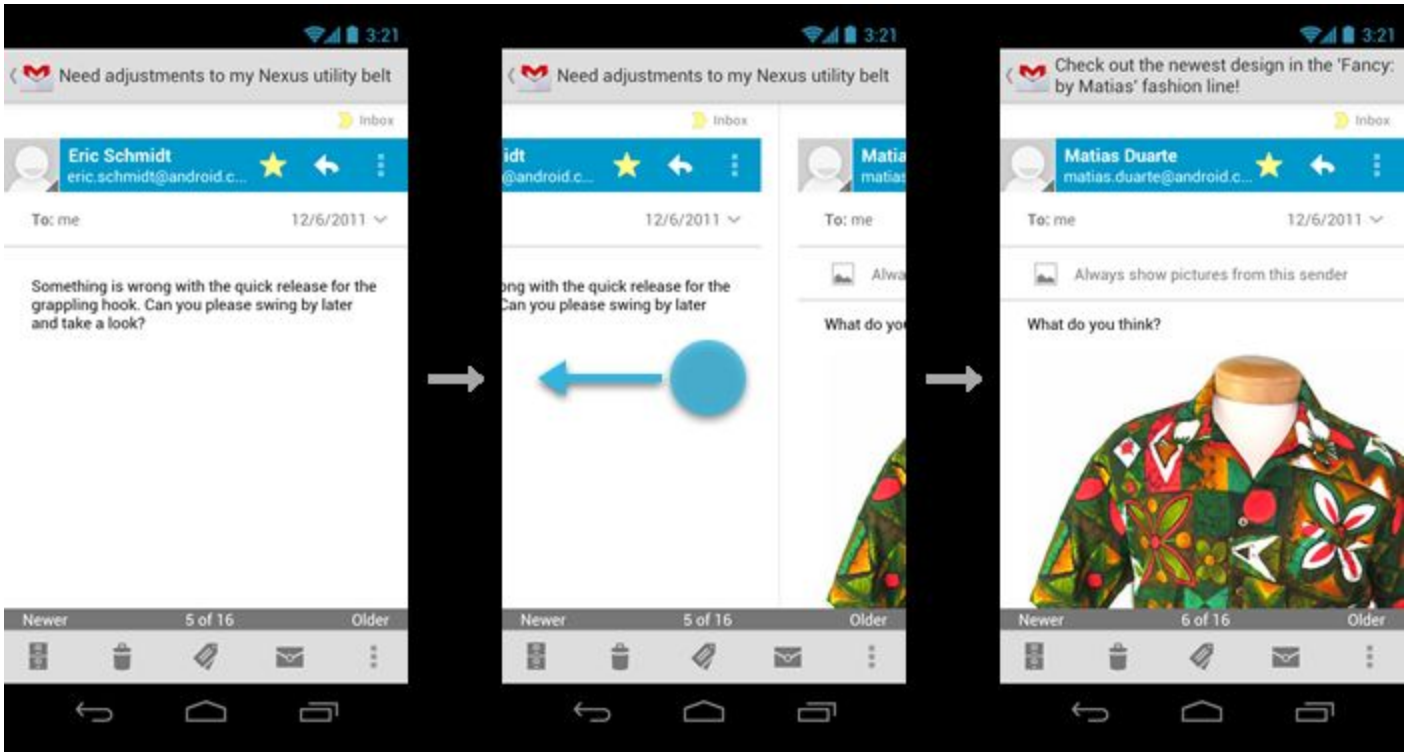
```
Spinner spinner = (Spinner) findViewById(R.id.spinner);
// Create an ArrayAdapter using the string array and a default spinner layout
ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this,
        R.array.planets_array, android.R.layout.simple_spinner_item);
// Specify the layout to use when the list of choices appears
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
// Apply the adapter to the spinner
spinner.setAdapter(adapter);
```

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="planets_array">
        <item>Mercury</item>
        <item>Venus</item>
        <item>Earth</item>
        <item>Mars</item>
        <item>Jupiter</item>
        <item>Saturn</item>
        <item>Uranus</item>
        <item>Neptune</item>
    </string-array>
</resources>
```

```
public class SpinnerActivity extends Activity implements OnItemSelectedListener {
    ...

