

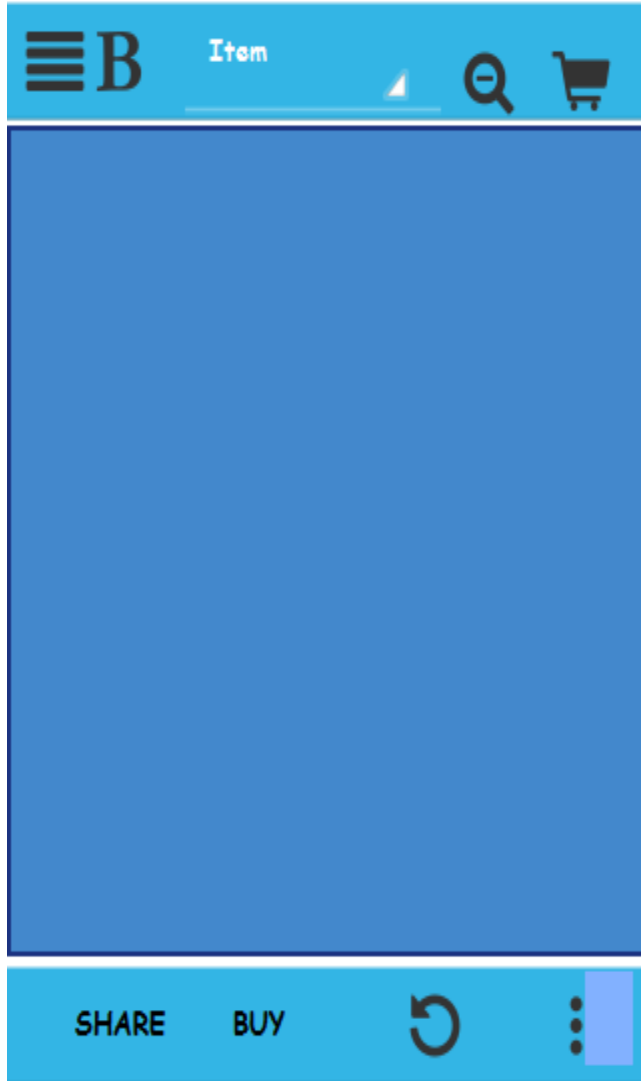


# Lesson 8

## ActionBars, Menus Dialog Boxes & Toast Widgets

# Android Design Strategies

## The ActionBar Design Model



### Header / Top Décor

An **ActionBar** occupies the top décor screen space. If this space is not sufficient, it splits its contents to the footer section of the app's GUI.

### Body: Main application's UI

### Footer:

ActionBar overflows to the bottom of the screen. Toolbars may also be placed at the bottom of the UI.

# Android Design Strategies

## Menus

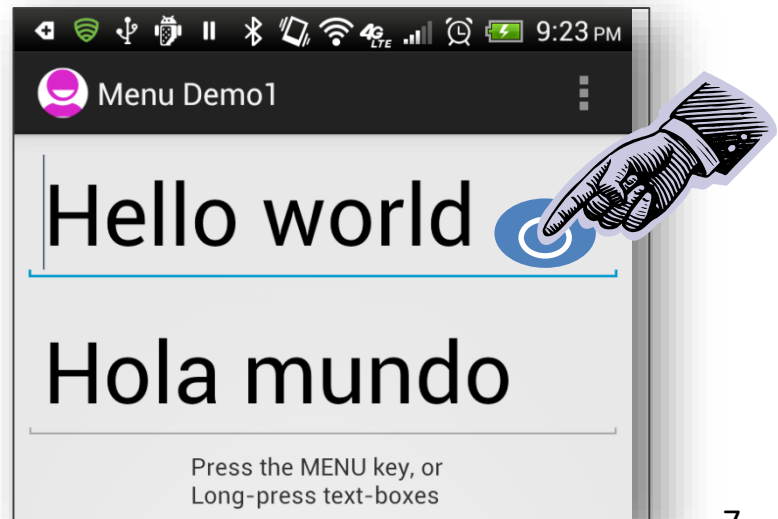
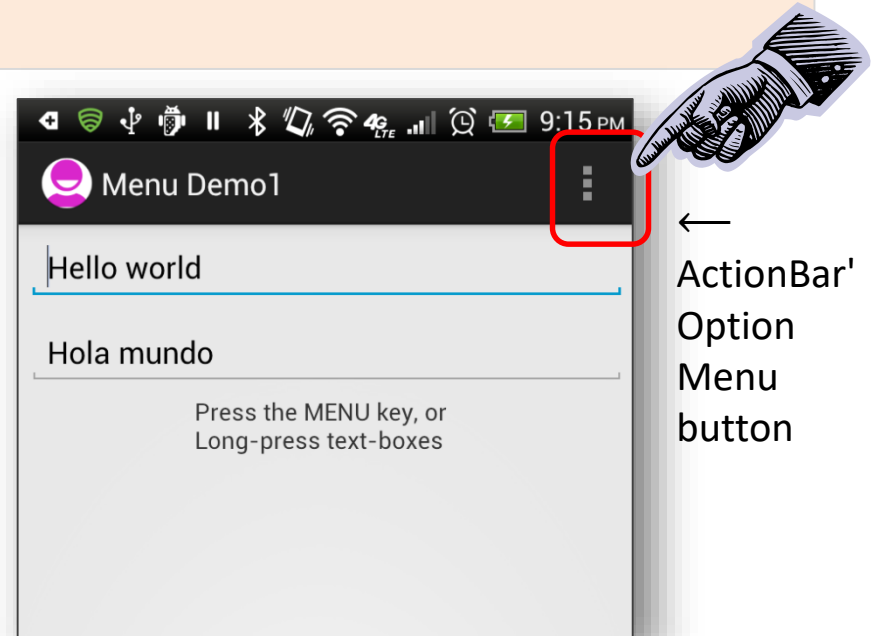
- Menus are a common design feature often included in Android solutions.
- *A good menu* provides a simple and unobtrusive interface that adds more capabilities to the app without occupying much space on the app's UI.
- Menus can be adjusted to the currently displayed UI. Each screen may have its own set of options.
- A screen could have any number of widgets and you may optionally attach a menu to any of those widgets.
- **Current design practices promote the integration of menus and the top décor component.**

# Using Menus

## Menu Types

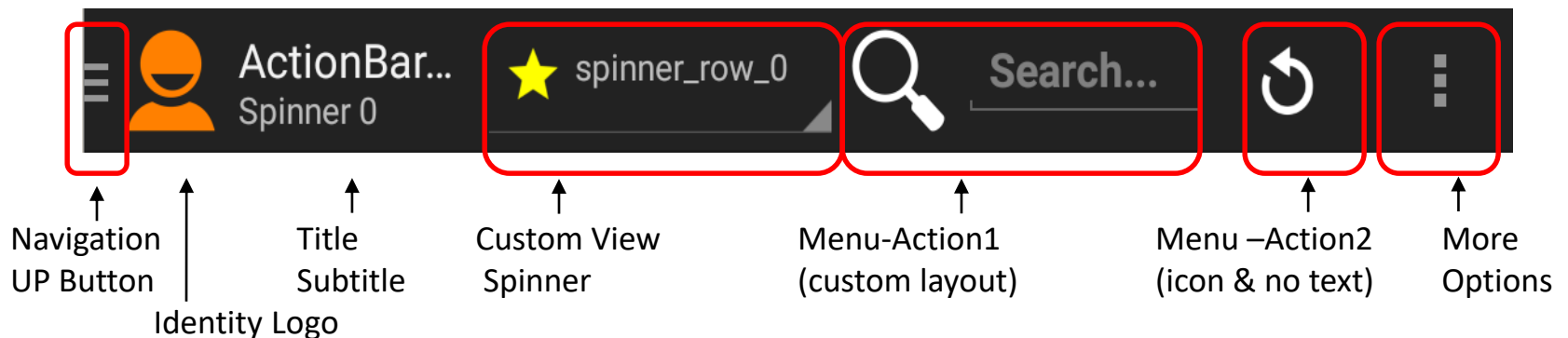
Android supports two types of menus:  
**Options Menu** and **Context Menu**.

