Lesson 5

List-Based Widgets:
Lists, Grids, and Scroll Views

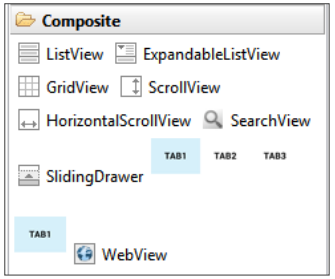
Victor Matos
Cleveland State University

List-Based Widgets

GUI Design for Selection Making

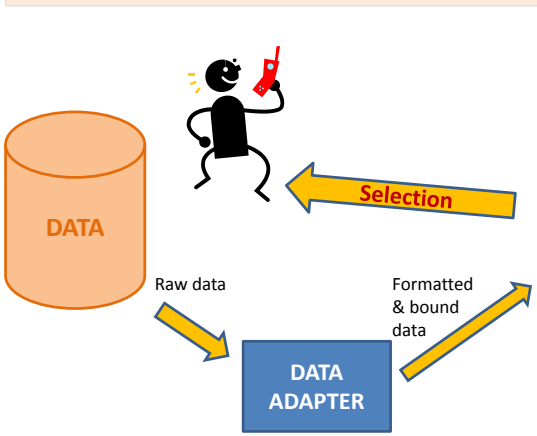
- RadioButtons and CheckButtons are widgets suitable for selecting options offered by a *small* set of choices. They are intuitive and uncomplicated; however they occupy a permanent space on the GUI (which is not a problem when only a few of them are shown)

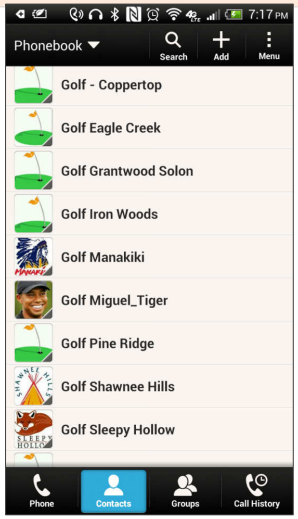
- When the set of values to choose from is large, other Android **List-Based Widgets** are more appropriate.
- Example of **List-Based Widgets** include:
  - ***ListViews,***
  - ***Spinner,***
  - ***GridView***
  - ***Image Gallery***
  - ***ScrollViews***, etc.



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.

Destination layout
Holding a **ListView**

List-Based App = ListView + Data + DataAdapter

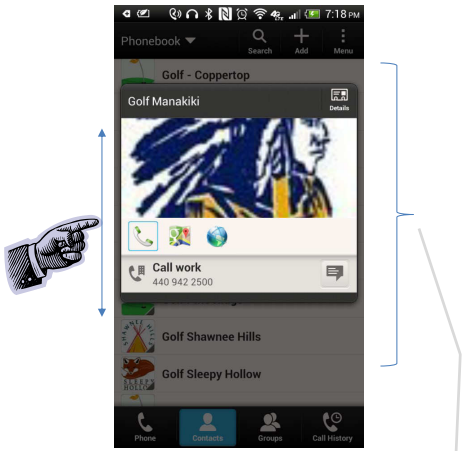
ListViews

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

ListViews 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**

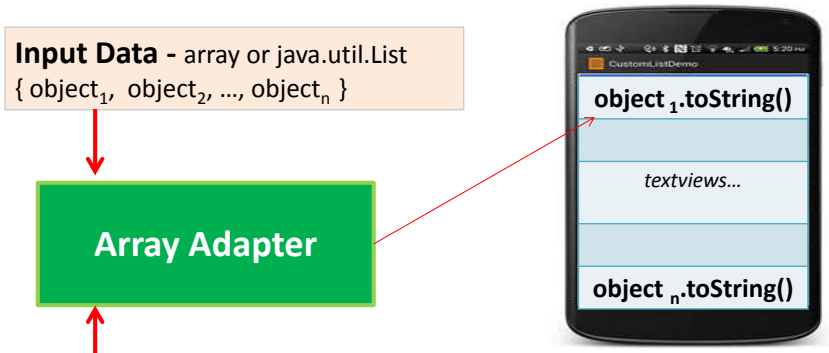
List-Based App = ListView + Data + DataAdapter

ArrayAdapter (A Data Beautifier)

- 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 ListViews 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. For a detailed sample see Example 8.

List-Based App = ListView + Data + DataAdapter

Output: 'Pretty' GUI



Input XML Specification

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


List-Based App = ListView + Data + DataAdapter

Using the ArrayAdapter<String> Class

```
String[] items = { "Data-0", "Data-1", "Data-2", "Data-3",
                   "Data-4", "Data-5", "Data-6", "Data-7"  };

ArrayAdapter<String> adapter = new ArrayAdapter<String>(
                     this,
                     android.R.layout.simple_list_item_1,
                     items );
```

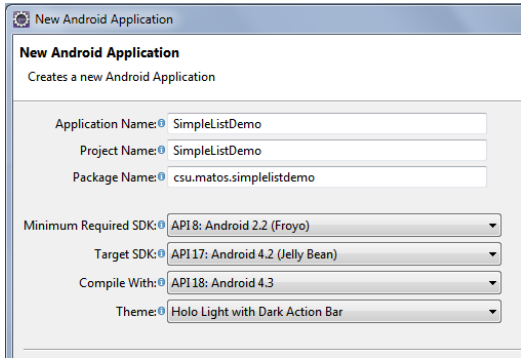
Parameters:

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

Using ListActivity + ArrayAdapter

Example1A: ListView showing a simple list (plain text)

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.



Using ListActivity + ArrayAdapter

Example1A: 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" >
    </ListView>

    <TextView
        android:id="@android:id/empty"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#ffff0000"
        android:text="empty list" />

</LinearLayout>
```

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

Android's built-in list layout

Used for empty lists

Using ListActivity + ArrayAdapter

Example1A: MainActivity (using a ListActivity !)

```
package csu.matos;

import ...

public class ListViewDemo extends ListActivity {

   TextView txtMsg;

   String[] items = {  "Data-0", "Data-1", "Data-2", "Data-3",
                       "Data-4", "Data-5", "Data-6", "Data-7"  };

   // next time try an empty list such as:
   // String[] items = {};
```

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

Data Source

```
...
    <ListView
        android:id="@android:id/list"
        android:layout_width="match_parent"
        android:layout_height="match_parent" >
    </ListView>
...
```

Fragment already defined in Layout: activity_main.xml

Using ListActivity + ArrayAdapter

Example1A: MainActivity (using a ListActivity !)

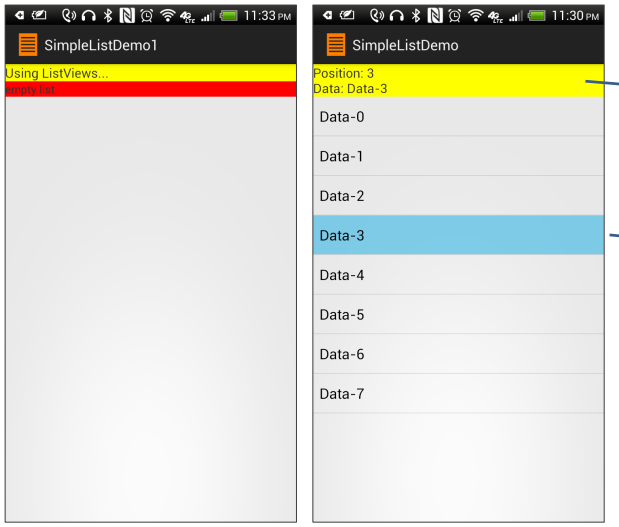
```
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      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 onListItemClick(ListView l, View v, int position, long id) {
      super.onListItemClick(l, v, position, id);
      String text = " Position: " + position + "    " + items[position];
      txtMsg.setText(text);
   }

}
```

List adapter

List Click Listener

Using ListActivity + ArrayAdapter

Example1A: MainActivity (using a ListActivity !)



Selection seen by the listener

Background flashes blue to acknowledge the users's selection

ListView displayed on a device running SDK4.3

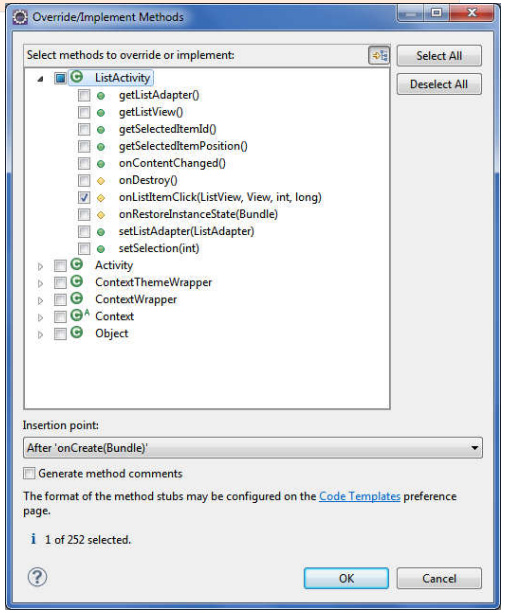
Using ListActivity + ArrayAdapter

A Comment on Example1A

Using Eclipse & ADT.
A simple way to add new code

Assume you have NOT written the click listener yet. In order to easily add a listener (or any other valid method) to the ListActivity class under construction do this:

1. Position the cursor in between the end of the onCreate method and the end of the class.
2. On the Eclipse Tool Bar Menu click Source > Override/Implement Methods… >
3. A list of pertinent methods is shown.
4. Check `onListItemClick` > Ok
5. Add your actions to the selected method.



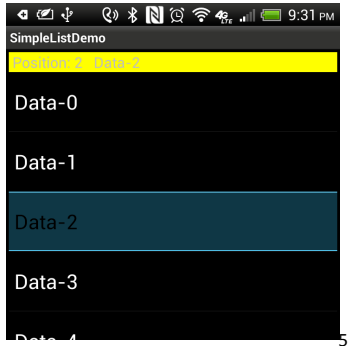
Using ListActivity + ArrayAdapter

An experiment based on Example1A

1. Open the AndroidManifest.xml file. Under the <Application> tag look for the clause
   `android:theme="@style/AppTheme"`

2. Change the previous line to the following value
   `android:theme="@android:style/Theme.Black"`

3. Try some of the other styles, such as:
   - `Theme.DeviceDefault`
   - `Theme.Dialog`
   - `Theme.Holo`
   - `Theme.Light`
   - `Theme.Panel`
   - `Theme.Translucent`
   - `Theme.Wallpaper`
   - etc.



Using ListActivity + ArrayAdapter

Another code experiment based on Example1A

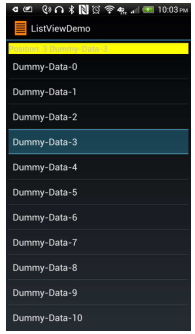
1. Open the AndroidManifest.xml file. Under the <Application> tag look for the clause `android:theme="@style/AppTheme"`

2. 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).

3. Remove from the manifest the entry *android:theme*.

4. Remove from the onCreate method the statement:
   `getListView().setBackgroundColor(Color.GRAY);`

3. Run the application again. Observe its new look.



Using Activity + ArrayAdapter

Example1B: Using Activity & ArrayAdapter

- You may use a common **Activity** class instead of a **ListActivity**.
- The Layout below uses a ListView identified as `@+id/my_list` (instead of `@android:id/list` used in the previous Example1).

