

# MOBILE DEVELOPMENT

---

LISTBASED-WIDGETS

A solid green horizontal bar spanning the width of the slide at the bottom.

# CONTENTS

---

List-Based widgets

Spinner

GridView

AutoComplete TextView

HorizontalScrollView

Image-based Gridviews

Custom-made ListViews

Using the SD card

Appendices

- Predefined Android resources
- EditText boxes & keyboarding
- Custom list supported by a BaseAdapter

# LIST-BASED WIDGETS

## (GUI design for selection making)

---

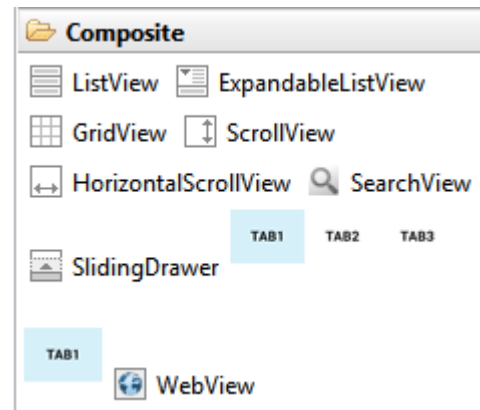
RadioButtons and CheckButtons are widgets suitable for selecting options offered by a small set of choices. They are intuitive and uncomplicated; however they occupy a permanent space on the GUI (which is not a problem when only a few of them are shown)



When the set of values to choose from is large, other Android List-Based Widgets are more appropriate.

Example of List-Based Widgets include:

- ListView
- Spinner
- GridView
- Image Gallery
- ScrollViews, etc.

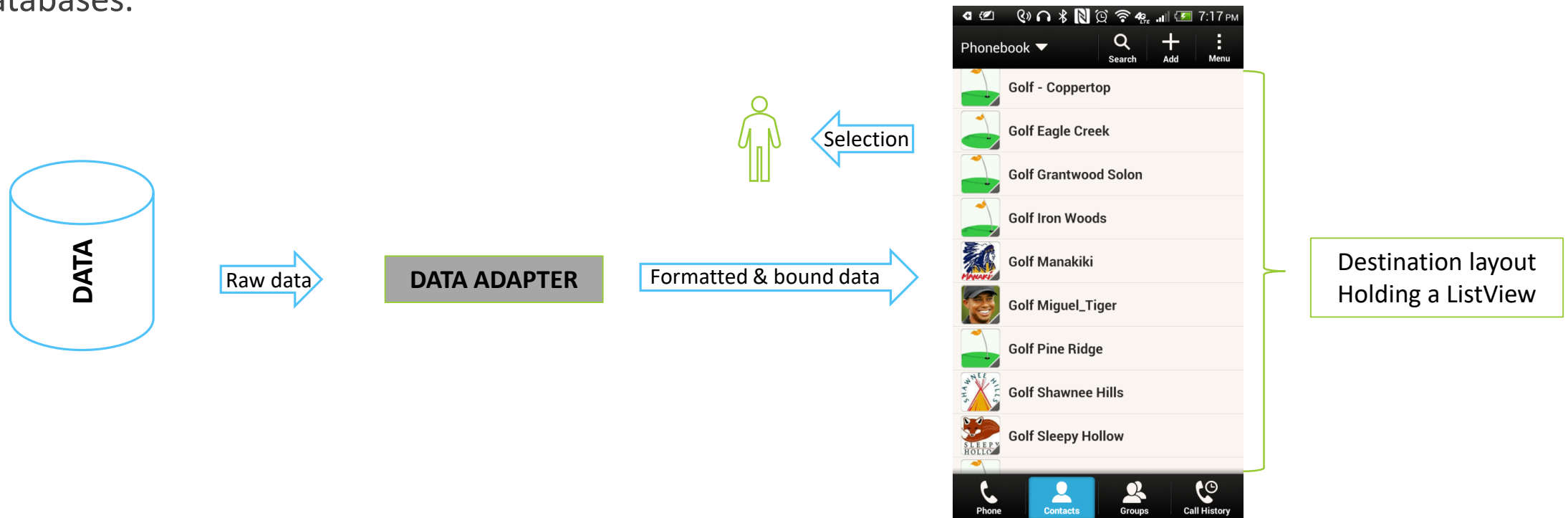


# LIST-BASED WIDGETS

## (Showing a large set of choices on the GUI)

The Android DataAdapter class is used to feed a collection of data items to a List-Based Widget.

The Adapter's raw data may come from a variety of sources, such as small arrays as well as large databases.



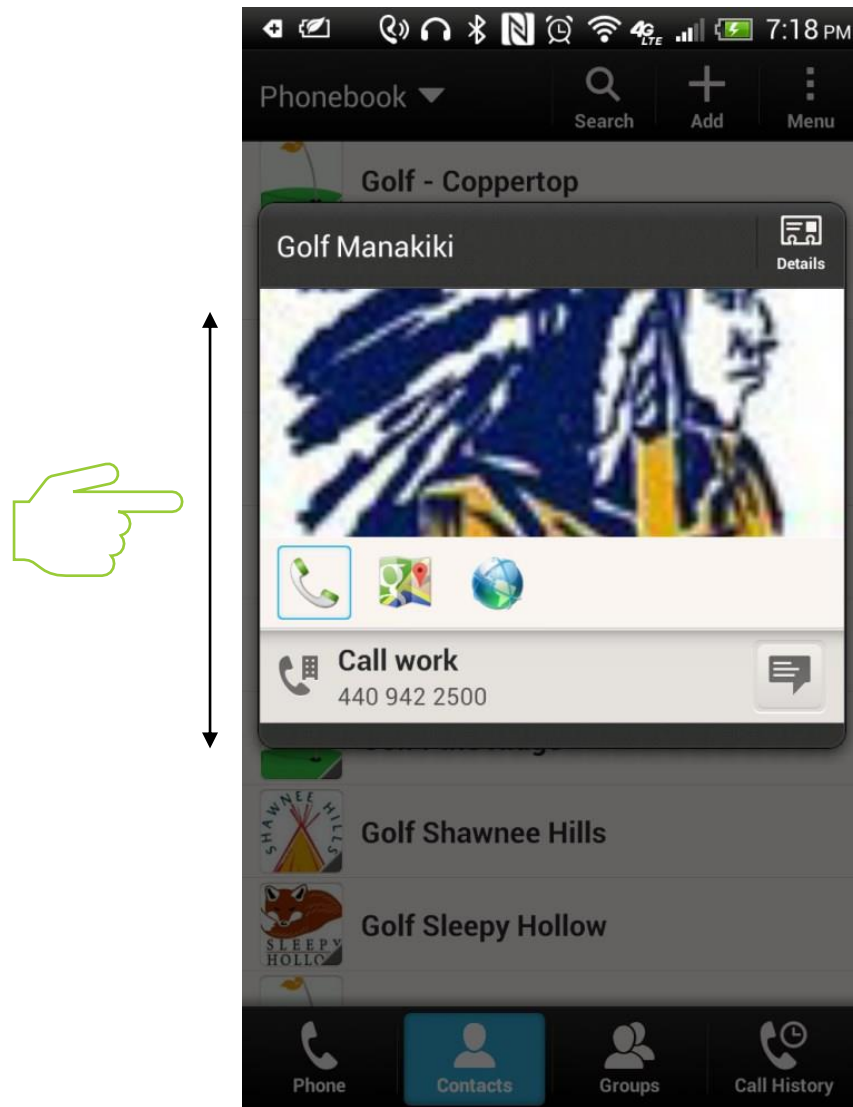
# LIST-BASED WIDGETS (ListView)

The Android ListView widget is the most common element used to display data supplied by a data adapter.

Lists are scrollable, each item from the base data set can be shown in an individual row.

Users can tap on a row to make a selection.

A row could display one or more lines of text as well as images.



Destination layout  
Holding a ListView

# LIST-BASED WIDGETS

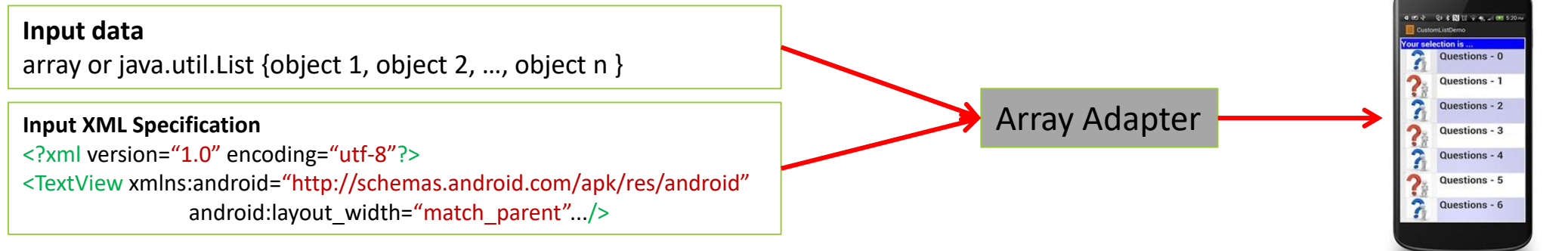
## (ListView - ArrayAdapter)

An `ArrayAdapter<T>` accepts for input an array (or `ArrayList`) of objects of some arbitrary type `T`.

The adapter works on each object by (a) applying its `toString()` method, and (b) moving its formatted output string to a `TextView`.

The formatting operation is guided by a user supplied XML layout specification which defines the appearance of the receiving `TextView`.

For `ListView`s showing complex arrangement of visual elements –such as text plus images- you need to provide a custom made adapter in which the `getView(...)` method explains how to manage the placement of each data fragment in the complex layout.



# LIST-BASED WIDGETS

(Using the `ArrayAdapter<String>` class)

### Parameters:

- The current activity's context ([this](#))
- The TextView layout indicating how an individual row should be written ( `android.R.id.simple_list_item_1`)
- The actual data source (Array or Java.List containing items to be shown)

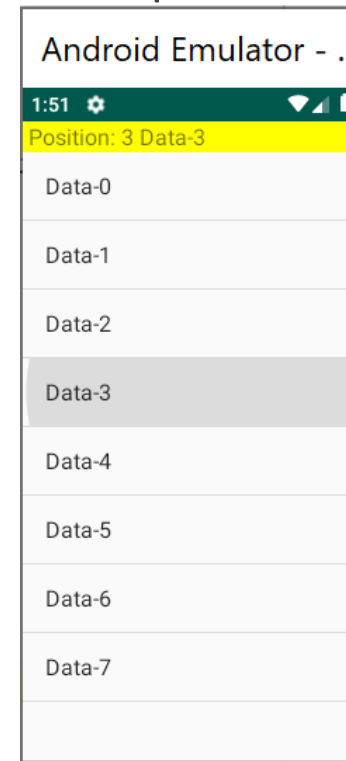
Code:

[illegible]

# LIST-BASED WIDGETS

## (Using ListActivity + ArrayAdapter)

Assume a large collection of input data items is held in a `String[]` array. Each row of the `ListView` must show a line of text taken from the array. In our example, when the user makes a selection, you must display on a `TextView` the selected item and its position in the list.



Selection seen  
by the listener

Background flashes  
blue to acknowledge  
the users's selection



# LIST-BASED WIDGETS

## (Using ListActivity + ArrayAdapter)

Layout:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android" xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <TextView android:id="@+id/txtMsg" android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#ffffff00"
        android:text="Using ListViews..."
        android:textSize="16sp"/>
    <ListView android:id="@android:id/list" android:layout_width="match_parent" android:layout_height="match_parent"/>
    <TextView android:id="@android:id/empty" android:layout_width="match_parent" android:layout_height="wrap_content"
        android:background="#ffff0000"
        android:text="empty list"/>
</LinearLayout>
```

Android's built-in list layout

Used for empty lists

Pay attention to the use of predefined Android components:  
@android:id/list  
@android:id/empty  
See Appendix A for a description of @android:id/list

# LIST-BASED WIDGETS

## (Using ListActivity + ArrayAdapter)

### MainActivity using ListActivity

```
public class MainActivity extends ListActivity {
    TextView txtMsg;
    String[] items = {"Data-0", "Data-1", "Data-2", "Data-3", "Data-4", "Data-5", "Data-6", "Data-7" };
    //String[] items = {};
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        setListAdapter(new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, items));
        //getListView().setBackgroundColor(Color.GRAY); //try this idea later
        txtMsg = (TextView) findViewById(R.id.txtMsg);
    }
    @Override
    protected void onItemClick(ListView l, View v, int position, long id) {
        super.onItemClick(l, v, position, id);
        String text = " Position: " + position + " " + items[position];
        txtMsg.setText(text);
    }
}
```

#### CAUTION:

A ListActivity is not a “plain” Activity. It is bound to a built-in ListView called @android:id/list

DATA  
SOURCE

LIST  
ADAPTER

LIST CLICK  
LISTENER

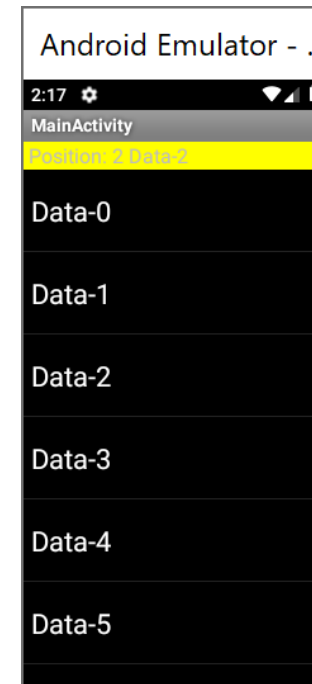
# LIST-BASED WIDGETS

## (Using ListActivity + ArrayAdapter)

---

An experiment:

- Open the AndroidManifest.xml file. Under the <Application> tag look for the clause android:theme="@style/AppTheme"
- Change the previous line to the following value android:theme="@android:style/Theme.Black"
- Try some of the other styles, such as:
  - Theme.DeviceDefault
  - Theme.Dialog
  - Theme.Holo
  - Theme.Light
  - Theme.Panel
  - Theme.Translucent
  - Theme.Wallpaper
  - etc.





# LIST-BASED WIDGETS

## (Using ListActivity + ArrayAdapter)

---

Another experiment:

- Open the AndroidManifest.xml file. Under the `<Application>` tag look for the clause `android:theme="@style/AppTheme"`
- Now open the res/values/styles folder. Look for the entry `<style name="AppTheme" parent="android:Theme.Light"/>` which indicates to use the "Light" theme (white background instead of black).
- Remove from the manifest the entry `android:theme`.
- Remove from the `onCreate` method the comment of statement: `getListView().setBackgroundColor(Color.GRAY);`
- Run the application again. Observe its new look.

# LIST-BASED WIDGETS

## (Using Activity + ArrayAdapter)

---

You may use a common Activity class instead of a ListActivity.

Layout below uses a ListView identified as `@+id/my_list` (instead of `@android:id/list` used in the previous example)

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <TextView android:id="@+id/txtMsg"
        android:layout_width="match_parent" android:layout_height="wrap_content"
        android:background="#ffffff00" android:text="Using ListViews..." android:textSize="16sp"/>
    <ListView android:id="@+id/my_list" android:layout_width="match_parent" android:layout_height="match_parent"/>
</LinearLayout>
```

# LIST-BASED WIDGETS

## (Using Activity + ArrayAdapter)

---

Instead of using a ListActivity (as we did on the previous example) we now employ a regular Android Activity. Observe that you must 'wired-up' the ListView to a Java proxy, and later bind it to an Adapter.

```
public class ListViewDemo2 extends Activity {
    String[] items = { "Data-0", "Data-1", "Data-2", "Data-3", "Data-4", "Data-5", "Data-6", "Data-7" };
    ListView myListView; TextView txtMsg;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        myListView = (ListView) findViewById(R.id.my_list);
        myListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> av, View v, int position, long id) {
                txtMsg.setText("Position: " + position + "\nData: " + items[position]);
            }
        });
        ArrayAdapter<String> aa = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, // R.layout.my_text, //try this later...
                                                         items);

        myListView.setAdapter(aa);
        txtMsg = (TextView) findViewById(R.id.txtMsg);
    } //onCreate
}
```

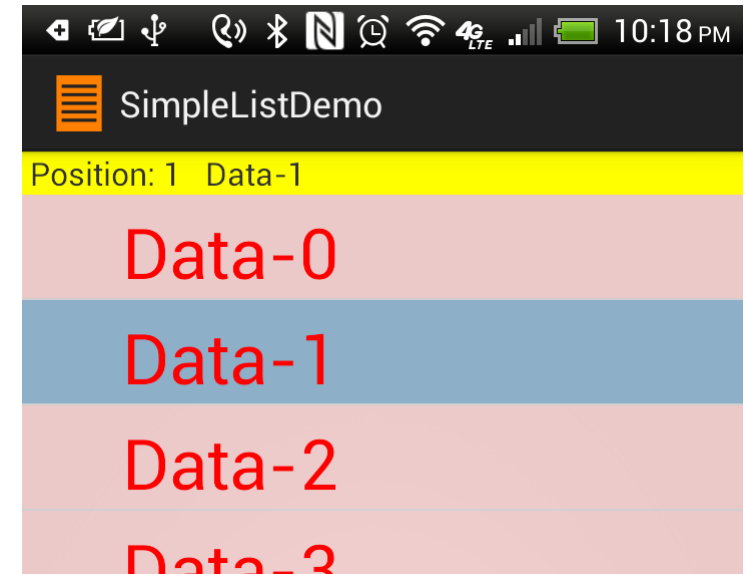
# LIST-BASED WIDGETS

## (Using Activity + ArrayAdapter)

You may want to modify the ListView control to use your own GUI design. For instance, you may replace `android.R.layout.simple_list_item_1` with `R.layout.my_custom_text`.

Where `my_custom_text` is the Layout specification listed below (held in the `res/layout` folder). It defines how each row is to be shown.

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="2dp"
    android:paddingTop="5dp"
    android:padding="5dp"
    android:textColor="#ffff0000"
    android:background="#22ff0000"
    android:textSize="35sp"/>
```



# LIST-BASED WIDGETS

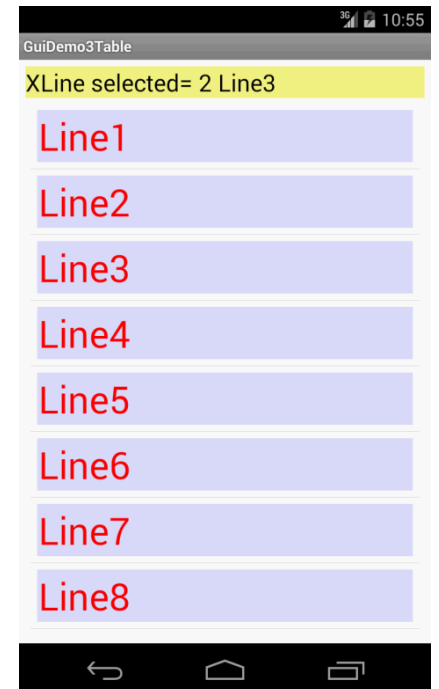
## (Using Activity + ArrayAdapter)

You may also create ArrayAdapter with more parameters. For instance, the following statement:

```
ArrayAdapter<String> adapter = new ArrayAdapter<String>(getApplication(), R.layout.my_custom_line3, R.id.my_custom_textview3, data );
```

Defines a custom list and textview layout to show the contents of the data array.