1. The global **options menu** is triggered by pressing the device's hardware or virtual **Menu** button. The global menu is also known as **action menu**. *There is only ONE option menu for each UI.*
2. A **context menu** is raised by a *tap-and-hold interaction (long-tap)* on the widget associated to the menu. *You may set one context menu on any widget.*



# ActionBar Design Strategy

- The **ActionBar** control was introduced in SDK 3.0 and plays a special role on the crafting of non-trivial Android apps. It is depicted as a graphical tool-bar at the top of each screen and it is usually persistent across the app.
- It normally contains the following pieces:
  - Navigation – UP Button (Hamburger or Arrow icon)
  - An Identity Logo,
  - Title and Subtitle
  - Optional custom view,
  - Action Tiles (clickable buttons showing icon/text/custom layouts),
  - Overflow Option Menu Button
  - Legacy app's may also include Navigation Tabs (*deprecated after SDK4.4*)



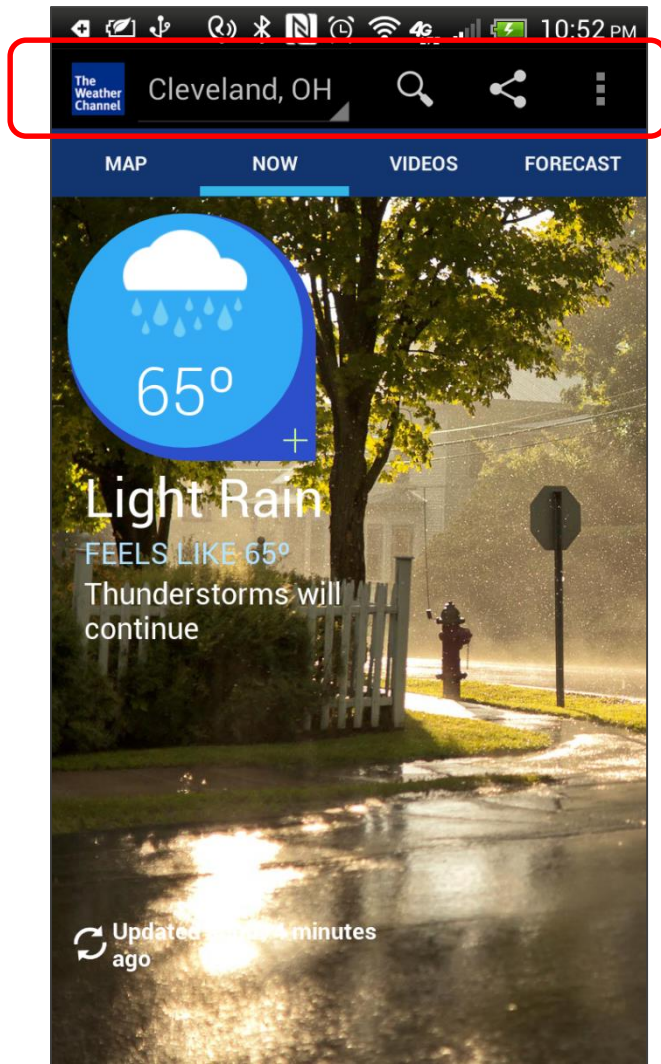
## Example of Apps based on the ActionBar Architecture

	Free App Name	Downloads	Rating	Category
1	Google Search	1B	4.4	Tools
2	Gmail	1B	4.3	Communication
3	Google Maps	1B	4.3	Travel & Local
4	YouTube	1B	4.1	Media & Video
5	Facebook	1B	4.0	Social
6	WhatsApp	500M	4.4	Communications
7	Instagram	100M	4.5	Social
8	Pandora	100M	4.4	Music & Audio
9	Netflix	100M	4.4	Entertainment
10	Adobe Reader	100M	4.3	Productivity
11	Skype	100M	4.1	Communications
12	Twitter	100M	4.1	Social
13	eBay	50M	4.3	Shopping
14	Weather Channel	50M	4.2	Weather
15	Kindle	50M	4.1	Books & References
16	Wikipedia	10M	4.4	Books & References
17	Zillow	10M	4.4	Lifestyle
18	ESPN SportCenter	10M	4.2	Sports
19	BBC News	10M	4.2	News & Magazines
20	Amazon (Tablets)	10M	4.0	Shopping
21	Expedia	10M	4.0	Travel & Local

### Source:

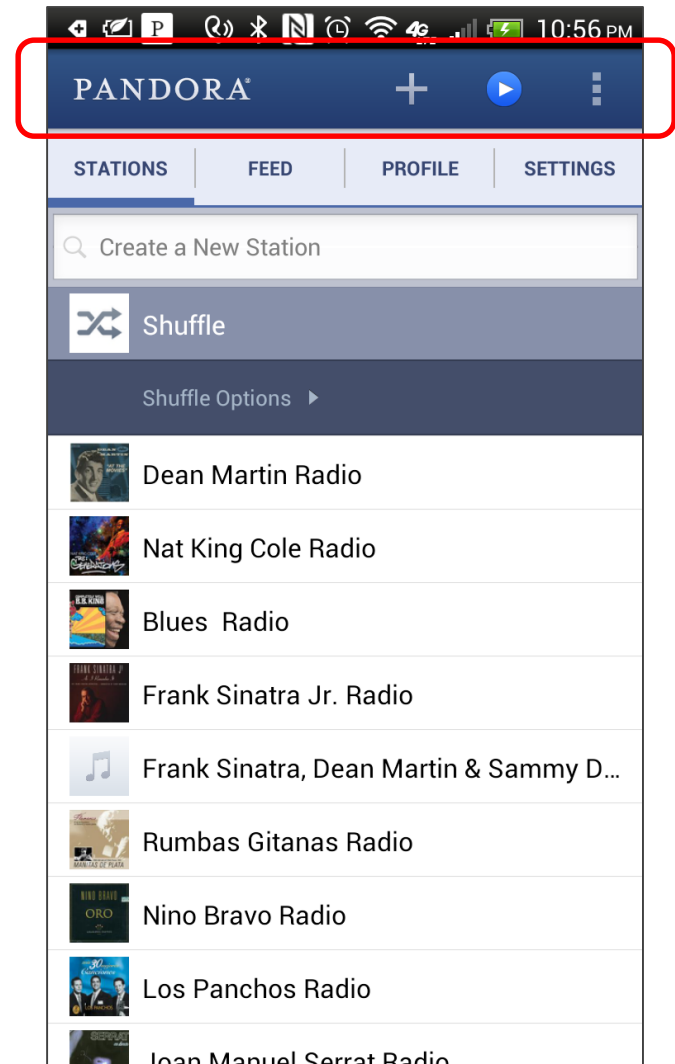
Google Play Store. Last visited: Feb 16, 2015. Link: <https://play.google.com/store?hl=en>

# Example of Apps based on the ActionBar Architecture



← ActionBar →

Two different apps showing a relatively similar navigation pattern and visual structure.



**Factoid:** According to [techcrunch.com](http://techcrunch.com) as of Q1-2013 the *Weather Channel* mobile application has been downloaded more than 100 million times. On the other hand, *Pandora* app exceeds 250 million downloads.

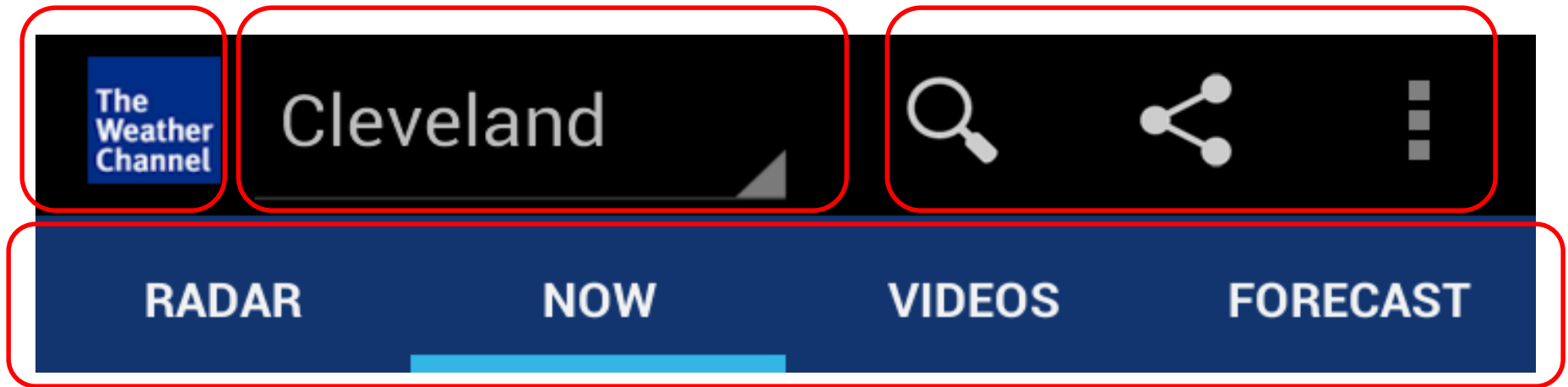
# Reverse Engineering an App's ActionBar

A possible interpretation of the Weather Channel App's ActionBar+PageViewer design follows

Identity  
Logo

Custom ListView

Menu Items (Search & Share)  
and rest of : Options Menu  
(*topic is explored later*)



PageViewer  
Control

Current Tab  
Selection

In this example UI navigation is supported through PageViewer control (discussed later). Older apps used the ActionBar to anchor their navigation tabs (this technique is now deprecated – see Appendix)

## Disclaimer:

The Weather Channel app is a copyrighted product and belongs to its authors and the Weather Channel LLC. Our interpretation of its organization is pure speculation intended to guide the reader into the appreciation of a well known and designed app.



