Développement Android (4.3)

Graphical User Interface (GUI)

BENJAMIN BILLET

WARNING

Le contenu de cette présentation est basé sur la documentation anglophone officielle d'Android, diffusée sous licence *Creative Commons Attribution 2.5* :

developer.android.com

La plupart des schémas qui composent ce cours proviennent de cette documentation et sont, par conséquent, soumis à cette même licence.

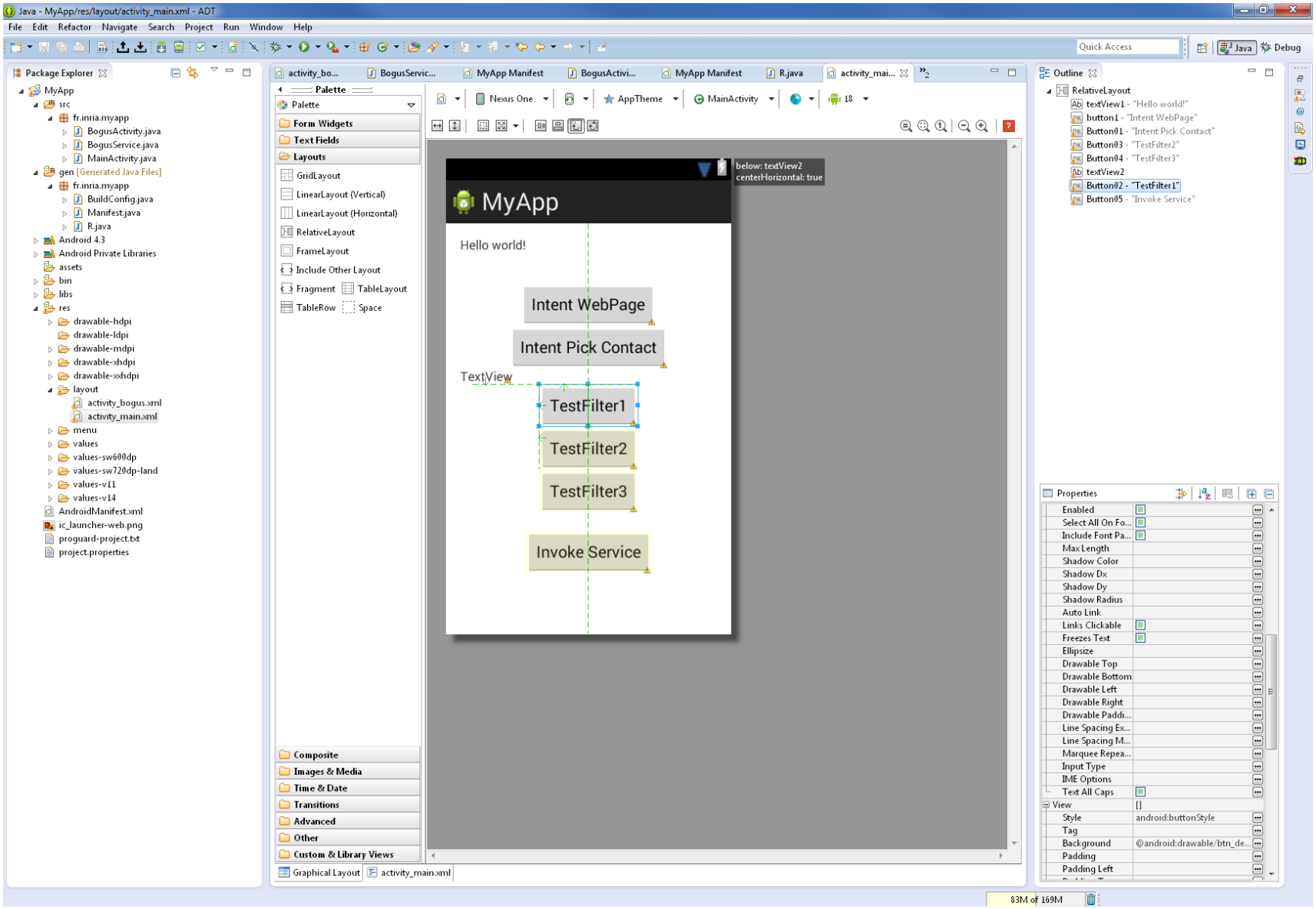
http://creativecommons.org/licenses/by/2.5/

Au Sommaire !

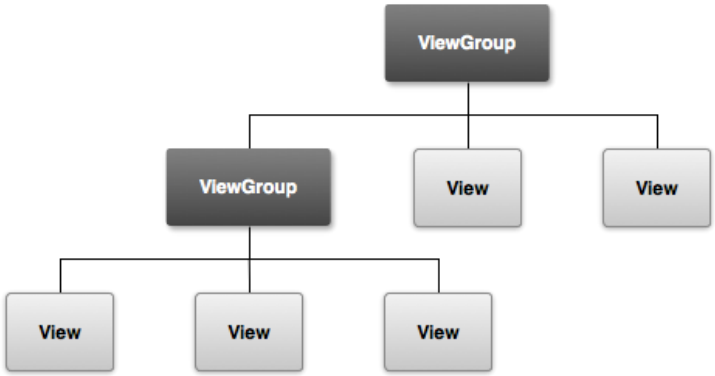
- Introduction
- Layout
- Widgets simples & Gesture
- Fragment
- Adapter
- Autres composants (Menu, Action Bar, Dialogs, etc.)
- Drag & Drop
- Notifications

INTRODUCTION

INTRODUCTION



INTRODUCTION




```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="I am a Button" />
</LinearLayout>
```

- Tous les composants graphiques héritent de View et ViewGroup (pattern Composite).
- L'interface graphique est décrite en XML.
- Le SDK transforme cette description en fichier R.java. La classe R permettra d'accéder aux instances concrètes de chaque composant grâce à leurs IDs
- Spoiler : Android gère l'interface graphique dans le thread principal (ou UI Thread).

Introduction

/res	
/layout	Layout descriptions
/menu	Menu descriptions
/values	
/strings.xml	Default strings
/dimens.xml	Default dimensions
/styles.xml	Default styles
/values-*	Specific values
/drawable-*	Graphical resources (images)

/res/values-en/strings.xml : traduction anglaise.
/res/drawable-ja : images optimisés pour le public japonais.
/res/values-v14/styles.xml : style spécifique à l'API 14
/res/drawable-hdpi : images optimisées pour écran haute densité (~240 dpi).

developer.android.com/guide/topics/resources/providing-resources.html

INTRODUCTION

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

<string name="app_name">MyApp</string>
<string name="hello_world">Hello world!</string>

</resources>
```

strings.xml

- @+id/id_of_an_element : déclare un id.
- @id/id_of_an_element : référence un id déjà déclaré.
- @string/string_name : une chaîne dans strings.xml.
- @drawable/my_image : my_image.png dans /res/drawable
- @anim/my_animation : my_animation.xml (descripteur d'animation) dans /res/anim (descripteur d'animation).
- @layout/my_layout : my_layout.xml dans /res/layout (pour inclusion, par exemple).

Introduction

- View :
  - Identifiant

    ```
    findViewById(R.id.myElementID);
    ```
  - layout_width, layout_height
    - Valeur exacte (px, mm, in, pt, dp).
    - wrap_content : s'adapter au contenu.
    - match_parent : s'adapter à la taille du parent.
  - Quelques méthodes :
    - getLeft(), getTop(), getRight(), getBottom()
    - getPaddingLeft(), getPaddingTop(), …
- ViewGroup :
  - layout_marginLeft, layout_marginTop, …

INTRODUCTION

```java
protected void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_layout); // res/layout/main_layout.xml

    Button myButton = (Button) findViewById(R.id.my_button);
}
```

```xml
<Button android:id="@+id/my_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/my_button_text" />
```

Remarque : Attention à bien manipuler l'interface après le setContentView, sinon findViewById retourne null à tous les coups.

Introduction

```
<Button android:layout_width="100dp" android:layout_height="wrap_content"
    android:text="@string/send"
    android:onClick="sendMessage"
    android:id="@+id/button_send"
/>

public class MyActivity extends Activity
{
    public void sendMessage(View v) // solution 1
    {
    }

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.my_activity);
        Button button = (Button) findViewById(R.id.button_send);
        button.setOnClickListener(new View.OnClickListener() // solution 2
        {
            @Override
            public void onClick(View v)
            {
            }
        });
    }
}
```

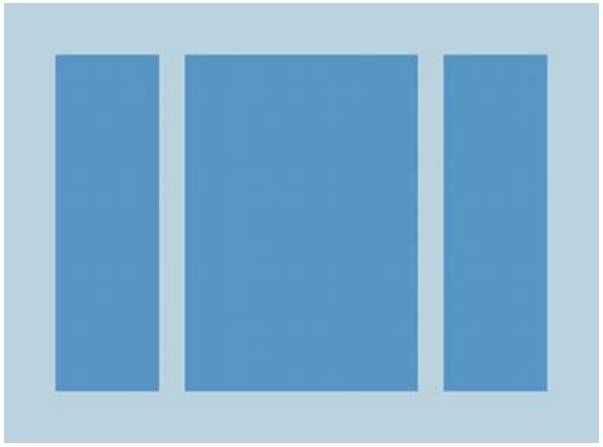
LAYOUT

Les layouts

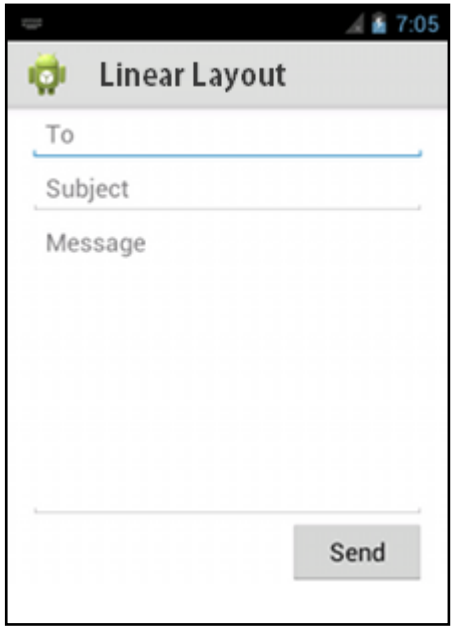
- Un layout définit une structure d'organisation pour tous les composants qu'il contient (ViewGroup).
- L'interface graphique d'une activité utilise un layout (la racine de la description XML).
- Les layouts peuvent contenir des layouts.
- Les layouts, comme la plupart des ressources, se trouvent dans le dossier "res", puis "layout".