```
<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" >
    </ListView>

</LinearLayout>
```

try:
wrap_content
to see limitations

Using Activity + ArrayAdapter

Example1B: 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.

Example 1B – MainActivity 1 of 2

```
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);

       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
}
```


Using Activity + ArrayAdapter

Example 1B – MainActivity 2 of 2

```
myListView.setOnItemClickListener(new OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> av, View v,
                                int position, long id) {

           String text = "Position: " + position
                       + "\nData: " + items[position];

           txtMsg.setText(text);
        }
        });
```

To provide a listener to the ListView control add the fragment above to the **onCreate** method.

Using Activity + ArrayAdapter

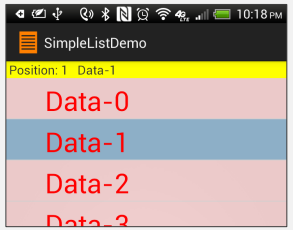
Example1C: Custom ListView

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" />
```



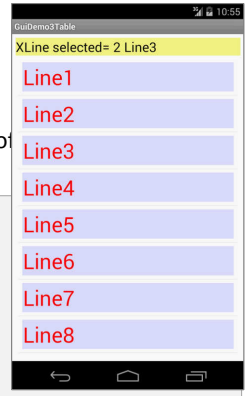
Note: As of SDK4.0 a TextView could also include an image (For example `.setDrawableLeft(some_image)`)

Using Activity + ArrayAdapter

Example1C: Custom ListView

You may also create the 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.

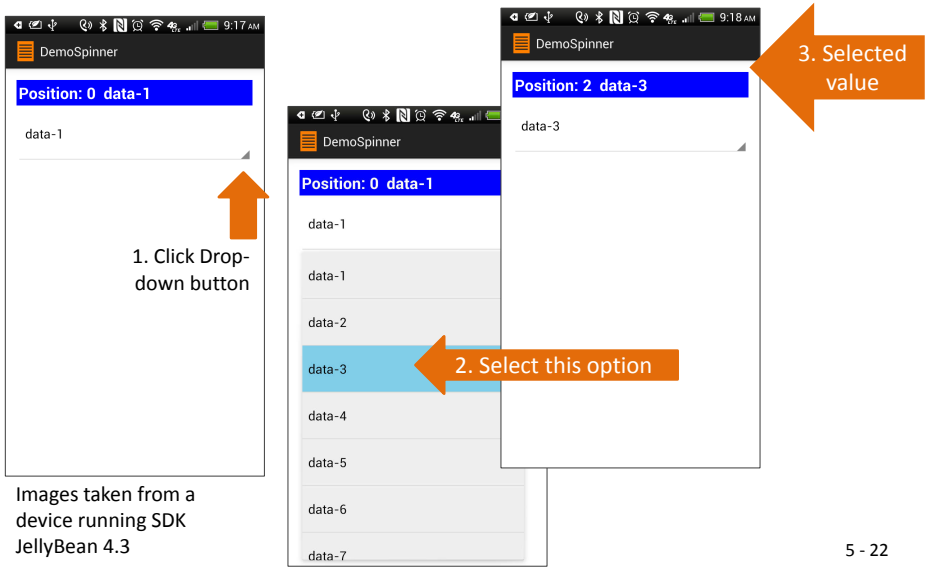
```
<!-- my_custom_line3 -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="6dp" >
    <TextView
        android:id="@+id/my_custom_textview3"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

The Spinner Widget

- 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

Example2: Using the Spinner Widget

Example 2. 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.

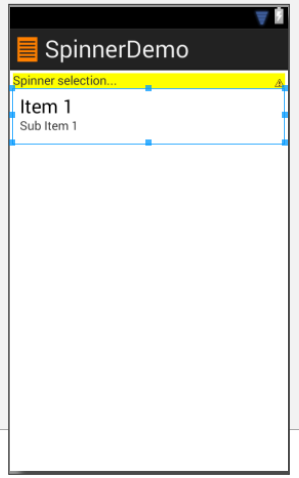


Images taken from a device running SDK JellyBean 4.3

Using the Spinner Widget

Example2: Spinner Demo - 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"
    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>
```



Using the Spinner Widget

Example2: Spinner Demo - MainActivity 1 of 2

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

   // GUI objects
   TextView txtMsg;
   Spinner spinner;

   // options to be offered by the 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);


      // use adapter to bind items array to GUI layout
      ArrayAdapter<String> adapter = new ArrayAdapter<String>(
            this,
            android.R.layout.simple_spinner_dropdown_item,
            items);
```

Using the Spinner Widget

Example2: Spinner Demo - MainActivity 2 of 2

```
        // bind everything together
        spinner.setAdapter(adapter);

        // add spinner a listener so user can meake selections by tapping an item
        spinner.setOnItemSelectedListener(this);

    }
    // next two methods implement the spinner's listener
    @Override
    public void onItemSelected(AdapterView<?> parent, View v, int position,
          long id) {
        // echo on the textbox the user's selection
        txtMsg.setText(items[position]);
    }

    @Override
    public void onNothingSelected(AdapterView<?> arg0) {
        // TODO do nothing - needed by the interface

    }

}
```

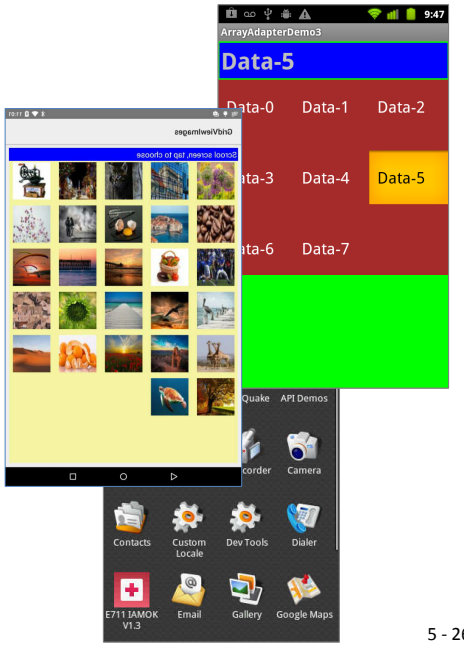

The GridView Widget

GridView

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

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

Grid cells can show text and/or images



The GridView Widget

GridView: Useful Properties

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 **dip**s.
- **android:stretchMode**
  indicates how to modify image size when there is available space not taken up by columns or spacing .

The GridView Widget

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).

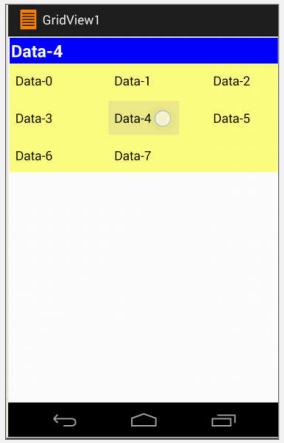
With **android:stretchMode** set to *columnWidth*, the three columns will each expand by 3-4 pixels to use up the remaining 10 pixels.

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

The GridView Widget

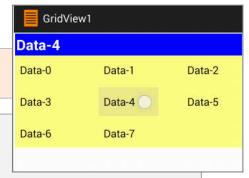
Example3A: GridView Demo - Layout

```
<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="2dip"   />

    <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>
```



The GridView Widget

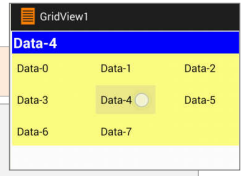
Example3A: GridView Demo – MainActivity 1 of 2




```
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(icicle);
                setContentView(R.layout.activity_main);

                txtMsg = (TextView) findViewById(R.id.txtMsg);
```


The GridView Widget

Example3A: GridView Demo – MainActivity 2 of 2




```
      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
```

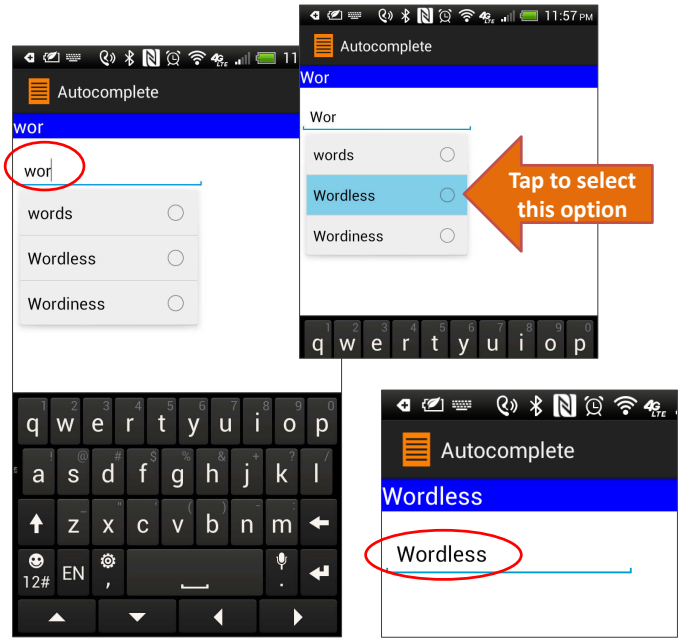

The AutoComplete TextView Widget

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.

NOTE: For other features of the TextView control see Appendix B

The AutoComplete TextView Widget



Example 4.
A list of selected words beginning with "**wor**" or "**set**" is being watched.

If any of these prefixes (3 letters) are entered the TextWatcher mechanism shows an option list.

The AutoComplete TextView Widget

Example4: AutoComplete Demo - 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="#ffffffff"
        android:background="#ff0000ff" >
       </TextView>

    <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>
```