```
<!-- my_custom_line3 -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="6dp" >
    <TextView android:id="@+id/my_custom_textview3"
        android:layout_width="match_parent" android:layout_height="wrap_content"
        android:background="#220000ff" android:padding="1dp"
        android:textColor="#ffff0000" android:textSize="35sp" />
</LinearLayout>
```





# SPINNER

---

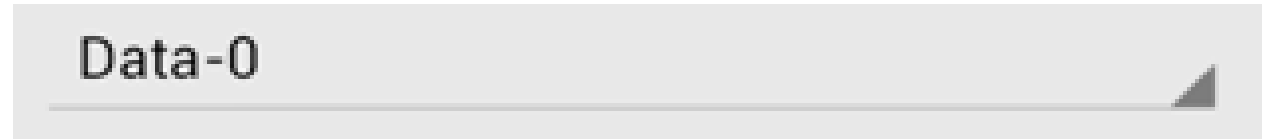
Android's Spinner is equivalent to a drop-down selector.

Spinners have the same functionality of a ListView but take less screen space.

An Adapter is used to supply its data using `setAdapter(...)`

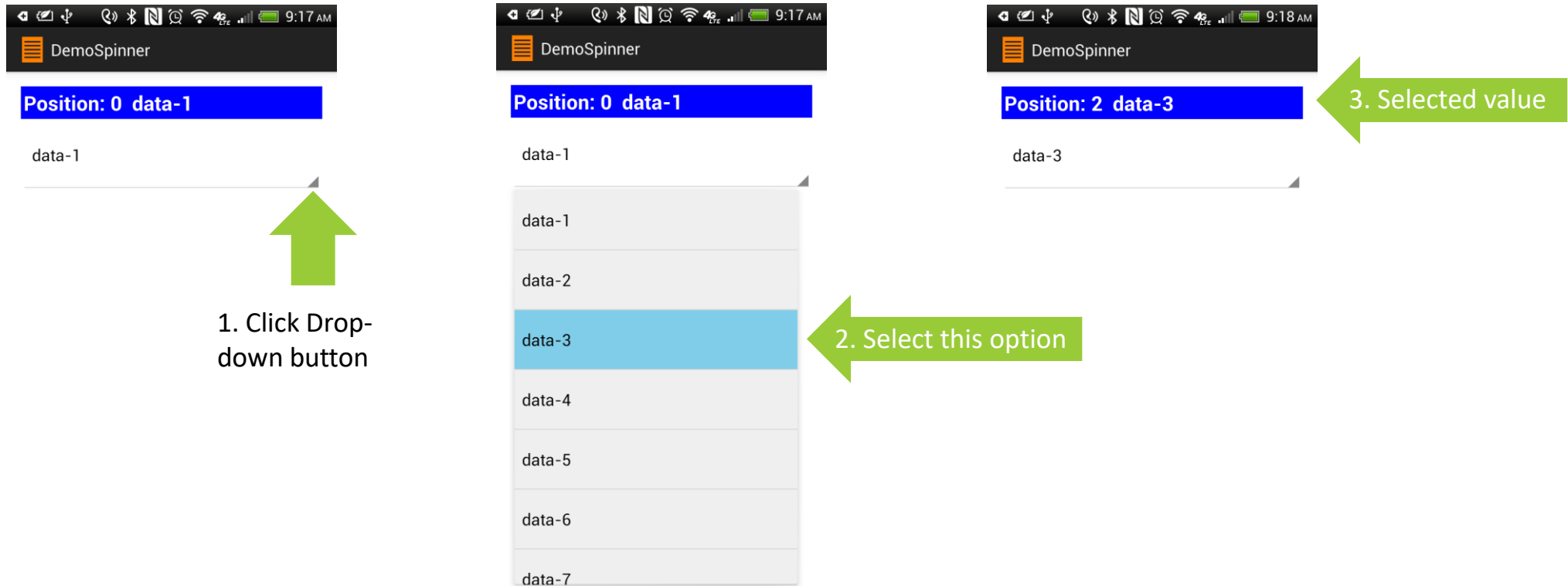
A listener captures selections made from the list with `setOnItemSelectedListener(...)`.

The `setDropDownViewResource(...)` method shows the drop-down multi-line window



# SPINNER (EXAMPLE)

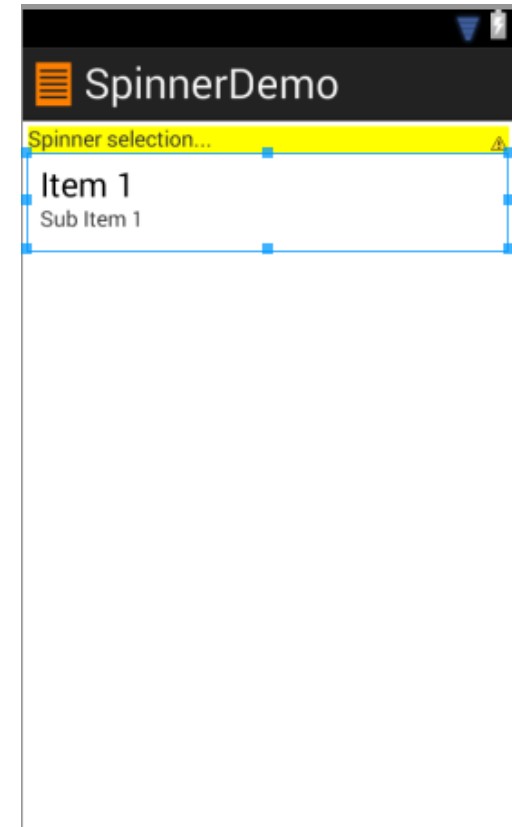
A list of options named 'Data-0', 'Data-1', 'Data-2' and so on, should be displayed when the user taps on the 'down-arrow' portion of the spinner



# SPINNER (EXAMPLE – LAYOUT)

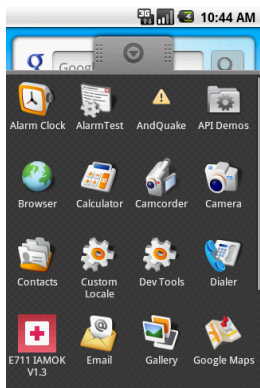
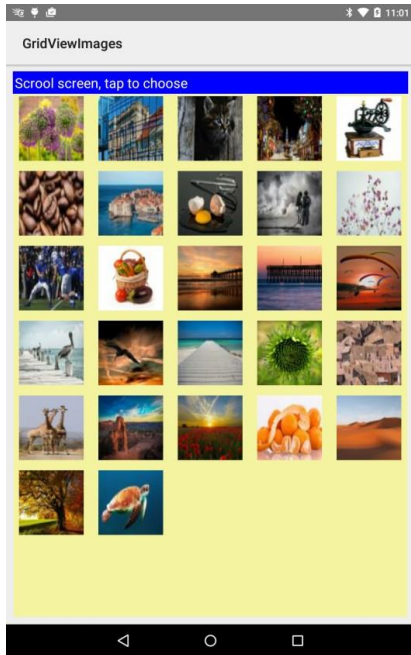
A list of options named 'Data-0', 'Data-1', 'Data-2' and so on, should be displayed when the user taps on the 'down-arrow' portion of the spinner

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="3dp"
    tools:context=".MainActivity" >
    <TextView android:id="@+id/txtMsg"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#ffffff00"
        android:text="@string/hello_world" />
    <Spinner android:id="@+id/spinner1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
</LinearLayout>
```



# SPINNER (EXAMPLE – MAINACTIVITY)

```
public class MainActivity extends Activity implements AdapterView.OnItemClickListener{
    TextView txtMsg; Spinner spinner;
    String[] items = { "Data-0", "Data-1", "Data-2", "Data-3", "Data-4", "Data-5", "Data-6", "Data-7" };
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        txtMsg = (TextView) findViewById(R.id.txtMsg);
        spinner = (Spinner) findViewById(R.id.spinner1);
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, android.R.layout.simple_spinner_dropdown_item, items);
        spinner.setAdapter(adapter);
        // add spinner a listener so user can meake selections by tapping an item
        spinner.setOnItemClickListener(this);
    }
    // next two methods implement the spinner's listener
    @Override
    public void onItemClick(AdapterView<?> parent, View v, int position, long id) { txtMsg.setText(items[position]); }
    @Override
    public void onNothingSelected(AdapterView<?> arg0) { /* TODO do nothing – needed by the interface*/ }
}
```



# GRIDVIEW

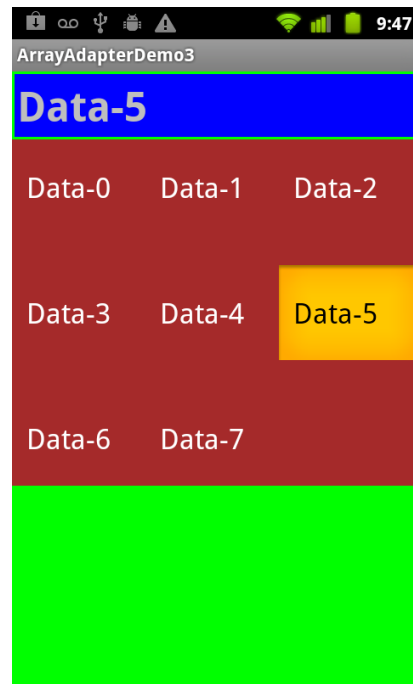
GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid.

Data items shown by grid are supplied by a data adapter.

Grid cells can show text and/or images

Some properties used to determine the number of columns and their sizes:

- `android:numColumns`: indicates how many columns to show. When used with option "auto\_fit", Android determines the number of columns based on available space and the properties listed below.
- `android:verticalSpacing` and `android:horizontalSpacing`: indicate how much free space should be set between items in the grid.
- `android:columnWidth`: column width in dips.
- `android:stretchMode`: indicates how to modify image size when there is available space not taken up by columns or spacing .



# GRIDVIEW

## (Fitting the view to the screen)

---

Suppose the screen is 320 (dip) pixels wide, and we have `android:columnWidth` set to 100dip and `android:horizontalSpacing` set to 5dip.

The user would see three columns taking 310 pixels (three columns of 100 pixels and two separators of 5 pixels).

With `android:stretchMode` set to `columnWidth`, the three columns will each expand by 3-4 pixels to use up the remaining 10 pixels.

With `android:stretchMode` set to `spacingWidth`, the two internal whitespaces will each grow by 5 pixels to consume the remaining 10 pixels.

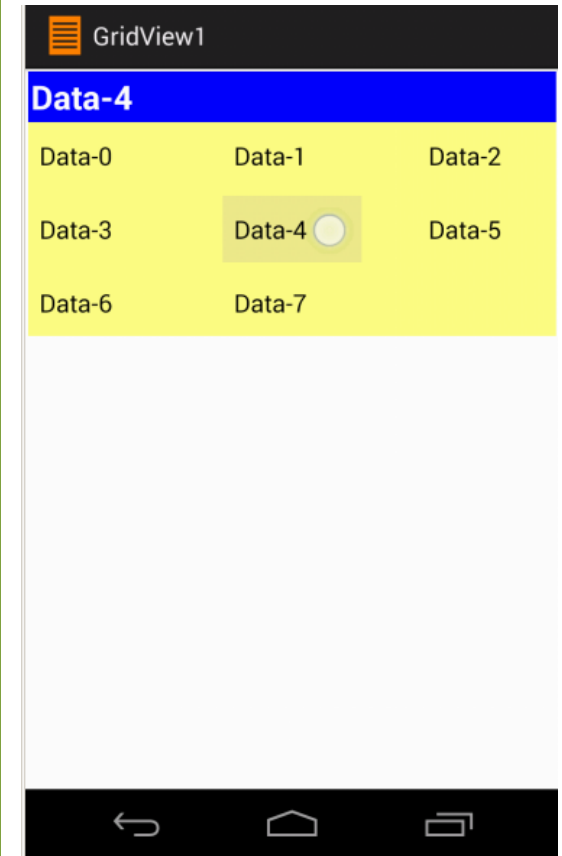
# GRIDVIEW (EXAMPLE - LAYOUT)

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="2dp"
    tools:context=".MainActivity" >

    <TextView android:id="@+id/txtMsg" android:layout_width="match_parent"
        android:layout_height="wrap_content" android:background="#ff0000ff"
        android:textSize="24sp" android:textStyle="bold"
        android:textColor="ffffffff" android:padding="2dp" />

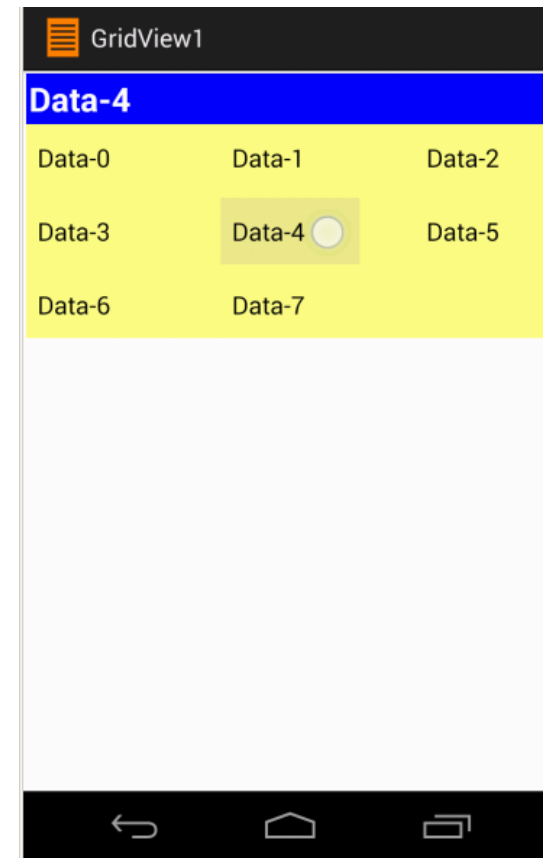
    <GridView android:id="@+id/grid" android:background="#77ffff00"
        android:layout_width="match_parent" android:layout_height="wrap_content"
        android:verticalSpacing="5dip" android:horizontalSpacing="5dip"
        android:numColumns="auto_fit" android:columnWidth="100dip"
        android:stretchMode="spacingWidth" />

</LinearLayout>
```



# GRIDVIEW (EXAMPLE - MAINACTIVITY)

```
public class ArrayAdapterDemo3 extends Activity {
    TextView txtMsg;
    String[] items = { "Data-0", "Data-1", "Data-2", "Data-3", "Data-4", "Data-5", "Data-6", "Data-7" };
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        txtMsg = (TextView) findViewById(R.id.txtMsg);
        GridView grid = (GridView) findViewById(R.id.grid);
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, items );
        grid.setAdapter(adapter);
        grid.setOnItemClickListener(new OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> container, View v, int position, long id) {
                txtMsg.setText(items[position]);
            }
        });
    }
    //onCreate
}
// class
```





# AUTOCOMPLETE TEXTVIEW

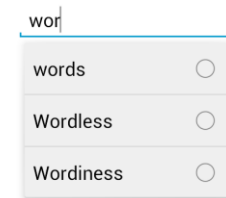
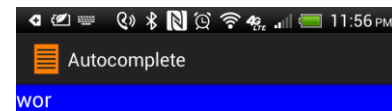
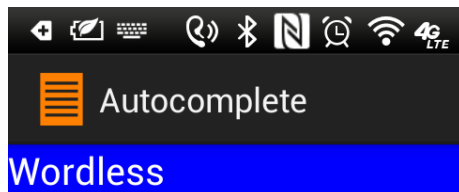
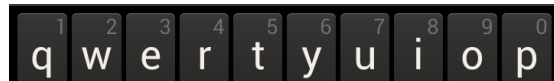
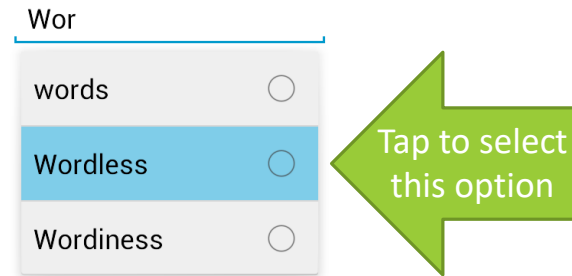
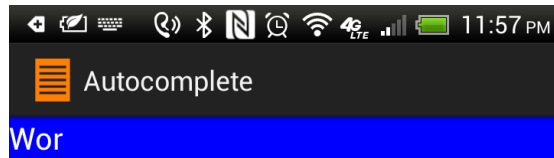
An AutoComplete box is a more specialized version of the EditText view.

Characters typed so far are compared with the beginning of words held in a user-supplied list of suggested values.

Suggestions matching the typed prefix are shown in a selection list.

The user can choose from the suggestion list or complete typing the word.

The `android:completionThreshold` property is used to trigger the displaying of the suggestion list. It indicates the number of characters to watch for in order to match prefixes.



# AUTOCOMPLETE TEXTVIEW (Example – layout)

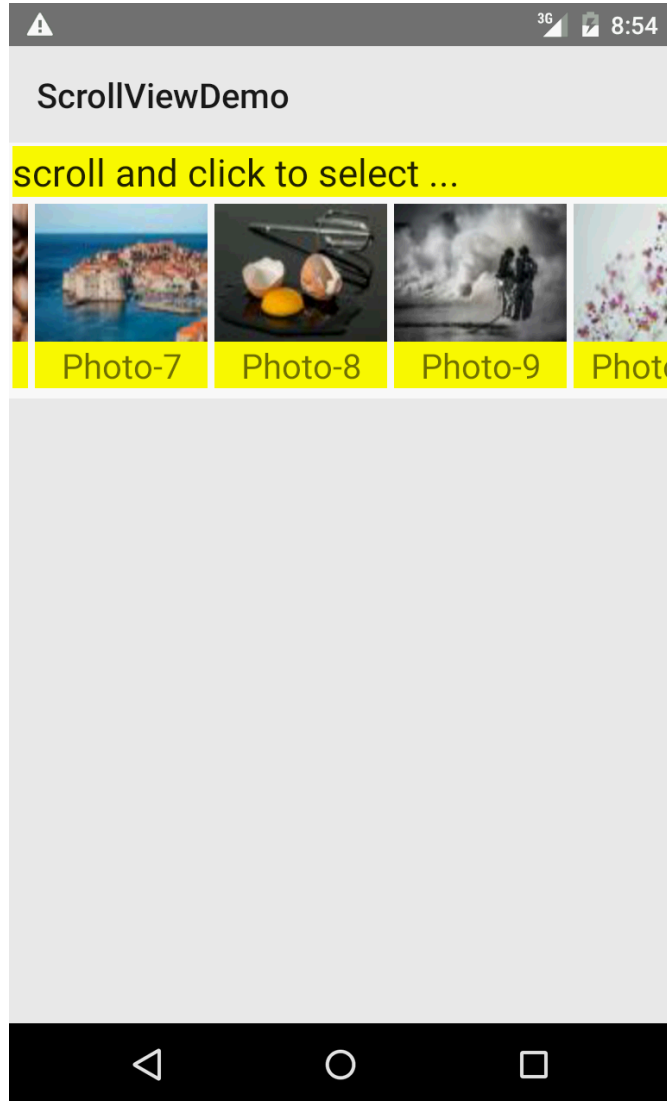
```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android" xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <TextView android:id="@+id/txtMsg"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="20sp"
        android:textColor="#ffffff"
        android:background="#ff0000ff"/>
    <AutoCompleteTextView android:id="@+id/autoCompleteTextView1"
        android:hint="type here..."
        android:completionThreshold="3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/txtMsg"
        android:layout_marginTop="15dp"
        android:ems="10" />
</RelativeLayout>
```

# AUTOCOMPLETE TEXTVIEW

## (Example – MainActivity)

---