```
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_layout); // res/layout/main_layout.xml
}
```

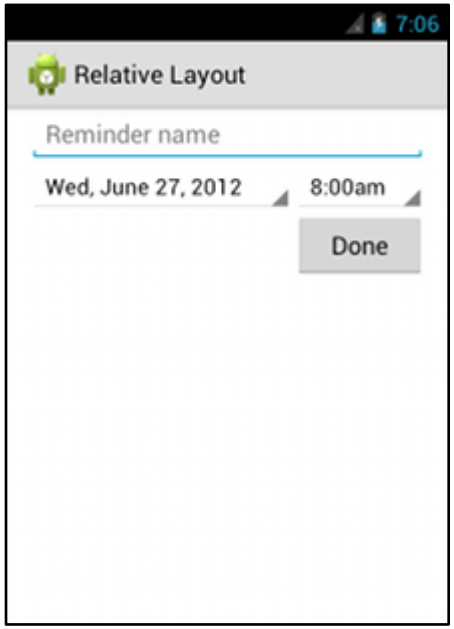
Les layouts

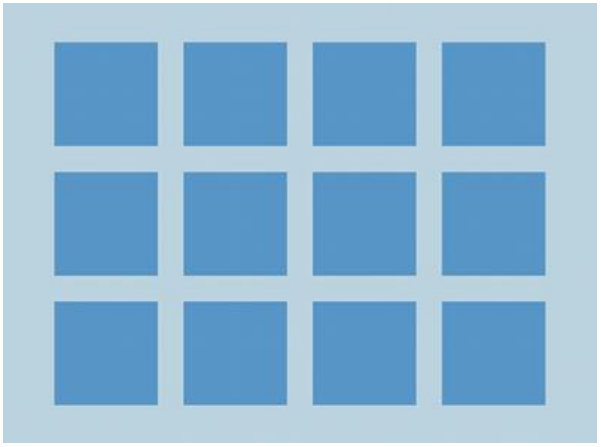


LinearLayout

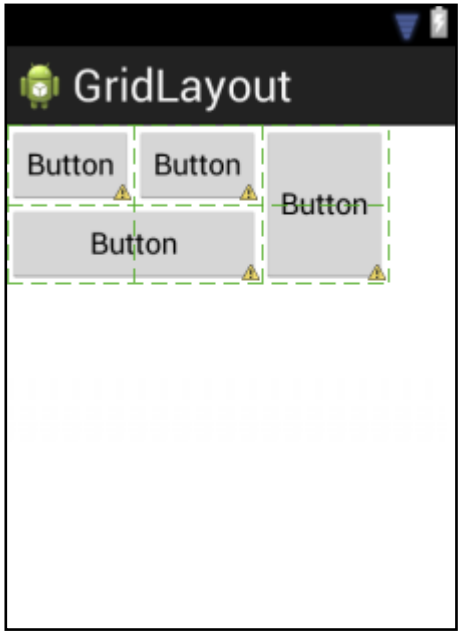


RelativeLayout





GridLayout



LinearLayout

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:orientation="vertical" >
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="@string/to" />
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="@string/subject" />
    <EditText
        android:layout_width="fill_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>
```

- Le weight indique que le composant doit utiliser l'espace restant (d'où height = 0).
- Si plusieurs composants définissent un poid, alors l'espace est partagé relativement à chaque poids.

RelativeLayout

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp" >
    <EditText
        android:id="@+id/name"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="@string/reminder" />
    <Spinner
        android:id="@+id/dates"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentLeft="true"
        android:layout_toLeftOf="@+id/times" />
    <Spinner
        android:id="@id/times"
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentRight="true" />
    <Button
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/times"
        android:layout_alignParentRight="true"
        android:text="@string/done" />
</RelativeLayout>
```

- Positionner un composant relativement à un autre :
  - toLeftOf, toRightOf
  - below, above
  - alignTop, alignLeft
  - …
- Positionner un composant relativement au parent :
  - alignParentLeft, alignParentTop
  - centerVertical, centerHorizontal
  - …

GridLayout

```
<?xml version="1.0" encoding="utf-8"?>
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/GridLayout1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:columnCount="3"
    android:rowCount="2"
    tools:context=".GridLayoutActivity" >

    <Button
        android:id="@+id/button3"
        android:layout_column="0"
        android:layout_gravity="left|top"
        android:layout_row="0"
    <Button
        android:id="@+id/button1"
        android:layout_column="1"
        android:layout_gravity="left|top"
        android:layout_row="0"
    <Button
        android:id="@+id/button2"
        android:layout_column="2"
        android:layout_gravity="fill_vertical"
        android:layout_row="0"
        android:layout_rowSpan="2"
    <Button
        android:id="@+id/button4"
        android:layout_column="0"
        android:layout_columnSpan="2"
        android:layout_gravity="fill_horizontal"
        android:layout_row="1"
</GridLayout>
```

- column, row : positionner le composant dans la grille.
- columnSpan, rowSpan : fusionner des colonnes et des lignes.

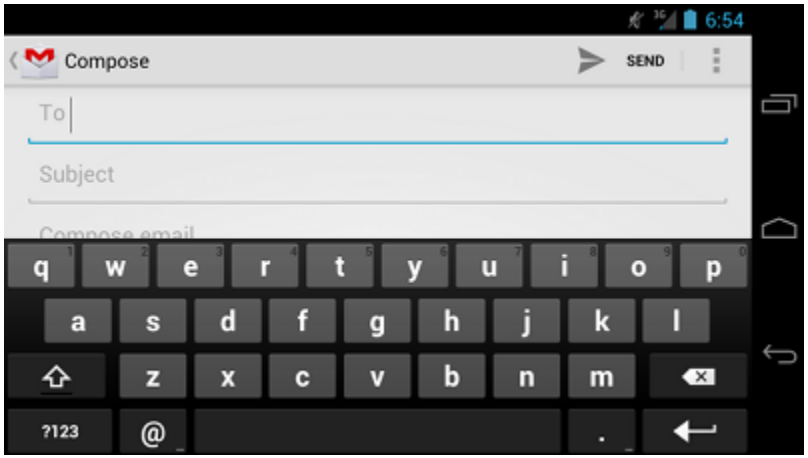
WIDGETS SIMPLES & GESTURE

Button


```
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/button_text"
... />

<ImageButton
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:src="@drawable/button_icon"
... />

<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/button_text"
android:drawableLeft="@drawable/button_icon"
... />
```

EditText

```
<EditText
    android:id="@+id/email_address"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="@string/email_hint"
    android:inputType="textEmailAddress" />
```

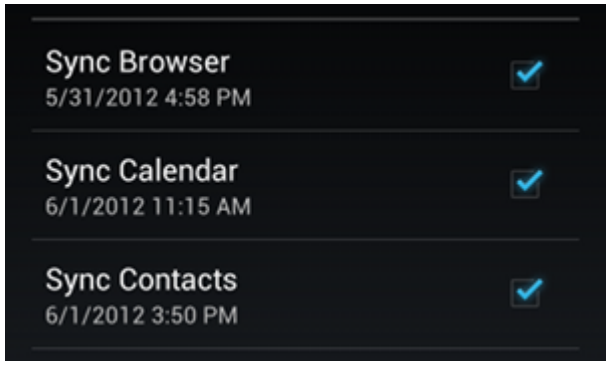


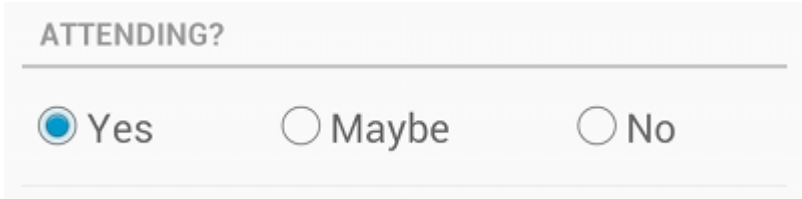
inputType = " textEmailAddress"

inputType = "phone"

CheckBox, RadioButton, ToggleButton

```
<CheckBox android:id="@+id/checkbox"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/cheese"
        android:onClick="onCheckboxClicked"/>

<RadioGroup
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <RadioButton android:id="@+id/radio_yes"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/yes"
        android:onClick="onRadioButtonClicked"/>
    <RadioButton android:id="@+id/radio_no"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/no"
        android:onClick="onRadioButtonClicked"/>
</RadioGroup>

<ToggleButton
    android:id="@+id/togglebutton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textOn="Vibrate on"
    android:textOff="Vibrate off"
    android:onClick="onToggleClicked"/>
```





Switch extends ToggleButton

Un petit exemple avec RadioButton

```
public void onRadioButtonClicked(View view)
{
    boolean checked = ((RadioButton) view).isChecked();
    switch(view.getId())
    {
        case R.id.radio_yes:
            if (checked)
                // yes
            break;
        case R.id.radio_maybe:
            if (checked)
                // maybe
            break;
        case R.id.radio_no:
            if (checked)
                // no
            break;
    }
}
```

Un petit exemple avec RadioButton

```
RadioGroup radio = (RadioGroup)findViewById(R.id.gender);
switch(user.getGender())
{
     case Male:
          radio.check(R.id.gender_male);
          break;
     case Female:
          radio.check(R.id.gender_female);
          break;
     case Unspecified:
          radio.check(R.id.gender_unspecified);
          break;
}

RadioGroup radio = (RadioGroup)findViewById(R.id.gender);
switch(radio.getCheckedRadioButtonId())
{
     case R.id.gender_male:
          user.setGender(Gender.Male);
          break;
     case R.id.gender_female:
          user.setGender(Gender.Female);
          break;
     default:
          user.setGender(Gender.Unspecified);
}
```

QUELQUES AUTRES

```
<SeekBar
        android:id="@+id/seekBar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:max="100"
        android:progress="75" />
```

```
<ProgressBar
        android:id="@+id/progressBar"
        style="?android:attr/progressBarStyleHorizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:indeterminate="true"
        android:max="100"
        android:progress="45" />
```