Wait 3 chars to work

The AutoComplete TextView Widget

Example4: AutoComplete Demo – MainActivity 1 of 2

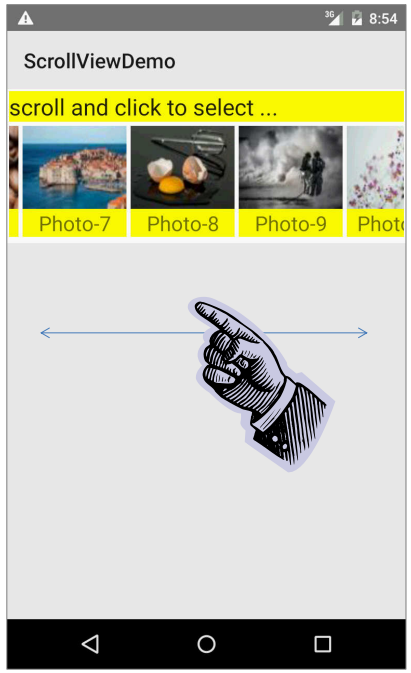
```
public class ArrayAdapterDemo4 extends Activity implements TextWatcher {

  TextView txtMsg;

  AutoCompleteTextView txtAutoComplete;

  String[] items = { "words", "starting", "with", "set", "Setback",
      "Setline", "Setoffs", "Setouts", "Setters", "Setting",
      "Settled", "Settler", "Wordless", "Wordiness", "Adios" };

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

        setContentView(R.layout.activity_main);

       txtMsg = (TextView) findViewById(R.id.txtMsg);
```


The AutoComplete TextView Widget

Example4: AutoComplete Demo – MainActivity 2 of 2

```
  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
```

The HorizontalScrollView Widget



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.

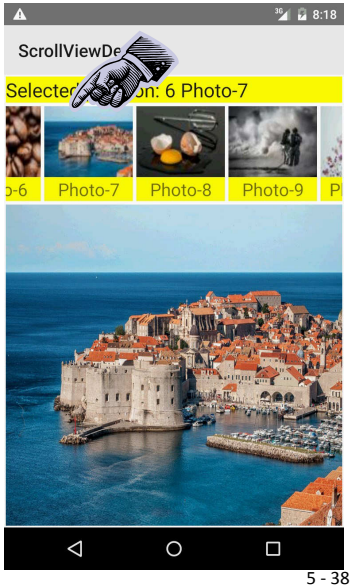
In our example, when the user clicks on a thumbnail the app responds by displaying a high-resolution version of the image

+. A typical thumbnail size is 100x100 pixels (or less).

The HorizontalScrollView Widget

Example5: HorizontalScrollView Demo

- 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.



The HorizontalScrollView Widget

Example5: How to make a thumbnail ?

- **Option-1.** The 100x100 thumbnails shown below were made visiting the site: http://makeathumbnail.com
- **Option-2**. Upload individual images to the Android_Asset_Studio_Tool http://android-ui-utils.googlecode.com/hg/asset-studio/dist/index.html

The free high quality images used in this example were obtained from **https://pixabay.com/**

The HorizontalScrollView Widget

Example5: How to make a thumbnail ?

GRAPHICAL RESOURCES

High-quality Pictures – Free distribution
Pixabay ©. Available at https://pixabay.com/
Last visited June 30, 2017.

Thumbnail makers
MakeAThumbnail ©. Available at http://makeathumbnail.com
Last visited June 30, 2017.

Android Asset Studio ©. Available at https://romannurik.github.io/AndroidAssetStudio/
Last visited June 30, 2017

The HorizontalScrollView Widget

Example5: Populating The HorizontalScrollView Widget

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

2. 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.

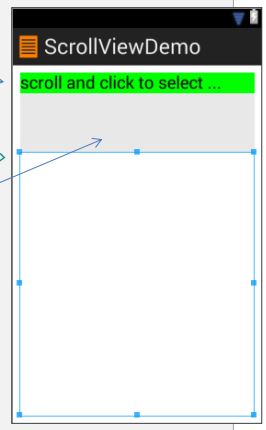
3. After the current *frame* is filled with data, it will be added to the growing set of views hosted by the *scrollViewgroup* container (scrollViewgroup is nested inside the horizontal scroller).

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

The HorizontalScrollView Widget

Example5: HorizontalScrollView Demo – Layout 1 of 2

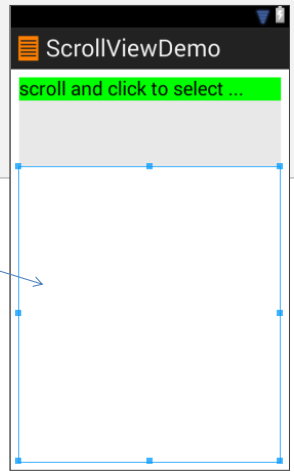
```
<?xml version="1.0" encoding="UTF-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="#ffffffff"
    android:orientation="vertical"
    android:padding="2dp" >

    <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" >
        </LinearLayout>

    </HorizontalScrollView>
```



The HorizontalScrollView Widget

Example5: HorizontalScrollView Demo – Layout 2 of 2

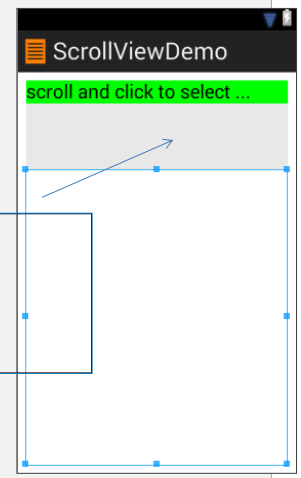
```
    <ImageView
        android:id="@+id/imageSelected"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="2" />

</LinearLayout>
```



The HorizontalScrollView Widget

Example5: Layout: *frame_icon_caption.xml*

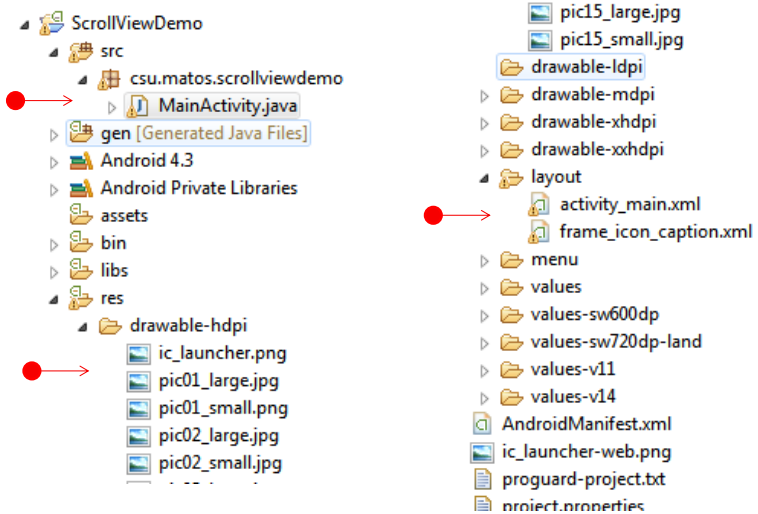
```
<?xml version="1.0" encoding="UTF-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:padding="2dp"
  android:orientation="vertical" >

  <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:gravity="center"
    android:textSize="20sp" />

</LinearLayout>
```



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

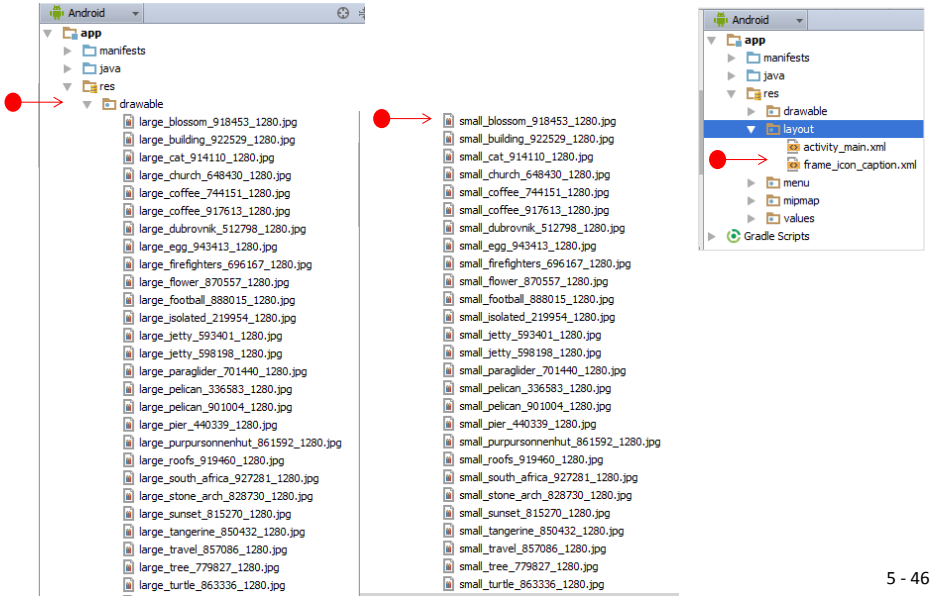
The HorizontalScrollView Widget

Example5: App's Structure - Eclipse's Package Explorer



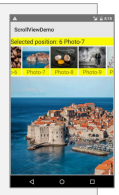
The HorizontalScrollView Widget

Example5: App's Structure - Android Studio Package Explorer



The HorizontalScrollView Widget

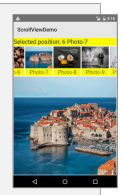
Example5: HorizontalScrollView Demo – MainActivity 1 of 5



```
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;

    //frame captions
    String[] items = {"Photo-1", "Photo-2", "Photo-3", "Photo-4", "Photo-5",
            "Photo-6",  "Photo-7",  "Photo-8",  "Photo-9",  "Photo-10", "Photo-11",
            "Photo-12", "Photo-13", "Photo-14", "Photo-15", "Photo-16", "Photo-17",
            "Photo-18", "Photo-19", "Photo-20", "Photo-21", "Photo-22", "Photo-23",
            "Photo-24", "Photo-25", "Photo-26", };
```