## ActionBar Architecture

The clickable *action tiles* (or action Items) exposed by an ActionBar are usually defined in a **res/menu** XML resource file. This resource file can be later inflated and shown as part of the app's global Option's Menu. Please notice that "Menu Items" and "Action Items" as well as "OptionsMenu" and "ActionOverflow" are overlapping concepts.

By default each action item is included in a simple dropdown text-only list activated by the clicking of the virtual : ActionOverflow button. However, selected tiles can be separately shown as icons (with or without text) as part of the ActionBar.

An OptionMenu generally persists for the lifetime of the app, however it could be dynamically enabled, disabled, and changed.

Two methods are responsible for most of the work related to interacting with the tiles on an ActionBar

### **onCreateOptionsMenu( ... )**

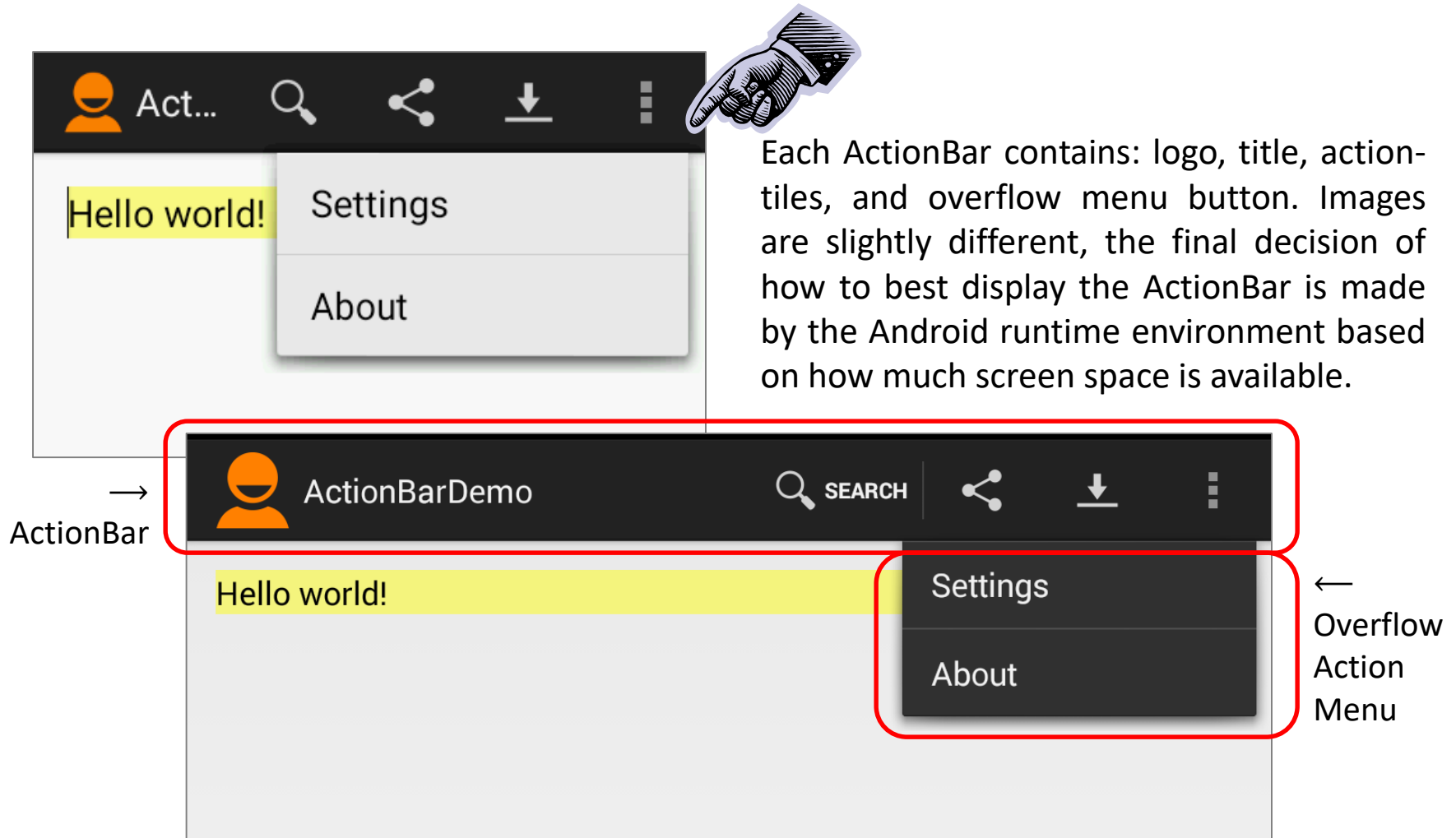
Inflates the XML specs defining each of the action-tiles.

### **onOptionsItemSelected( ... )**

Captures the click-event on any tile and dispatches the proper response to the user's request.

## Example 1 – Creating a Simple ActionBar

This example uses the “Blank Application” ADT-wizard to generate a basic Android app. We modify its **res/menu/main.xml** file to produce a custom ActionBar. The screen-shots below are taken from a small handset and a tablet running the app.



## Example 1 – Creating a Simple ActionBar

Below is the `res/menu/main.xml` definition used to create the app's ActionBar.

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto"
      xmlns:tools="http://schemas.android.com/tools"
      tools:context="csu.matos.MainActivity" >

    <item
        android:id="@+id/action_search"
        android:icon="@drawable/ic_action_search"
        android:orderInCategory="120"
        android:showAsAction="always|withText"
        android:title="Search"/>

    <item
        android:id="@+id/action_share"
        android:icon="@drawable/ic_action_share"
        android:orderInCategory="140"
        android:showAsAction="always"
        android:title="Share"/>

    <item
        android:id="@+id/action_download"
        android:icon="@drawable/ic_action_download"
        android:orderInCategory="160"
        android:showAsAction="always"
        android:title="Download"/>