```
 style="?android:attr/progressBarStyleLarge"
```

GESTURES

- Tristement, seul android:onClick est disponible pour les descriptions XML. Les autres évènements doivent être implémentés manuellement :
  - View.onClickListener
  - View.onLongClickListener
  - View.OnFocusChangeListener
  - View.OnKeyListener
  - View.OnTouchListener
- Certains listeners retournent un booléen pour indiquer si le système doit continuer à propager l'évènement.
- Certains listeners s'annulent. Par exemple, onTouch() annule la plupart des évènements de clic.

GESTURES

```
// Solution 1
public class MyActivity extends Activity implements OnClickListener
{
	@Override
	protected void onCreate(Bundle savedInstanceState)
	{
		super.onCreate(savedInstanceState);
		Button button1 = (Button)findViewById(R.id.my_button1);
		button1.setOnClickListener(this);
		Button button2 = (Button)findViewById(R.id.my_button2);
		button2.setOnClickListener(this);
	}

	@Override
	public void onClick(View v)
	{
		switch (v.getId())
		{
			case R.id.my_button1:
				break;
			case R.id.my_button2:
				break;
			default:
				break;
		}
	}
}
```

```
// Solution 2
public class MyActivity extends Activity
{
	private OnClickListener listener = new OnClickListener()
	{
		public void onClick(View v)
		{
		}
	};

	@Override
	protected void onCreate(Bundle savedInstanceState)
	{
		super.onCreate(savedInstanceState);
		Button button = (Button)findViewById(R.id.my_button);
		button.setOnClickListener(listener);
	}
}
```

Gestures complexes avec onTouch

- onTouch reçoit un MotionEvent, qui contient des informations sur les zones de l'écran qui sont activées (pointers) : taille, pression, position.
- La détection d'évènements complexes s'effectue au travers de classes dédiées.
  - Tap, Double Tap, Scroll, Long Press, Fling, Swipe : android.view.GestureDetector
  - Pinch : android.view.ScaleGestureDetector
- Retourner false dans onTouch signifie qu'on ne s'intéresse pas à l'event. Le callback ne sera pas invoqué pour les évènements sous-jacent (mouvement de doigt, retrait, etc.).

Exemple du double tap

```
public MyActivity extends Activity implements OnTouchListener
{
     private GestureDetector detector;

     @Override
     protected void onCreate(Bundle savedInstanceState)
     {
          super.onCreate(savedInstanceState);
          setContentView(R.layout.my_activity);
          detector = new GestureDetector(this, new GestureListener());
          findViewById(R.id.mainLayout).setOnTouchListener(this);
     }

     @Override
     public boolean onTouch(View v, MotionEvent ev)
     {
          detector.onTouchEvent(ev);
          return true;
     }

     private class GestureListener extends GestureDetector.SimpleOnGestureListener
     {
          @Override
          public boolean onDoubleTap(MotionEvent e)
          {
               return true;
          }

          @Override
          public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX, float velocityY)
          {
               return true;
          }
     }
}
```

EXEMPLE DU PINCH

```
public MyActivity extends Activity implements OnTouchListener
{
     private ScaleGestureDetector detector;

     @Override
     protected void onCreate(Bundle savedInstanceState)
     {
          super.onCreate(savedInstanceState);
          setContentView(R.layout.my_activity);
          detector = new ScaleGestureDetector(this, new ScaleListener());
          findViewById(R.id.mainLayout).setOnTouchListener(this);
     }

     @Override
     public boolean onTouch(View v, MotionEvent ev)
     {
          detector.onTouchEvent(ev);
          return true;
     }

     private class ScaleListener extends ScaleGestureDetector.SimpleOnScaleGestureListener
     {
          @Override
          public boolean onScale(ScaleGestureDetector detector)
          {
               scaleFactor *= detector.getScaleFactor();
               return true;
          }

          @Override
          public void onScaleEnd(ScaleGestureDetector detector)
          {
          }
     }
}
```

FOCUS

- L'enchaînement du focus est géré automatiquement par Android, mais celui-ci peut éventuellement être redéfini.

```
<LinearLayout
    android:orientation="vertical"
    ... >
  <Button android:id="@+id/top"
          android:nextFocusUp="@+id/bottom"
          ... />
  <Button android:id="@id/bottom"
          android:nextFocusDown="@id/top"
          ... />
</LinearLayout>
```

- Selon le type de device, certains éléments ne sont pas focusable (e.g., un bouton sur un écran tactile) :
  - isFocusableInTouchMode(), setFocusableInTouchMode(), android:focusableInTouchMode
  - android:focusable et setFocusable() pour forcer le comportement.
  - isInTouchMode()

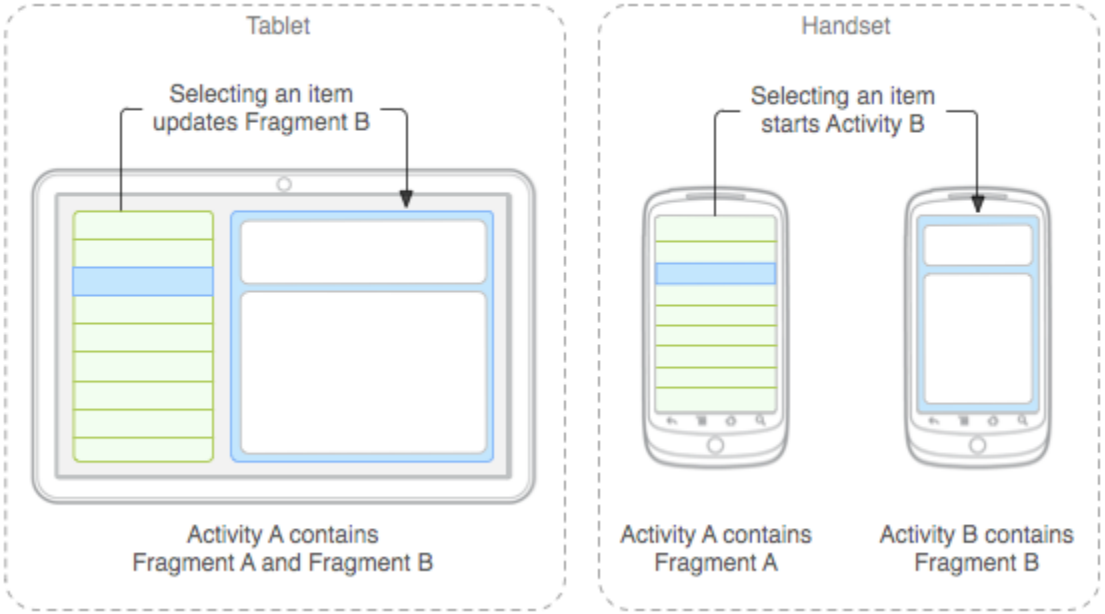
FOCUS

```
public class MyActivity extends Activity implements OnFocusListener
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        EditText text = (EditText)findViewById(R.id.my_text);
        text.setOnClickListener(this);
        text.requestFocus(); // ask for the focus explicitely
    }