    public void onItemSelected(AdapterView<?> parent, View view, int pos, long id) {
        // An item was selected. You can retrieve the selected item using
        // parent.getItemAtPosition(pos)
    }

    public void onNothingSelected(AdapterView<?> parent) {
        // Another interface callback
    }
}
```

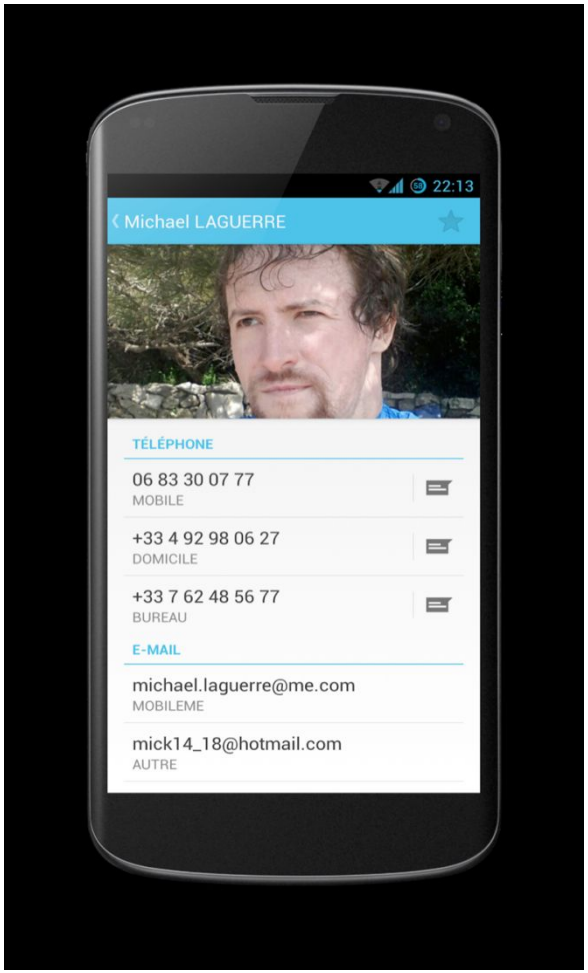
```
Spinner spinner = (Spinner) findViewById(R.id.spinner);
spinner.setOnItemSelectedListener(this);
```

ViewPager

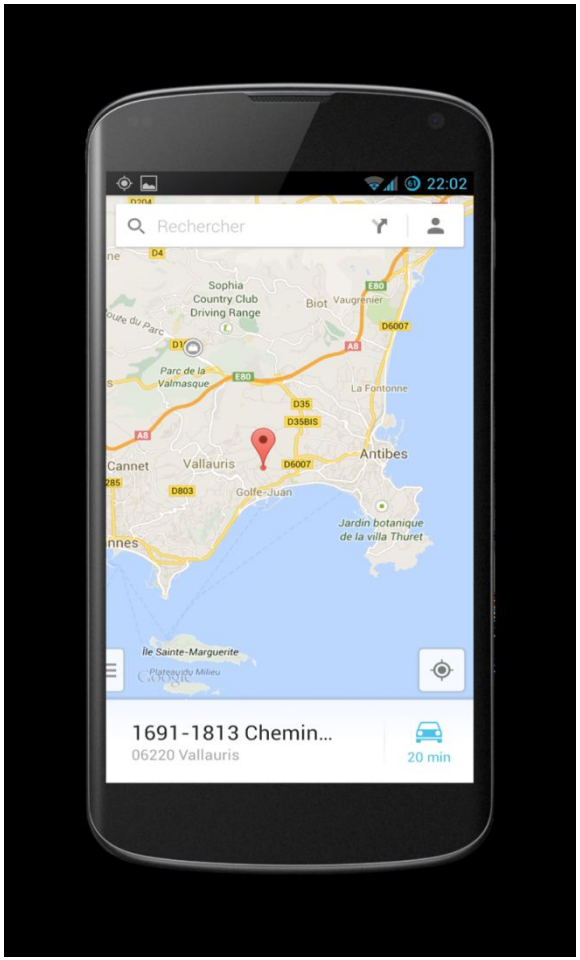

3:21
Need adjustments to my Nexus utility belt
Inbox
Eric Schmidt
eric.schmidt@android.c...
To: me
12/6/2011
Something is wrong with the quick release for the grappling hook. Can you please swing by later and take a look?
Newer
5 of 16
Older
3:21
Need adjustments to my Nexus utility belt
Inbox
12/6/2011
To: me
Newer
5 of 16
Older
3:21
Check out the newest design in the 'Fancy: by Matias' fashion line!
Inbox
Matias Duarte
matias.duarte@android.c...
To: me
12/6/2011
Always show pictures from this sender
What do you think?
Newer
6 of 16
Older

Vues spécialisées

WebView



ScrollView



MapView

ViewPager

Pour présenter une galerie en plein écran : pour scroller horizontalement d'un écran à un autre.

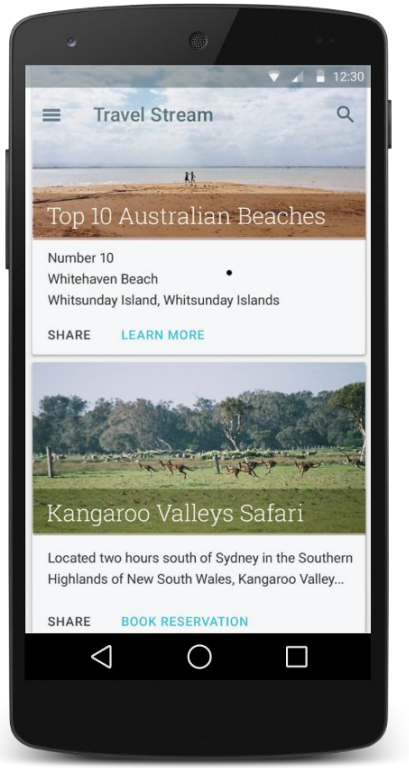
1 Déclarer le Viewpager dans un layout.
2 Utiliser un adapter pour remplir le ViewPager
Le PageAdapter est le plus courant.
Avec **getCount()**: retourne le nombres de pages du ViewPager