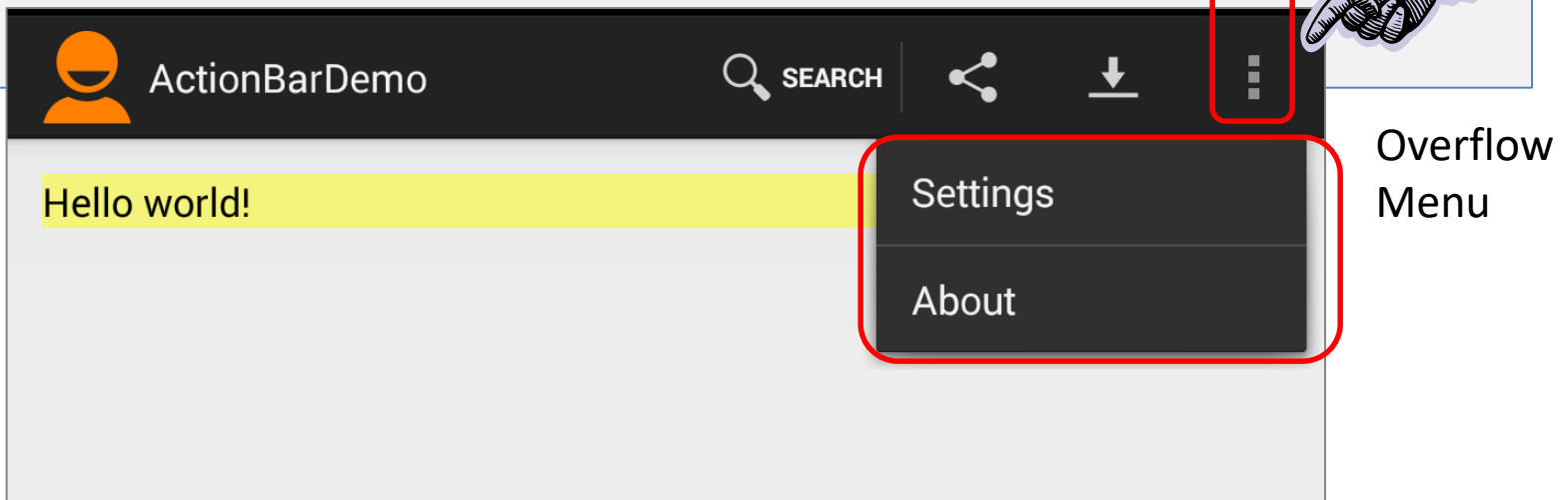
</menu>
```



## Example 1 – Creating a Simple ActionBar

Below is the `res/menu/main.xml` definition of the app's ActionBar.

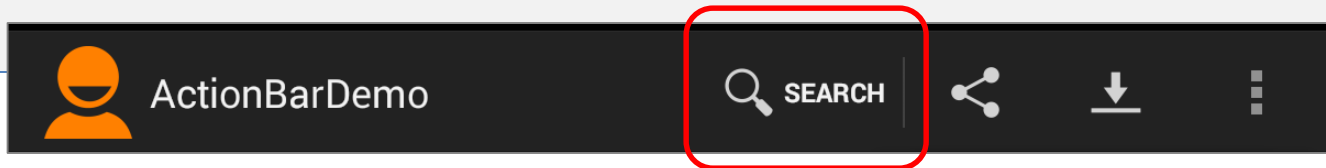
```
<item
    android:id="@+id/action_settings"
    android:orderInCategory="180"
    android:showAsAction="never"
    android:title="Settings"/>
<item
    android:id="@+id/action_about"
    android:orderInCategory="200"
    android:showAsAction="never"
    android:title="About"/>
</menu>
```



## Example 1 – Creating a Simple ActionBar

Each menu **<item>** element represents an action-tile. Consider the following sample:

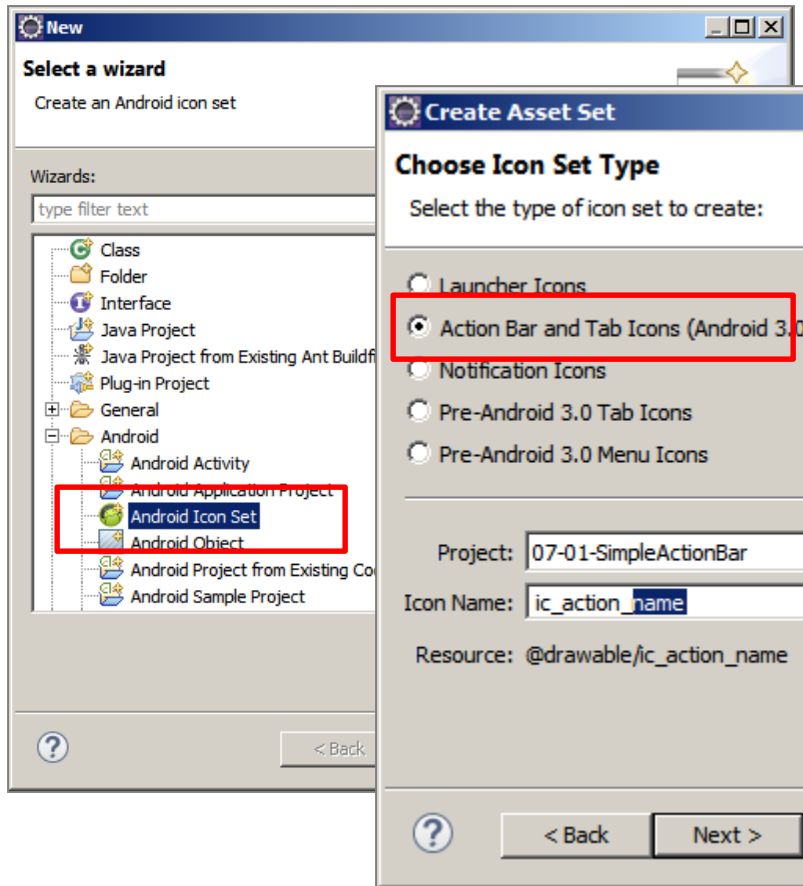
```
<item
    android:id="@+id/action_search"
    android:icon="@drawable/ic_action_search"
    android:orderInCategory="120"
    android:showAsAction="always|withText"
    android:title="Search"/>
```



<b>android:id</b>	Action tile ID ( <a href="#">@+id/action_search</a> ), needed to identify what action has been selected.
<b>android:icon</b>	Optional icon to be displayed with this entry. For guidance on how to create an action icon consult <a href="http://developer.android.com/design/style/iconography.html">http://developer.android.com/design/style/iconography.html</a>
<b>:orderInCategory</b>	Relative position of the tile on the ActionBar ( <a href="#">100</a> , <a href="#">120</a> , <a href="#">140</a> , ... )
<b>:showAsAction</b>	Custom placement of an individual tile is determined using the clauses: “never”, “ifRoom”, “always”, “withText”, and “collapseActionView”.
<b>:title</b>	Optional text ( <a href="#">‘SEARCH’</a> ) describing the action-tile

# Example 1 – Creating a Simple ActionBar

## ActionBar Icons



Predefined collection  
of ActionBar icons

# Example 1 – Creating a Simple ActionBar

## ActivityMain.java

```
public class MainActivity extends Activity {
    EditText txtMsg;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        txtMsg = (EditText)findViewById(R.id.txtMsg);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; add items to the action bar
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // user clicked a menu-item from ActionBar
        int id = item.getItemId();

        if (id == R.id.action_search) {
            txtMsg.setText("Search...");
            // perform SEARCH operations...
            return true;
        }
    }
}
```

1

2

3

## Example 1 – Creating a Simple ActionBar

### ActivityMain.java

3

```
    else if (id == R.id.action_share) {  
        txtMsg.setText("Share...");  
        // perform SHARE operations...  
        return true;  
    }  
    else if (id == R.id.action_download) {  
        txtMsg.setText("Download...");  
        // perform DOWNLOAD operations...  
        return true;  
    }  
    else if (id == R.id.action_about) {  
        txtMsg.setText("About...");  
        // perform ABOUT operations...  
        return true;  
    }  
    else if (id == R.id.action_settings) {  
        txtMsg.setText("Settings...");  
        // perform SETTING operations...  
        return true;  
    }  
  