    @Override
    public void onFocusChange(View v, boolean hasFocus)
    {
        switch (v.getId())
        {
            case R.id.my_text:
                EditText text = (EditText)v;
                Toast.makeText(this, text.getText().toString(), Toast.LENGTH_LONG);
                break;
            default:
                break;
        }
    }
}
```

FRAGMENT

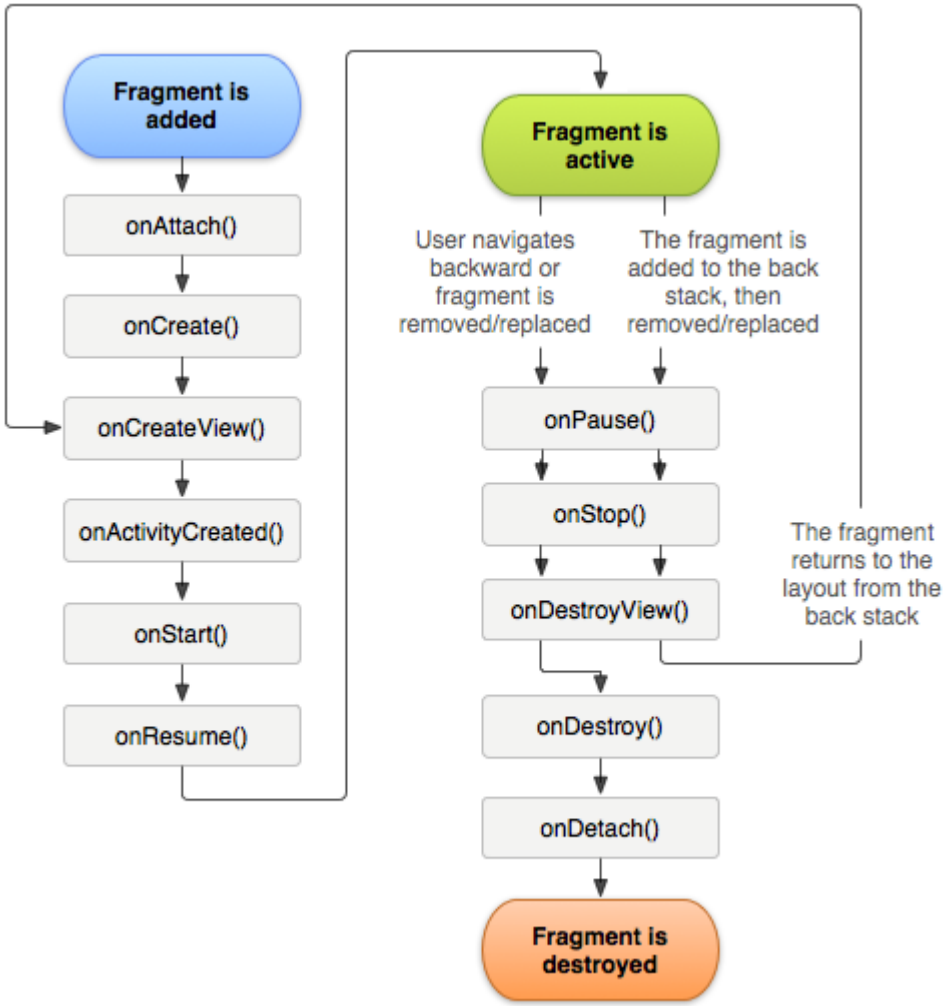
Les fragments

- Un fragment est un morceau d'activité, habituellement utilisé pour encapsuler une partie réutilisable de l'UI.
- Une activité peut combiner plusieurs fragments, qui peuvent être ajouté ou retirés dynamiquement.



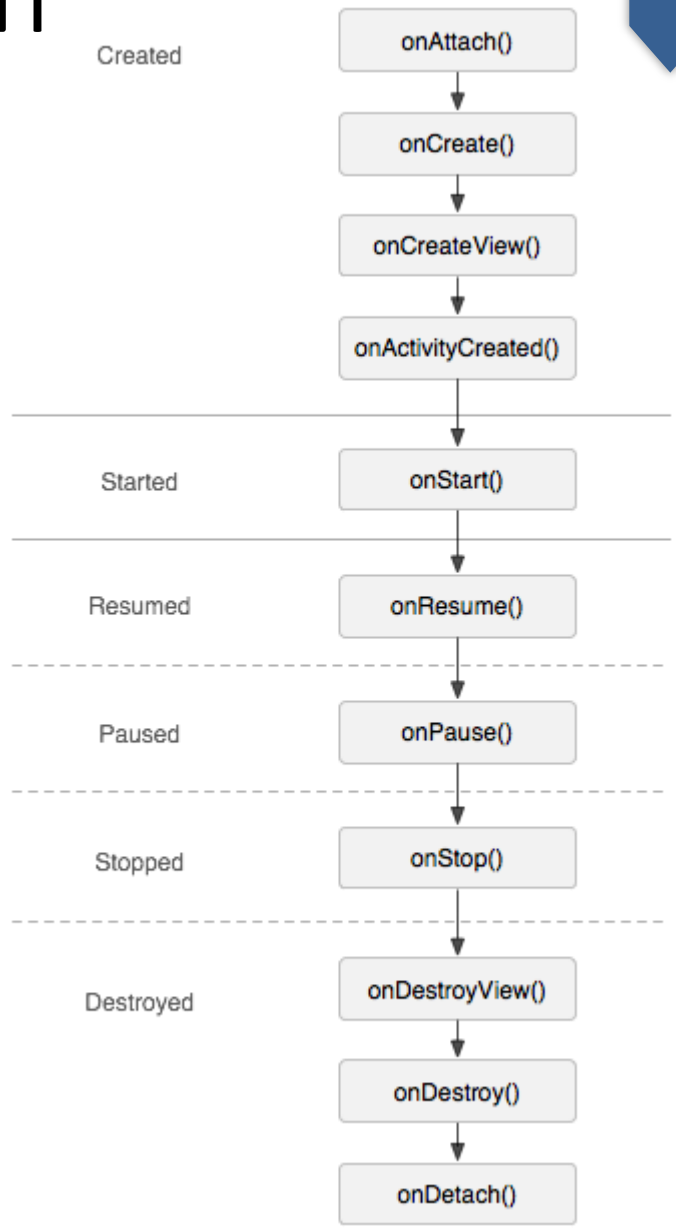
Cycle de vie d'un fragment

- Un fragment possède son propre cycle de vie, mais celui-ci est conditionné par celui de son parent (si le parent est mis en pause, ses fragments sont mis en pause aussi).
- onCreateView retourne la View représentant le fragment.
- Tout comme une activité, onSaveInstanceState() peut être implémentée.



Cycle de vie d'un fragment

- Resumed : Le fragment est affiché par l'activité au premier plan.
- Paused : Une autre activité est au premier plan, mais l'activité contenant le fragment est encore visible.
- Stopped : Le fragment n'est plus affiché, soit l'activité a été arrêtée, soit le fragment a été retiré (mais reste présent dans la back stack). Dans cet état, il sera détruit avec l'activité.



UN FRAGMENT SIMPLE

```java
public static class MyFragment extends Fragment
{
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)
    {
        Activity a = getActivity(); // return null, as the activity has not been created yet.
        return inflater.inflate(R.layout.my_fragment, container, false);
    }
}
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <fragment android:name=".MyFragment"
            android:id="@+id/my_fragment"
            android:layout_weight="1"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
</LinearLayout>
```

Le fragment manager

- Le FragmentManager gère les fragments dans l'activité.
- Les opérations sur un fragment sont gérées sous forme de transactions.

```
FragmentManager fragmentManager = getFragmentManager()
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
MyFragment fragment = new MyFragment();
fragmentTransaction.add(R.id.fragment_container, fragment);
fragmentTransaction.add(fragment, "myFragment"); // fragment without UI, findFragmentByTag()
fragmentTransaction.commit();
```

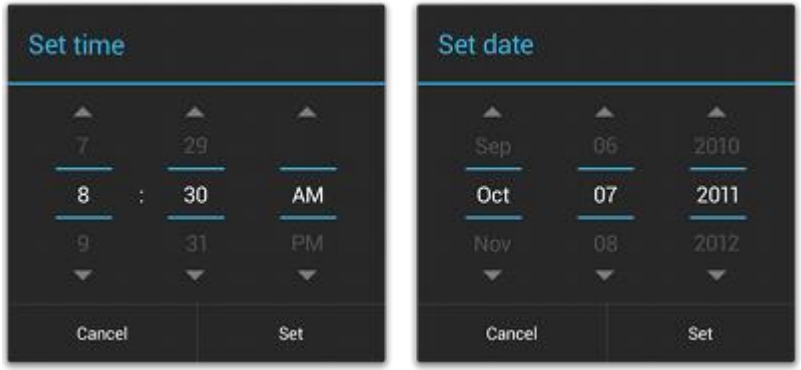
- Les fragments peuvent être ajoutés à la back stack : addBackStack(), popBackStack()
- addOnBackStackChangedListener()
- FragmentManager.OnBackStackChangedListener

```
FragmentTransaction transaction = getFragmentManager().beginTransaction();
transaction.replace(R.id.fragment_container, newFragment);
transaction.addToBackStack(null);
transaction.commit();
```

Time and Date pickers

- Les composants TimePickerDialog et DatePickerDialog sont des boîtes de dialogue pour la sélection de dates/heures.
- Si l'on désire les intégrer à une interface autrement qu'en boîte de dialogue (premier plan), les fragments sont la seule solution.

```
TimePickerDialog(
context,
callback,
initialHour,
initialMinute,
is24HourView
);

TimePickerDialog(this, this, 8, 30,
DateFormat.is24HourFormat(this));

TimePickerDialog.OnTimeSetListener
```



Time and Date pickers

```
public static class TimePickerFragment extends DialogFragment
    implements TimePickerDialog.OnTimeSetListener
{
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState)
    {
        final Calendar c = Calendar.getInstance();
        int hour = c.get(Calendar.HOUR_OF_DAY);
        int minute = c.get(Calendar.MINUTE);

        return new TimePickerDialog(getActivity(), this, hour, minute,
                DateFormat.is24HourFormat(getActivity()));
    }

    public void onTimeSet(TimePicker view, int hourOfDay, int minute)
    {
    }
}
```

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/pick_time"
    android:onClick="showTimePickerDialog" />

public void showTimePickerDialog(View v)
{
    DialogFragment newFragment = new TimePickerFragment();
    newFragment.show(getFragmentManager(), "timePicker");
}
```

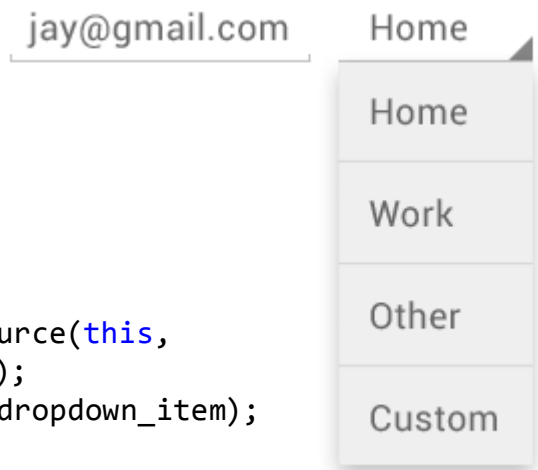
ADAPTER

Les adapters

- Un Adapter est une passerelle entre une source de données et l'interface graphique.
- Plusieurs composants nécessitent des adapters (GridView, ListView, Spinners, etc.).
- Tous les adapters implémentent l'interface Adapter :
  - BaseAdapter
  - ArrayAdapter : depuis un tableau.
  - CursorAdapter : depuis un curseur.
  - SimpleAdapter : depuis une liste de lignes

SPINNER

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




```
Spinner spinner = (Spinner)findViewById(R.id.my_spinner);
ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this,
          R.array.my_data, android.R.layout.simple_spinner_item);
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
spinner.setAdapter(adapter);
```

res/values/my_data.xml :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="my_data">
        <item>Home</item>
        <item>Work</item>
        <item>Other</item>
        <item>Custom</item>
    </string-array>
</resources>
```

- simple_spinner_item, simple_spinner_dropdown_item : layouts par défaut pour les spinners.

SPINNER

```
public class SpinnerActivity extends Activity implements OnItemSelectedListener
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        Spinner spinner = (Spinner)findViewById(R.id.my_spinner);
        ...
        spinner.setAdapter(adapter);
        spinner.setOnItemSelectedListener(this);
    }

    public void onItemSelected(AdapterView<?> parent, View view, int pos, long id)
    {
        parent.getItemAtPosition(pos);
        parent.getSelectedItem();
    }

    public void onNothingSelected(AdapterView<?> parent)
    {
    }
}
```

Un adapter personnalisé

```
public class MyAdapter extends BaseAdapter
{
     private List<User> users = Collections.emptyList();
     private final Context context;

     public MyAdapter(Context context)
     {
          this.context = context;
     }
     public void updateUsers(List<User> users)
     {
          checkOnMainThread();
          this.users = users;
          notifyDataSetChanged();
     }

     @Override
     public int getCount()
     {
          return users.size();
     }
     @Override
     public User getItem(int position)
     {
          return users.get(position);
     }
     @Override
     public long getItemId(int position)
     {
          return position;
     }
     @Override
     public View getView(int position, View convertView, ViewGroup parent)
     {
     }
}
```

ArrayAdapter est protégé contre les accès concurrents. On peut toutefois forcer l'utilisation d'un Adapter uniquement dans l'UI Thread pour s'économiser le coût des synchronisation.

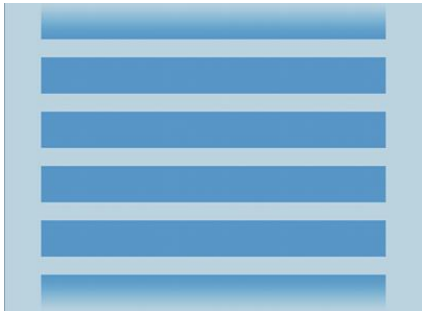
```
public void checkOnMainThread()
{
     if(Thread.currentThread() != Looper.getMainLooper().getThread())
          throw new IllegalStateException("Wrong thread");
}
```

Un adapter personnalisé