The HorizontalScrollView Widget

Example5: HorizontalScrollView Demo – MainActivity 2 of 5

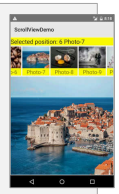


```
    //frame-icons ( 100X100 thumbnails )
    Integer[] thumbnails = {R.drawable.small_blossom_918453_1280
            , R.drawable.small_building_922529_1280
            , R.drawable.small_cat_914110_1280
            , R.drawable.small_church_648430_1280
            , R.drawable.small_coffee_744151_1280
            , R.drawable.small_coffee_917613_1280
            , R.drawable.small_dubrovnik_512798_1280
            , R.drawable.small_egg_943413_1280
            , R.drawable.small_firefighters_696167_1280
            , R.drawable.small_flower_870557_1280
            , R.drawable.small_football_888015_1280
            , R.drawable.small_isolated_219954_1280
            , R.drawable.small_jetty_593401_1280
            , R.drawable.small_jetty_598198_1280
            , R.drawable.small_paraglider_701440_1280
            , R.drawable.small_pelican_336583_1280
            , R.drawable.small_pelican_901004_1280
            , R.drawable.small_pier_440339_1280
            , R.drawable.small_purpursonnenhut_861592_1280
            , R.drawable.small_roofs_919460_1280
            , R.drawable.small_south_africa_927281_1280
            , R.drawable.small_stone_arch_828730_1280
            , R.drawable.small_sunset_815270_1280
            , R.drawable.small_tangerine_850432_1280
            , R.drawable.small_travel_857086_1280
            , R.drawable.small_tree_779827_1280
            , R.drawable.small_turtle_863336_1280};
```

The HorizontalScrollView Widget

Example5: HorizontalScrollView Demo – MainActivity 3 of 5

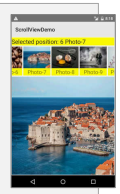
```
    //frame-icons ( 100X100 thumbnails )
    Integer[] largeImages = {R.drawable.large_blossom_918453_1280
            , R.drawable.large_building_922529_1280
            , R.drawable.large_cat_914110_1280
            , R.drawable.large_church_648430_1280
            , R.drawable.large_coffee_744151_1280
            , R.drawable.large_coffee_917613_1280
            , R.drawable.large_dubrovnik_512798_1280
            , R.drawable.large_egg_943413_1280
            , R.drawable.large_firefighters_696167_1280
            , R.drawable.large_flower_870557_1280
            , R.drawable.large_football_888015_1280
            , R.drawable.large_isolated_219954_1280
            , R.drawable.large_jetty_593401_1280
            , R.drawable.large_jetty_598198_1280
            , R.drawable.large_paraglider_701440_1280
            , R.drawable.large_pelican_336583_1280
            , R.drawable.large_pelican_901004_1280
            , R.drawable.large_pier_440339_1280
            , R.drawable.large_purpursonnenhut_861592_1280
            , R.drawable.large_roofs_919460_1280
            , R.drawable.large_south_africa_927281_1280
            , R.drawable.large_stone_arch_828730_1280
            , R.drawable.large_sunset_815270_1280
            , R.drawable.large_tangerine_850432_1280
            , R.drawable.large_travel_857086_1280
            , R.drawable.large_tree_779827_1280
            , R.drawable.large_turtle_863336_1280};
```



The HorizontalScrollView Widget

Example5: HorizontalScrollView Demo – MainActivity 4 of 5

```
    @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);
```



The HorizontalScrollView Widget

Example5: HorizontalScrollView Demo – MainActivity 5 of 5

```
            //each single frame gets its own click listener
            singleFrame.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {

                    String text = "Selected position: " + singleFrame.getId()
                            + " " + items[singleFrame.getId()];
                    txtMsg.setText(text);
                    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);

        //ALTERNATIVE SOLUTION (for APIs prior to API-21)
        Drawable selectedLargeImage = ContextCompat.getDrawable(getApplication(),
                                                        largeImages[position]);
        largeImageSelected.setImageDrawable(selectedLargeImage);

    }
}
```

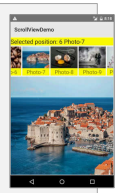


Image-Based GridViews (*again…*)

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:

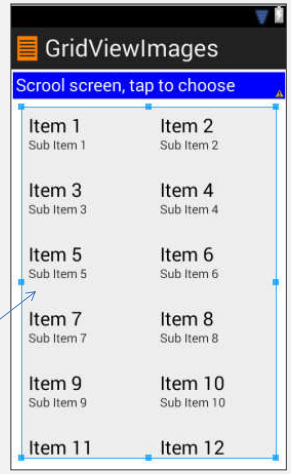
1. A screen shows an array of thumbnails.
2. The user makes her selection by tapping on one of them.
3. The app displays on a new screen a bigger & better image of the selected option.
4. The programmer must provide a custom data adapter to manage the displaying of thumbnails from the data set.

This example is based on the tutorial:
http://developer.android.com/guide/topics/ui/layout/gridview.html

Using The GridView Widget to Show Images

Example6: GridView Images Demo – Layout1 (activity_main)

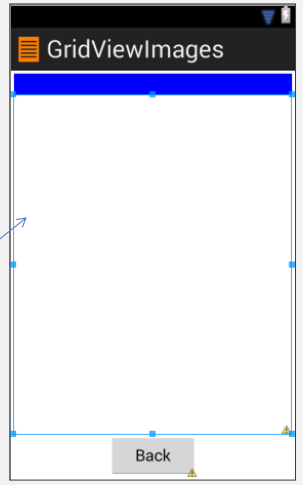
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="3dp" >

    <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="#ffffffff"
        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>
```



Using The GridView Widget to Show Images

Example6: GridView Images Demo – Layout2 (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:padding="3dp"
    android:orientation="vertical" >

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

    <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>
```

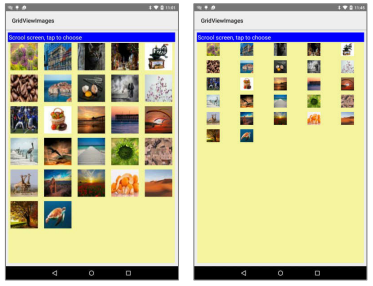


Using The GridView Widget to Show Images

Example6: GridView Images Demo – res/values/dimens/

```
<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>
```

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.



On the left:

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

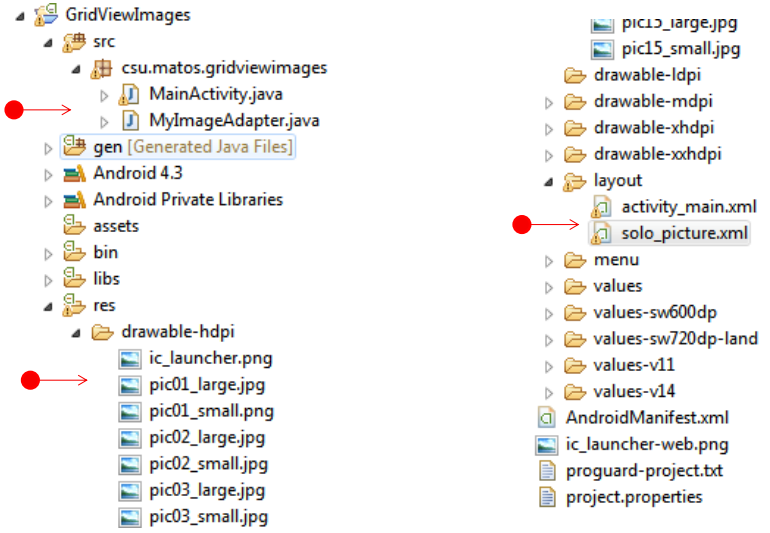
On the right:

```
imageView.setLayoutParams(new
        GridView.LayoutParams(100, 100));
```

(*see class MyImageAdapter*)

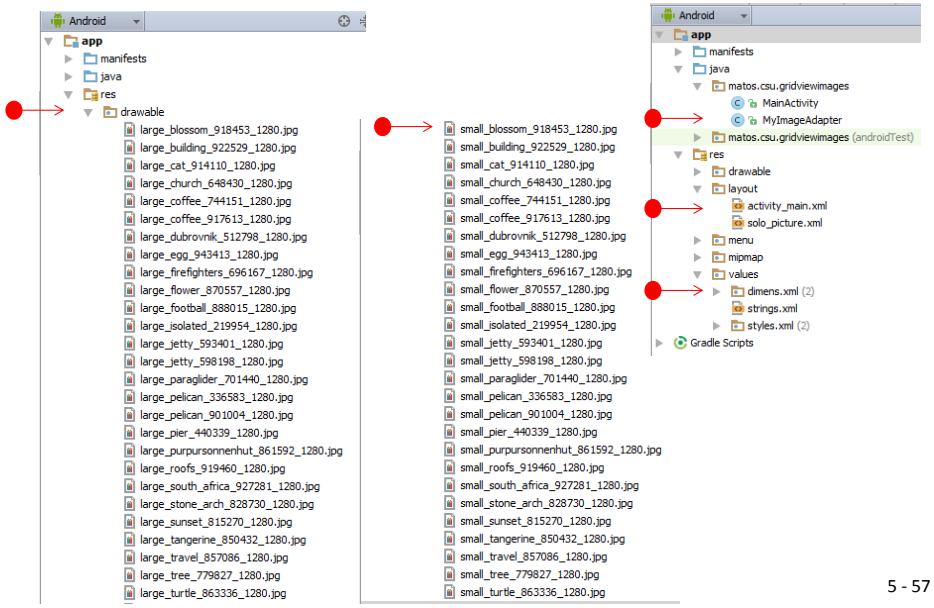
Using The GridView Widget to Show Images

Example6: App's Structure – Eclipse's Package Explorer



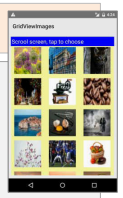
Using The GridView Widget to Show Images

Example6: App's Structure – Android Studio



Using The GridView Widget to Show Images

Example6: MainActivity 1 of 5



```
public class MainActivity extends Activity {

    //GUI control  bound to screen1 (holding GidView)
    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-2", "Photo-3", "Photo-4", "Photo-5",
            "Photo-6",  "Photo-7",  "Photo-8",  "Photo-9",  "Photo-10", "Photo-11",
            "Photo-12", "Photo-13", "Photo-14", "Photo-15", "Photo-16", "Photo-17",
            "Photo-18", "Photo-19", "Photo-20", "Photo-21", "Photo-22", "Photo-23",
            "Photo-24", "Photo-25", "Photo-26", };
```