    return false;  
}
```



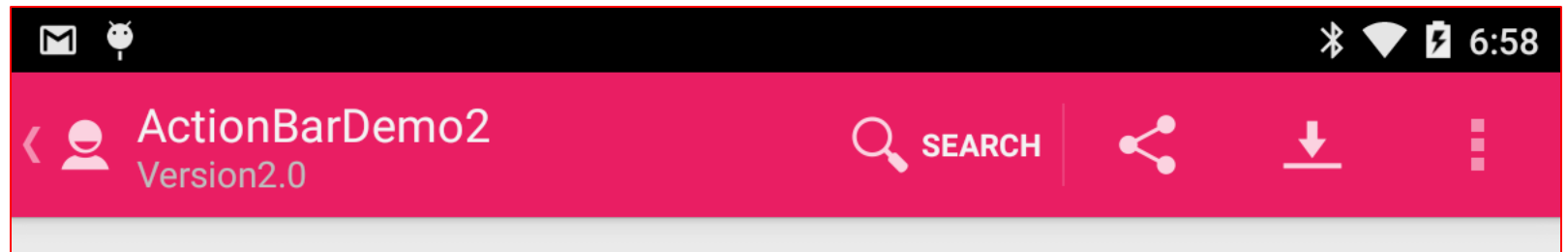
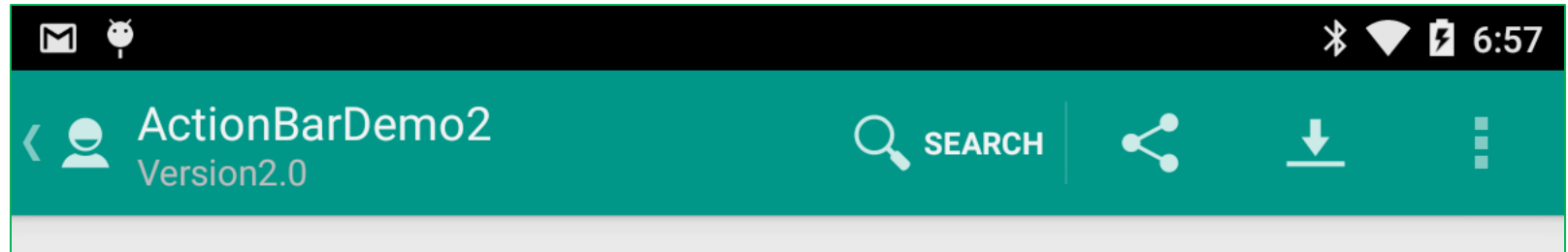
## Example 1 – Creating a Simple ActionBar

### Comments: ActivityMain.java

1. Plumbing operation. Establish access to the GUI's **EditText** field displaying the “Hello World” line.
2. The method `onCreateOptionsMenu()` is called to prepare the app's `OptionsMenu`. The xml file `res/memu/main.xml` containing the `ActionBar` item specifications is inflated using a `MenuInflater` object. Some action items will be shown on the `ActionBar` as an `Icon/Text` tile and the rest moved to the overflow menu window.
3. When the user clicks on a reactive portion of the `ActionBar`, its item's ID is supplied to the `onOptionsItemSelected()` method. There you branch to the appropriated service routine where the action is served. Finally return **true** to signal the event has been fully consumed.

## Example 2 – Modifying the ActionBar

This example is a minor extension of Example1. The ActionBar is modified so it can show a different background color, logo, and UP affordance. Colors selected from: <http://www.google.com/design/spec/style/color.html#color-color-palette>



Colors: (1) Teal-500, (2) Pink-500, (3) Gradient Blue-500-700-900

## Example 2 – Modifying the ActionBar

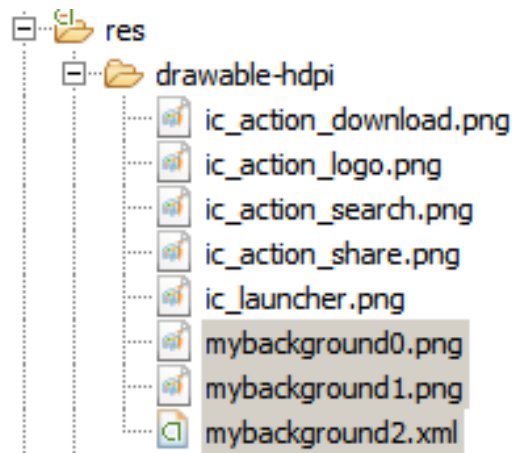
The ActionBar is programmatically changed as follows

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    txtMsg = (EditText)findViewById(R.id.txtMsg);  
  
    // setup ActionBar  
1 → actionBar = getActionBar();  
  
2 → actionBar.setTitle("ActionBarDemo2");  
   actionBar.setSubtitle("Version2.0");  
   actionBar.setIcon(R.drawable.ic_action_logo);  
  
    // choose one type of background  
3 → actionBar.setBackgroundDrawable(getResources().getDrawable(R.drawable.mybackground0));  
   actionBar.setBackgroundDrawable(getResources().getDrawable(R.drawable.mybackground1));  
   actionBar.setBackgroundDrawable(getResources().getDrawable(R.drawable.mybackground2));  
  
4 → actionBar.setDisplayShowCustomEnabled(true);    // allow custom views to be shown  
   actionBar.setDisplayHomeAsUpEnabled(true);        // show 'UP' affordance < button  
   actionBar.setDisplayShowHomeEnabled(true);       // allow app icon - logo to be shown  
   actionBar.setHomeButtonEnabled(true);            // needed for API14 or greater  
}
```

## Example 2 – Modifying the ActionBar

### Comments

1. A call to the `getActionBar()` method returns a handle to the app's ActionBar. Using this reference you now have programmatic control to any of its components.
2. In this example a new title & subtitle is assigned to the ActionBar. Notice that when you work with a complex app exposing many screens, changing title and/or subtitle becomes a simple, yet powerful way of guiding the user through the app.
3. The app's identifying logo could be changed with a call to `.setLogo(drawable)`. Similarly, the ActionBar's background image could be changed to any drawable you chose. In our example the first two backgrounds are just a pair of solid rectangular Teal and Pink color swatches stored as .PNG images and added to the **res/drawable** folder.



mybackground0.png

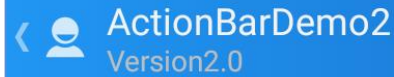


mybackground1.png



## Example 2 – Modifying the ActionBar

### Comments



3. (*cont.*) You may also provide an XML gradient definition for a background. For instance

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <gradient
        android:angle="0"
        android:centerColor="#1976D2"
        android:centerX="50%"
        android:endColor="#0D47A1"
        android:startColor="#2196F3"
        android:type="linear" />
</shape>
```

4. You may set/reset features such as: show custom views, show Up-affordance, and show application's logo or icon.

```
// set ActionBar options
actionBar.setDisplayShowCustomEnabled(true); // allow custom views to be shown
actionBar.setDisplayHomeAsUpEnabled(true); // show 'UP' affordance < button
actionBar.setDisplayShowHomeEnabled(true); // allow app icon - logo to be shown
actionBar.setHomeButtonEnabled(true); // needed for API.14 or greater
```

## Example 2 – Modifying the ActionBar

### Comments

4. (*cont.*) Alternatively, you may set/reset the ActionBar features using a single statement.

```
// set ActionBar options

actionBar.setDisplayOptions( ActionBar.DISPLAY_SHOW_TITLE
                             | ActionBar.DISPLAY_SHOW_HOME
                             | ActionBar.DISPLAY_HOME_AS_UP
                             | ActionBar.DISPLAY_SHOW_CUSTOM );
```

### MISCELLANEOUS. Drawing Resources


**Gradients**    <http://angrytools.com/gradient/>

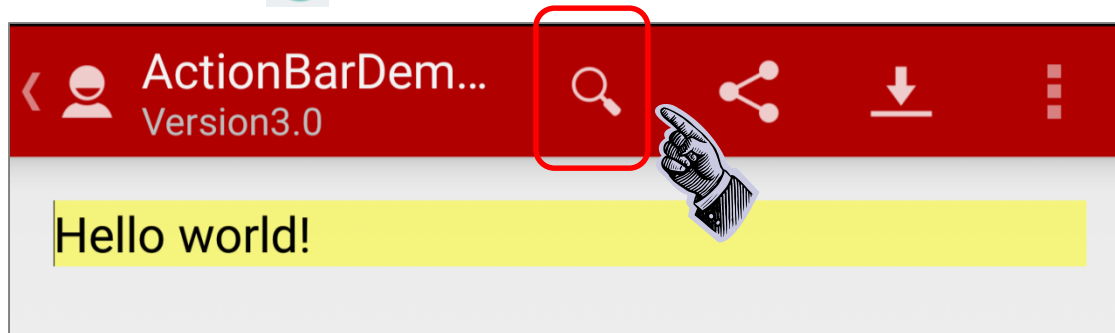
**Color Chart**    <http://www.google.com/design/spec/style/color.html#color-color-palette>


**Drawables**    <http://developer.android.com/guide/topics/resources/drawable-resource.html>

## Example 3 – Search ActionBar Tile

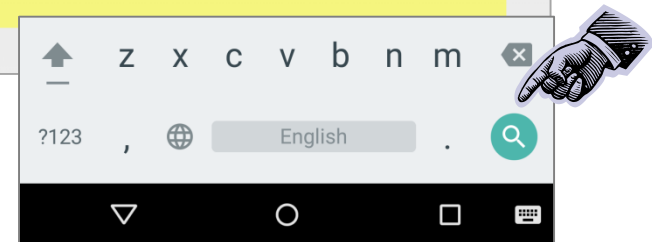
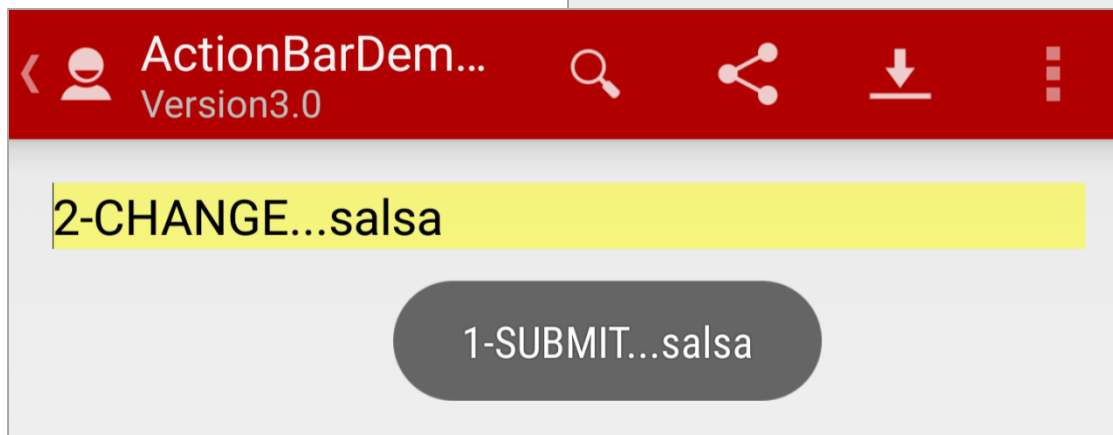
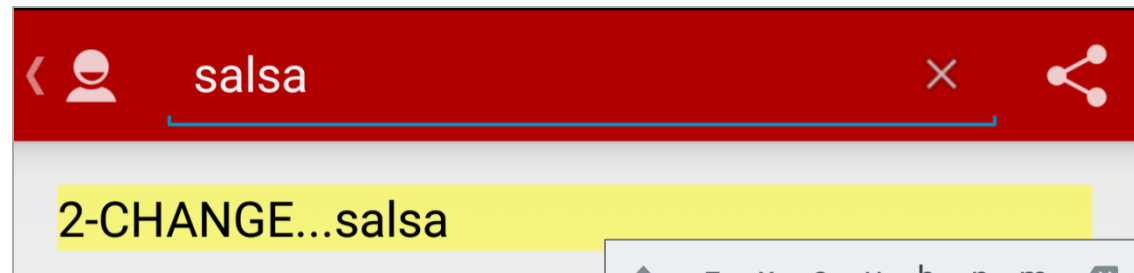



This is an extension of Example1. The ActionBar *search option* is implemented using a **SearchView** widget as a menu item. When the user finally taps on the keyboard's SEARCH key  the text captured in the SearchView box is used to trigger an inquiry.



Press  to clear query string

A listener watches for each new character added to the query



Press  to close SearchView and submit query



The **SearchView** box is defined as part of the ActionBar through an **XML MENU** file similar To the following code sample