```
public View getView(int position, View convertView, ViewGroup parent)
{
    User user = getItem(position);
    TextView myView;

    if(convertView == null)
    {
        // Solution 1
        myView = new TextView(context);
        // Solution 2
        myView = (TextView)LayoutInflater.from(context).inflate(R.layout.line, parent, false);
    }
    else
        myView = (TextView)convertView;

    myView.setText(user.getFirstName() + " " + user.getLastName());
    return myView;
}
```

- Certains composants gèrent un pool d'éléments affichés à l'écran et les réutilisent plutôt que de construire continuellement des éléments. Ces éléments sont transmis à l'adapter qui peut les utiliser ou non.

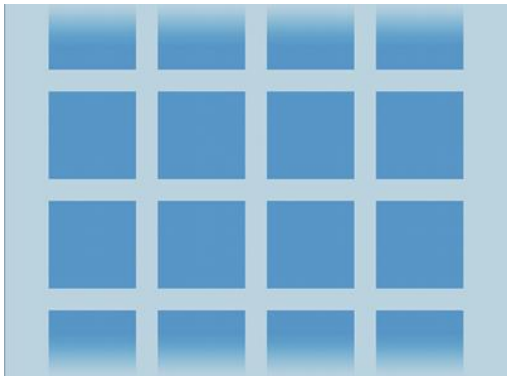
List view



```
<ListView
        android:id="@+id/my_list"
        android:dividerHeight="1dp"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
    </ListView>
```

```
public class MyActivity extends Activity implements OnItemClickListener
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        MyAdapter adapter = new MyAdapter();
        ListView list = (ListView)findViewById(R.id.my_list);
        list.setAdapter(adapter);
        list.setOnItemClickListener(this);
    }

    public void onItemClick(AdapterView<?> parent, View view, int pos, long id)
    {
        parent.getItemAtPosition(pos);
    }
}
```

GRID VIEW : GALERIE D'IMAGES



```
<GridView
    android:id="@+id/my_grid"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:columnWidth="90dp"
    android:numColumns="auto_fit"
    android:verticalSpacing="10dp"
    android:horizontalSpacing="10dp"
    android:stretchMode="columnWidth"
    android:gravity="center"
/>
```

```
public class MyActivity extends Activity implements OnItemClickListener
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        ImageAdapter adapter = new ImageAdapter ();
        GridView grid = (GridView)findViewById(R.id.my_grid);
        grid.setAdapter(adapter);
        grid.setOnItemClickListener(this);
    }

    public void onItemClick(AdapterView<?> parent, View view, int pos, long id)
    {
        parent.getItemAtPosition(pos);
    }
}
```

GRID VIEW : GALERIE D'IMAGES

```
public class ImageAdapter extends BaseAdapter
{
     private List<Bitmap> images = new ArrayList<Bitmap>();
     private final Context context;

     public MyAdapter(Context context)
     {
          this.context = context;
     }
     public void addImage(Bitmap image)
     {
          checkOnMainThread();
          images.add(image);
          notifyDataSetChanged();
     }

     @Override
     public int getCount()
     {
          return images.size();
     }
     @Override
     public Bitmap getItem(int position)
     {
          return images.get(position);
     }
     @Override
     public long getItemId(int position)
     {
          return position;
     }
}
```

GRID VIEW : GALERIE D'IMAGES

```
@Override
public View getView(int position, View convertView, ViewGroup parent)
{
    ImageView imageView;
    if (convertView == null)
    {
        imageView = new ImageView(context);
        imageView.setLayoutParams(new ViewGroup.LayoutParams(85, 85));
        imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
        imageView.setPadding(8, 8, 8, 8);
    }
    else
        imageView = (ImageView) convertView;

    imageView.setImageBitmap(getItem(position));
    return imageView;
}
```

Le ViewHolder pattern

- On peut être tenté d'utiliser des parent.findViewById() pour remplir un layout dans getView().
- Cependant, la recherche dans le layout pour l'affichage de chaque item est coûteuse et peut ralentir le scrolling (saccades).
- Le pattern ViewHolder permet de conserver des références sur les composants au sein du parent, évitant alors de les rechercher continuellement.
- Deux implémentations possibles :
  - Après l'API15 : référencer directement les composants avec un simple appel de méthode.
  - De manière générale, créer son propre composant ViewGroup pour gérer les items.

Le ViewHolder pattern : méthode 1

```
@Override
public View getView(int position, View convertView, ViewGroup parent)
{
    ImageView avatarView;
    TextView nickView;
    if (convertView == null)
    {
        convertView = LayoutInflater.from(context).inflate(R.layout.line, parent, false);
        avatarView = (ImageView) convertView.findViewById(R.id.avatar);
        nickView = (TextView) convertView.findViewById(R.id.nick);
        convertView.setTag(R.id.avatar, avatarView);
        convertView.setTag(R.id.nick, nickView);
    }
    else
    {
        avatarView = (ImageView) convertView.getTag(R.id.avatar);
        nickView = (TextView) convertView.getTag(R.id.nick);
    }

    ...
}
```

- Les tags sont stockés dans un SparseArray, une structure de tableau sans continuité d'index.
- Avant l'API15, les tags sont gérés différemment et des fuites de mémoire peuvent survenir.

Le ViewHolder pattern : méthode 2A

```
public class MyViewGroup extends LinearLayout
{
    private TextView nickView;
    private ImageView avatarView;

    public MyViewGroup(Context context)
    {
        super(context);
        nickView = new TextView(context);
        avatarView = new ImageView(context);
        ...
        addView(nickView);
        addView(avatarView);
    }

    ...

    public void update(Bitmap avatar, String nick)
    {
        nickView.setText(nick);
        avatarView.setImageBitmap(avatar);
    }
}
```

Le ViewHolder pattern : méthode 2A

```
@Override
public View getView(int position, View convertView, ViewGroup parent)
{
     MyViewGroup view;
     if (convertView == null)
          view = new MyViewGroup(context);
     else
          view = (MyViewGroup)convertView;

     ...
     view.update(aBitmap, aString);
     return view;
}
```

Le ViewHolder pattern : méthode 2B

```java
public class MyViewGroup extends LinearLayout
{
    private TextView nickView;
    private ImageView avatarView;

    public MyViewGroup(Context context)
    {
        super(context);
    }
    public MyViewGroup(Context context, AttributeSet attrs)
    {
        super(context, attrs, defStyle);
    }
    public MyViewGroup(Context context, AttributeSet attrs, int defStyle)
    {
        super(context, attrs, defStyle);
    }

    @Override
    protected void onFinishedInflate()
    {
        nickView = (TextView) findViewById(R.id.nick);
        avatarView = (ImageView) findViewById(R.id.avatar);
    }

    public void update(Bitmap avatar, String nick)
    {
        nickView.setText(nick);
        avatarView.setImageBitmap(avatar);
    }
}
```

```xml
<fr.inria.MyViewGroup
  android:orientation="horizontal"
  ... >
      <TextView
       android:id="@+id/nick"
       ... />
      <ImageView
       android:id="@+id/avatar"
       ... />
</fr.inria.MyViewGroup>
```

Le ViewHolder pattern : méthode 2B

```
@Override
public View getView(int position, View convertView, ViewGroup parent)
{
     MyViewGroup view;
     if (convertView == null)
          view = (MyViewGroup)LayoutInflater.from(context).inflate(R.layout.line, parent, false);
     else
          view = (MyViewGroup)convertView;

     ...
     view.update(aBitmap, aString);
     return view;
}
```

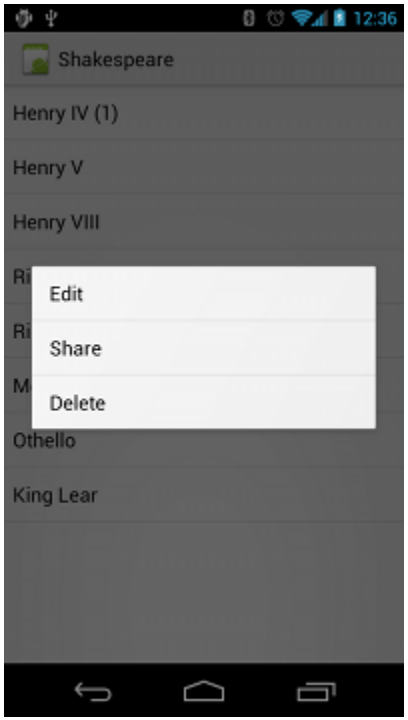
AUTRES COMPOSANTS

Contextual Menu

res/menu/my_menu.xml :

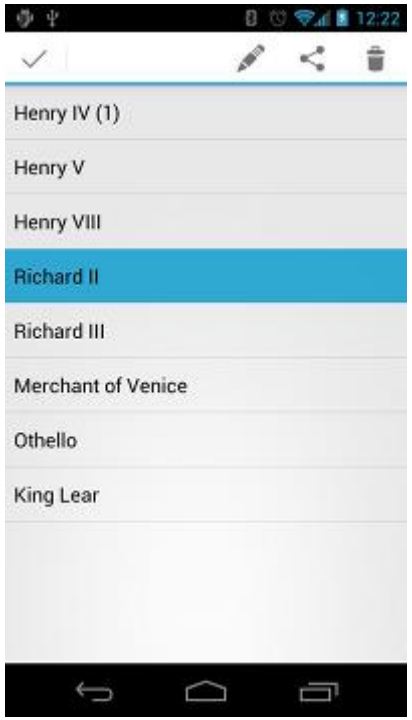
```xml
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/note" android:title="@string/note" >
        <menu> <!-- submenu -->
            <item android:id="@+id/edit" android:title="@string/edit" />
            <item android:id="@+id/delete" android:title="@string/delete" />
        </menu>
    </item>
</menu>
```