```
public class ArrayAdapterDemo4 extends Activity implements TextWatcher {
    TextView txtMsg;
    AutoCompleteTextView txtAutoComplete;
    String[] items = {"words", "starting", "with", "set", "Setback", "Setline", "Setoffs", "Setouts", "Setters", "Setting", "Settled", "Wordless", "Wordiness", "Adios"};
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        txtMsg = (TextView) findViewById(R.id.txtMsg);
        txtAutoComplete = (AutoCompleteTextView) findViewById(R.id.autoCompleteTextView1);
        txtAutoComplete.addTextChangedListener(this);
        txtAutoComplete.setAdapter(new ArrayAdapter<String>(this, android.R.layout.simple_list_item_single_choice, items));
    } //onCreate
    public void onTextChanged(CharSequence s, int start, int before, int count) {txtMsg.setText(txtAutoComplete.getText());}
    public void beforeTextChanged(CharSequence s, int start, int count, int after) { /*needed for interface, but not used*/ }
    public void afterTextChanged(Editable s) { /*needed for interface, but not used*/ }
} //class
```



## HORIZONTALSCROLLVIEW

HorizontalScrollViews allow the user to graphically select an option from a set of small images called thumbnails

The user interacts with the viewer using two simple actions:

- 1. Scroll the list (left ↔ right)
- 2. Click on a thumbnail to pick the option it offers.

In our example, when the user clicks on a thumbnail the app responds by displaying a high-resolution version of the image



# HORIZONTALSCROLLVIEW (Example)

In this example we place a `HorizontalScrollView` at the top of the screen, this view will show a set of thumbnail options.

The user may scroll through the images and finally tap on a particular selection.

A better-quality version of the selected picture will be displayed in an `ImageView` widget placed below the horizontal scroller.

Make a thumbnail:

- <https://www.fotojet.com/>
- <https://romannurik.github.io/AndroidAssetStudio/>

# HORIZONTALSCROLLVIEW

## (Example - Populating the HorizontalScrollView)

---

Our HorizontalScrollView will expose a list of frames, each containing an icon and a caption below the icon.

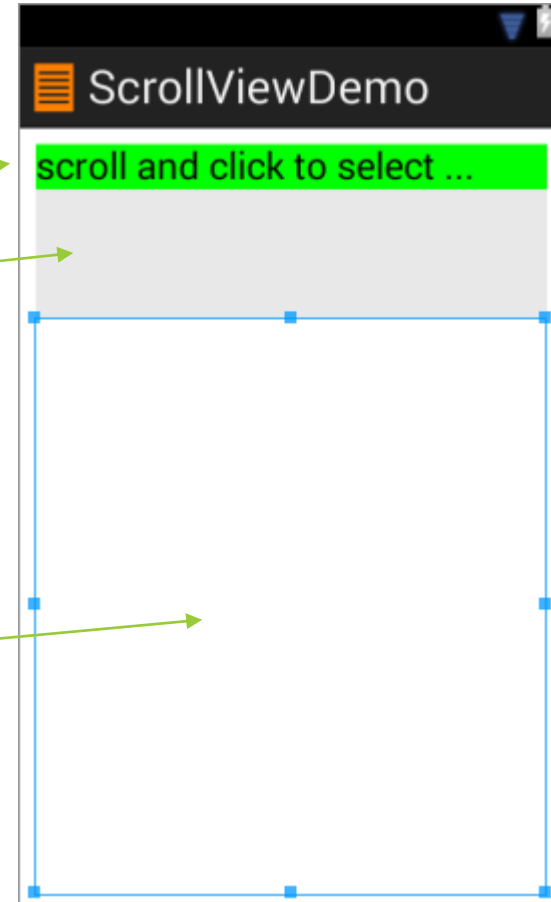
The frame\_icon\_caption.xml layout describes the formatting of icon and its caption. This layout will be inflated in order to create run-time GUI objects.

After the current frame is filled with data, it will be added to the growing set of views hosted by the scrollViewgroup container (scrollViewgroup is nested inside the horizontal scroller).

Each frame will receive an ID (its current position in the scrollViewgroup) as well as an individual onClick listener.

# HORIZONTALSCROLLVIEW (Example – layout)

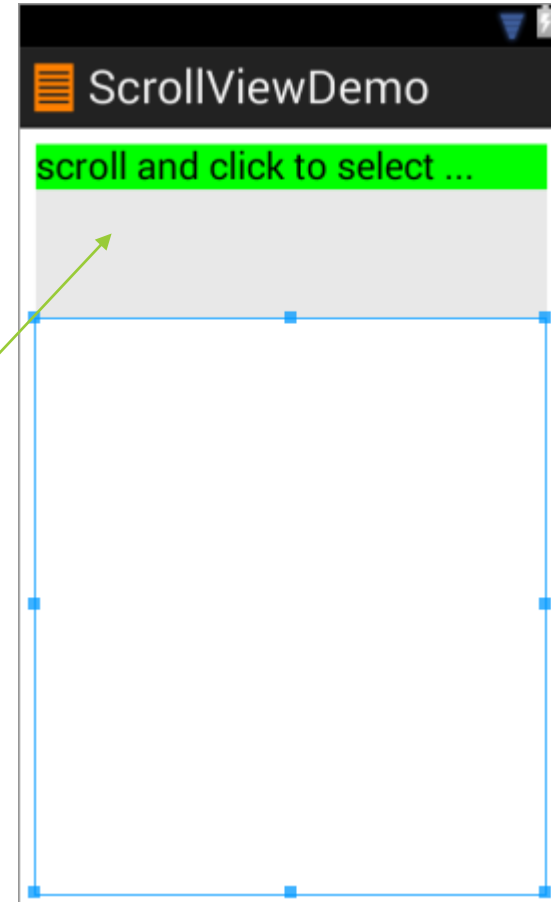
```
<?xml version="1.0" encoding="UTF-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="wrap_content"
    android:background="#ffffff" android:orientation="vertical" android:padding="2dp" >
    <TextView android:id="@+id/txtMsg"
        android:layout_width="match_parent" android:layout_height="wrap_content"
        android:background="#ff00ff00" android:text="scroll and click to select ..."
        android:textAppearance="?android:attr/textAppearanceLarge" />
    <HorizontalScrollView android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#44aaaaaa" >
        <LinearLayout android:id="@+id/viewgroup"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:orientation="horizontal"
            android:padding="10dip" />
    </HorizontalScrollView>
    <ImageView android:id="@+id/imageSelected"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="2" />
</LinearLayout>
```



# HORIZONTALSCROLLVIEW

## (Example – 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="wrap_content"
    android:orientation="vertical"
    android:padding="2dp" >
    <ImageView android:id="@+id/icon"
        android:layout_width="100dp"
        android:layout_height="80dp"
        android:scaleType="fitXY"
        android:src="@mipmap/ic_launcher" />
    <TextView android:id="@+id/caption"
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:background="#33ffff00"
        android:textSize="20sp"
        android:gravity="center" />
</LinearLayout>
```

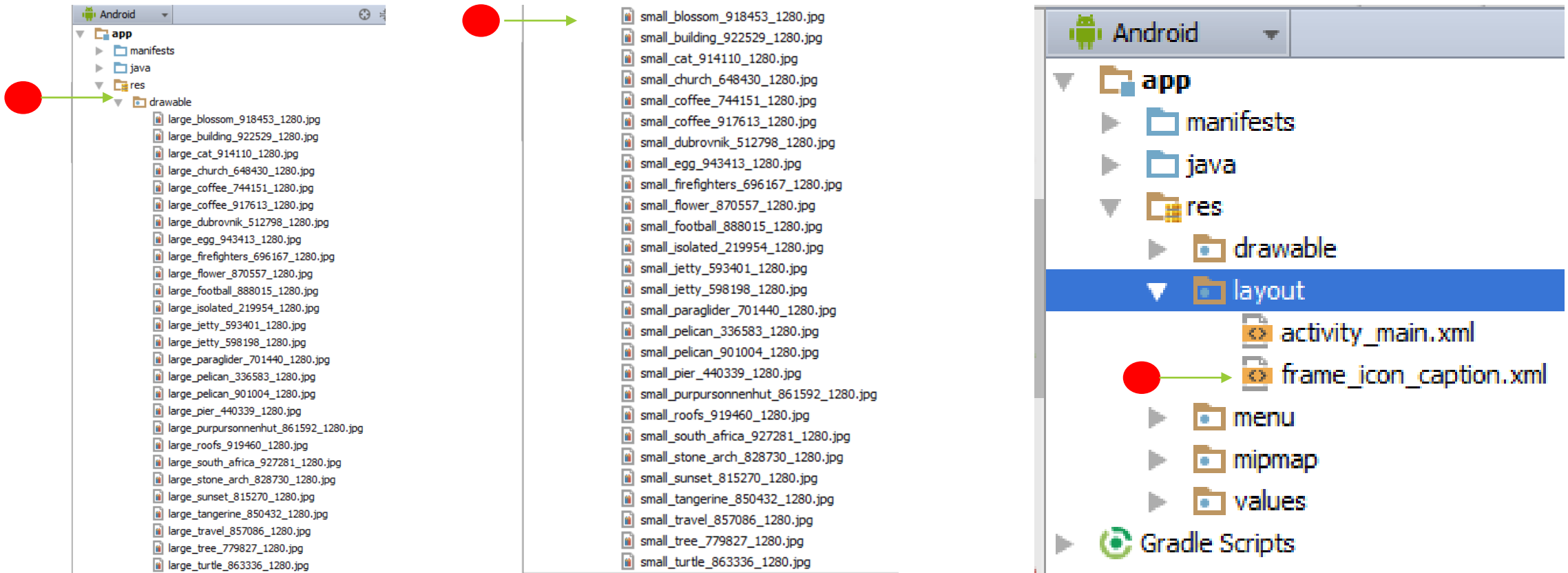


This layout'll be used by an inflater to dynamically create new views. These views'll be added to linear layout contained inside the HorizontalScrollerView



# HORIZONTALSCROLLVIEW

(Example – app's structure)



# HORIZONTALSCROLLVIEW

## (Example - MainActivity)

```
public class MainActivity extends Activity {
    TextView txtMsg; ViewGroup scrollViewgroup;
    //each frame in the HorizontalScrollView has [icon, caption]
    ImageView icon; TextView caption;
    //large image frame for displaying high-quality selected image
    ImageView imageSelected;
    //frame captions
    String[] items = {"Photo-1", ..., "Photo-26"};
    //frame-icons ( 100×100 thumbnails )
    Integer[] thumbnails = {R.drawable.small_blossom_918453_1280, ...};
    //frame-icons ( 100×100 thumbnails )
    Integer[] largeImages = {R.drawable.large_blossom_918453_1280, ...};
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //bind GUI controls to Java classes
        txtMsg = (TextView) findViewById(R.id.txtMsg);
        imageSelected = (ImageView) findViewById(R.id.imageSelected);
        // this layout goes inside the HorizontalScrollView
        scrollViewgroup = (ViewGroup) findViewById(R.id.viewgroup);
```

```
        // populate the ScrollView
        for (int i = 0; i < items.length; i++) {
            //create single frames [icon & caption] using XML inflater
            final View singleFrame = getLayoutInflater().inflate(R.layout.frame_icon_caption, null);
            //frame: 0, frame: 1, frame: 2, ... and so on
            singleFrame.setId(i);
            //internal plumbing to reach elements inside single frame
            TextView caption = (TextView) singleFrame.findViewById(R.id.caption);
            ImageView icon = (ImageView) singleFrame.findViewById(R.id.icon);
            //put data [icon, caption] in each frame
            icon.setImageResource(thumbnails[i]);
            caption.setText(items[i]); caption.setBackgroundColor(Color.YELLOW);
            //add frame to the scrollView
            scrollViewgroup.addView(singleFrame);
            //each single frame gets its own click listener
            singleFrame.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    txtMsg.setText("Selected position: " + singleFrame.getId() + " " + items[singleFrame.getId()]);
                    showLargeImage(singleFrame.getId());
                } // listener
            }); // for – populating ScrollView
        } //onCreate
        //display a high-quality version of the image selected using thumbnails
        protected void showLargeImage(int frameId) {
            Drawable selectedLargeImage = getResources().getDrawable(largeImages[frameId], getTheme()); //API-21 or newer
            imageSelected.setBackground(selectedLargeImage);
        }
    }
}
```

# IMAGE-BASED GRIDVIEW



Perhaps a more interesting version of the GridView control involves the displaying of images instead of text.

The following example illustrates how to use this control:

- 1. A screen shows an array of thumbnails.
- 2. The user makes her selection by tapping on one of them.
- 3. The app displays on a new screen a bigger & better image of the selected option.
- 4. The programmer must provide a custom data adapter to manage the displaying of thumbnails from the data set.

# IMAGE-BASED GRIDVIEW

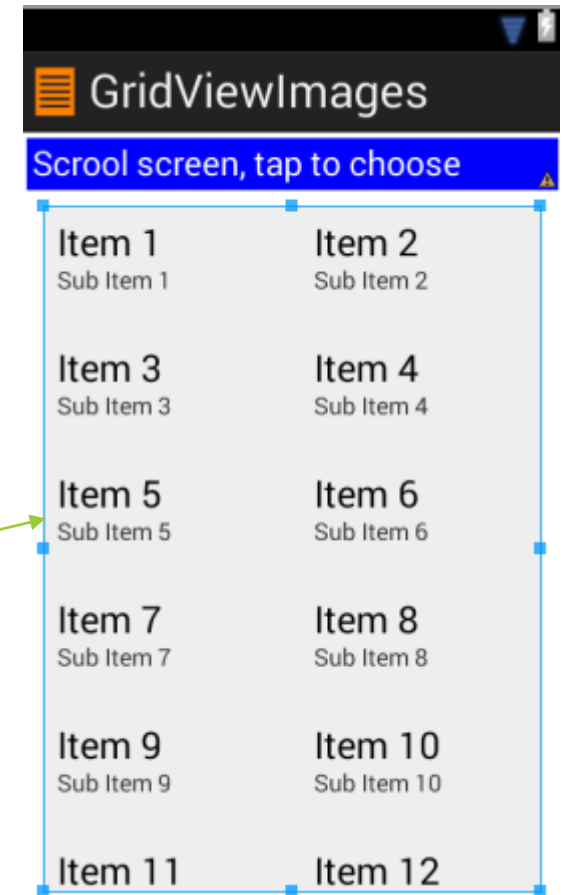
## (Example – layout activity\_main)

```
<?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:orientation="vertical"
    android:padding="3dp" >

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#ff0000ff"
        android:padding="3dp"
        android:text="Scrool screen, tap to choose"
        android:textColor="#ffffff"
        android:textSize="20sp" />

    <GridView
        android:id="@+id/gridview"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_margin="1dp"
        android:columnWidth="100dp"
        android:gravity="center"
        android:horizontalSpacing="5dp"
        android:numColumns="auto_fit"
        android:stretchMode="columnWidth"
        android:verticalSpacing="10dp" />

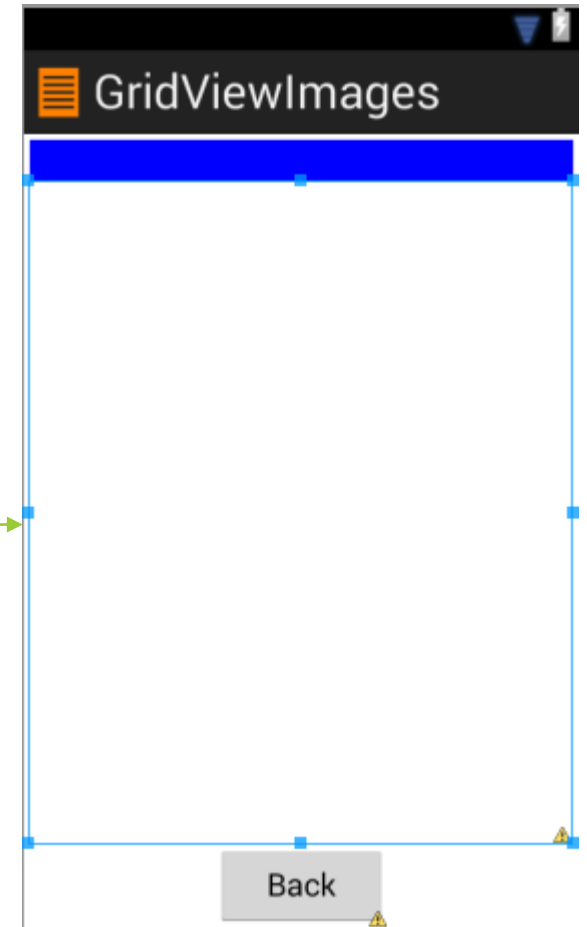
</LinearLayout>
```



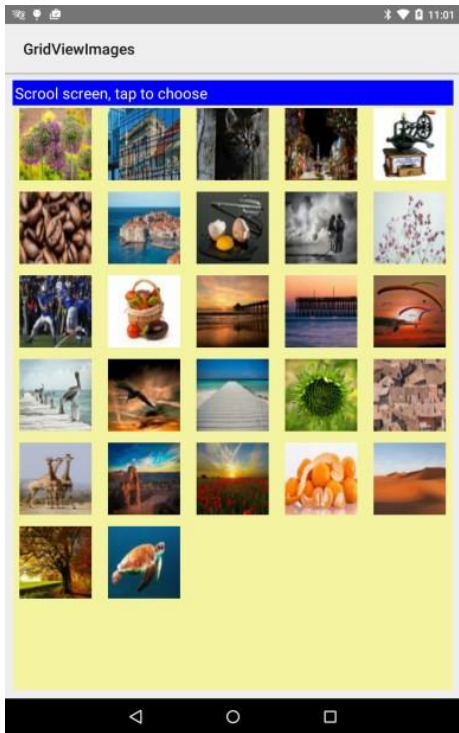
# IMAGE-BASED GRIDVIEW

## (Example – layout solo\_picture)

```
<?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:orientation="vertical"
    android:padding="3dp" >
    <TextView
        android:id="@+id/txtSoloMsg"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#ff0000ff"
        android:textColor="@android:color/white" />
    <ImageView
        android:id="@+id/imgSoloPhoto"
        android:layout_width="match_parent"
        android:layout_height="0dip"
        android:layout_gravity="center|fill"
        android:layout_weight="2" />
    <Button
        android:id="@+id/btnSoloBack"
        android:layout_width="100dip"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:text="Back" />
</LinearLayout>
```