```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto"
      xmlns:tools="http://schemas.android.com/tools"
      tools:context="csu.matos.MainActivity" >

    <item
        android:id="@+id/action_search"
        android:actionViewClass="android.widget.SearchView"
        android:orderInCategory="120"
        android:showAsAction="always|withText"
        android:title="Search"/>

    <item
        android:id="@+id/action_share"
        android:icon="@drawable/ic_action_share"
        android:orderInCategory="140"
        android:showAsAction="always"
        android:title="Share"/>

    <item
        android:id="@+id/action_download"
        android:icon="@drawable/ic_action_download"
        android:orderInCategory="160"
        android:showAsAction="always"
        android:title="Download"/>

</menu>
```

1

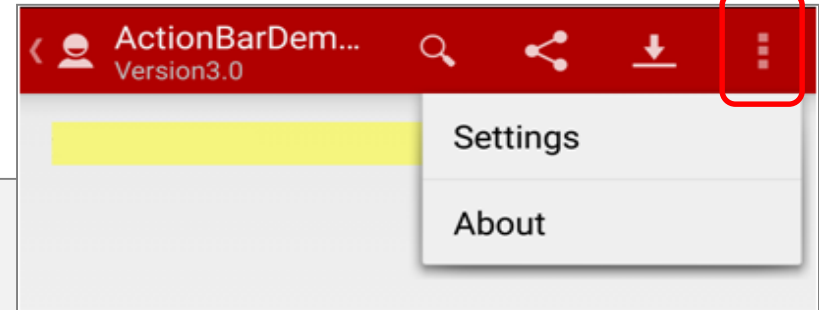




The **SearchView** box is defined as part of the ActionBar through an **XML MENU** file similar To the following code sample

```
<item
    android:id="@+id/action_settings"
    android:orderInCategory="180"
    android:showAsAction="never"
    android:title="Settings"/>
<item
    android:id="@+id/action_about"
    android:orderInCategory="200"
    android:showAsAction="never"
    android:title="About"/>

</menu>
```



### Comments

1. The item *action\_search* does not include an **:icon** clause, instead it relies on the clause **android:actionViewClass="android.widget.SearchView"** to set a collapsible view that –when expanded- allows the user to enter a search query and later submit it to a search provider.



```
public class MainActivity extends Activity {
    EditText txtMsg;
    ActionBar actionBar;
    SearchView txtSearchValue;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        txtMsg = (EditText) findViewById(R.id.txtMsg);

        // setup the ActionBar
        actionBar = getActionBar();
        actionBar.setTitle("ActionBarDemo3");
        actionBar.setSubtitle("Version3.0");
        actionBar.setIcon(R.drawable.ic_action_logo);
        actionBar.setBackgroundDrawable(getResources().getDrawable(
            R.drawable.mybackground1));

        actionBar.setDisplayShowCustomEnabled(true); // allow custom views to be shown
        actionBar.setDisplayHomeAsUpEnabled(true); // show 'UP' affordance < button
        actionBar.setDisplayShowHomeEnabled(true); // allow app icon - logo to be shown
        actionBar.setHomeButtonEnabled(true); // needed for API14 or greater
    }
}
```

**Setting up the ActionBar**  
The code on this page is taken from Example2.



```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the options menu to adds items from menu/main.xml into the ActionBar
    getMenuInflater().inflate(R.menu.main, menu);
    // get access to the collapsible SearchView
1 → txtSearchValue = (SearchView) menu.findItem(R.id.action_search)
                                   .getActionView();
    // set searchView listener (look for text changes, and submit event)
    txtSearchValue.setOnQueryTextListener(new OnQueryTextListener() {

        @Override
        2 → public boolean onQueryTextSubmit(String query) {
            Toast.makeText(getApplicationContext(), "1-SUBMIT..." + query,
                           Toast.LENGTH_SHORT).show();
            // recreate the 'original' ActionBar (collapse the SearchBox)
            invalidateOptionsMenu();
            // clear searchView text
            txtSearchValue.setQuery("", false);
            return false;
        }

        @Override
        3 → public boolean onQueryTextChange(String newText) {
            // accept input one character at the time
            txtMsg.append("\n2-CHANGE..." + newText);
            return false;
        }
    });
    return true;
}
```



4

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle ActionBar item clicks here.
    // NOTE: Observe that SEARCH menuItem is NOT processed in this
    // method (it has its own listener set by onCreateOptionsMenu)

    int id = item.getItemId();

    if (id == android.R.id.home) {
        txtMsg.setText("Home...");
        return true;
    } else if (id == R.id.action_share) {
        txtMsg.setText("Share...");

        return true;
    } else if (id == R.id.action_download) {
        txtMsg.setText("Download...");
        return true;
    } else if (id == R.id.action_about) {
        txtMsg.setText("About...");
        return true;
    } else if (id == R.id.action_settings) {
        txtMsg.setText("Settings...");
        return true;
    }

    return false;
} //onOptionsItemSelected

} //MainActivity
```



### Comments

1. During the execution of **onCreateOptionsMenu()** the items defined in the resource file *menu/main.xml* are added to the ActionBar. The statement  

```
txtSearchValue = (SearchView) menu.findItem(R.id.action_search)  
                    .getActionView();
```

gives you access to the SearchView, which is by default shown as a *search* icon.
2. The next step consists in defining a **QueryTextListener** on top of the SearchView. The listener has two methods. The first is **onQueryTextSubmit()** which is executed after the user taps on the virtual keyboard's the 'search' key. At this point, the text collected in the SearchView could be sent to a user-defined *search provider*. The statement *invalidateOptionsMenu()* closes the current search view and re-draws the ActionBar. The statement *.setQuery("", false)* is used to clear the text area of the newly created SearchView (not yet visible).
3. The second listening method **onQueryTextChange** is a text-watcher called after a new character is added to the SearchView. You may use this method to show suggestions progressively refined as more symbols are added to the query string.