```
@Override
protected void onCreate(Bundle savedInstanceState)
{
     super.onCreate(savedInstanceState);
     ...
     registerForContextMenu(myView);
}
@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo info)
{
     super.onCreateContextMenu(menu, v, info);
     getMenuInflater().inflate(R.menu.my_menu, menu);
}
@Override
public boolean onContextItemSelected(MenuItem i)
{
     AdapterContextMenuInfo info = (AdapterContextMenuInfo)i.getMenuInfo();
     switch (i.getItemId())
     {
          case R.id.edit:
               editNote(info.id);
               return true;
          default:
               return super.onContextItemSelected(i);
     }
}
```



Contextual Bar

```
private ActionMode currentMode = null;
@Override
protected void onCreate(Bundle savedInstanceState)
{
      int layoutId = android.R.layout.simple_list_item_activated_1;
      myListView.setAdapter(new ArrayAdapter<String>(this, layoutId , new String[] { ... }));
      myListView.setChoiceMode(ListView.CHOICE_MODE_SINGLE);
      myListView.setOnItemLongClickListener(new AdapterView.OnItemLongClickListener()
      {
            @Override
            public boolean onItemLongClick(AdapterView<?> parent, View view, int position, long id)
            {
                  if (currentMode != null)
                        return false;
                  currentMode = MainActivity.this.startActionMode(actionModeCallback);
                  view.setSelected(true);
                  return true;
            }
      });
}
```



Contextual Bar

```
private ActionMode.Callback actionModeCallback = new ActionMode.Callback()
{
	@Override
	public boolean onCreateActionMode(ActionMode mode, Menu menu)
	{
		mode.getMenuInflater().inflate(R.menu.my_menu, menu);
		return true;
	}

	@Override
	public boolean onPrepareActionMode(ActionMode mode, Menu menu)
	{
		return false;
	}

	@Override
	public boolean onActionItemClicked(ActionMode mode, MenuItem item) // called when shown
	{
		switch (item.getItemId())
		{
			case R.id.edit:
				mode.finish(); // action picked, so close the bar
				return true;
			default:
				return false;
		}
	}

	@Override
	public void onDestroyActionMode(ActionMode mode)
	{
		currentMode = null;
	}
};
```

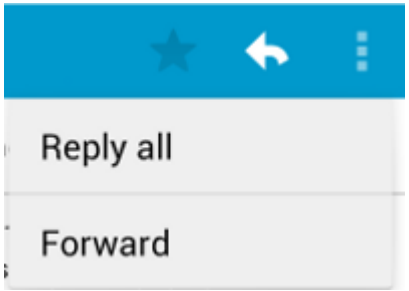
Contextual Bar (batch)

```
private ActionMode currentMode = null;
@Override
protected void onCreate(Bundle savedInstanceState)
{
     int layoutId = android.R.layout.simple_list_item_activated_1;
     myListView.setAdapter(new ArrayAdapter<String>(this, layoutId , new String[] { ... }));
     myListView.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE_MODAL);
     myListView.setMultiChoiceModeListener(multiModeCallback);
}
```

Popup Menu

```
<ImageButton android:id="@+id/image" android:src="@drawable/something" android:onClick="showPopup" />
```

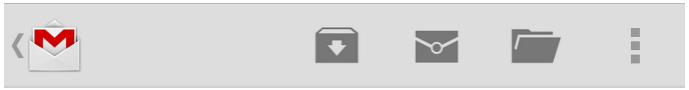
```
public void showPopup(View v)
{
    PopupMenu popup = new PopupMenu(this, v);
    popup.setOnMenuItemClickListener(myListener);
    MenuInflater inflater = popup.getMenuInflater();
    inflater.inflate(R.menu.my_menu, popup.getMenu());
    popup.show();
}

private PopupMenu.OnMenuItemClickListener myListener = new PopupMenu.OnMenuItemClickListener()
{
    @Override
    public boolean onMenuItemClick(MenuItem item)
    {
        switch (item.getItemId())
        {
            ...
        }
    }
}
```



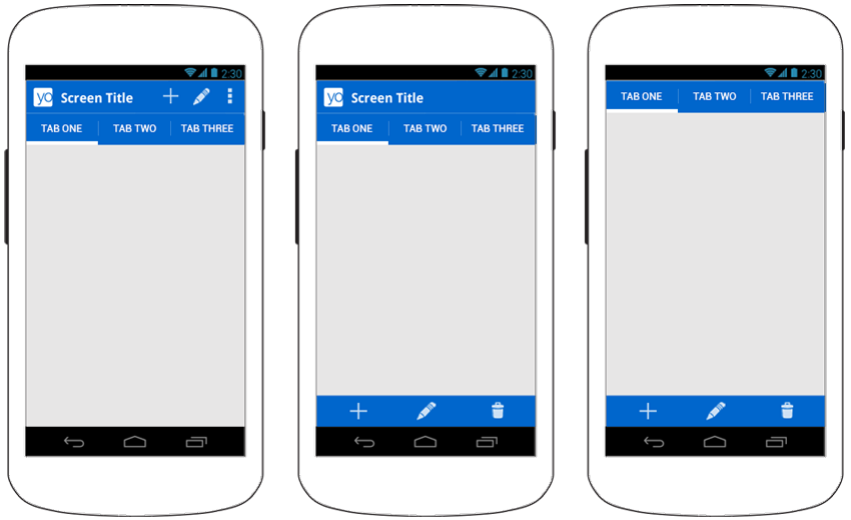
ACTION BAR

```
public class MyActivity extends Activity
{
    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {
        getMenuInflater().inflate(R.menu.main, menu);
        return super.onCreateOptionsMenu(menu);
    }

    // getActionBar().hide();

    @Override
    public boolean onOptionsItemSelected(MenuItem i)
    {
        switch(i.getItemId())
        {
            ...
            default:
                return super.onOptionsItemSelected(i);
        }
    }
}
```

```
<manifest ...>
    <activity uiOptions="splitActionBarWhenNarrow" ... >
        ...
    </activity>
</manifest>
```



```
<menu ...>
    <item android:id="@+id/action_search"
          android:icon="@drawable/something1"
          android:title="@string/action_search"
          android:showAsAction="always" />
    <item android:id="@+id/action_compose"
          android:icon="@ drawable/something2"
          android:title="@string/action_compose"
          android:showAsAction="ifRoom|withText" />
    <item android:id="@+id/action_clear"
          android:icon="@ drawable/something3"
          android:title="@string/action_compose"
          android:showAsAction="never" />
</menu>
```



Action View

- Une action view remplace un bouton lorsque celui-ci est activé.

```
@Override
public boolean onCreateOptionsMenu(Menu menu)
{
    getMenuInflater().inflate(R.menu.main, menu);
    MenuItem i = menu.findItem(R.id.action_search);
    SearchView view = (SearchView)i.getActionView();
    ...
    return super.onCreateOptionsMenu(menu);
}
```

```
<item android:id="@+id/action_search"
    android:icon="@drawable/something1"
    android:title="@string/action_search"
    android:actionViewClass="android.widget.SearchView"
    android:showAsAction="always" />
```

```
i.setOnActionExpandListener(new OnActionExpandListener()
{
    @Override
    public boolean onMenuItemActionCollapse(MenuItem i)
    {
        return true;
    }
    @Override
    public boolean onMenuItemActionExpand(MenuItem i)
    {
        return true;
    }
});
```

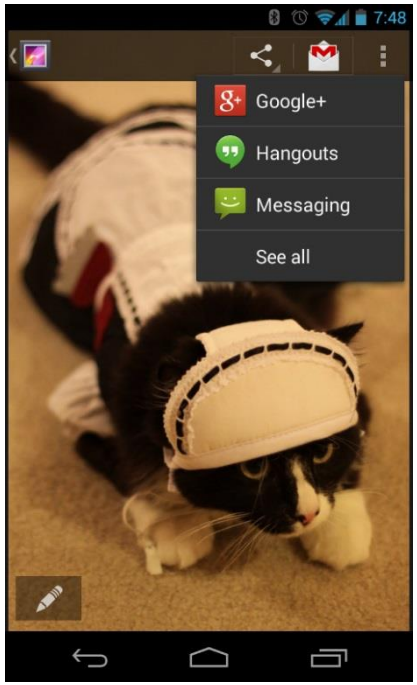
Action Provider

- Un action provider est similaire à une action view, mais gère ses propres callbacks.

```
<item android:id="@+id/action_share"
    android:icon="@drawable/something1"
    android:title="@string/action_share"
    android:actionProviderClass="android.widget.ShareActionProvider"
    android:showAsAction="always" />
```

```
@Override
public boolean onCreateOptionsMenu(Menu menu)
{
    getMenuInflater().inflate(R.menu.main, menu);
    MenuItem i = menu.findItem(R.id.action_share);
    ShareActionProvider provider = (ShareActionProvider)i.getActionProvider();
    ...
    return super.onCreateOptionsMenu(menu);
}
```

```
// anywhere in the code
Intent i = new Intent(Intent.ACTION_SEND);
i.setData(Uri.fromFile(new File("/sdcard/cat_maid.jpg")));
setShareIntent(i);
```



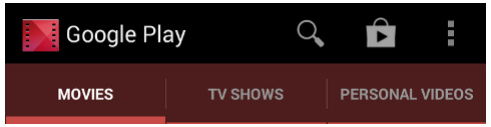
Action Provider

```
public class MyProvider extends ActionProvider
{
    private Context context;
    public MyProvider(Context context)
    {
        this.context = context;
    }

    @Override
    public View onCreateActionView(MenuItem i)
    {
        View v = LayoutInflater.from(context).inflate(R.layout.action_provider, null);