instantiateItem(View collection, int position): Crée une view pour une position donnée.
destroyItem(View collection, int position, Object view): Supprime une vue d'une position donnée.
3 **Agir sur le Viewer depuis le code.**
Attacher un listener au viewPager grâce à la méthode setOnPageChangeListener(myPageChangeListener).
Cela permet par exemple d'indiquer le numéro de page sur une TextView.

Souvent lié aux Fragments pour réutiliser des parties d'activité dans d'autres activités (cf. **FragmentPagerAdapter**).

http://blog.oxiane.com/2011/12/19/utiliser-le-viewpager-pour-slider-dun-ecran-a-un-autre/

CardView

CardView est un élément respectant le style Material Design.
Ajout d'une profondeur dans l'application : 3ème index (z-index en css), l'élévation.
Une image et un texte associé



CardView hérite de FrameLayout
Les informations sont visualisées dans des « Cartes »
Peuvent être personalisées avec des ombres etdes coins arrondis

Ombres : via l'attribut card_view:cardElevation.
ATTENTION Simplification en Android 5.0 (API level 21).
Autres attributs utiles :
card_view:cardCornerRadius ouCardView.setRadius.
card_view:cardBackgroundColor.

http://tutos-android-france.com/material-design-recyclerview-et-cardview/

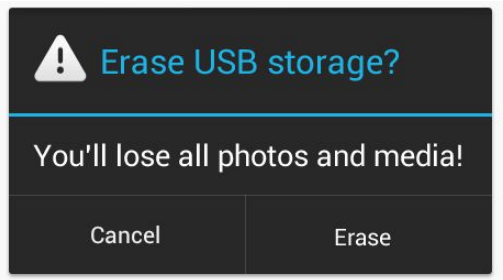
Exemple

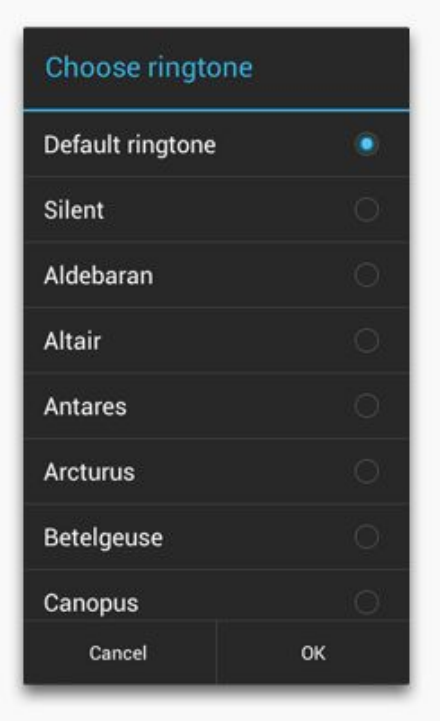
```
?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.CardView xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="5dp"