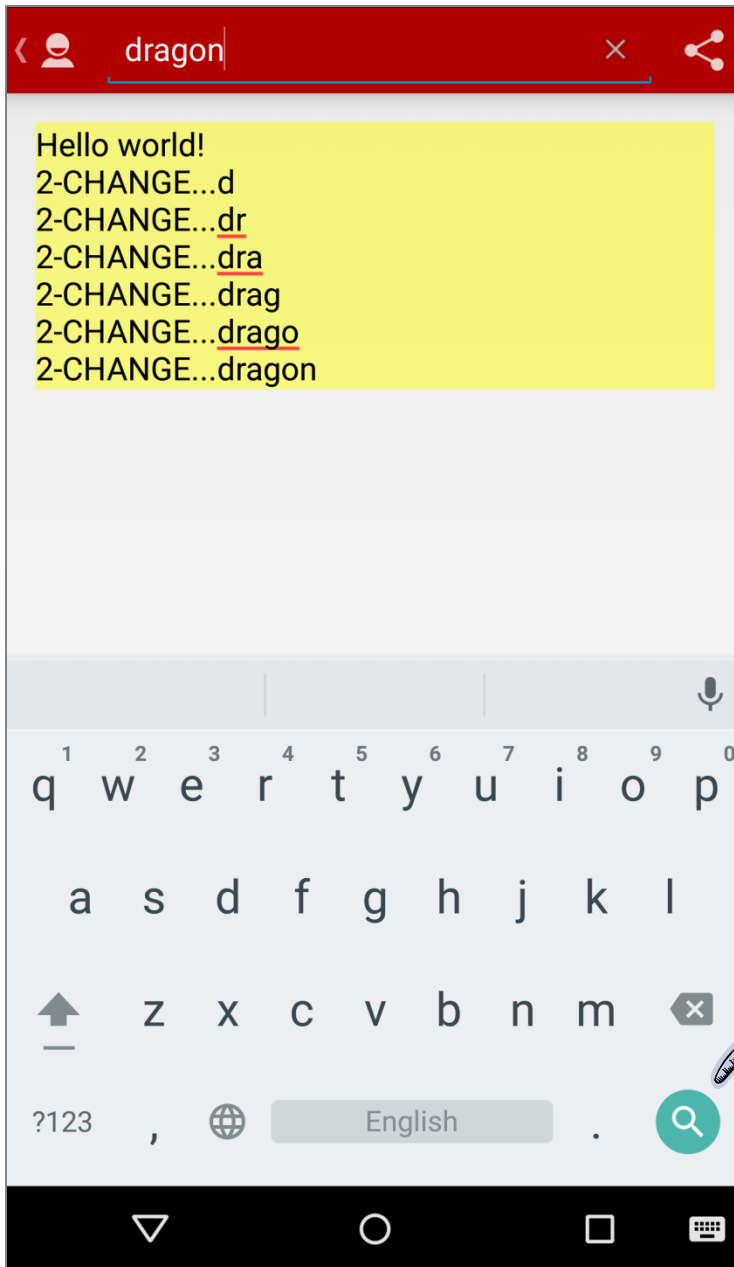


### Compatibility

```
// Get action bar
ActionBar actionBar = getSupportActionBar();

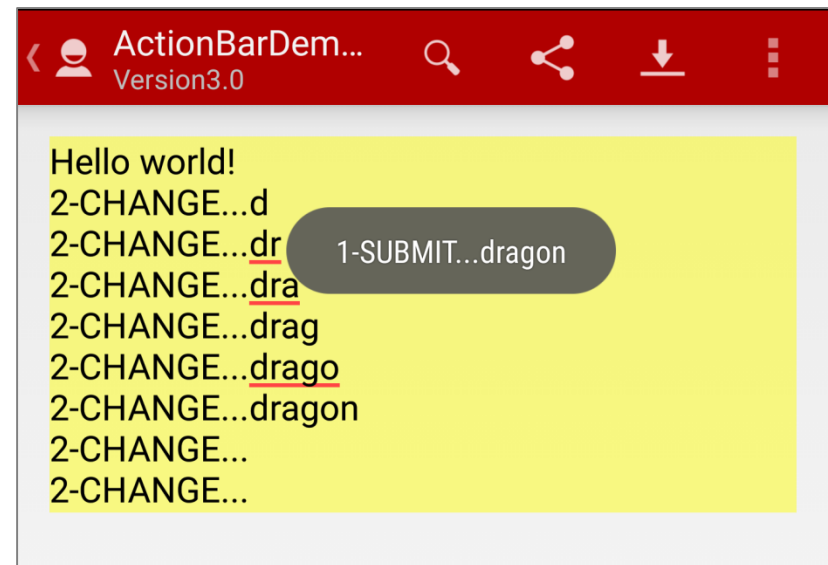
// Set actionView as a SearchView
app:actionViewClass="android.support.v7.widget.SearchView"
```

## Example 3 – Search ActionBar Tile



A sample screen illustrating the execution of Example3. Notice how the listener reacts to the typing of each new character into the query area of the SearchBox.

The toast-message appears after submitting the query string.



## Example 4 – CustomView - Spinner

In this example a custom view showing a **Spinner** widget is added to the ActionBar. The app's **res/menu/main.xml** is the same used in Example1 (showing Search, Share, Download, and Overflow-Items).

For example, The **Weather Channel** app presents in its ActionBar a **spinner** widget to set location.





## Example 4 – CustomView - Spinner

First, you need to define the XML layout for the custom view that is going to be added to the ActionBar. In our example the custom view is a simple Spinner specified as follows:

### custom\_spinner\_view\_on\_actionbar.xml

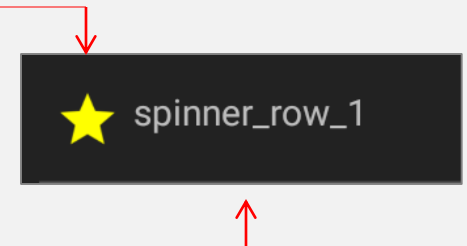
```
<Spinner
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/spinner_data_row"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```



In the next step , you state the layout of individual lines to be held by the Spinner.

### custom\_spinner\_row\_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:orientation="horizontal"    android:padding="6dp" >
    <ImageView
        android:id="@+id/imgSpinnerRowIcon"
        android:layout_width="25dp" android:layout_height="25dp"
        android:layout_marginRight="5dp"
        android:src="@drawable/ic_launcher" />
    <TextView
        android:id="@+id/txtSpinnerRowCaption"
        android:layout_width="wrap_content"    android:layout_height="wrap_content" />
</LinearLayout>
```



## Example 4 – CustomView - Spinner

We have chosen the **onCreate** method to attach the spinner custom view to the ActionBar. Then you add a data adapter and an 'ItemSelected' listener to the spinner

```
@Override
protected void onResume() {
    super.onResume();
    actionBar = getActionBar();                // setup the ActionBar
    actionBar.setDisplayShowCustomEnabled(true); // allow custom views to be shown
    actionBar.setDisplayHomeAsUpEnabled(true);  // show 'UP' affordance < button
    actionBar.setDisplayShowHomeEnabled(true); // allow app icon - logo to be shown
    actionBar.setHomeButtonEnabled(true);       // needed for API.14 or greater

    // move the spinner to the actionBar as a CustomView
    actionBar.setCustomView(R.layout.custom_spinner_view_on_actionbar);

    // create the custom adapter to feed the spinner
    customSpinnerAdapter = new SpinnerCustomAdapter(
        getApplicationContext(),
        SpinnerDummyContent.customSpinnerList);

    // plumbing - get access to the spinner widget shown on the actionBar
    customSpinner = (Spinner)
        actionBar.getCustomView().findViewById(R.id.spinner_data_row);

    // bind spinner and adapter
    customSpinner.setAdapter(customSpinnerAdapter);

    // put a listener to wait for spinner rows to be selected
    customSpinner.setOnItemClickListener(this);
    customSpinner.setSelection(selectedSpinnerRow);
} //onResume
```

## Example 4 – CustomView - Spinner

1 of 2

**SpinnerCustomAdapter.** The following is a custom adapter that inflates spinner rows. Each row holds an image and a caption as defined by **custom\_spinner\_row\_icon\_caption.xml**.

```
public class SpinnerCustomAdapter extends BaseAdapter {

    private ImageView spinnerRowIcon;
    private TextView spinnerRowCaption;
    private ArrayList<SpinnerRow> spinnerRows;
    private Context context;

    public SpinnerCustomAdapter(Context applicationContext,
                                ArrayList<SpinnerRow> customSpinnerList) {
        this.spinnerRows = customSpinnerList;
        this.context = applicationContext;
    }

    @Override
    public int getCount() {
        return spinnerRows.size();
    }

    @Override
    public Object getItem(int index) {
        return spinnerRows.get(index);
    }

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

## Example 4 – CustomView - Spinner

2 of 2

**SpinnerCustomAdapter.** The following is a custom adapter that inflates spinner rows. Each row holds an image and a caption as defined by **custom\_spinner\_row\_icon\_caption.xml**.