        // remember : view holder is better ;)
        ImageButton button = (ImageButton)v.findViewById(R.id.button);

        button.setOnClickListener(new View.OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                ...
            }
        });
        return v;
    }
} s
```

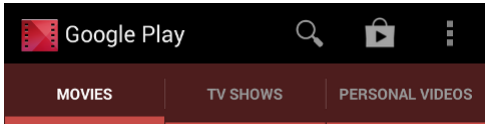
Navigation Tab

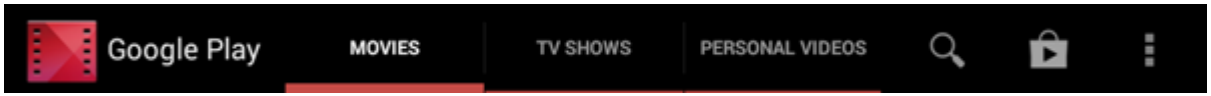
```
public class MyTabListener<T extends Fragment> implements ActionBar.TabListener
{
     private Fragment fragment;
     private final Activity activity;
     private final Class<T> fragmentClass;

     public MyTabListener(Activity activity, Class<T> fragmentClass)
     {
          this.activity = activity;
          this.fragmentClass = fragmentClass;
     }
     @Override
     public void onTabSelected(Tab tab, FragmentTransaction ft)
     {
          if (fragment == null)
          {
               fragment = Fragment.instantiate(activity, fragmentClass.getName());
               ft.add(android.R.id.content, fragment);
          }
          else
               ft.attach(fragment);
     }
     @Override
     public void onTabUnselected(Tab tab, FragmentTransaction ft)
     {
          if (fragment != null)
               ft.detach(fragment);
     }
     @Override
     public void onTabReselected(Tab tab, FragmentTransaction ft)
     {
     }
}
```

Attention : pas de commit ou de backstack.



NAVIGATION TAB

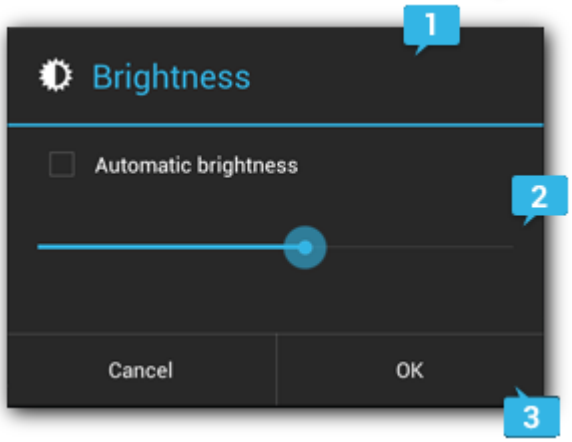





```
@Override
protected void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    ...
    ActionBar actionBar = getActionBar();
    actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);
    actionBar.setDisplayShowTitleEnabled(false);

    Tab tab = actionBar.newTab()
            .setText("Test1")
            .setTabListener(new MyTabListener<Fragment1>(this, Fragment1.class));
    actionBar.addTab(tab);

    tab = actionBar.newTab()
            .setText("Test2")
            .setTabListener(new MyTabListener<Fragment2>(this, Fragment2.class));
    actionBar.addTab(tab);
}
```

Alert Dialog

```
public class MyDialog extends DialogFragment
{
	@Override
	public Dialog onCreateDialog(Bundle savedInstanceState)
	{
		AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
		builder.setMessage(R.string.dialog_message);
		builder.setTitle(R.string.dialog_title);

		builder.setPositiveButton(R.string.ok, new DialogInterface.OnClickListener()
		{
			@Override
			public void onClick(DialogInterface dialog, int id)
			{
			// user clicked OK button
			}
		});
		builder.setNegativeButton(R.string.cancel, new DialogInterface.OnClickListener()
		{
			@Override
			public void onClick(DialogInterface dialog, int id)
			{
				// user cancelled the dialog
		}
		});
		builder.setNeutralButton(R.string.remind, new DialogInterface.OnClickListener()
		{
			@Override
			public void onClick(DialogInterface dialog, int id)
			{
				// user wanted a neutral action (e.g., a reminder)
			}
		});
		...
		AlertDialog dialog = builder.create();
	}
}
```

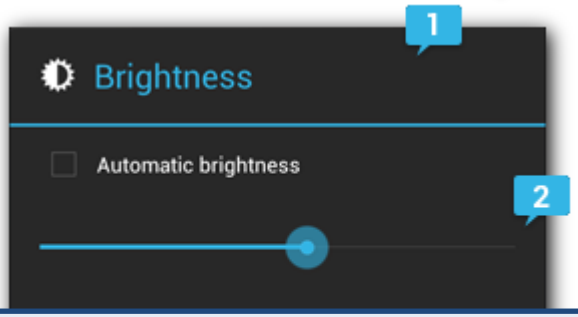


1. Title
2. Content area
3. Action buttons

L'utilisation de fragments assure la bonne gestion des évènements du cycle de vie (back, etc.) et permet éventuellement d'intégrer le composant directement dans une view.

Alert Dialog

```
public class MyDialog extends DialogFragment
{
     AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
     builder.setMessage(R.string.dialog_message);
     builder.setTitle(R.string.dialog_title);

     builder.setPositiveButton(R.string.ok, new DialogInterface.OnClickListener()
     {
          @Override
```



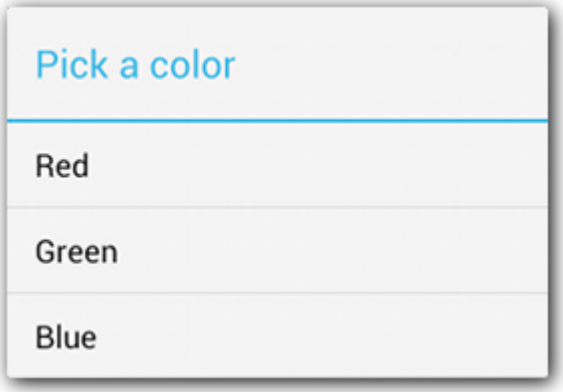

```
public void pleaseConfirm()
{
     DialogFragment newFragment = new MyDialog();
     newFragment.show(getFragmentManager(), "myDialog");
}
```

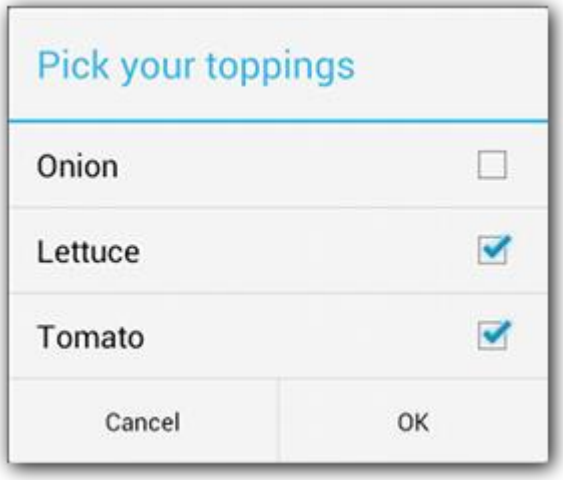
```
     });
     builder.setNeutralButton(R.string.remind, new DialogInterface.OnClickListener()
     {
          @Override
          public void onClick(DialogInterface dialog, int id)
          {
               // user wanted a neutral action (e.g., a reminder)
          }
     });
     ...
     AlertDialog dialog = builder.create();
}
```

L'utilisation de fragments assure la bonne gestion des évènements du cycle de vie (back, etc.) et permet éventuellement d'intégrer le composant directement dans une view.

Alert Dialog

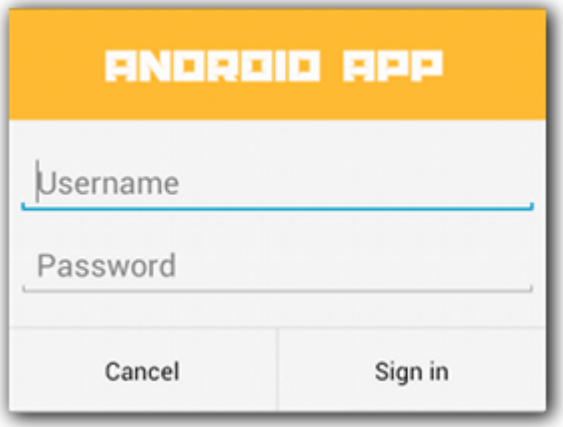
```
builder.setItems(R.array.colors, new DialogInterface.OnClickListener()
{
    @Override
    public void onClick(DialogInterface dialog, int which)
    {
    }
});

// setMultiChoiceItems() : checkboxes
// setSingleChoiceItems() : radio buttons
```





Alert Dialog avec layout

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
    <ImageView
        android:src="@drawable/header_logo"
        android:layout_width="match_parent"
        android:layout_height="64dp"
        android:scaleType="center"
        android:background="#FFFFBB33"
        android:contentDescription="@string/app_name" />
    <EditText
        android:id="@+id/username"
        android:inputType="textEmailAddress"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:layout_marginLeft="4dp"
        android:layout_marginRight="4dp"
        android:layout_marginBottom="4dp"
        android:hint="@string/username" />
    <EditText
        android:id="@+id/password"
        android:inputType="textPassword"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="4dp"
        android:layout_marginLeft="4dp"
        android:layout_marginRight="4dp"
        android:layout_marginBottom="16dp"
        android:fontFamily="sans-serif"
        android:hint="@string/password"/>
</LinearLayout>
```



Alert Dialog avec layout

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState)
{
     AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
     LayoutInflater inflater = getActivity().getLayoutInflater();

     builder.setView(inflater.inflate(R.layout.dialog_signin, null));
     builder.setPositiveButton(R.string.signin, new DialogInterface.OnClickListener()
     {
          @Override
          public void onClick(DialogInterface dialog, int id)
          {
          }
     });
     builder.setNegativeButton(R.string.cancel, new DialogInterface.OnClickListener()
     {
          public void onClick(DialogInterface dialog, int id)
          {
               getDialog().cancel();
          }
     });
     return builder.create();
}