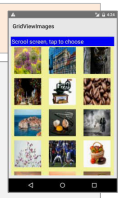

Using The GridView Widget to Show Images

Example6: MainActivity 2 of 5

```
    //frame-icons ( 100X100 thumbnails )
    Integer[] thumbnails = {R.drawable.small_blossom_918453_1280
            , R.drawable.small_building_922529_1280
            , R.drawable.small_cat_914110_1280
            , R.drawable.small_church_648430_1280
            , R.drawable.small_coffee_744151_1280
            , R.drawable.small_coffee_917613_1280
            , R.drawable.small_dubrovnik_512798_1280
            , R.drawable.small_egg_943413_1280
            , R.drawable.small_firefighters_696167_1280
            , R.drawable.small_flower_870557_1280
            ,R.drawable.small_football_888015_1280
            ,R.drawable.small_isolated_219954_1280
            ,R.drawable.small_jetty_593401_1280
            ,R.drawable.small_jetty_598198_1280
            ,R.drawable.small_paraglider_701440_1280
            ,R.drawable.small_pelican_336583_1280
            ,R.drawable.small_pelican_901004_1280
            ,R.drawable.small_pier_440339_1280
            ,R.drawable.small_purpursonnenhut_861592_1280
            ,R.drawable.small_roofs_919460_1280
            ,R.drawable.small_south_africa_927281_1280
            ,R.drawable.small_stone_arch_828730_1280
            ,R.drawable.small_sunset_815270_1280
            ,R.drawable.small_tangerine_850432_1280
            ,R.drawable.small_travel_857086_1280
            ,R.drawable.small_tree_779827_1280
            ,R.drawable.small_turtle_863336_1280 };
```


Using The GridView Widget to Show Images

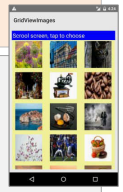
Example6: MainActivity 3 of 5



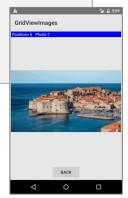
```
    //large images (high quality pictures)
    Integer[] largeImages = {R.drawable.large_blossom_918453_1280
            , R.drawable.large_building_922529_1280
            , R.drawable.large_cat_914110_1280
            , R.drawable.large_church_648430_1280
            , R.drawable.large_coffee_744151_1280
            , R.drawable.large_coffee_917613_1280
            , R.drawable.large_dubrovnik_512798_1280
            , R.drawable.large_egg_943413_1280
            , R.drawable.large_firefighters_696167_1280
            , R.drawable.large_flower_870557_1280
            ,R.drawable.large_football_888015_1280
            ,R.drawable.large_isolated_219954_1280
            ,R.drawable.large_jetty_593401_1280
            ,R.drawable.large_jetty_598198_1280
            ,R.drawable.large_paraglider_701440_1280
            ,R.drawable.large_pelican_336583_1280
            ,R.drawable.large_pelican_901004_1280
            ,R.drawable.large_pier_440339_1280
            ,R.drawable.large_purpursonnenhut_861592_1280
            ,R.drawable.large_roofs_919460_1280
            ,R.drawable.large_south_africa_927281_1280
            ,R.drawable.large_stone_arch_828730_1280
            ,R.drawable.large_sunset_815270_1280
            ,R.drawable.large_tangerine_850432_1280
            ,R.drawable.large_travel_857086_1280
            ,R.drawable.large_tree_779827_1280
            ,R.drawable.large_turtle_863336_1280 };
```

Using The GridView Widget to Show Images

Example6: MainActivity 4 of 5



```
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
```



Using The GridView Widget to Show Images

Example6: MainActivity 5 of 5

```
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
```


Using The GridView Widget to Show Images

Example6: Custom Adapter - MyImageAdapter 1 of 2

```
// 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;
   }
```


Using The GridView Widget to Show Images

Example6: Custom Adapter - MyImageAdapter 2 of 2

```
  // create a view for each thumbnail in the data set
  public View getView(int position, View convertView, ViewGroup parent) {

     ImageView imageView;

     // if possible, reuse (convertView) image already held in cache
     if (convertView == null) {
        // new image in GridView formatted to:
        // 100x75 pixels (its actual size)
        // center-cropped, and 5dp padding all around

        imageView = new ImageView(context);
        imageView.setLayoutParams( new GridView.LayoutParams(100, 75) );
        imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
        imageView.setPadding(5, 5, 5, 5);

     } else {
        imageView = (ImageView) convertView;
     }

     imageView.setImageResource(smallImages[position]);

     return imageView;
  }

}//MyImageAdapter
```

Using The GridView Widget to Show Images

Example6: Custom Adapter - MyImageAdapter 2 of 2

```
// 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.setLayoutParams(new GridView.LayoutParams(100, 100));//NOT a good practice
        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
```


Using The GridView Widget to Show Images

Results generated by Example6 running on different devices

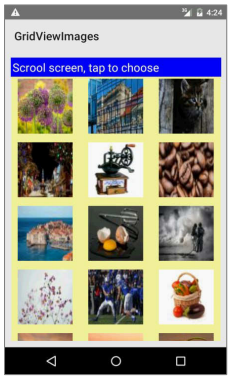


Image taken from the Emulator

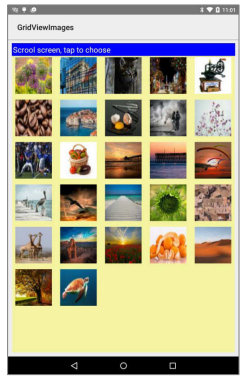


Image displayed on a Nexus7 (1028x728) tablet. The GridView's clause:

`android:numColumns="auto_fit"`

determines the best way to fill up each row.

Using The GridView Widget to Show Images

Example6B: An Experiment - 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.

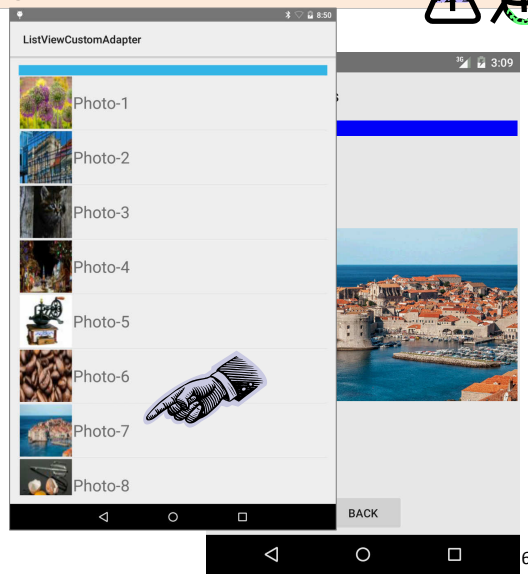
Using The GridView Widget to Show Images

Example6B: An Experiment - Changing from GridView to ListView

The new app should display the following screens.

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

More on this issue on Example 8.

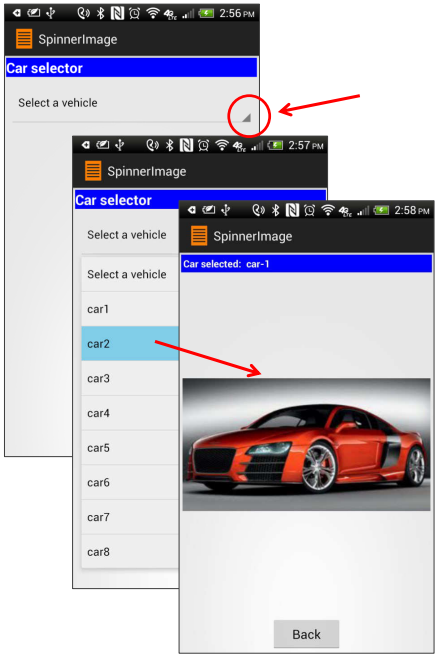


Using The Spinner Widget (again…)

This is a simple variation of the previous example.

A list of choices is offered through a drop-down spinner control.

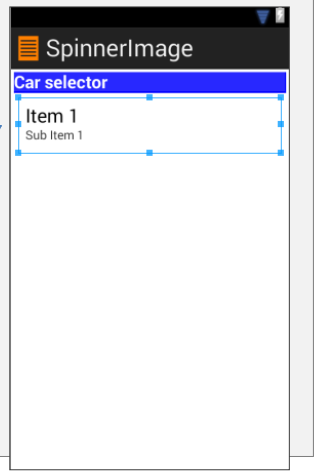
The user taps on a row and an image for the selected choice is displayed on a new screen.



Using The Spinner Widget (again…)

Example7: Spinner Demo2 - Layout1 (activity_main)

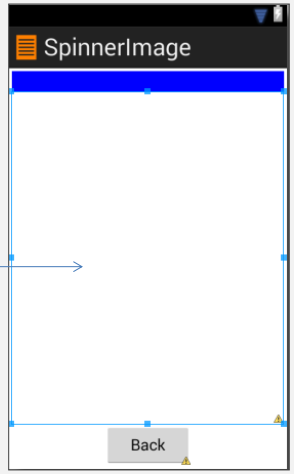
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="3dp" >

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#ff0000ff"
        android:text="Car selector"
        android:textColor="#ffffffff"
        android:textSize="20sp"
        android:textStyle="bold" />

    <Spinner
        android:id="@+id/spinner"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="5dip" />

</LinearLayout>
```



Using The Spinner Widget (again…)

Example7: Spinner Demo2 - Layout2 (solo_picture)

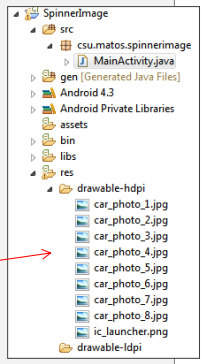
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="3dp"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/txtSoloMsg"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textStyle="bold"
        android:textColor="#FFFFFFFF"
        android:padding="3dp"
        android:background="#ff0000ff" />

    <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>
```



Using The Spinner Widget (again…)

Example7: Spinner Demo2 - MainActivity 1 of 3

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

    // 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_2, R.drawable.car_photo_3,
          R.drawable.car_photo_4, R.drawable.car_photo_5,
          R.drawable.car_photo_6, R.drawable.car_photo_7,
          R.drawable.car_photo_8, };
    Bundle myStateInfo;

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

       setContentView(R.layout.activity_main);
```



Using The Spinner Widget (again…)

Example7: Spinner Demo2 - MainActivity 2 of 3

```
        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) {
                // redraw the main screen showing the spinner
                onCreate(myStateInfo);
            }
        });

    }// showBigScreen
```


Using The Spinner Widget (again…)

Example7: Spinner Demo2 - MainActivity 3 of 3

```
    // next two methods implement the spinner listener
    @Override
    public void onItemSelected(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 ListViews

Example8: Defining your own ListViews

- 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.***
- The next example shows how to do that.