   app:cardBackgroundColor="@android:color/white"
    app:cardCornerRadius="2dp"
    app:cardElevation="2dp">
      <LinearLayout
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:orientation="vertical">

       <ImageView
          android:id="@+id/image"
          android:layout_width="match_parent"
          android:layout_height="200dp"
          android:scaleType="centerCrop"
          tools:src="@drawable/parisguidetower" />


       <TextView
          android:id="@+id/text"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:background="?android:selectableItemBackground"
          android:padding="20dp"
          tools:text="Paris"
          android:fontFamily="sans-serif"
          android:textColor="#333"
          android:textSize="18sp" />
   </LinearLayout>
</android.support.v7.widget.CardView>
```

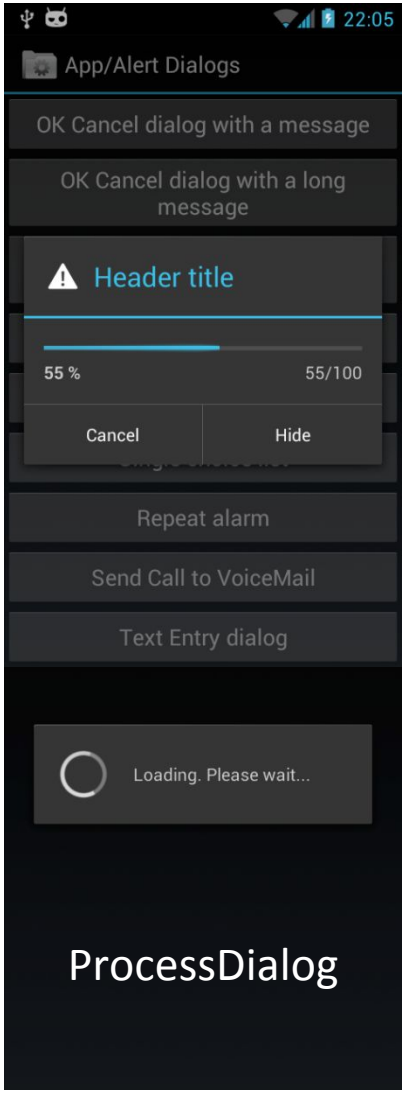
http://developer.android.com/design/building-blocks/dialogs.html

Dialogues

AlertDialog

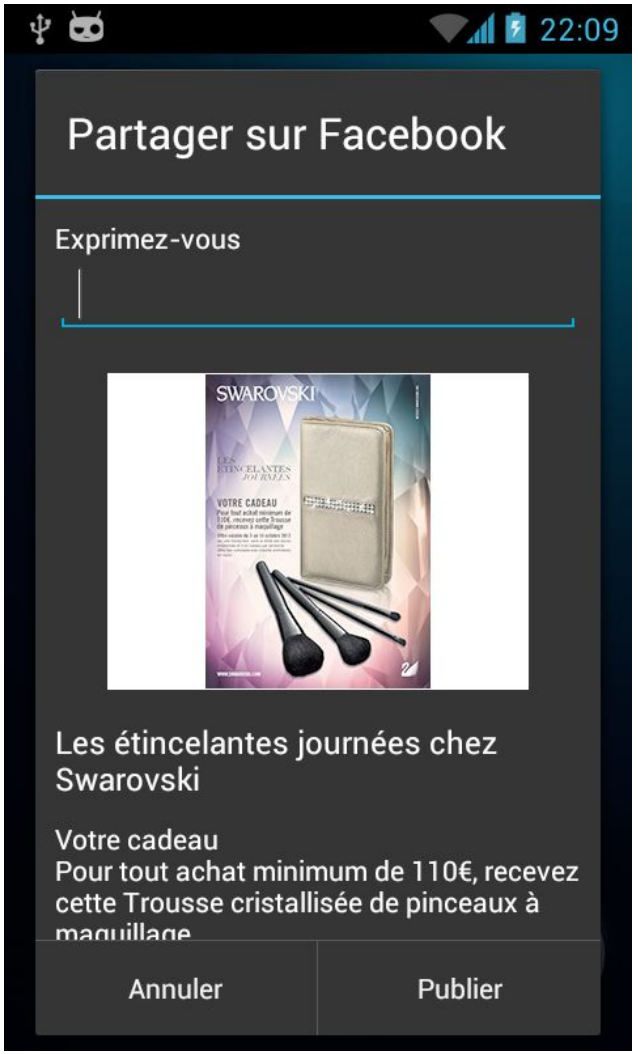






ProcessDialog

CustomDialog



Alert Dialog


```
// 1. Instantiate an AlertDialog.Builder with its constructor
AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());

// 2. Chain together various setter methods to set the dialog characteristics
builder.setMessage(R.string.dialog_message)
       .setTitle(R.string.dialog_title);

// 3. Get the AlertDialog from create()
AlertDialog dialog = builder.create();
```

Il y a des boutons : Positifs, négatifs et neutres

```
AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
// Add the buttons
builder.setPositiveButton(R.string.ok, new DialogInterface.OnClickListener() {
          public void onClick(DialogInterface dialog, int id) {
             // User clicked OK button
          }
      });
builder.setNegativeButton(R.string.cancel, new DialogInterface.OnClickListener() {
          public void onClick(DialogInterface dialog, int id) {
             // User cancelled the dialog
          }
      });
// Set other dialog properties
...

// Create the AlertDialog
AlertDialog dialog = builder.create();
```