@Override
public void onDismiss(DialogInterface dialog)
{
}

@Override
public void onCancel(DialogInterface dialog)
{
}

...
```

DRAG & DROP

Le drag & drop process

- Started :
  - Appel de startDrag() avec les données à transférer, des métadonnées et un callback pour la fabrication du drag shadow qui sera invoqué immédiatement.
  - Le système broadcaste un évènement de type ACTION_DRAG_STARTED à toutes les view intéressées (= qui retournent true).
- Continuing
  - Lorsque le drag rencontre une view, le système emet des évènements à destination de celle-ci (ACTION_DRAG_ENTERED) qui peut modifier son apparence.
- Dropped
  - L'utilisateur relâche la drag shadow à l'intérieur d'une view qui reçoit un évènement ACTION_DROP contenant les données. La vue indique qu'elle a accepté les données en retournant true.
- Ended
  - Le système broadcaste un évènement de type ACTION_DRAG_ENDED pour indiquer que le drag est terminé.

Le drag & drop process

getAction() value	Meaning
ACTION_DRAG_STARTED	L'application a invoqué startDrag();
ACTION_DRAG_ENTERED	La drag shadow est entré dans le périmètre de la vue (retourner true pour accepter).
ACTION_DRAG_LOCATION	La drag shadow s'est déplacée périmètre de la vue, après qu'un ACTION_DRAG_ENTERED ait été reçu.
ACTION_DRAG_EXITED	La drag shadow est sortie périmètre de la vue, après qu'un ACTION_DRAG_ENTERED ait été reçu ainsi qu'un ou plusieurs ACTION_DRAG_LOCATION.
ACTION_DROP	L'utilisateur a relâché la drag shadow sur la vue (retourner false pour indiquer une erreur).
ACTION_DRAG_ENDED	Le drag est terminé. Utiliser getResult() pour savoir si ACTION_DROP a été émis sur une vue (true).

getAction()	getClipDescription()	getLocalState()	getX()	getY()	getClipData()	getResult()
ACTION_DRAG_STARTED	X	X	X			
ACTION_DRAG_ENTERED	X	X	X	X		
ACTION_DRAG_LOCATION	X	X	X	X		
ACTION_DRAG_EXITED	X	X				
ACTION_DROP	X	X	X	X	X	
ACTION_DRAG_ENDED	X	X				X

getAction(), describeContents() et writeToParcel() retournent toujours des données cohérentes.

Démarrer le drag

```
final TextView view = new TextView(this);
...
view.setOnLongClickListener(new View.OnLongClickListener()
{
    @Override
    public boolean onLongClick(View v)
    {
        ClipData.Item item = new ClipData.Item(view.getText().toString());
        ClipData dragData = new ClipData("aLabel", ClipData.MIMETYPE_TEXT_PLAIN, item);
        // or ClipData.newPlainText("aLabel", view.getText().toString());

        View.DragShadowBuilder myShadow = new View.DragShadowBuilder(view);

        v.startDrag( dragData,    // the data to be dragged
                     myShadow,    // the drag shadow builder
                     null,        // no need to use local data
                     0            // flags (not currently used, set to 0)
        );
    }
}
```

- ClipData est une structure contenant une ou plusieurs instances de ClipData.Item, représentant chacun une donnée sous forme de texte, d'URI ou d'Intent, avec un type MIME associé.

La drag shadow

```
private class MyDragShadowBuilder extends View.DragShadowBuilder
{
    private Drawable shadow;
    public MyDragShadowBuilder(View v)
    {
        super(v);
        // create a gray rectangle
        shadow = new ColorDrawable(Color.LTGRAY);
    }

    @Override
    public void onProvideShadowMetrics (Point size, Point touch)
    {
        int width, height;
        width = getView().getWidth() / 2;
        height = getView().getHeight() / 2;

        shadow.setBounds(0, 0, width, height);

        size.set(width, height);
        touch.set(width / 2, height / 2);
    }

    @Override
    public void onDrawShadow(Canvas canvas)
    {
        shadow.draw(canvas);
    }
}
```

- size et touch sont des références qui seront utilisées par le système pour dessiner la shadow.
- size = la taille de la shadow.
- touch = la position du doigt par rapport au bord supérieur gauche de la shadow.

Drag Listener

- Implémenter View.OnDragEventListener
- Utiliser setOnDragListener() sur une vue pour l'associer.

```
public class listener implements View.OnDragEventListener
{
    public boolean onDrag(View v, DragEvent event)
    {
        int action = event.getAction();
        switch(action)
        {
            ...

            default:
                return false;
        }
    }
};
```

Drag Listener (started)

- Vérifier si la view accepte ce type de données et éventuellement le signifier visuellement.
- Si oui, retourner true.

```
case DragEvent.ACTION_DRAG_STARTED:
    ClipDescription d = event.getClipDescription();
    if(d.hasMimeType(ClipDescription.MIMETYPE_TEXT_PLAIN))
    {
        // change the view tint
        v.setColorFilter(Color.BLUE);
        v.invalidate();
        return true;
    }
    else
        return false;
```

Drag Listener (continuing)

```
case DragEvent.ACTION_DRAG_ENTERED:
    v.setColorFilter(Color.GREEN);
    v.invalidate();
    return true;

case DragEvent.ACTION_DRAG_LOCATION:
    return true; // ignore the event

case DragEvent.ACTION_DRAG_EXITED:
    v.setColorFilter(Color.BLUE);
    v.invalidate();
    return true;
```

Drag Listener (dropped, Ended)

- La valeur de retour pourra être accédée par getResult() lors du dernier évènement ACTION_DRAG_ENDED.

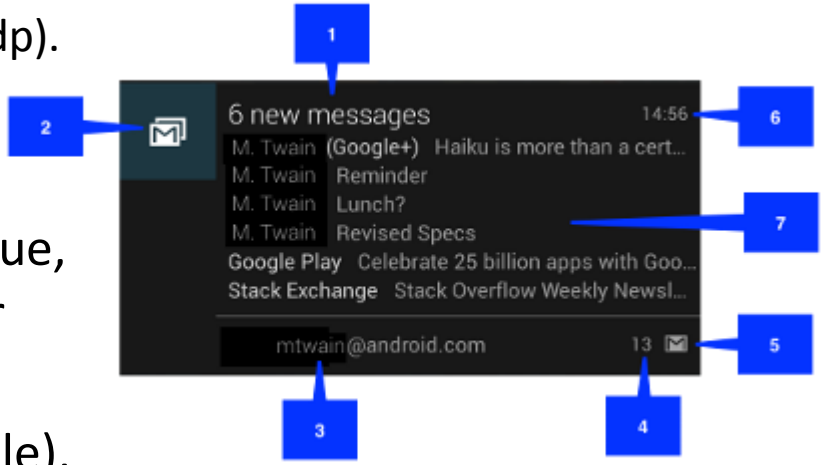
```
case DragEvent.ACTION_DROP:
    ClipData.Item item = event.getClipData().getItemAt(0);
    dragData = item.getText();

    Toast.makeText(this, "Dragged data " + dragData, Toast.LENGTH_LONG);

    v.clearColorFilter();
    v.invalidate();
    return true;

case DragEvent.ACTION_DRAG_ENDED:
    v.clearColorFilter();
    v.invalidate();
    return true;
```

NOTIFICATIONS

NOTIFICATIONS

- Normal view :
  1. Content title
  2. Large icon
  3. Content text
  4. Content info
  5. Small icon
  6. Time, ou spécifié par setWhen()

- Big view (7. Details)
  - Big picture style : une image (max 256 dp).
  - Big text style : un texte .
  - Inbox style : plusieurs lignes de texte.
- La première big view est affichée étendue, les autres sont condensées (l'utilisateur doit les déployer).
- Un titre supplémentaire (big content title), affiché quand la notification est déployée.



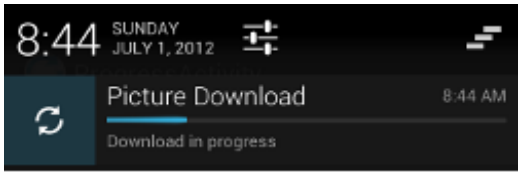
Une simple notification

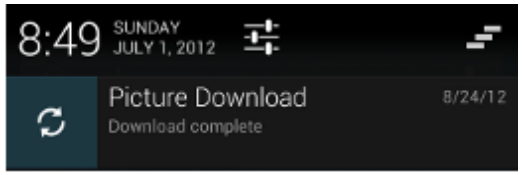
```
NotificationCompat.Builder builder = new NotificationCompat.Builder(this);
builder.setSmallIcon(R.drawable.notification_icon);
builder.setContentTitle("Event");
builder.setContentText("My event !");
builder.setNumber(1);

Intent intent = new Intent(this, ResultActivity.class);
builder.setContentIntent(intent);

NotificationManager notifMgr = (NotificationManager)getSystemService(Context.NOTIFICATION_SERVICE);
notifMgr.notify(id, builder.build());

// big view
...
NotificationCompat.InboxStyle style = new NotificationCompat.InboxStyle();
style.setBigContentTitle("Pending events:");
String[] events = new String[6];
for (int i = 0; i < events.length; i++)
{
    style.addLine(events[i]);
}
builder.setNumber(events.length);
builder.setStyle(style);
...
```

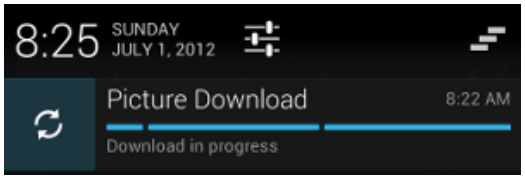
- Si une notification avec le même id existe déjà, celle-ci est mise à jour.
- Une notification est supprimée si l'utilisateur la retire ou si :
  - setAutoCancel() a été utilisé à la création (supprimée dès que l'utilisateur clique dessus).
  - cancel(id) ou cancelAll().

Progress Bar Notification

```
NotificationManager notifMgr = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
builder = new NotificationCompat.Builder(this);
builder.setContentTitle("Picture Download")
builder.setContentText("Download in progress")
builder.setSmallIcon(R.drawable.ic_notification);

new Thread(new Runnable()
{
    @Override
    public void run()
    {
        for (int i = 0; i <= 100; i += 5)
        {
            builder.setProgress(100, incr, false);
            notifMgr.notify(id, builder.build());
            try
            {
                Thread.sleep(5000);
            }
            catch (InterruptedException e) { }
        }

        builder.setContentText("Download complete");
        builder.setProgress(0,0,false);
        notifMgr.notify(ID, builder.build());
    }
}).start();
```








```
builder.setProgress(0, 0, true);
```

TOASTS

```
Context context = getApplicationContext();
CharSequence text = "Hello toast!";
int duration = Toast.LENGTH_SHORT;

Toast toast = Toast.makeText(context, text, duration);
toast.setGravity(Gravity.TOP|Gravity.LEFT, 0, 0);
toast.setDuration(Toast.LENGTH_LONG);

toast.show();
```


```
// custom layout for the toast
View toastRoot = (ViewGroup) findViewById(R.id.toast_layout_root); // standard root
View layout = inflater.inflate(R.layout.custom_toast, toastRoot);
toast.setView(layout);
```