Custom-made ListViews

Example8: Create your own DataAdapter

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.

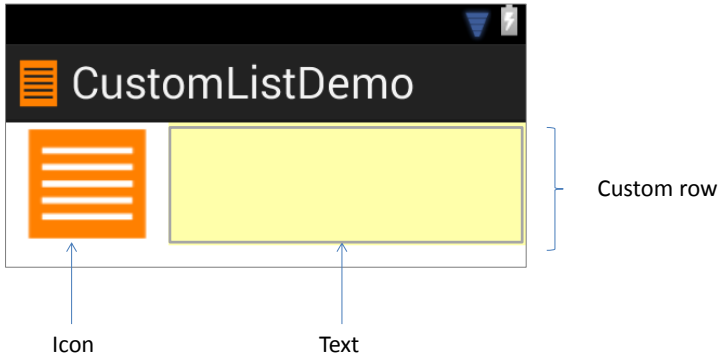
For each data element supplied by the adapter, the method **`getView()`** returns its 'visible' View.

```
public class MyCustomAdapter extends ArrayAdapter{

    // class variables go here ...

    public MyCustomAdapter(...) {  }

    public View getView(...) {    }

}//MyCustomAdapter
```

Custom-made ListViews

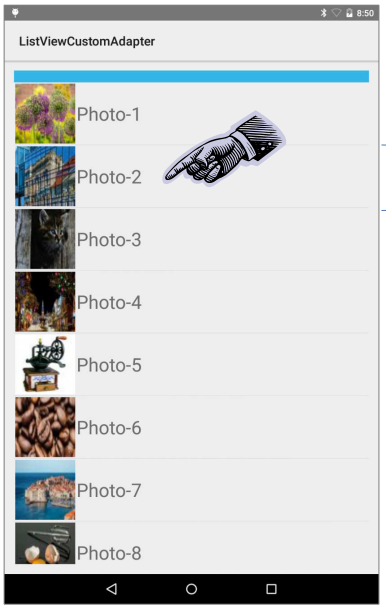
Example8: Designing Custom-Rows

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 ListViews

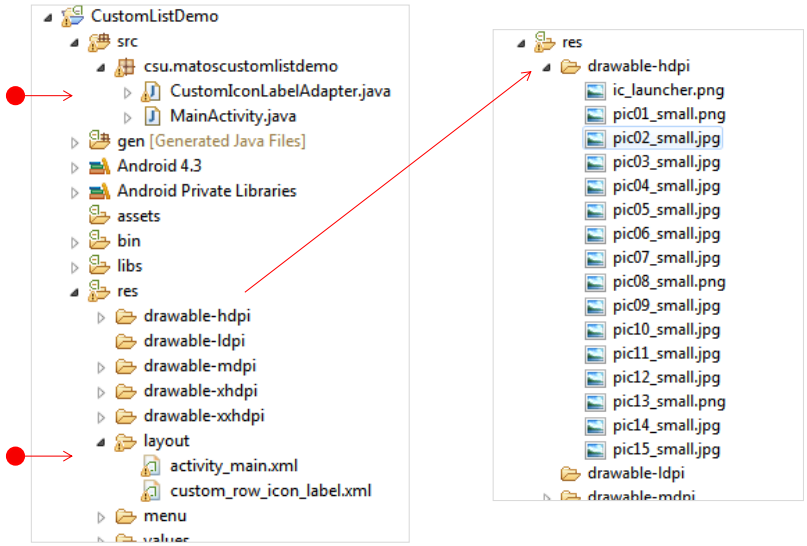
Example8: Designing Custom-Rows



Custom row consists of icon & text

Custom-made ListViews

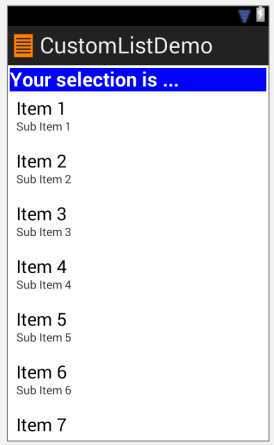
Example8: Custom ListView Demo - App's Structure



Custom-made ListViews

Example8: Layout1 – activity_main.xml

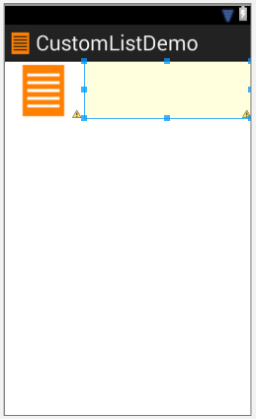
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="3dp"
    android:orientation="vertical" >

    <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="#ffffffff"
        android:textSize="24sp"
        android:textStyle="bold" />

    <ListView
        android:id="@android:id/list"
        android:layout_width="match_parent"
        android:layout_height="match_parent" >
    </ListView>

</LinearLayout>
```



Custom-made ListViews

Example8: Layout2 – custom_row_icon_label.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="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="@+id/label"
        android:layout_width="match_parent"
        android:layout_height="75dp"
        android:background="#22ffff00"
        android:textSize="20sp" />

</LinearLayout>
```



Custom-made ListViews

Example8: Custom ListView Demo – MainActivity 1 of 3

```
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-2", "Data-3", "Data-4", "Data-5",
         "Data-6", "Data-7", "Data-8", "Data-9", "Data-10", "Data-11",
         "Data-12", "Data-13", "Data-14", "Data-15" };

   Integer[] thumbnails = { R.drawable.pic01_small, R.drawable.pic02_small,
         R.drawable.pic03_small, R.drawable.pic04_small,
         R.drawable.pic05_small, R.drawable.pic06_small,
         R.drawable.pic07_small, R.drawable.pic08_small,
         R.drawable.pic09_small, R.drawable.pic10_small,
         R.drawable.pic11_small, R.drawable.pic12_small,
         R.drawable.pic13_small, R.drawable.pic14_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);
```


Custom-made ListViews

Example8: Custom ListView Demo – MainActivity 2 of 3

```
      // the arguments of the custom adapter are:
      // activityContex, 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

   // react to user's selection of a row
   @Override
   protected void onListItemClick(ListView l, View v, int position, long id) {
      super.onListItemClick(l, v, position, id);
      String text = " Position: " + position + "    " + items[position];
      txtMsg.setText(text);
   }//listener

}//class
```


Custom-made ListViews

Example8: Custom ListView Demo – MainActivity 3 of 3

```
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
```

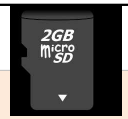
Custom-made ListViews

Example8: 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 C for an example of a better built custom-adapter using the **ViewHolder** design strategy.

Using the SD card



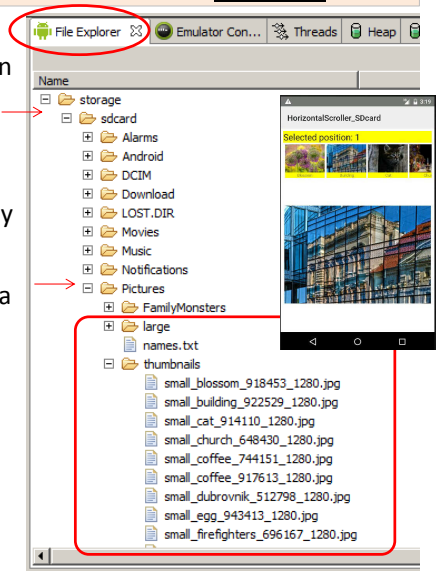
Example 9: Storing Images on the SD card

(*A Variation on Example5*)

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/**



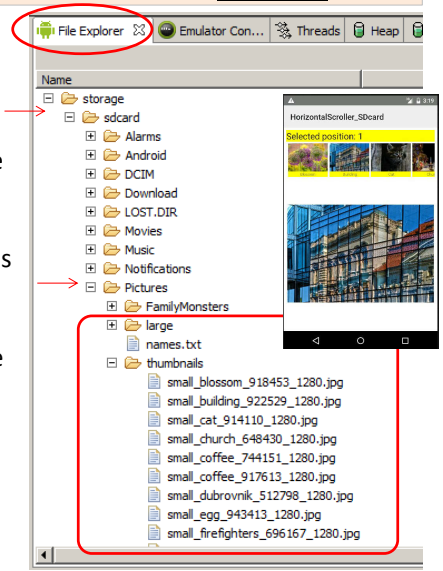
Using the SD card

Example 9: Storing Images on the SD card

(*A Variation on Example5*)

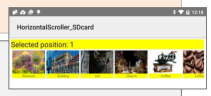
The folder **/Pictures/large/** contains a high-quality image for each thumbnail. These larger pictures are shown after the user makes a selection using the *scroller*.

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, ...`).



Using the SD card

In this app we use the same layouts already introduced in Example5

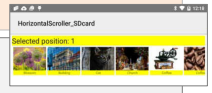


Example 9: SD card Demo – MainActivity 1 of 4

```
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);

    //plumbing: 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);
```

Using the SD card

In this app we use the same layouts already introduced in Example5

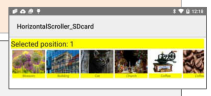


Example 9: SD card Demo – MainActivity 2 of 4

```
    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 - polulate 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);
```


Using the SD card

In this app we use the same layouts already introduced in Example5



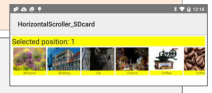
Example 9: SD card Demo – MainActivity 3 of 4

```
      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) {
          String text = "Selected position: " + frame.getId();
          txtMsg.setText(text);
          showLargeImage(frame.getId());
        }
      }); // listener

    }// for

  } catch (Exception e) {

    txtMsg.append("\nError: " + e.getMessage());
  }

}//onCreate
```

*A **ViewGroup** is a special view that can contain other views (called children). The superclass ViewGroup is the base class for layouts and views containers* [From Android Documentation].