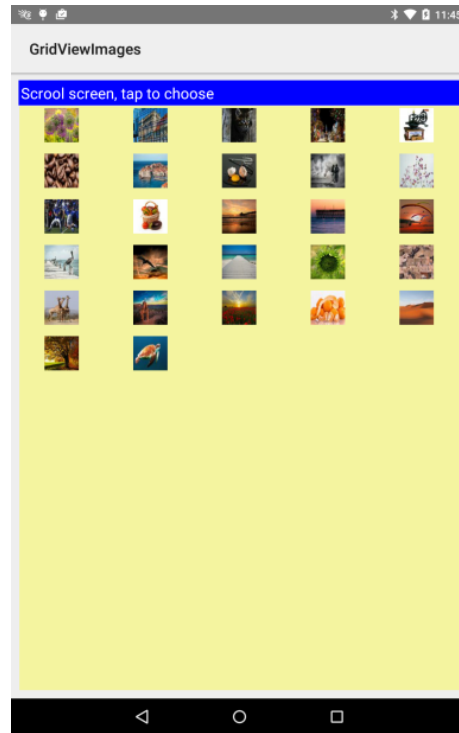
# IMAGE-BASED GRIDVIEW (Example – res/values/dimens/)



On the left:

```
int gridsize = context.getResources().getDimensionPixelOffset(R.dimen.gridview_size);  
imageView.setLayoutParams(new GridView.LayoutParams(gridsize, gridsize));
```

On the right: `imageView.setLayoutParams(new GridView.LayoutParams(100, 100));`

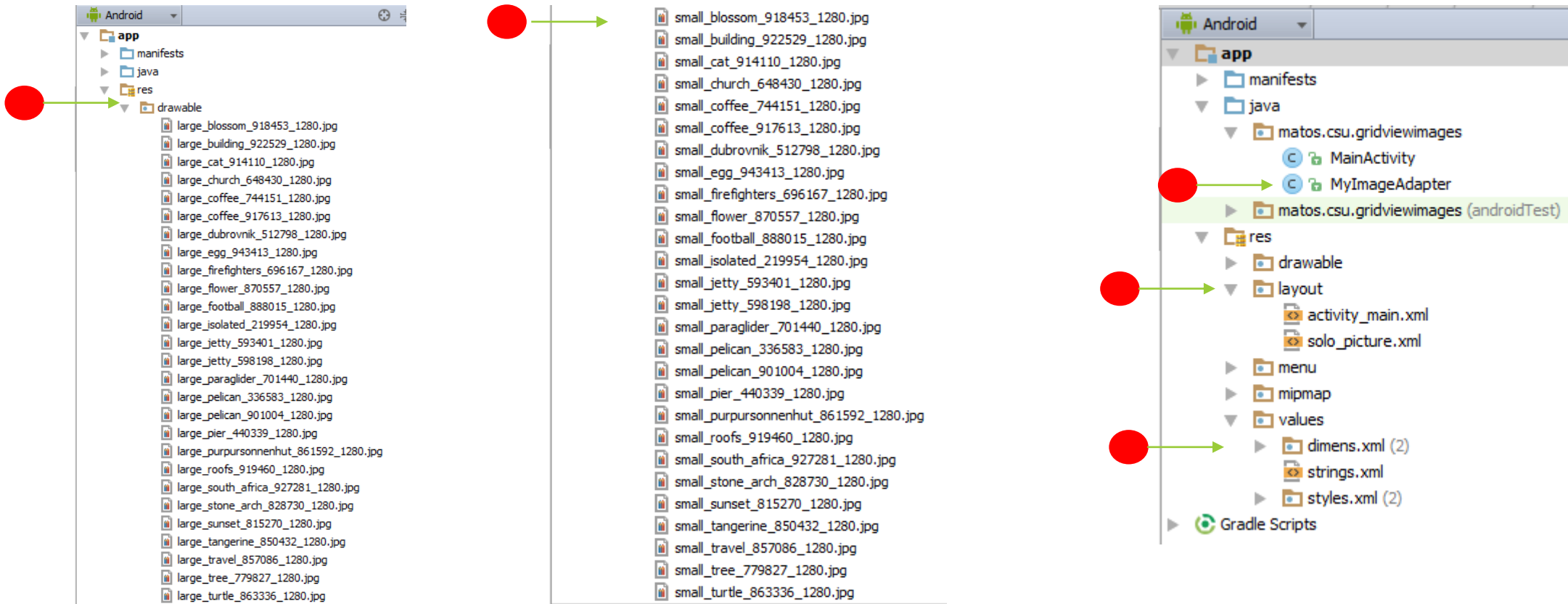


Best Practice: Defining the GridView's high and width dimensions on dips is safer than in pixels. Later on, images can be automatically scaled to devices of various densities.

```
<resources>  
<!-- Default screen margins, per the Android Design guidelines. -->  
<dimen name="activity_horizontal_margin">16dp</dimen>  
<dimen name="activity_vertical_margin">16dp</dimen>  
<dimen name="gridview_size">100dp</dimen>  
</resources>
```

# IMAGE-BASED GRIDVIEW

## (Example – App's structure)





# IMAGE-BASED GRIDVIEW (Example – MainActivity)

```
public class MainActivity extends Activity {
    //GUI control bound to screen1 (holding GridView)
    GridView gridView;
    //GUI controls bound to screen2 (holding single ImageView)
    TextView txtSoloMsg; ImageView imgSoloPhoto; Button btnSoloBack;
    //in case you want to use-save state values
    Bundle myOriginalMemoryBundle;
    //frame captions
    String[] items = {"Photo-1", ..., "Photo-26"};
    //frame-icons (100×100 thumbnails)
    Integer[] thumbnails = {R.drawable.small_blossom_918453_1280, ...};
    //large images (high quality pictures)
    Integer[] largeImages = {R.drawable.large_blossom_918453_1280, ...};
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        myOriginalMemoryBundle = savedInstanceState;
        // setup GridView with its custom adapter and listener
        gridView = (GridView) findViewById(R.id.gridView);
        gridView.setAdapter(new MyImageAdapter(this, thumbnails));
        gridView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int position, long id) { showBigScreen(position); }
        });
    }
    //onCreate
```

```
private void showBigScreen(int position) {
    // show the selected picture as a single frame in the second layout
    setContentView(R.layout.solo_picture);
    // plumbing – second layout
    txtSoloMsg = (TextView) findViewById(R.id.txtSoloMsg);
    imgSoloPhoto = (ImageView) findViewById(R.id.imgSoloPhoto);
    // set caption-and-large picture
    txtSoloMsg.setText(" Position= " + position + " " + items[position]);
    imgSoloPhoto.setImageResource( largeImages[position] );
    // set GO BACK button to return to layout1 (GridView)
    btnSoloBack = (Button) findViewById(R.id.btnSoloBack);
    btnSoloBack.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            // redraw the main screen showing the GridView
            onCreate(myOriginalMemoryBundle);
        }
    });
}
// showBigScreen
//Activity
```



# IMAGE-BASED GRIDVIEW

## (Example – Custom adapter - MyImageAdapter)

```
// This custom adapter populates the GridView with a visual representation of each thumbnail in the input data set. It also implements a method -getView()- to access individual cells in the GridView.
public class MyImageAdapter extends BaseAdapter{
    private Context context; // main activity's context
    Integer[] smallImages; // thumbnail data set
    public MyImageAdapter(Context mainActivityContext, Integer[] thumbnails) { context = maiActivityContext; smallImages = thumbnails; }
    // how many entries are there in the data set?
    public int getCount() { return smallImages.length; }
    // what is in a given 'position' in the data set?
    public Object getItem(int position) { return smallImages[position]; }
    // what is the ID of data item in given 'position'?
    public long getItemId(int position) { return position; }
    // create a view for each thumbnail in the data set, add it to gridview
    public View getView(int position, View convertView, ViewGroup parent) {
        ImageView imageView;
        // if possible, reuse (convertView) image already held in cache
        if (convertView == null) {
            // no previous version of thumbnail held in the scrapview holder define entry in res/values/dimens.xml for grid height,width in dips <dimen name="gridview_size">100dp</dimen>
            // setLayoutParams will do conversion to physical pixels
            imageView = new ImageView(context);
            int gridsize = context.getResources().getDimensionPixelOffset(R.dimen.gridview_size);
            imageView.setLayoutParams(new GridView.LayoutParams(gridsize, gridsize)); imageView.setScaleType(ImageView.ScaleType.FIT_XY); imageView.setPadding(5, 5, 5, 5);
        }
        else { imageView = (ImageView) convertView; }
        imageView.setImageResource(smallImages[position]); imageView.setId(position);
        return imageView;
    } //getView
} //MyImageAdapter
```

# IMAGE-BASED GRIDVIEW (Example - Result)



**Image taken from the Emulator**



**Image displayed on a Nexus7 (1028x728) tablet.**

The GridView's clause: `android:numColumns="auto_fit"` determines the best way to fill up each row

# IMAGE-BASED GRIDVIEW

## (Changing from GridView to ListView)

---

Modify the previous example to show the set of thumbnails in a ListView instead of a GridView. As before when a thumbnail is tapped a high-quality image is shown in a new screen.

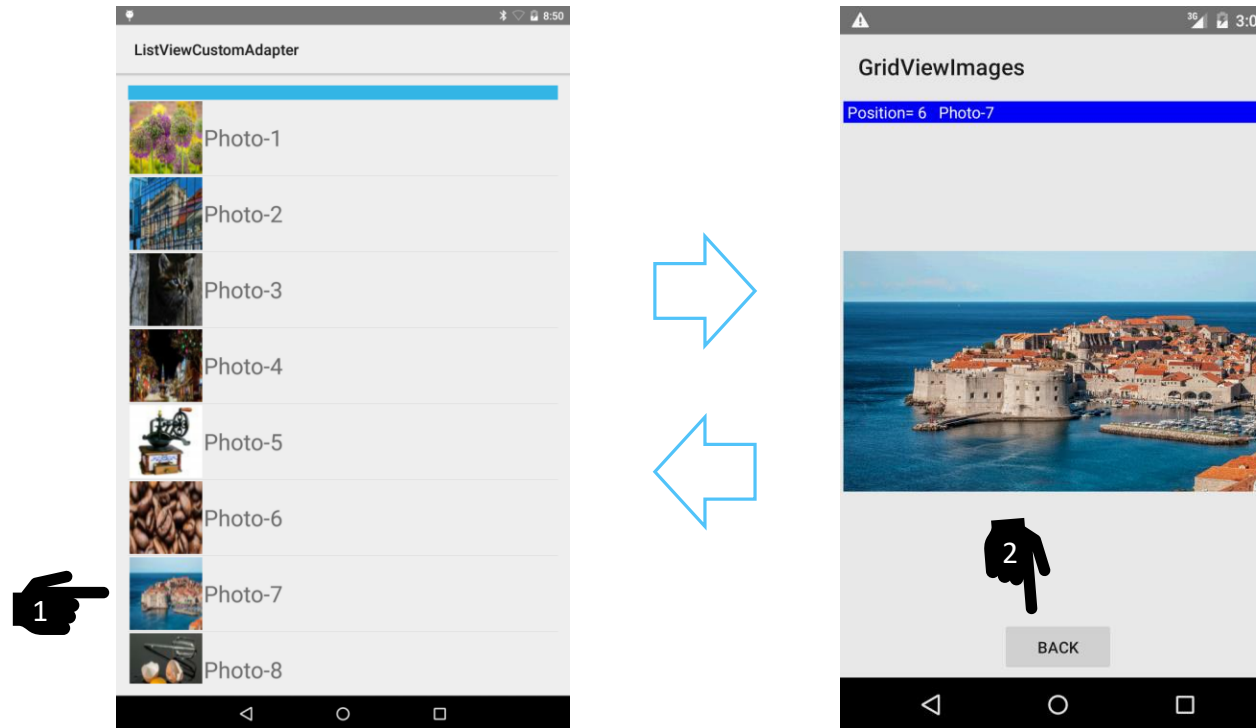
### STEPS

- 1. Modify the layout activity\_main.xml. Change the tag <GridView ... to <ListView. Leave the rest unchanged.
- 2. In the main Java class, replace each reference to the GridView type with ListView. The new statements should be:
  - `ListView gridview;`
  - ...
  - `gridview = (ListView) findViewById(R.id.gridview);`
- 3. In the custom Image adapter make the following change to indicate the new imageView should be added to a ListView (instead that a GridView)
  - `imageView.setLayoutParams(new ListView.LayoutParams(100, 75) );`
- 4. Keep the rest of the adapter code unchanged.

# IMAGE-BASED GRIDVIEW (Changing from GridView to ListView)

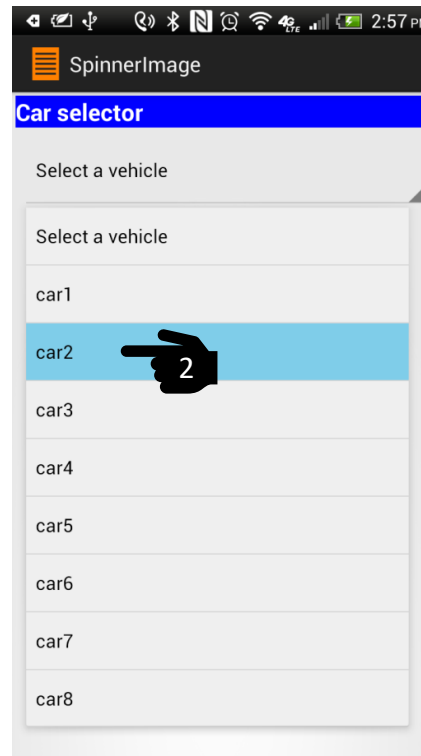
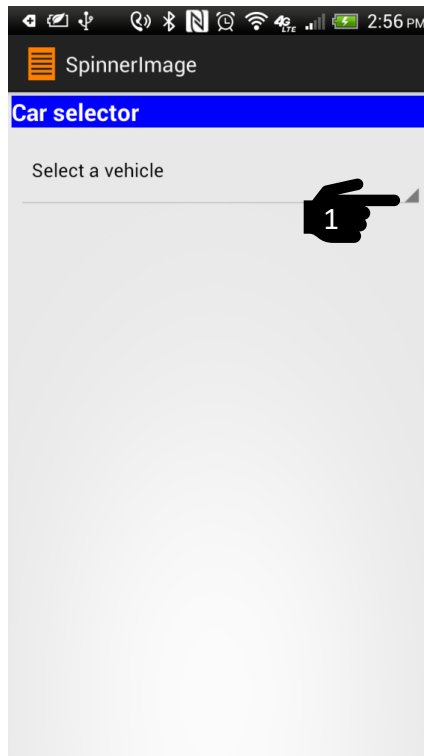
The new app should display the following screens.

Observe the main screen arranges the thumbnails in a ListView container.



# IMAGE-BASED SPINNER

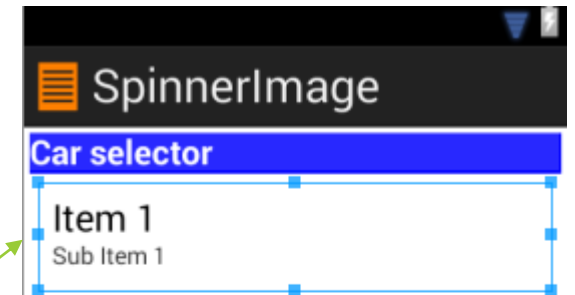
This is a simple variation of the previous example. A list of choices is offered through a drop-down spinner control. The user taps on a row and an image for the selected choice is displayed on a new screen.



# IMAGE-BASED SPINNER

## (Example – layout - MainActivity)

```
<?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:orientation="vertical"
    android:padding="3dp" >
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#ff0000ff"
        android:text="Car selector"
        android:textColor="#ffffff"
        android:textSize="20sp"
        android:textStyle="bold" />
    <Spinner
        android:id="@+id/spinner"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="5dp" />
</LinearLayout>
```



# IMAGE-BASED SPINNER

## (Example – layout – solo\_picture)

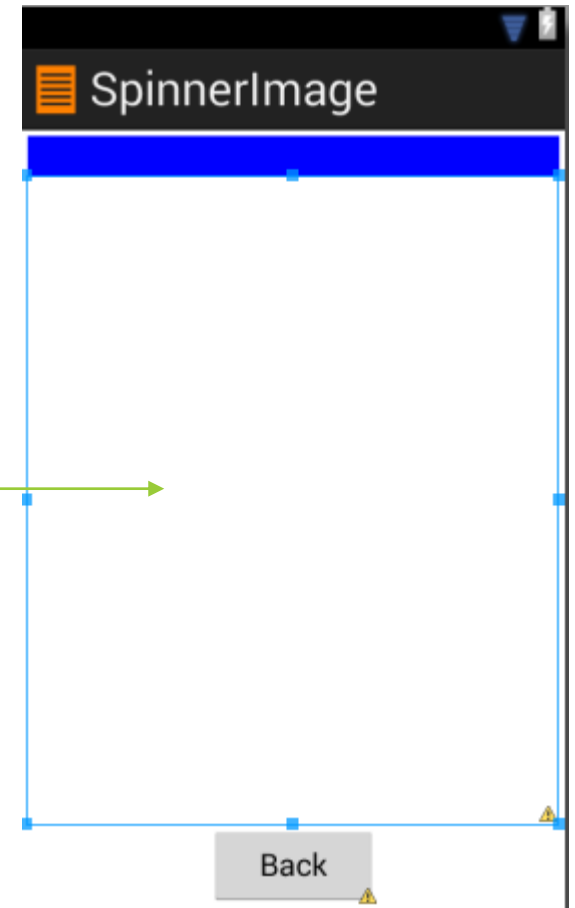
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="3dp" >

    <TextView
        android:id="@+id/txtSoloMsg" android:layout_width="match_parent"
        android:layout_height="wrap_content" android:background="#ff0000ff"
        android:textColor="#ffffff" android:textStyle="bold" />

    <ImageView
        android:id="@+id/imgSoloPhoto"
        android:layout_width="match_parent"
        android:layout_height="0dip"
        android:layout_gravity="center|fill"
        android:layout_weight="2" />

    <Button
        android:id="@+id/btnBack"
        android:layout_width="100dip"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:text="Back" />

</LinearLayout>
```



# IMAGE-BASED SPINNER

## (Example – MainActivity)

```
public class MainActivity extends Activity
    implements AdapterView.OnItemClickListener {

    // GUI controls in the main screen
    Spinner spinner;

    // GUI controls in the solo_picture screen
    TextView txtSoloMsg; ImageView imageSelectedCar; Button btnBack;

    // captions to be listed by the spinner
    String[] items = { "Select a vehicle", "car1", "car2", "car3", "car4",
        "car5", "car6", "car7", "car8" };

    // object IDs of car pictures
    Integer[] carImageArray = new Integer[] { R.drawable.car_photo_1, ...,
        R.drawable.car_photo_8, };

    Bundle myStateInfo;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        myStateInfo = savedInstanceState;
        setContentView(R.layout.activity_main);
        spinner = (Spinner) findViewById(R.id.spinner);
        spinner.setAdapter(new ArrayAdapter<String>(this,
            android.R.layout.simple_spinner_dropdown_item, items));
        spinner.setOnItemSelectedListener(this);
    } // onCreate
```

```
// display screen showing image of the selected car
private void showBigImage(int position) {
    // show the selected picture as a single frame
    setContentView(R.layout.solo_picture);
    txtSoloMsg = (TextView) findViewById(R.id.txtSoloMsg);
    imageSelectedCar = (ImageView) findViewById(R.id.imgSoloPhoto);
    txtSoloMsg.setText("Car selected: car-" + position);
    imageSelectedCar.setImageResource(carImageArray[position]);
    btnBack = (Button) findViewById(R.id.btnBack);
    btnBack.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) { onCreate(myStateInfo); }
    });
} // showBigScreen

// next two methods implement the spinner listener
@Override
public void onItemClick(AdapterView<?> parent, View v, int position, long id) {
    // ignore position 0. It holds just a label ("SELECT A VEHICLE...")
    if (position > 0) { showBigImage(position - 1); }
}

@Override
public void onNothingSelected(AdapterView<?> parent) { /* DO NOTHING - needed by the interface */ }
}
```



# CUSTOM-MADE LISTVIEW

---

Android provides several predefined row layouts for displaying simple lists (such as: `android.R.layout.simple_list_item_1`, `android.R.layout.simple_list_item_2`, etc ).

However, there are occasions in which you want a particular disposition and formatting of elements displayed in each list-row. In those cases, you should create your own subclass of a Data Adapter.

In order to customize a Data Adapter, you need to:

- 1. Create a class extending the concrete `ArrayAdapter` class
- 2. Override its `getView()`, and
- 3. Construct (inflate) your rows yourself.

```
public class MyCustomAdapter extends ArrayAdapter{  
    // class variables go here ...  
    public MyCustomAdapter(...) {}  
    public View getView(...) {}  
} // MyCustomAdapter
```

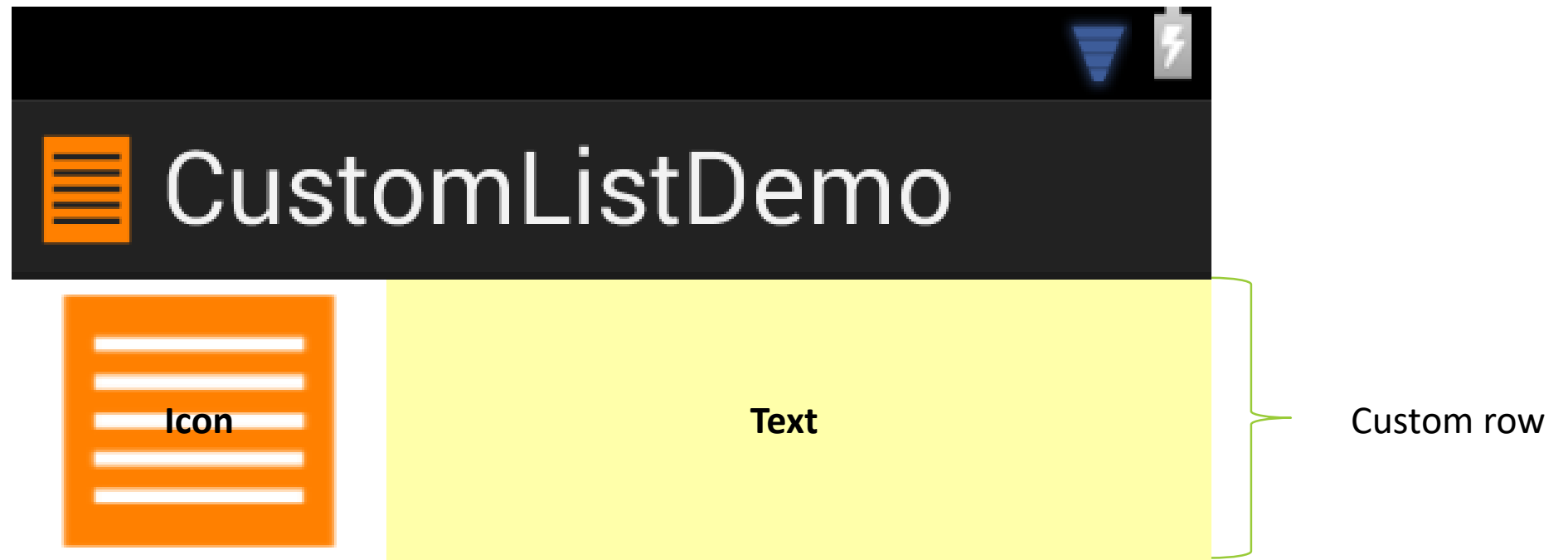
For each data element supplied by the adapter, the method `getView()` returns its 'visible' View.



# CUSTOM-MADE LISTVIEW (Designing custom-rows)

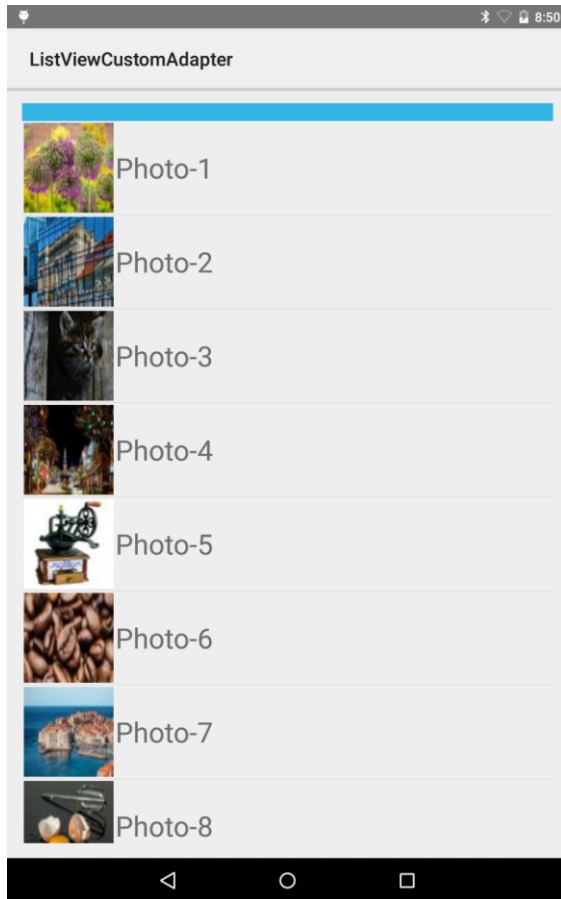
---

In our example each UI row will show an icon (on the left side) and text following the icon to its right side.



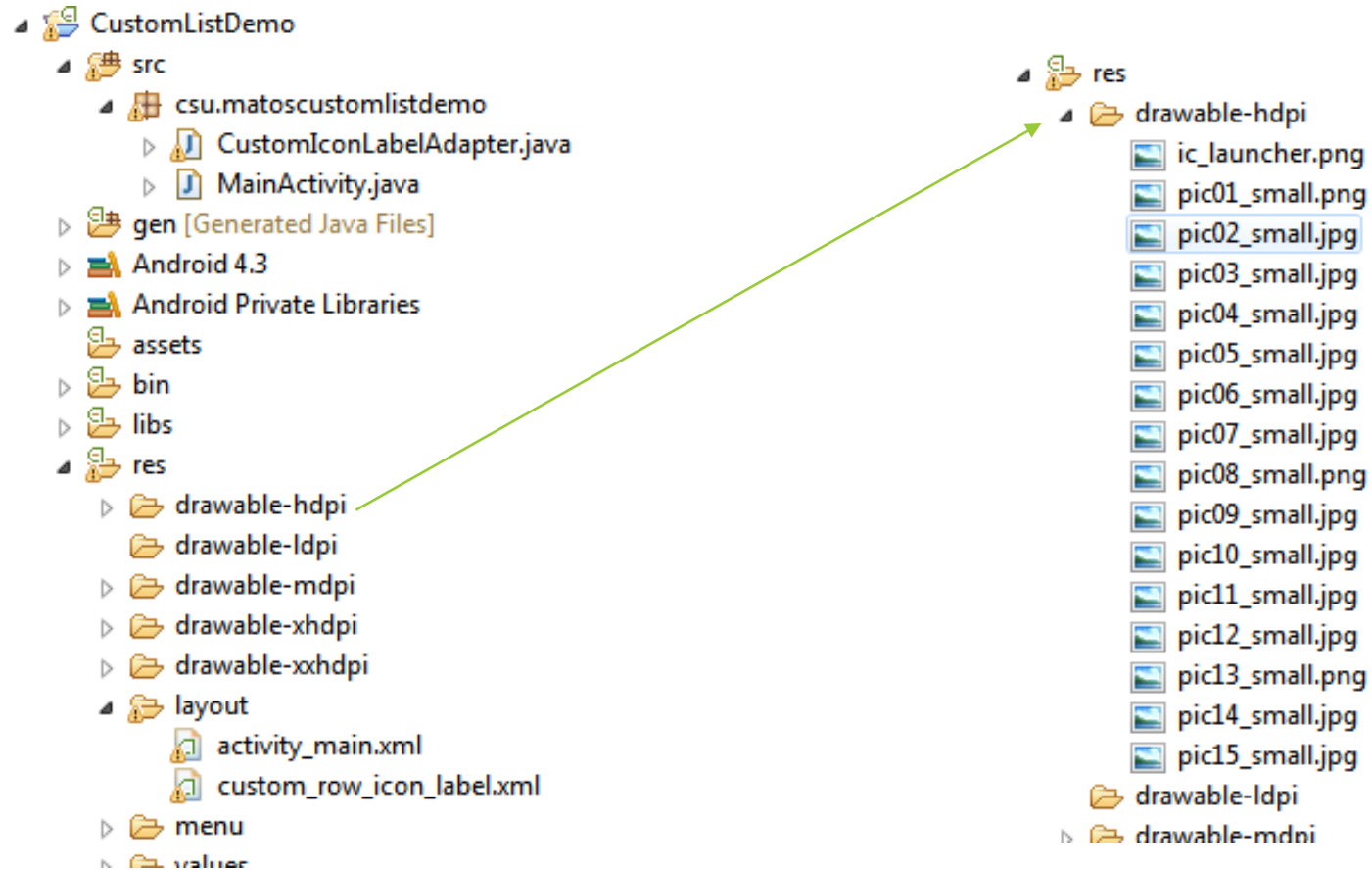
# CUSTOM-MADE LISTVIEW (Designing custom-rows)

---



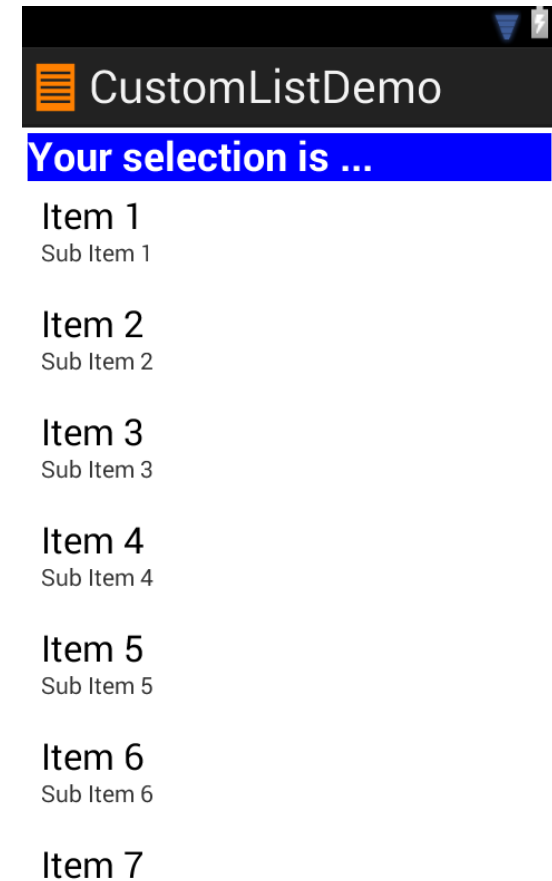
Custom row consists of icon & text

# CUSTOM-MADE LISTVIEW (APP'S STRUCTURE)



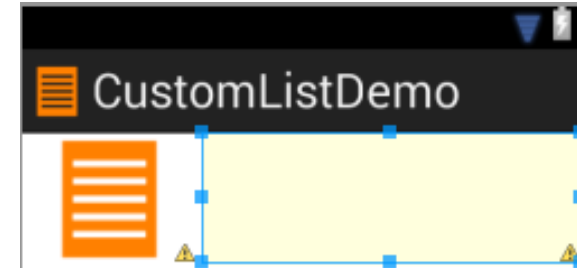
# CUSTOM-MADE LISTVIEW (Layout – activity\_main.xml)

```
<?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:padding="3dp"
    android:orientation="vertical" >
    <TextView android:id="@+id/txtMsg"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#ff0000ff"
        android:text="Your selection is ..."
        android:textColor="#ffffff"
        android:textSize="24sp"
        android:textStyle="bold" />
    <ListView android:id="@android:id/list"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</LinearLayout>
```



# CUSTOM-MADE LISTVIEW (Layout – activity\_main.xml)

```
<?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="wrap_content"
    android:orientation="horizontal" >
    <ImageView android:id="@+id/icon"
        android:layout_width="100dp"
        android:layout_height="75dp"
        android:layout_marginRight="3dp"
        android:src="@drawable/ic_launcher"/>
    <TextView android:id="@android:id/label"
        android:layout_width="match_parent"
        android:layout_height="75dp"
        android:background="#22ffff00"
        android:textSize="20sp"/>
</LinearLayout>
```



# CUSTOM-MADE LISTVIEW (MainActivity)

```
public class MainActivity extends ListActivity {
    TextView txtMsg;
    // The n-th row in the list will consist of [icon, label] where icon = thumbnail[n] and label=items[n]
    String[] items = { "Data-1", ..., "Data-15" };
    Integer[] thumbnails = { R.drawable.pic01_small, R.drawable.pic02_small, ..., R.drawable.pic15_small };
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        txtMsg = (TextView) findViewById(R.id.txtMsg);
        // the arguments of the custom adapter are: activityContext, layout-to-be-inflated, labels, icons
        CustomIconLabelAdapter adapter = new CustomIconLabelAdapter(this, R.layout.custom_row_icon_label, items, thumbnails);
        // bind intrinsic ListView to custom adapter
        setListAdapter(adapter);
    } //onCreate
    @Override
    protected void onItemClick(ListView l, View v, int position, long id) {
        super.onItemClick(l, v, position, id);
        txtMsg.setText(" Position: " + position + " " + items[position]);
    } //listener
} //class
```

# CUSTOM-MADE LISTVIEW (MainActivity)

---

```
class CustomIconLabelAdapter extends ArrayAdapter<String> {
    Context context; Integer[] thumbnails; String[] items;
    public CustomIconLabelAdapter( Context context, int layoutToBeInflated, String[] items, Integer[] thumbnails) {
        super(context, R.layout.custom_row_icon_label, items);
        this.context = context;
        this.thumbnails = thumbnails;
        this.items = items;
    }
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        LayoutInflater inflater = ((Activity) context).getLayoutInflater();
        View row = inflater.inflate(R.layout.custom_row_icon_label, null);
        TextView label = (TextView) row.findViewById(R.id.label);
        ImageView icon = (ImageView) row.findViewById(R.id.icon);
        label.setText(items[position]);
        icon.setImageResource(thumbnails[position]);
        return (row);
    }
} // CustomAdapter
```



# CUSTOM-MADE LISTVIEW

## (The LayoutInflater class)

---

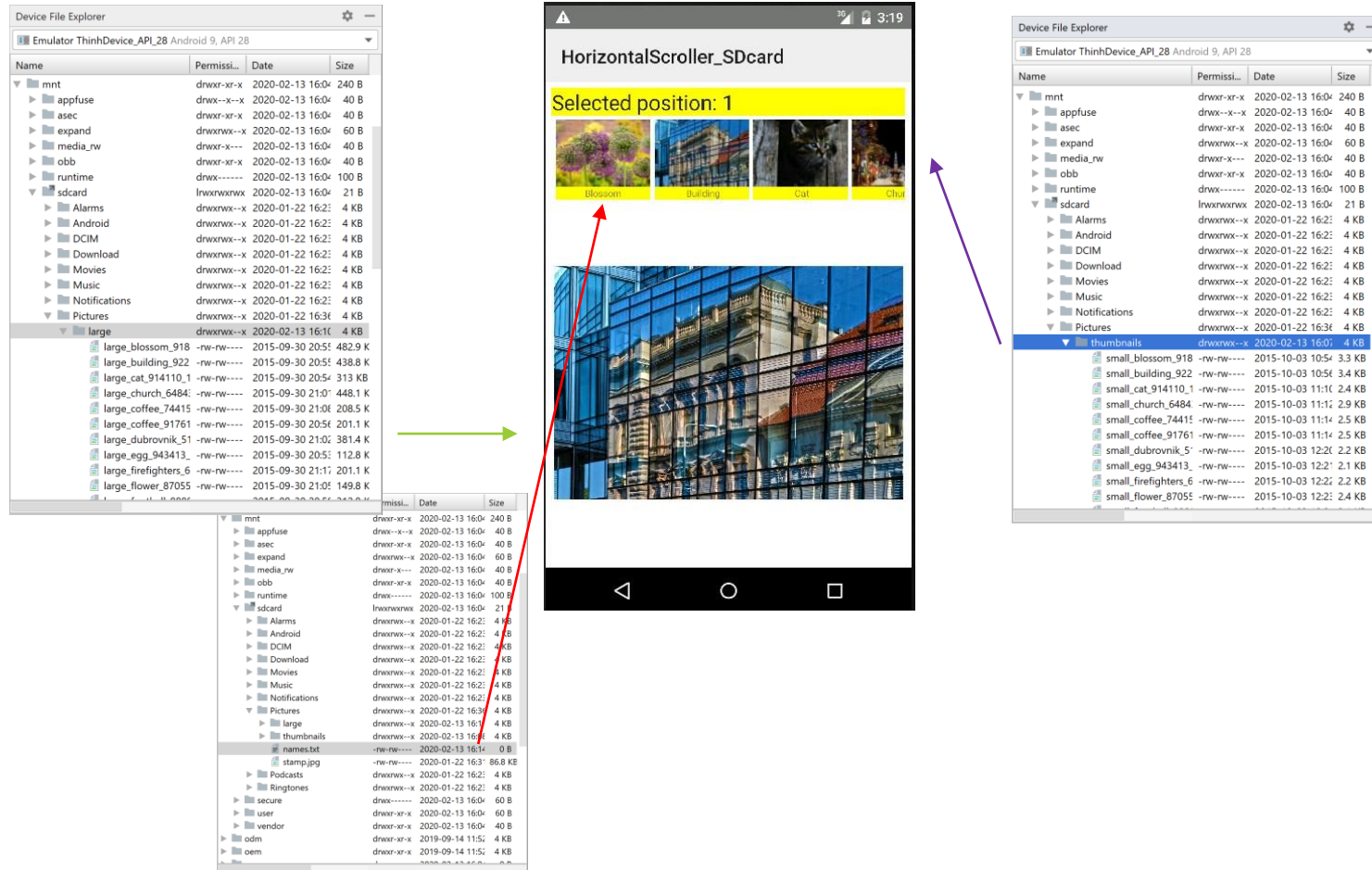
The LayoutInflater class converts an XML layout specification into an actual tree of View objects. The objects inflated by code are appended to the selected UI view. It typically works in cooperation with an ArrayAdapter.

A basic ArrayAdapter requires three arguments: current context, layout on which output rows are shown, source data items (data to feed the rows).

- The overridden getView() method inflates the row layout by custom allocating icons and text taken from data source in the user designed row.
- Once assembled, the View (row) is returned.
- This process is repeated for each item supplied by the ArrayAdapter.
- See Appendix for an example of a better built custom-adapter using the ViewHolder design strategy.

# USING THE SD-CARD

## (Storing images on the SD card)



In previous examples, images were kept in main storage using the application's memory space.

This time we will use the external disk (SD-card) to store the app's data (arguably a better space/speed tradeoff practice).

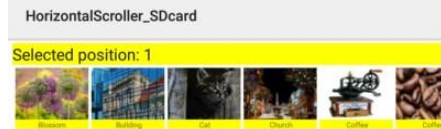
Assume the user has already transferred a copy of the pictures to the SD folder /Pictures/thumbnails/

The folder /Pictures/large/ contains a high-quality image for each thumbnail. These larger pictures are shown after the user makes a selection using the scroller.

The text file /Pictures/names.txt contains the caption associated to each picture. Data has been entered as a single line of comma separated strings (Something like Blossom, Building, Cat, Church, Coffee, ... ).

# CUSTOM-MADE LISTVIEW (MainActivity)

In this app we use the same layouts already introduced in previous example