Using the SD card

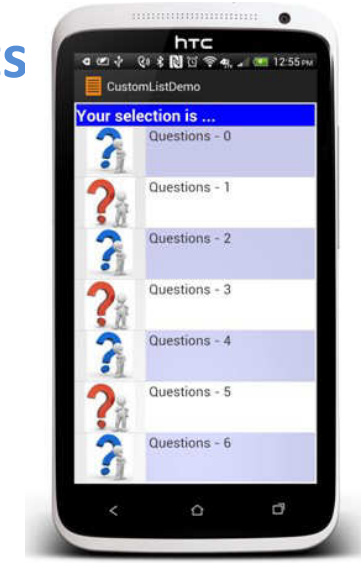
In this app we use the same layouts already introduced in Example5



Example 9: SD card Demo – MainActivity 4 of 4

```
  //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
```


List-Based Widgets



Images made with the "Device Frame Generator Tool", available at
http://android-ui-utils.googlecode.com/hg/asset-studio/dist/device-frames.html

Appendix A: 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

Example:
The following is the definition of the 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" />
```


Appendix A: Predefined Android Resources

Android's Predefined Layouts

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

Link:
http://code.google.com/p/pdn-slatedroid/source/browse/trunk/eclair/frameworks/base/core/res/res/layout/simple_spinner_dropdown_item.xml?r=44

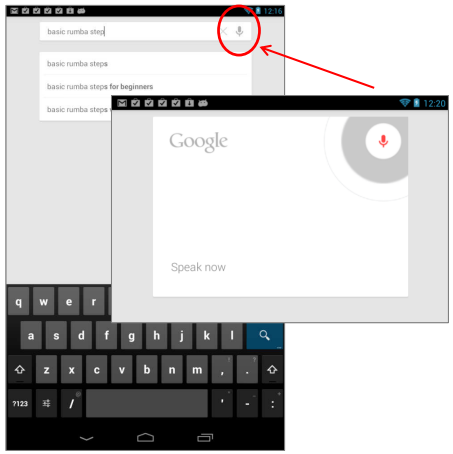
```
<?xml version="1.0" encoding="utf-8"?>
<!--
** Copyright 2008, The Android Open Source Project
** etc...
-->
<CheckedTextView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@android:id/text1"
    style="?android:attr/spinnerDropDownItemStyle"
    android:singleLine="true"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


Appendix B: 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



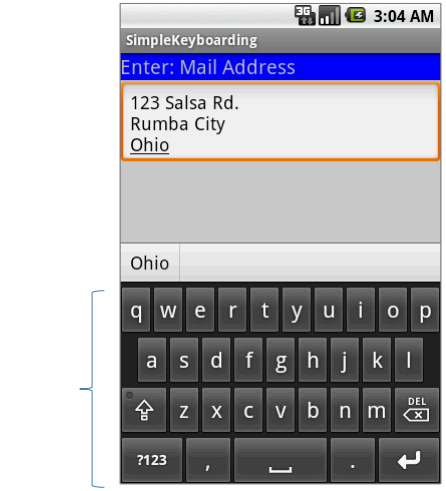
Input accepted from Virtual keyboard and/or voice recognition

Appendix B: 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.



Telling Android what data to expect

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

XML

```
android:inputType="phone"
```

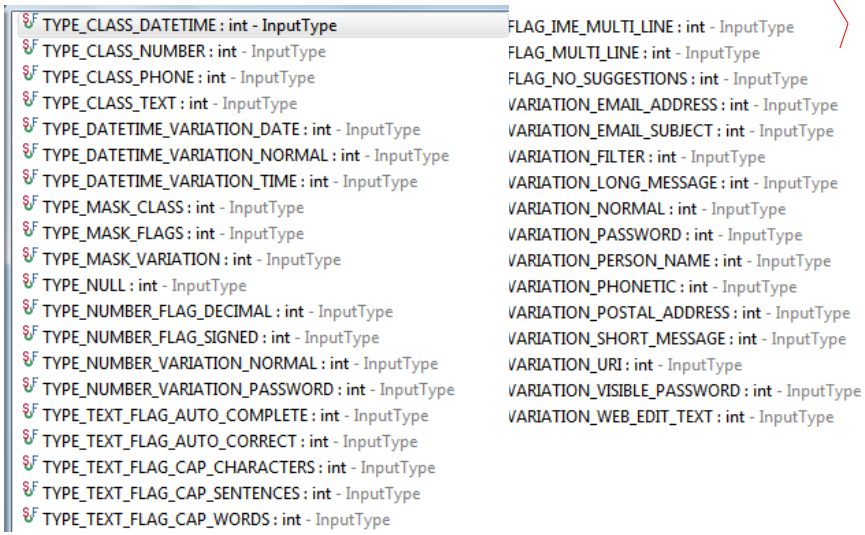
Java

```
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)

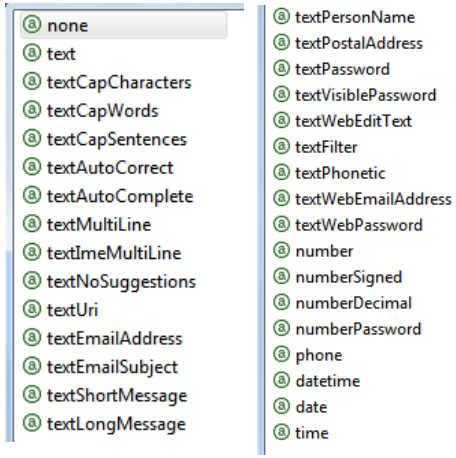
Java Usage – inputType Classes

```
editTextBox.setInputType( android.text.InputType.XXX );
```



XML Usage – inputType Classes

```
<EditText
        ...
        android:inputType="numberSigned|numberDecimal"
        ... />
```



Reference:
http://developer.android.com/reference/android/R.styleable.html#TextView_inputType

Example10: 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="#ffcccccc"
     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>
```

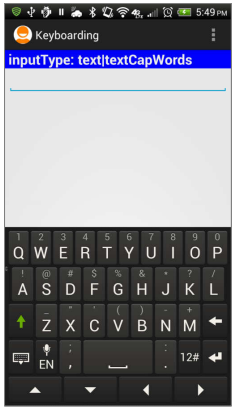
Use "**pipe**" symbol **|** to separate the options.

In this example a soft **text keyboard** will be used.

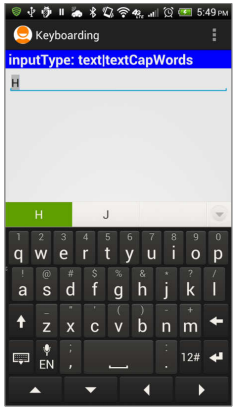
Each word will be capitalized.

Appendix B: EditText Boxes & Keyboarding

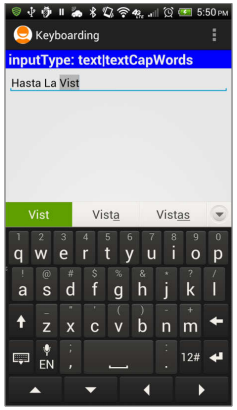
Example10: Using android:inputType= "*text|textCapWords*"



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



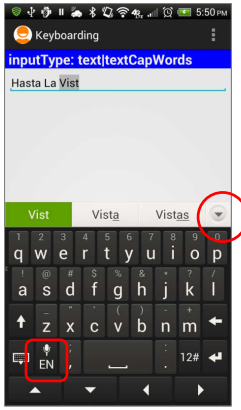
After first letter is typed the keyboard automatically switches to LOWER case mode



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

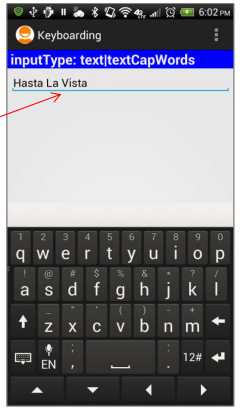
Appendix B: EditText Boxes & Keyboarding

Example10: Using android:inputType= "*text|textCapWords*"



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

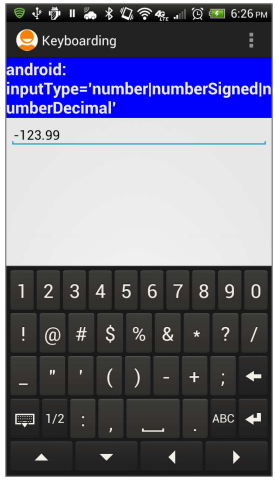
You may speed up typing by tapping on an option from the list of suggested words (bilingual choices)



Selected word is introduced in the EditText box

Appendix B: EditText Boxes & Keyboarding

Example 11: Using
android:inputType="*number|numberSigned|numberDecimal*"



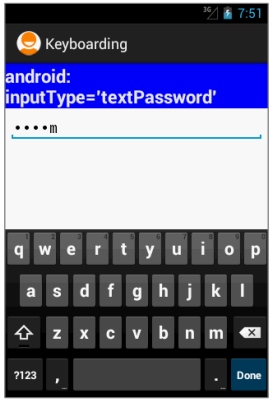
1. The keyboard displays numbers.
2. *Non-numeric* keys (such as !@#$%&*?/_) are visible but disable.
3. Only valid numeric expressions can be entered.
4. Type **number|numberSigned** accepts integers.
5. Type **numberDecimal** accepts real numbers.

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

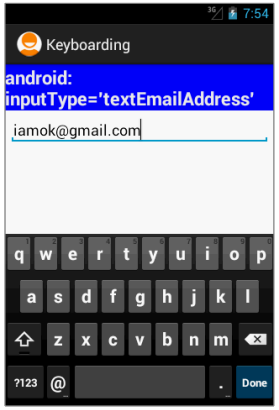
```
editTextBox.setInputType(
  android.text.InputType.TYPE_CLASS_NUMBER |
  android.text.InputType.TYPE_NUMBER_FLAG_SIGNED);
```


Appendix B: EditText Boxes & Keyboarding

Example 12: Using
android:inputType="*textPassword*"



Example 13: Using
android:inputType="*textEmailAddress*"



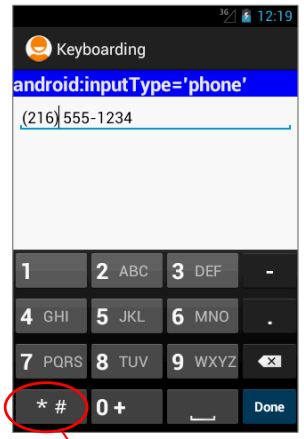
Soft keyboard shows characters used in email addresses (such as letters, **@,** dot**).**

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

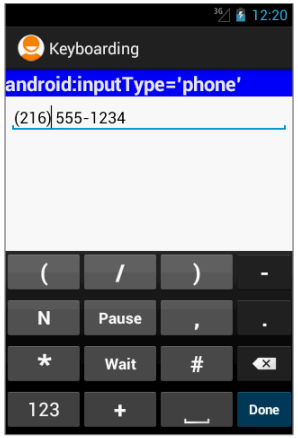
- The keyboard displays all possible keys.
- Current character is briefly displayed for verification purposes.
- The current character is hidden and a *heavy-dot* is displayed.