```
public class MainActivity extends Activity {
    //GUI controls
    TextView txtMsg; ViewGroup scrollViewgroup;
    //each frame in the HorizontalScrollView has [icon, caption]
    ImageView icon; TextView caption;
    //large image frame for displaying high-quality selected image
    ImageView imageSelected; String[] items; int index;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState); setContentView(R.layout.activity_main);
        txtMsg = (TextView) findViewById(R.id.txtMsg);
        imageSelected = (ImageView) findViewById(R.id.imageSelected);
        scrollViewgroup = (ViewGroup) findViewById(R.id.viewgroup);
        try {
            // Environment class allows access to your 'MEDIA' variables
            String absolutePath2SdCard = Environment.getExternalStorageDirectory().getAbsolutePath();
            //photo captions are held in a single-line comma-separated file
            String pathPictureCaptionFile = absolutePath2SdCard + "/Pictures/names.txt";
            File nameFile = new File(pathPictureCaptionFile);
            Scanner scanner = new Scanner(nameFile);
            String line = scanner.nextLine();
            items = line.split(",");
            //get access to the small thumbnails - populate the horizontal scroller
            String pathThumbnailsFolder = absolutePath2SdCard + "/Pictures/thumbnails/";
            File sdPictureFiles = new File(pathThumbnailsFolder);
            File[] thumbnailArray = sdPictureFiles.listFiles();
```

```
txtMsg.append("\nNum files: " + thumbnailArray.length);
File singleThumbnailFile;
for (index = 0; index < thumbnailArray.length; index++) {
    singleThumbnailFile = thumbnailArray[index];
    final View frame = getLayoutInflater().inflate(R.layout.frame_icon_caption, null);
    TextView caption = (TextView) frame.findViewById(R.id.caption);
    ImageView icon = (ImageView) frame.findViewById(R.id.icon);
    // convert (jpg, png,...) file into a drawable
    icon.setImageDrawable(Drawable.createFromPath(singleThumbnailFile.getAbsolutePath()));
    caption.setText(items[index]); scrollViewgroup.addView(frame); frame.setId(index);
    frame.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) { txtMsg.setText("Selected position: " + frame.getId()); showLargeImage(frame.getId()); }
    }); // listener
} // for
} catch (Exception e) { txtMsg.append("\nError: " + e.getMessage()); }
} // onCreate
//display a high-quality version of the selected thumbnail image
protected void showLargeImage(int frameId) {
    String pathLargePictureFolder = Environment.getExternalStorageDirectory().getAbsolutePath() + "/Pictures/large/";
    // convert SD image file(jpg, png,...) into a drawable, then show into ImageView
    File sdLargePictureFolder = new File(pathLargePictureFolder);
    File[] largePictureArray = sdLargePictureFolder.listFiles();
    File largeImageFile = largePictureArray[frameId];
    imageSelected.setImageDrawable(Drawable.createFromPath(largeImageFile.getAbsolutePath()));
} // ShowLargeImage
} // Activity
```

# APPENDICES (Predefined Android resources)

---

Android SDK includes a number of predefined layouts & styles. Some of those resources can be found in the folders:

- C:\ Your-Path \Android\android-sdk\platforms\android-xx\data\res\layout
- C:\ Your-Path \Android\android-sdk\platforms\android-xx\data\res\values\styles.xml

Ex: the following is the definition of layout called: android.R.layout.simple\_list\_item\_1. It consists of a single TextView field named “text1”, its contents are centered, large font, and some padding.

```
<!-- Copyright (C) 2006 The Android Open Source Project Licensed under the Apache License, Version 2.0 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License. -->
```

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

```
<TextView xmlns:android="http://schemas.android.com/apk/res/android" android:id="@android:id/text1"
    android:layout_width="match_parent" android:layout_height="wrap_content"
    android:gravity="center_vertical" android:minHeight="?android:attr/listPreferredItemHeight"
    android:paddingLeft="6dip" android:textAppearance="?android:attr/textAppearanceLarge" />
```

# APPENDICES (Predefined Android resources)

---

This is the definition of: `simple_spinner_dropdown_item` in which a single row holding a radio-button and text is shown.

```
<?xml version="1.0" encoding="utf-8"?>
<!--
** Copyright 2008, The Android Open Source Project
** etc...
-->
<CheckedTextView xmlns:android="http://schemas.android.com/apk/res/android" android:id="@android:id/text1"
    android:id="@android:id/text1" style="?android:attr/spinnerDropDownItemStyle"
    android:singleLine="true" android:layout_width="match_parent"
    android:layout_height="?android:attr/listPreferredItemHeight"
    android:ellipsize="marquee" />
```

# APPENDICES (EditText boxes & keyboarding)

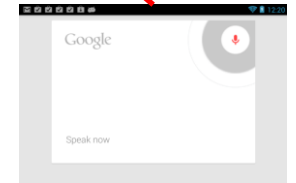
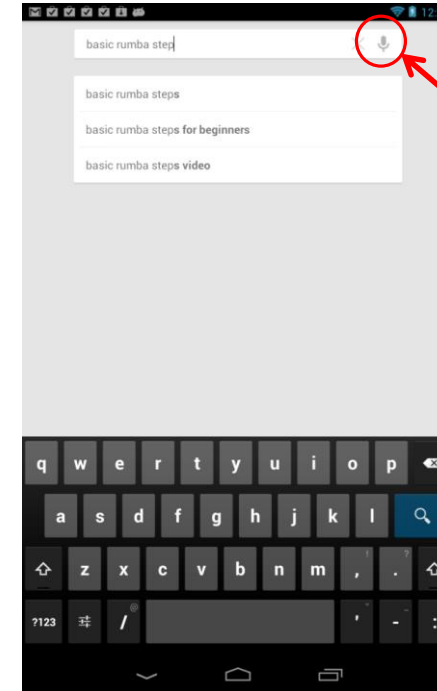
Keyboarding data into Android's applications is functionally dependent of the hardware present in the actual device.



Sliding Window in this unit exposes a hard keyboard.



This device has a permanently exposed hard keyboard and Stylus pen appropriate for handwriting



Input accepted from Virtual keyboard and/or voice recognition



IME Soft  
Keyboard

# APPENDICES (EditText boxes & keyboarding)

When the user taps on an EditText box, the Input Media Framework (IMF) provides access to

- 1. a hard (or real) keyboard (if one is present) or
- 2. a soft (or virtual) keyboard known as IME that is the most appropriated for the current input type.

You may close the virtual keyboard by tapping the hardware BackArrow key.



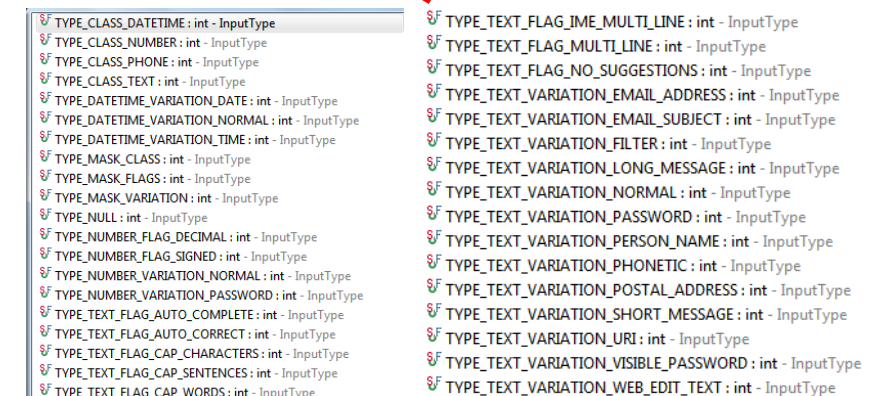
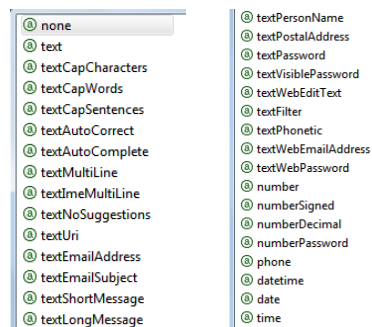
# APPENDICES (EditText boxes & keyboarding)

EditText can use either XML elements or Java code to tell the type of textual data they should accept. For example:

- XML: `android:inputType="phone"`
- Java: `editTextBox.setInputType(android.text.InputType.TYPE_CLASS_PHONE);`

Knowing the `inputType` has an impact on virtual keyboards (the software can expose the best layout for the current input class)

- Java Usage – `inputType` Classes: `editTextBox.setInputType( android.text.InputType.XXX );`
- XML Usage – `inputType` Classes
  - `<EditText ...android:inputType="numberSigned|numberDecimal" ... />`





# APPENDICES (EditText boxes & keyboarding)

Using multiple XML attributes: android:inputType="text|textCapWords"

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:layout_width="match_parent" android:layout_height="match_parent"
    android:background="#ffcccc" android:orientation="vertical"
    xmlns:android="http://schemas.android.com/apk/res/android" >
    <TextView android:layout_width="match_parent" android:layout_height="wrap_content"
        android:background="#ff0000ff" android:text="inputType: text|textCapWords"
        android:textStyle="bold" android:textSize="22sp" />
    <EditText android:id="@+id/editTextBox" android:layout_width="match_parent"
        android:layout_height="wrap_content" android:padding="5dip"
        android:textSize="18sp"
        android:inputType="text|textCapWords" />
</LinearLayout>
```

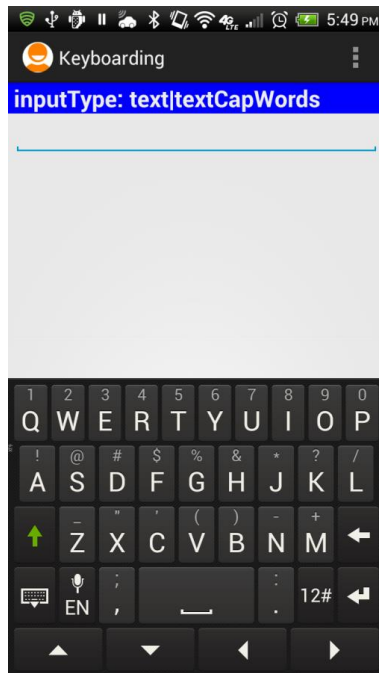
Use “pipe” symbol | to separate the options.

In this example a soft text keyboard will be used.

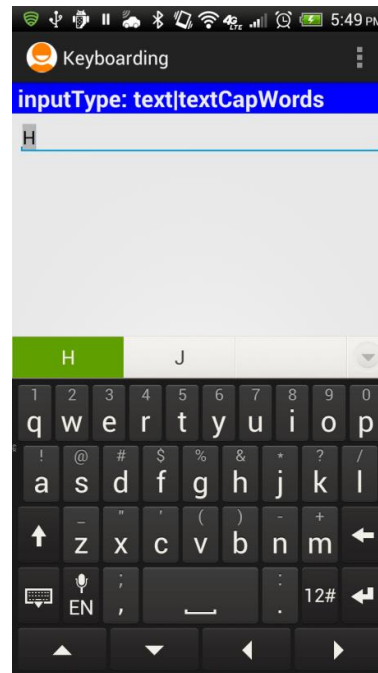
Each word will be capitalized.

# APPENDICES (EditText boxes & keyboarding)

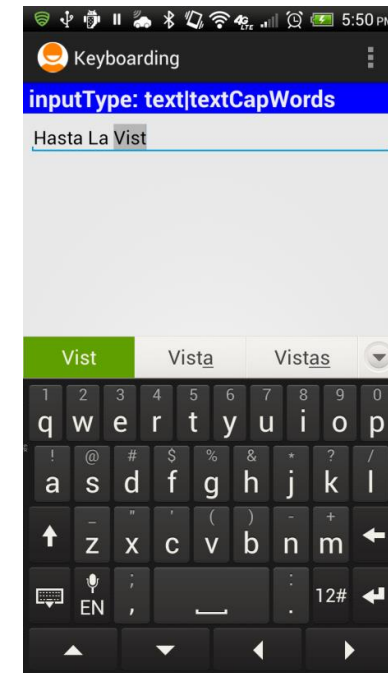
Using multiple XML attributes: android:inputType="text|textCapWords"



After tapping the EditText box to gain focus, a soft keyboard appears showing CAPITAL letters



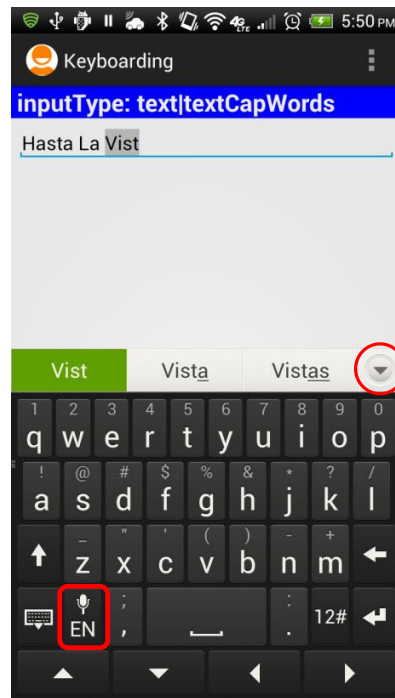
After first letter is typed the keyboard automatically switches to LOWER case mode



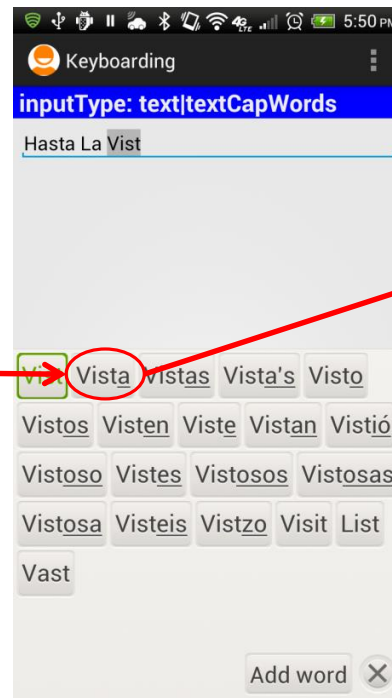
After entering space the keyboard repeats cycle beginning with UPPER case, then LOWER case letters

# APPENDICES (EditText boxes & keyboarding)

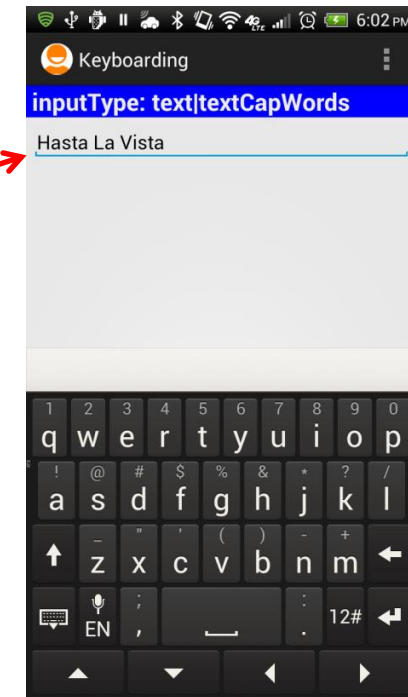
Using multiple XML attributes: `android:inputType="text|textCapWords"`



English and Spanish are the user's selected languages in this device



You may speed up typing by tapping on an option from the list of suggested words (bilingual choices)



Selected word is introduced in the EditText box

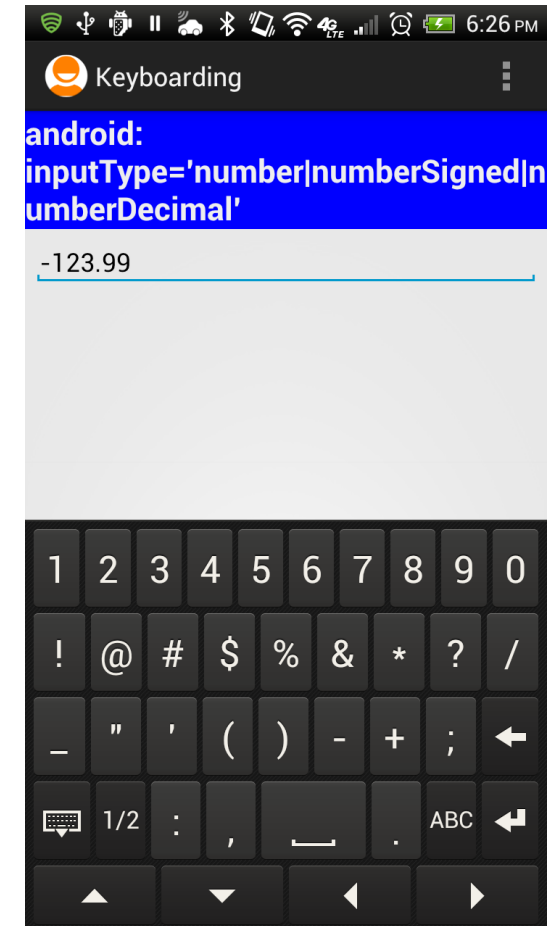
# APPENDICES (EditText boxes & keyboarding)

Using `android:inputType="number|numberSigned|numberDecimal"`

- 1. The keyboard displays numbers.
- 2. Non-numeric keys (such as !@#\$%&\*?/\_ ) are visible but disable.
- 3. Only valid numeric expressions can be entered.
- 4. Type `number|numberSigned` accepts integers.
- 5. Type `numberDecimal` accepts real numbers.