Appendix B: EditText Boxes & Keyboarding

Example 14: Using `android:inputType="phone"`



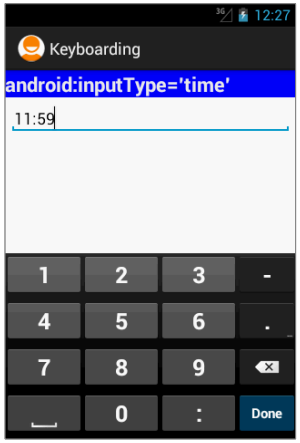
Additional symbols



Soft keyboard displays the layout of a typical *phone keypad* plus additional non digit symbols such as:
( ) . / Pause Wait # - +

Appendix B: EditText Boxes & Keyboarding

Example 15: Using `android:inputType="time"`



Soft keyboard displays a numerical layout.

Only digits and colon-char "**:**" can be used.

Appendix B: EditText Boxes & Keyboarding

Example 16: 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"
```

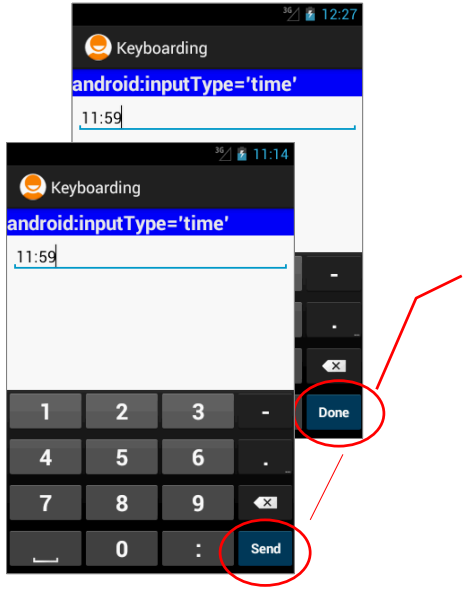
Later, your Java code could provide an implementation of the method:

```
editTextBox
   .setOnEditorActionListener()
```

to do something with the event.

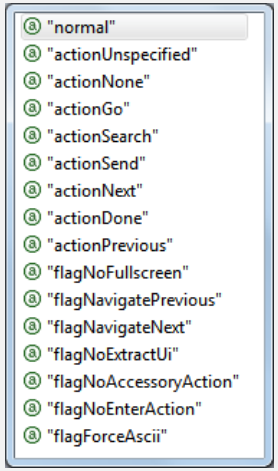
Appendix B: EditText Boxes & Keyboarding

Example 16: Using `android:inputType="time"`



Other options for
`android:imeAction=". . ."`

are



Appendix B: EditText Boxes & Keyboarding

Example 17: Using `android:inputType="datetime"`

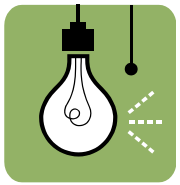
Soft keyboard displays a numerical layout.

Only digits and date/time valid characters are allowed.

Examples of valid dates are:

```
12/21/2012   12:12
12/31/2011
12:30
```


Appendix B: 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);
}
```

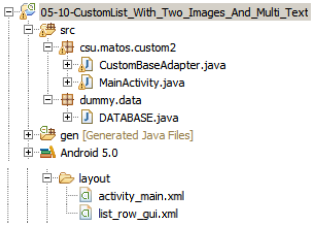
- To **display** the virtual keyboard, call the method:

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

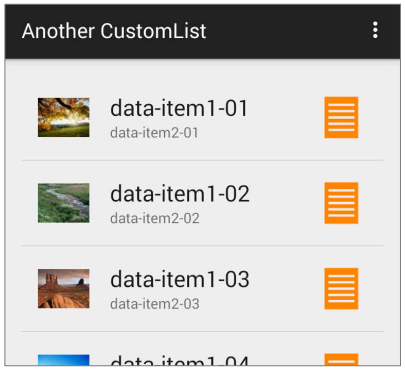

Appendix C. Custom List Supported by a BaseAdapter

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.

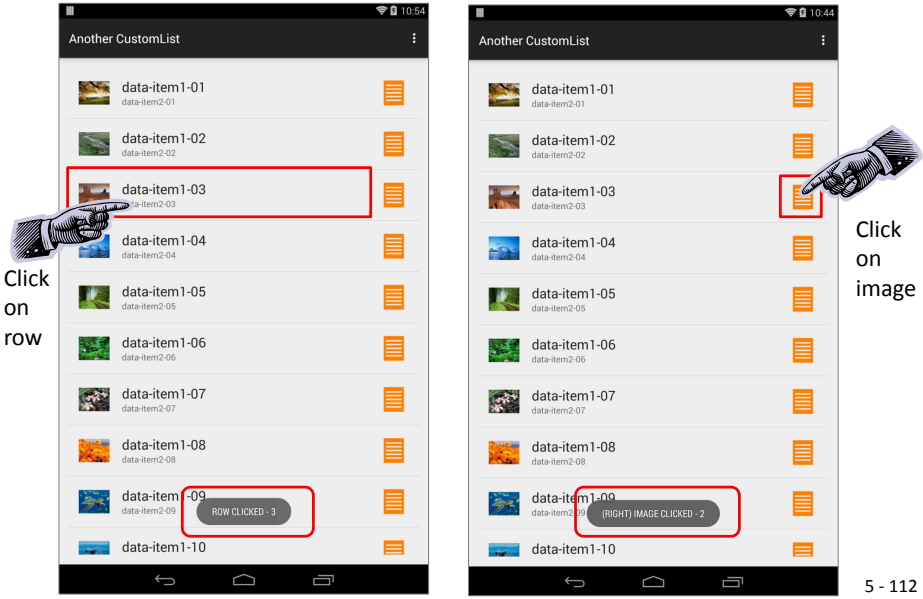
An **onClick** listener is set to recognize the user's tapping on the image to the right side, and another listener is set for clicking anything from the rest of the row.



The app consists of two classes: **MainActivity** and **CustomBaseAdapter**. It has two 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.



Appendix C. Custom List Supported by a BaseAdapter

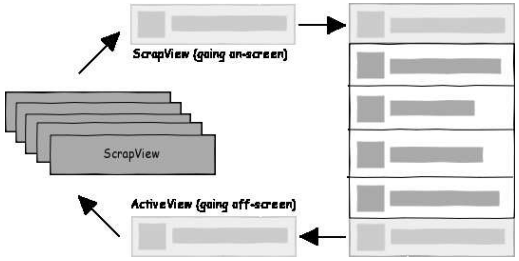


Click on row

Click on image

Appendix C. Custom List – ViewHolder Pattern

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.

Appendix C. Custom List – activity_main.xml

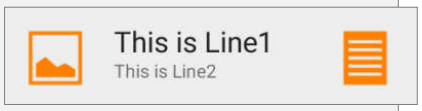
Layout *activity_main.xml* shows a ListView.

```
<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>
```

Appendix C. Custom List – list_row_gui.xml 1 of 2

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" />
```



Appendix C. Custom List – list_row_gui.xml 2 of 2

Layout *list_gui_row.xml* shows a custom-made row holding two lines of text and two images.

```
        <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>
```

Appendix C. Custom List – MainActivity.java 1 of 1

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
```


Appendix C. Custom List – CustomBaseAdapter.java 1 of 5

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);
   }
```


Appendix C. Custom List – CustomBaseAdapter.java 2 of 5

```
   @Override
   public long getItemId(int position) {
      return position;
   }

   @Override
   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();
```


Appendix C. Custom List – CustomBaseAdapter.java 3 of 5

```
         // 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();
      }

      // 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);
```

Appendix C. Custom List – CustomBaseAdapter.java 4 of 5

```
      // 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
```


Appendix C. Custom List – CustomBaseAdapter.java 5 of 5

```
   // 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;
      TextView textview2;
      ImageView imageview1;
      ImageView imageview2;
   }


}// CustomMadeListener
```


Appendix C. Custom List – DATABASE.java 1 of 2

```
public class DATABASE {    // TEST DATABASE
   public String[] text1array = { "data-item1-01", "data-item1-02",
         "data-item1-03", "data-item1-04", "data-item1-05", "data-item1-06",
         "data-item1-07", "data-item1-08", "data-item1-09", "data-item1-10",
         "data-item1-11", "data-item1-12", "data-item1-13", "data-item1-14",
         "data-item1-15" };
   public String[] text2array = { "data-item2-01", "data-item2-02",
         "data-item2-03", "data-item2-04", "data-item2-05", "data-item2-06",
         "data-item2-07", "data-item2-08", "data-item2-09", "data-item2-10",
         "data-item2-11", "data-item2-12", "data-item2-13", "data-item2-14",
         "data-item2-15" };
   public Integer[] icon1array = { csu.matos.custom2.R.drawable.pic01_small,
         csu.matos.custom2.R.drawable.pic02_small,
         csu.matos.custom2.R.drawable.pic03_small,
         csu.matos.custom2.R.drawable.pic04_small,
         csu.matos.custom2.R.drawable.pic05_small,
         csu.matos.custom2.R.drawable.pic06_small,
         csu.matos.custom2.R.drawable.pic07_small,
         csu.matos.custom2.R.drawable.pic08_small,
         csu.matos.custom2.R.drawable.pic09_small,
         csu.matos.custom2.R.drawable.pic10_small,
         csu.matos.custom2.R.drawable.pic11_small,
         csu.matos.custom2.R.drawable.pic12_small,
         csu.matos.custom2.R.drawable.pic13_small,
         csu.matos.custom2.R.drawable.pic14_small,
         csu.matos.custom2.R.drawable.pic15_small, };
```


Appendix C. Custom List – DATABASE.java 2 of 2

```
   public class DbRecord {
      public String text1;
      public String 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 DbRecods:[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
```