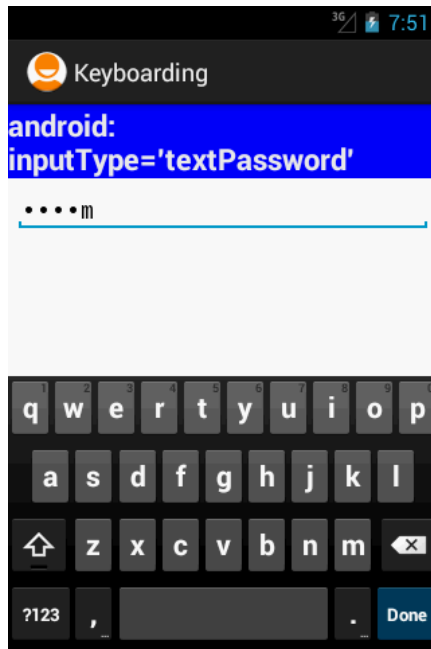
Assume the EditText field is named: `editTextBox`, In Java code we could at run-time set the input method by issuing the command:

- `editTextBox.setInputType(android.text.InputType.TYPE_CLASS_NUMBER | android.text.InputType.TYPE_NUMBER_FLAG_SIGNED);`



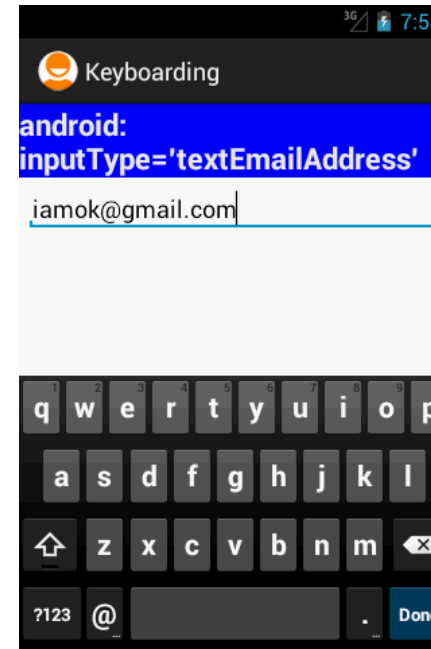
# APPENDICES (EditText boxes & keyboarding)

Using android:inputType="textPassword"



- The keyboard displays all possible keys.
- Current character is briefly displayed for verification purposes.
- The current character is hidden and a heavy-dot is displayed.

android:inputType="textEmailAddress"

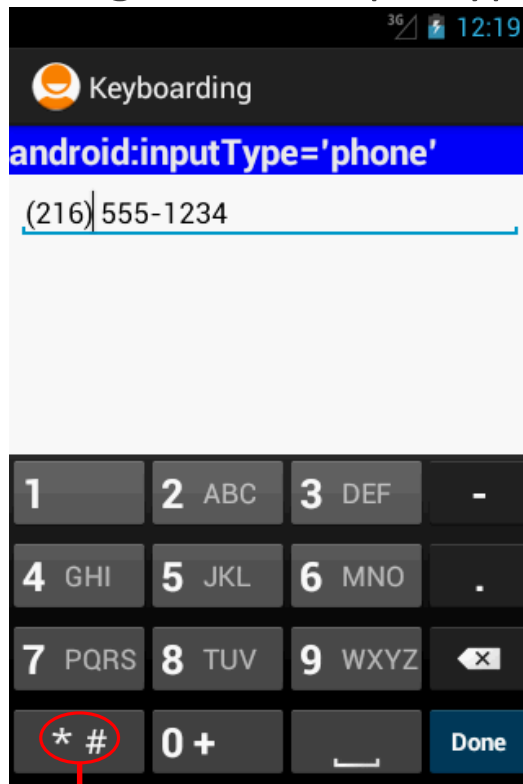


Soft keyboard shows characters used in email addresses (such as letters, @, dot).

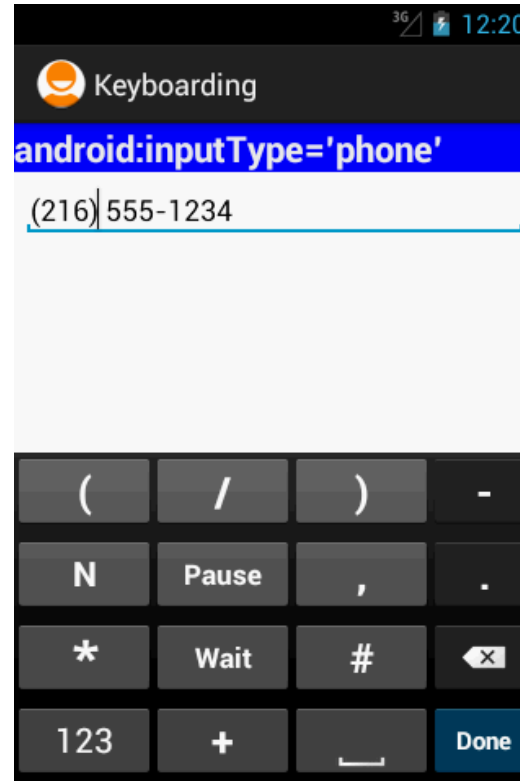
Click on [?123 ] key (lower-left) for additional characters

# APPENDICES (EditText boxes & keyboarding)

Using android:inputType="phone"



Additional symbols

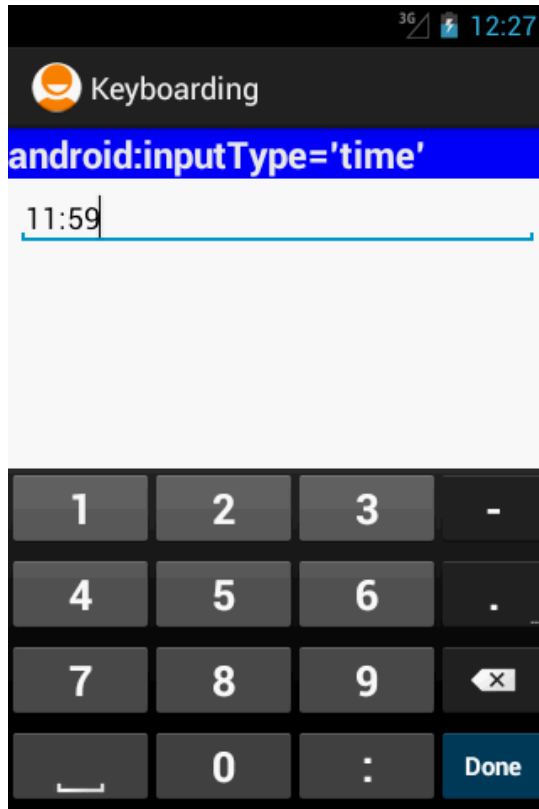


Soft keyboard displays the layout of a typical phone keypad plus additional non digit symbols such as:

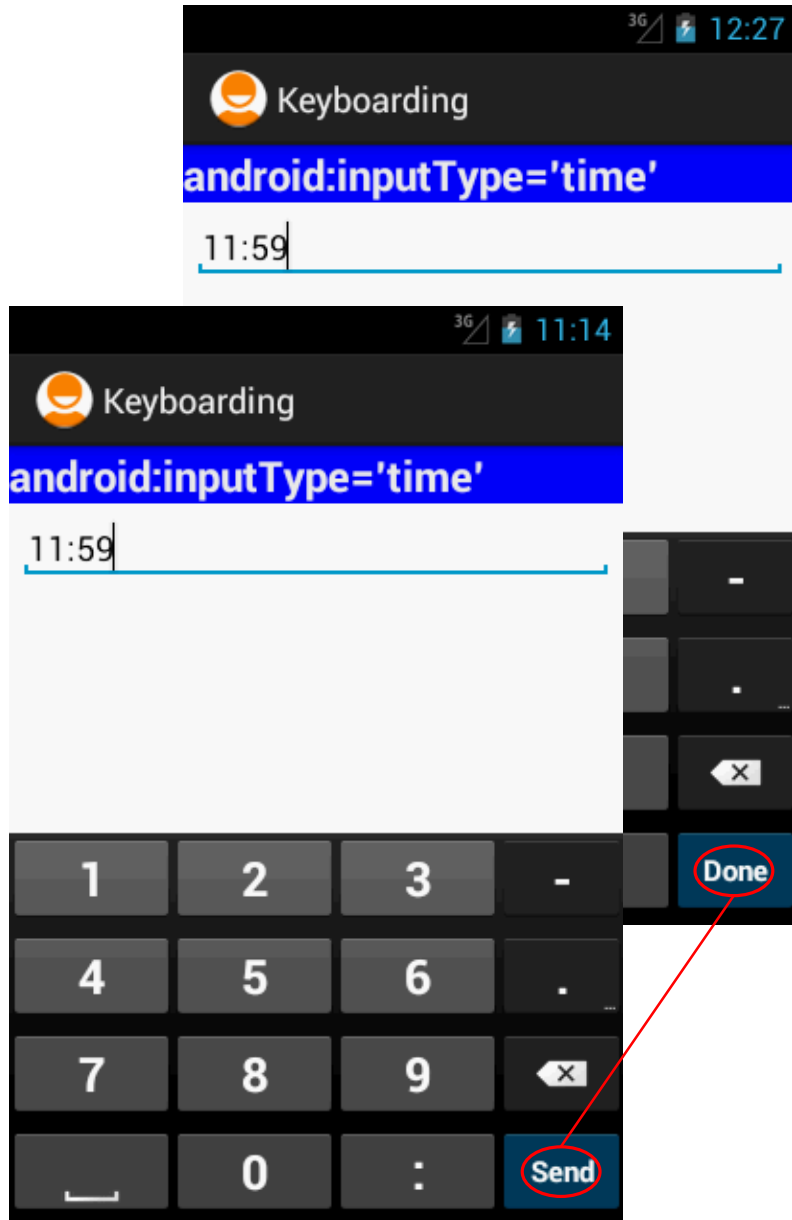
( ) . / Pause Wait # - +

# APPENDICES (EditText boxes & keyboarding)

Using android:inputType="time"



Soft keyboard displays a numerical layout.  
Only digits and colon-char ":" can be used.



# APPENDICES (EditText boxes & keyboarding)

Using `android:inputType="time"`

When clicked, the Auxiliary button DONE will removed the keyboard from the screen (default behavior).

You may change the button's caption and set a listener to catch the click event on the Auxiliary button. To do that add the following entry to your XML specification of the EditText box: `android:imeAction="actionSend"`

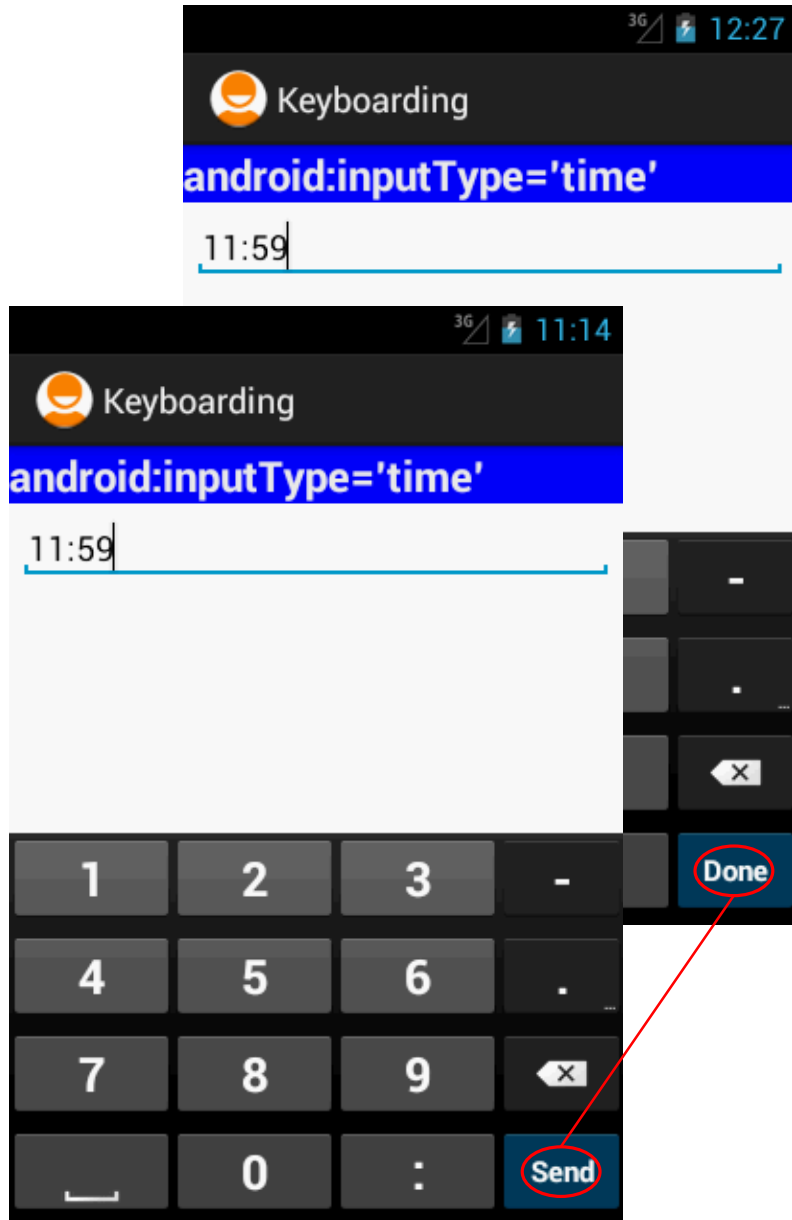
Later, your Java code could provide an implementation of the method: `editTextBox.setOnEditorActionListener()` to do something with the event.



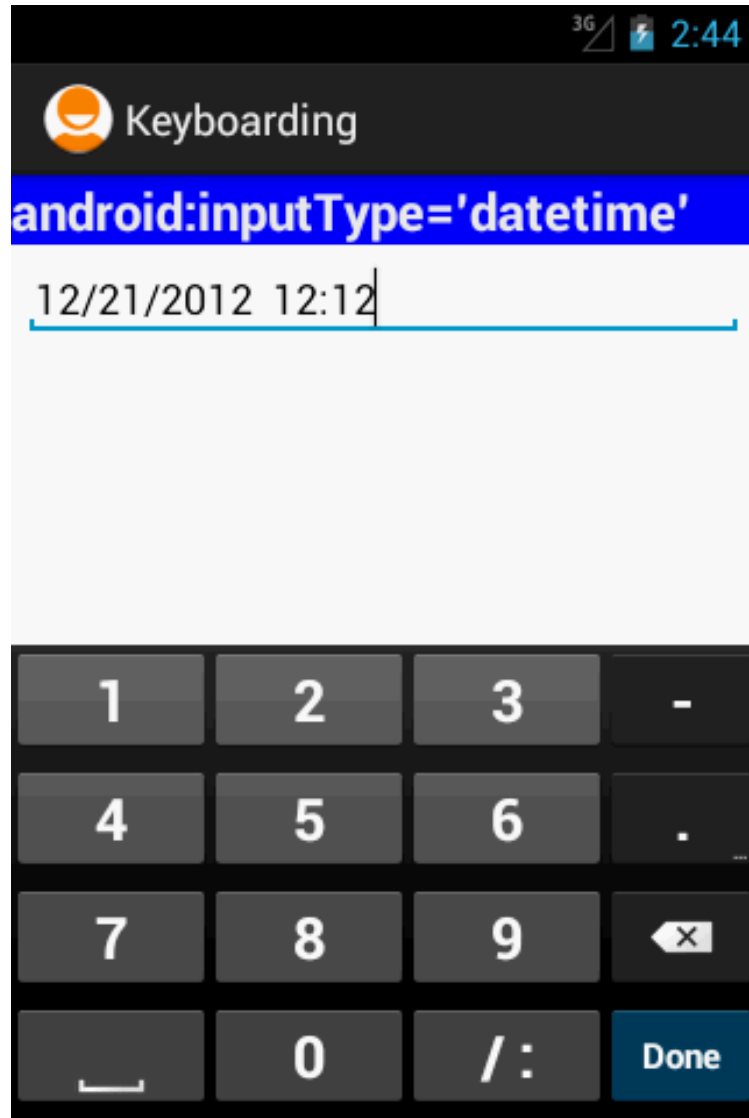
# APPENDICES (EditText boxes & keyboarding)

Using android:inputType = **"time"**

Other options for android:imeAction = **"..."** are



- ⓐ "normal"
- ⓐ "actionUnspecified"
- ⓐ "actionNone"
- ⓐ "actionGo"
- ⓐ "actionSearch"
- ⓐ "actionSend"
- ⓐ "actionNext"
- ⓐ "actionDone"
- ⓐ "actionPrevious"
- ⓐ "flagNoFullscreen"
- ⓐ "flagNavigatePrevious"
- ⓐ "flagNavigateNext"
- ⓐ "flagNoExtractUi"
- ⓐ "flagNoAccessoryAction"
- ⓐ "flagNoEnterAction"
- ⓐ "flagForceAscii"



# APPENDICES (EditText boxes & keyboarding)

---

Using android:inputTpe = **"datetime"**

Soft keyboard displays a numerical layout.

Only digits and date/time valid characters are allowed.

Examples of valid dates are:

12/21/2012 12:12

12/31/2011

12:30

# APPENDICES (EditText boxes & keyboarding)

---

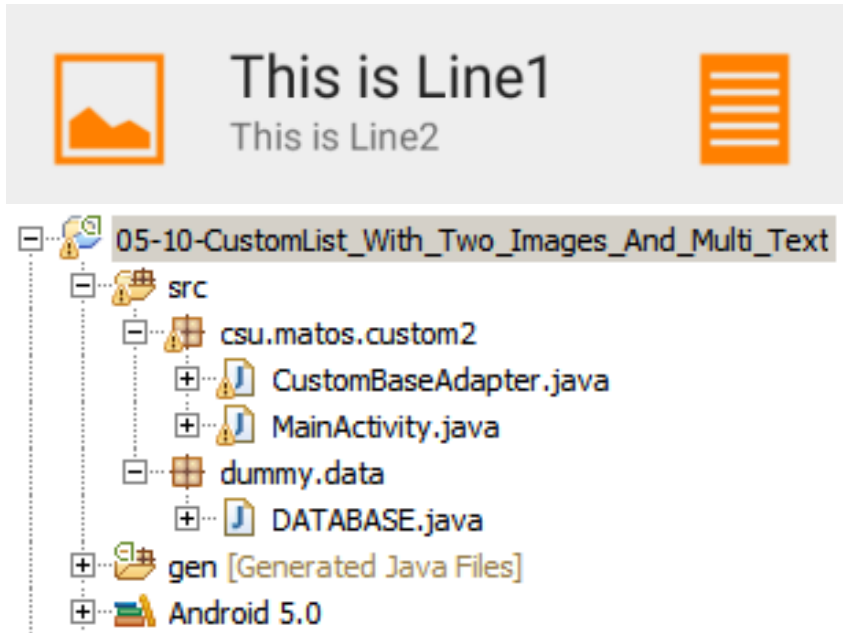
## Disable Soft Keyboarding on an EditText View

- To disable the action of the soft keyboard on an EditText you should set its input type to null, as indicated below: `editTextBox.setInputType( InputType.TYPE_NULL );`
- To temporarily hide the virtual keyboard, call the following method:

```
public void hideVirtualKeyboard() {  
    Context context = getActivity().getApplicationContext();  
    ((InputMethodManager) context.getSystemService(Activity.INPUT_METHOD_SERVICE)).toggleSoftInput(InputMethodManager.SHOW_IMPLICIT, 0);  
}
```

- To display the virtual keyboard, call the method:

```
public void showVirtualKeyboard() {  
    Context context = getActivity().getApplicationContext();  
    ((InputMethodManager) context.getSystemService(Activity.INPUT_METHOD_SERVICE)).toggleSoftInput(InputMethodManager.SHOW_IMPLICIT, 1);  
}
```



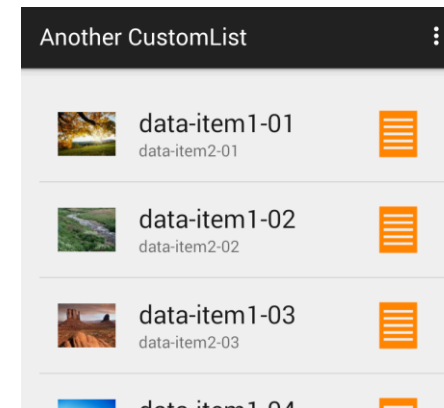
The app consists of **2 classes**: MainActivity & CustomBaseAdapter. It has **2 layouts**: activity\_main showing the list (see image on the right) and list\_row\_gui describing the structure of individual rows. Test data is placed in a separate class called DATABASE.

## APPENDICES

### (Custom list supported by a BaseAdapter)

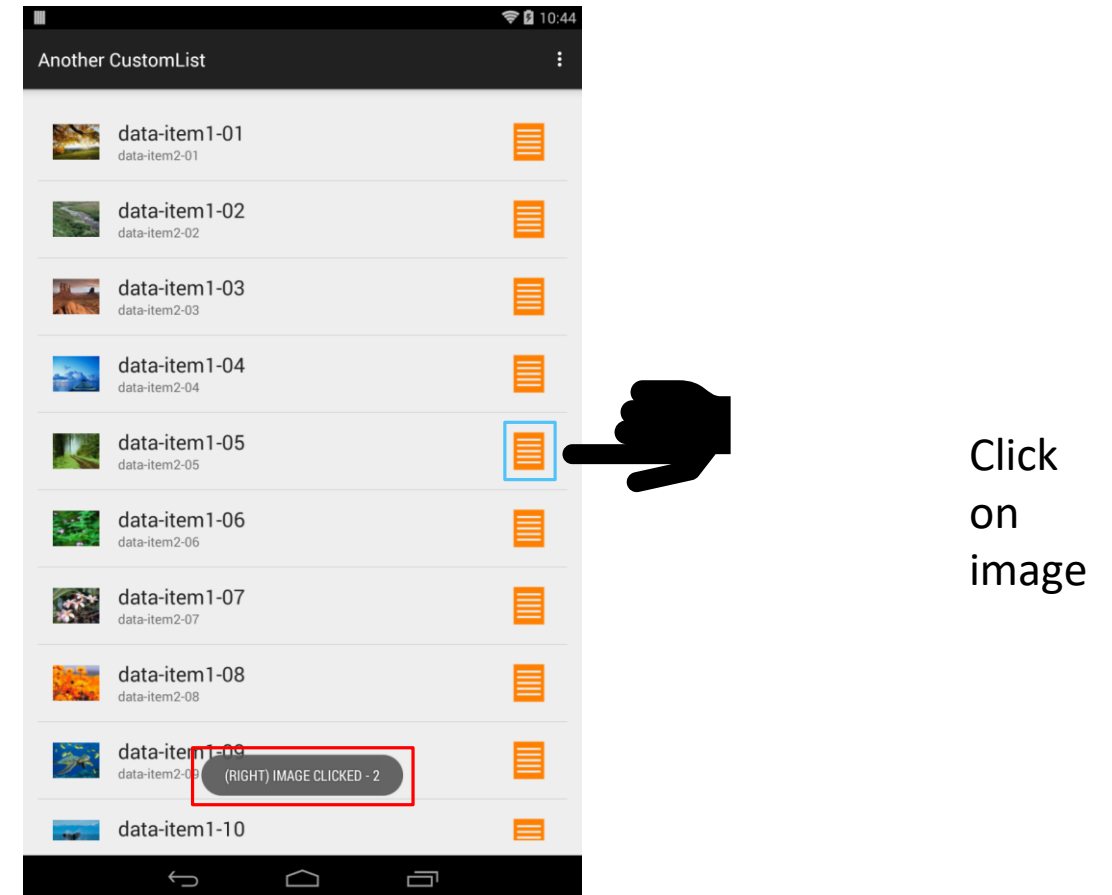
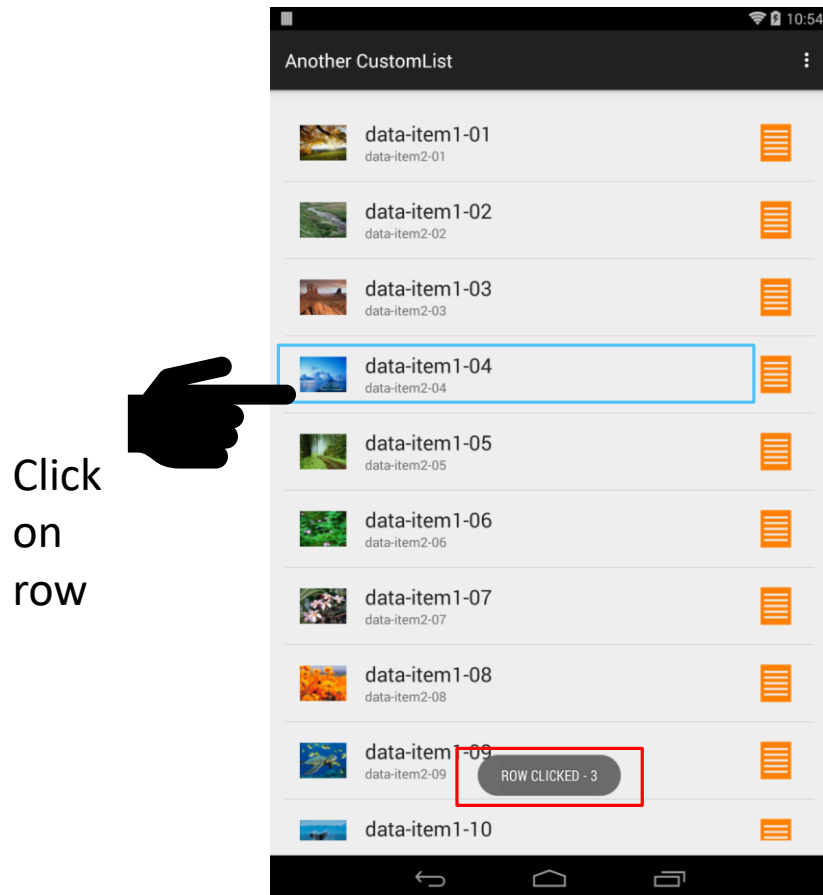
In this example a list holding rows showing multiple lines of text and images, is populated with a custom made BaseAdapter that uses the ViewHolder strategy for better performance.

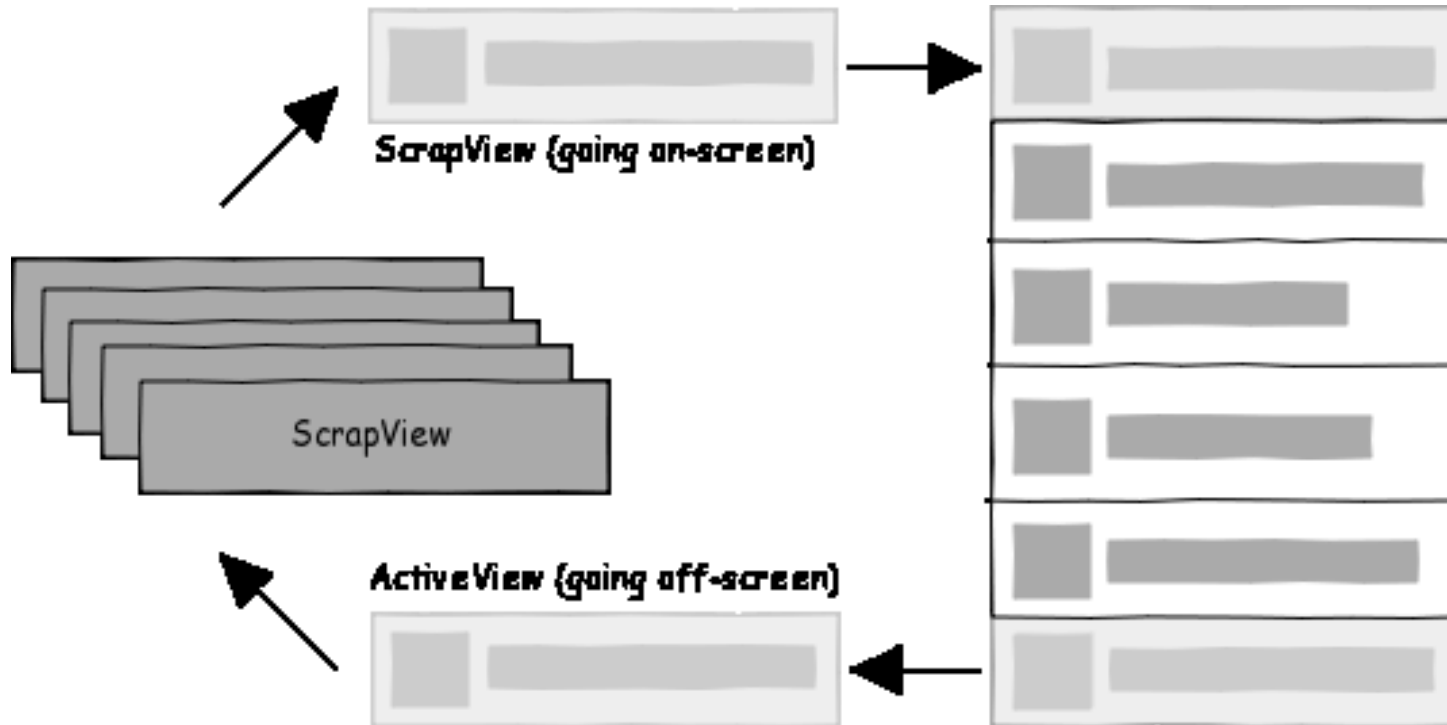
An onClick listener is set to recognize the user's tapping on the image to the right side, and another listener is set for clicking anything from the rest of the row.



# APPENDICES

## (Custom list supported by a BaseAdapter)





## APPENDICES

### (Custom list supported by a BaseAdapter)

The figure below is from "Performance Tips for Android's ListView" by Lucas Rocha <http://lucasr.org/2012/04/05/performance-tips-for-androids-listview/> [Dec, 2014]. It shows a set of rows presented to the user inside a ListView container.

When a row gets out of sight, the memory of its layout is saved in a scrapview collection silently kept by the ListView. If the row comes back to a visible state, you may reuse its scrapview skeleton instead of redoing the row from scratch.

The strategy of reusing these scrapviews is known as the ViewHolder Design Pattern. It cuts down on the number of times you have to inflate a row-layout and then get access to its internal widgets by calling the 'findViewById()' method.

When reusing the scrapviews (made available as 'convertView') all you need to do is move the appropriate data to the internal widgets and set their onClick listeners.

# APPENDICES

## (Custom list supported by a BaseAdapter)

---

Layout activity\_main.xml shows a ListView.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp" >
    <ListView
        android:id="@+id/listView1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >
    </ListView>
</LinearLayout>
```

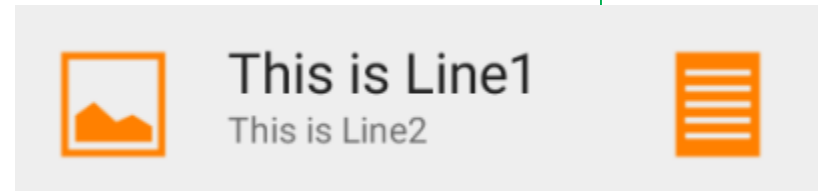


# APPENDICES

## (Custom list supported by a BaseAdapter)

Layout list\_gui\_row.xml shows a custom-made row holding two lines of text and two images.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android" xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent" android:layout_height="match_parent"
    android:orientation="horizontal"
    android:padding="16dp">
    <ImageView android:id="@+id/rowImageView1"
        android:layout_width="50dp" android:layout_height="50dp"
        android:contentDescription="@string/image_left" android:src="@drawable/ic_pic_left" />
    <LinearLayout android:layout_width="0dp"
        android:layout_height="match_parent" android:layout_marginLeft="20dp"
        android:layout_weight="2" android:orientation="vertical" >
        <TextView
            android:id="@+id/rowTextView1"
            android:layout_width="wrap_content" android:layout_height="wrap_content"
            android:text="@string/text1" android:textAppearance="?android:attr/textAppearanceLarge" />
        <TextView
            android:id="@+id/rowTextView2"
            android:layout_width="wrap_content" android:layout_height="wrap_content"
            android:text="@string/text2" android:textAppearance="?android:attr/textAppearanceSmall" />
    </LinearLayout>
    <ImageView android:id="@+id/rowImageView2"
        android:layout_width="50dp" android:layout_height="50dp"
        android:contentDescription="@string/image_right" android:src="@drawable/ic_launcher" />
</LinearLayout>
```





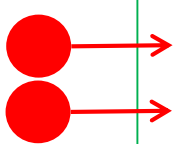
# APPENDICES

## (Custom list supported by a BaseAdapter)

---

The main activity exposes a ListView. A custom adapter is tied to the ListView. The adapter gets a reference to a test 'database' and the custom row layout.

```
public class MainActivity extends ActionBarActivity {  
    DATABASE database_records;  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        ListView listview = (ListView) findViewById(R.id.listView1);  
        //create an instance of our fake database: {[text1, text2, icon]}  
        List database = new DATABASE().dbList;  
        CustomBaseAdapter adapter = new CustomBaseAdapter(this, database, R.layout.list_row_gui);  
        listview.setAdapter(adapter);  
    } //onCreate  
}
```



# APPENDICES

## (Custom list supported by a BaseAdapter)

---

The getView method in this extended BaseAdapter inflates a supplied row layout, gets access to its internal widgets, fills them with data and set listeners on some of them.

```
public class CustomBaseAdapter extends BaseAdapter {
    Context context; int layoutToBeInflated; List<DATABASE.DbRecord> dbList;
    public CustomBaseAdapter(Context context, List<DATABASE.DbRecord> databaseList, int resource) {
        this.context = context; this.dbList = databaseList; layoutToBeInflated = resource;
    }
    @Override public int getCount() { return dbList.size(); }
    @Override public DATABASE.DbRecord getItem(int position) { return dbList.get(position); }
    @Override public long getItemId(int position) { return position; }
    //...
```

# APPENDICES

## (Custom list supported by a BaseAdapter)

---

The getView method in this extended BaseAdapter inflates a supplied row layout, gets access to its internal widgets, fills them with data and set listeners on some of them.

```
public View getView(final int position, View convertView, ViewGroup parent) {  
    // use View-Holder pattern to reduce calls to inflate, findViewById holder is a POJO for the GUI rows [textview1,textview2, img1, img2]  
    MyViewHolder holder;  
    // hopefully convertView is a scrapview already made (but out of sight)  
    View row = convertView;  
    // has this row-layout been already created?  
    if (row == null) {  
        // first time this row has to be created: (1) inflate custom layout holding images and text, (2) invoke findViewById to access its sub-components  
        LayoutInflater inflater = ((Activity) context).getLayoutInflater();  
        row = inflater.inflate(layoutToBeInflated, null);  
        holder = new MyViewHolder();  
        // plumbing - provide access to each widget in the inflated layout (two images & two lines of text)  
        holder = new MyViewHolder();  
        holder.textview1 = (TextView) row.findViewById(R.id.rowTextView1); holder.textview2 = (TextView) row.findViewById(R.id.rowTextView2);  
        holder.imageview1 = (ImageView) row.findViewById(R.id.rowImageView1); holder.imageview2 = (ImageView) row.findViewById(R.id.rowImageView2);  
        // identify this row with the POJO holder just created  
        row.setTag(holder);  
    } else { // row was already created- no need to inflate and invoke findViewById getTag() returns the object originally stored in this view  
        holder = (MyViewHolder) row.getTag();  
    }  
    //...
```

# APPENDICES

## (Custom list supported by a BaseAdapter)

---

The getView method in this extended BaseAdapter inflates a supplied row layout, gets access to its internal widgets, fills them with data and set listeners on some of them.

```
// enter(or restore) data that goes in this frame (from database 'position')
DATABASE.DbRecord dbRec = getItem(position);
holder.textview1.setText(dbRec.text1); holder.textview2.setText(dbRec.text2);
holder.imageview1.setImageResource(dbRec.img1); holder.imageview2.setImageResource(R.drawable.ic_launcher);
// EXTRA: individual listeners go here - if you need only a single listener for the entire row, put it into ActivityMain.
// This is a CLICK listener on top of the right icon (imageview2) (for example, here you start an intent to call phone[position])
holder.imageview2.setOnClickListener(new OnClickListener() {
    @Override public void onClick(View v) { Toast.makeText(context, "(RIGHT) IMAGE CLICKED - " + position, 1).show(); }
});
// row listener (user clicks on any other part of the row)
row.setOnClickListener(new OnClickListener() {
    @Override public void onClick(View v) { Toast.makeText(context, "ROW CLICKED - " + position, 1).show(); }
});
return row;
} // getView
// A humble POJO holding references to GUI widgets that are part of rows shown by the list. They have already been made and their IDs are
//known, therefore there is no need to issue 'findViewById' calls again.
public class MyViewHolder { TextView textview1, textview2; ImageView imageview1, imageview2; }
} // CustomMadeListener
```

# APPENDICES

## (Custom list supported by a BaseAdapter)

---

DATABASE.java

```
public class DATABASE { // TEST DATABASE
    public String[] text1array = { "data-item1-01", ..., "data-item1-15" }, text2array = { "data-item2-01", ..., "data-item2-15" };
    public Integer[] icon1array = { csu.matos.custom2.R.drawable.pic01_small, ..., csu.matos.custom2.R.drawable.pic15_small };
    public class DbRecord {
        public String text1, text2;
        public Integer img1;
        public DbRecord(String text1, String text2, Integer img1) { this.text1 = text1; this.text2 = text2; this.img1 = img1; }
    } //dbRecord
    // dbList is a 'database' holding a list of DbRecords:[string,string,int]
    public ArrayList<DbRecord> dbList = new ArrayList<DbRecord>();
    // populate the 'database' with data items
    public DATABASE () {
        for(int i=0; i<text1array.length; i++){ dbList.add(new DbRecord(text1array[i], text2array[i], icon1array[i]) ); }
    }
} // DATABASE
```