```
@Override
public View getView(final int position, View convertView, ViewGroup parent) {

    if (convertView == null) {
        LayoutInflater mInflater = (LayoutInflater) context.getSystemService(
            Activity.LAYOUT_INFLATER_SERVICE);
        convertView=mInflater.inflate(R.layout.custom_spinner_row_icon_caption, null);
    }

    spinnerRowIcon = (ImageView) convertView.findViewById(R.id.imgSpinnerRowIcon);
    spinnerRowCaption = (TextView) convertView.findViewById(
        R.id.txtSpinnerRowCaption);

    spinnerRowIcon.setImageResource(spinnerRows.get(position).getIcon());

    spinnerRowCaption.setText(spinnerRows.get(position).getCaption());

    convertView.setId(position);

    return convertView;
}
}
```

## Example 4 – CustomView - Spinner

1 of 2

**Dummy Data (SpinnerDummyContent.java).** Use the following code fragment to generate spinner's data.

```
public class SpinnerDummyContent {  
  
    public static ArrayList<SpinnerRow> customSpinnerList = new  
        ArrayList<SpinnerRow>();  
  
    static {  
  
        // preparing spinner data (a set of [caption, icon] objects)  
        customSpinnerList.add(new SpinnerRow("spinner_row_0",  
            R.drawable.ic_spinner_row_icon));  
        customSpinnerList.add(new SpinnerRow("spinner_row_1",  
            R.drawable.ic_spinner_row_icon));  
        customSpinnerList.add(new SpinnerRow("spinner_row_2",  
            R.drawable.ic_spinner_row_icon));  
        customSpinnerList.add(new SpinnerRow("spinner_row_3",  
            R.drawable.ic_spinner_row_icon));  
  
    }  
}
```

## Example 4 – CustomView - Spinner

2 of 2

**Dummy Data (SpinnerDummyContent.java).** Use the following code fragment to generate spinner's data.

```
public static class SpinnerRow { // each row consists of [caption, icon]
    private String caption;
    private int icon;

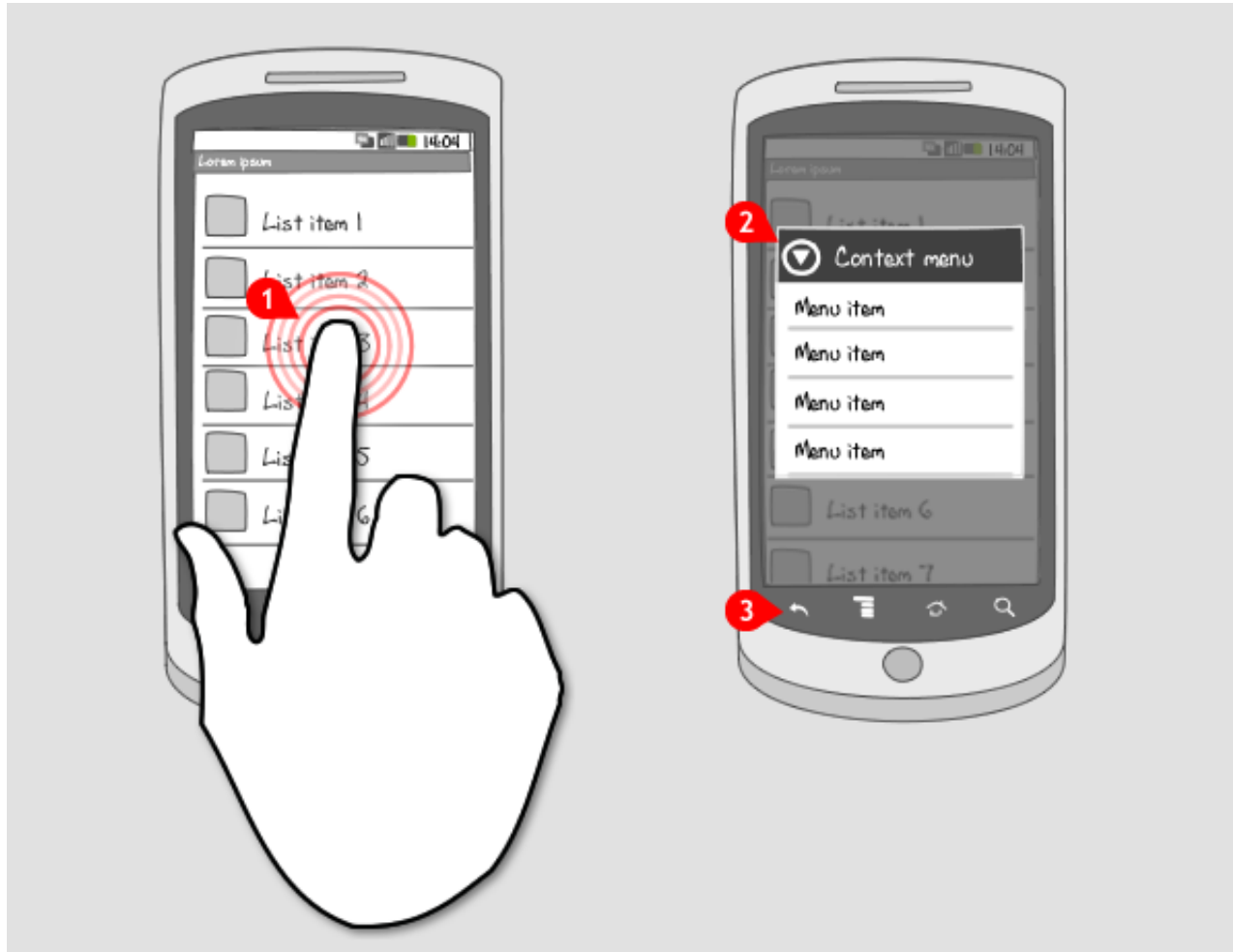
    public SpinnerRow(String caption, int icon) {
        this.caption = caption; this.icon = icon;
    }

    public String getCaption() {
        return this.caption;
    }

    public int getIcon() {
        return this.icon;
    }

    @Override
    public String toString() {
        return caption;
    }
} //SpinnerRow
}
```

## Example 5 – Context Menu



Tap and hold on an item to open the context menu

## Example 5 – Context Menu

```
registerForContextMenu(imageView);
```

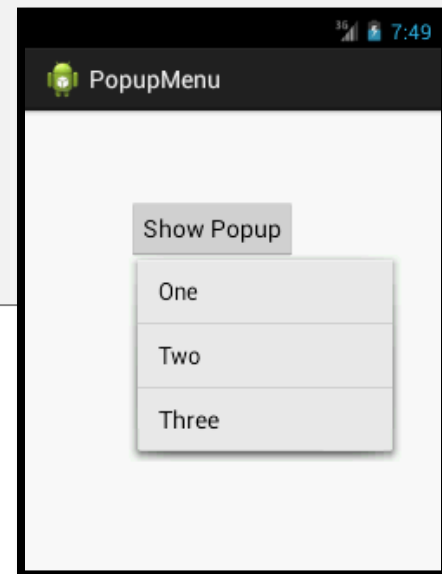
```
@Override  
public void onCreateContextMenu(ContextMenu menu, View v,  
ContextMenu.ContextMenuInfo menuInfo) {  
    super.onCreateContextMenu(menu, v, menuInfo);  
    menu.setHeaderTitle("Select action");  
    menu.add(0, CALL_ID, 0, "Call"); // groupId, itemId, order, title  
    menu.add(0, SMS_ID, 0, "SMS");  
}
```

```
@Override  
public boolean onContextItemSelected(MenuItem item) {  
    if(item.getTitle() == "Call") {  
        Toast.makeText(getApplicationContext(), "calling code",  
Toast.LENGTH_LONG).show();  
    }  
    else if(item.getTitle() == "SMS") {  
        Toast.makeText(getApplicationContext(), "sending sms code",  
Toast.LENGTH_LONG).show();  
    } else {  
        return false;  
    }  
    return true;  
}
```



## Example 6 – Popup Menu

```
findViewById(R.id.buttonPopup).setOnClickListener(new
View.OnClickListener() {
    @Override
    public void onClick(View view) {
        PopupMenu popupMenu = new PopupMenu(MainActivity.this,
buttonPopup);
        popupMenu.getMenuInflater().inflate(R.menu.popup_menu,
popupMenu.getMenu());
        popupMenu.setOnMenuItemClickListener(new
PopupMenu.OnMenuItemClickListener() {
            @Override
            public boolean onMenuItemClick(MenuItem menuItem) {
                Toast.makeText(MainActivity.this, menuItem.getTitle() + "
is selected", Toast.LENGTH_SHORT).show();
                return true;
            }
        });
        popupMenu.show();
    }
});
```

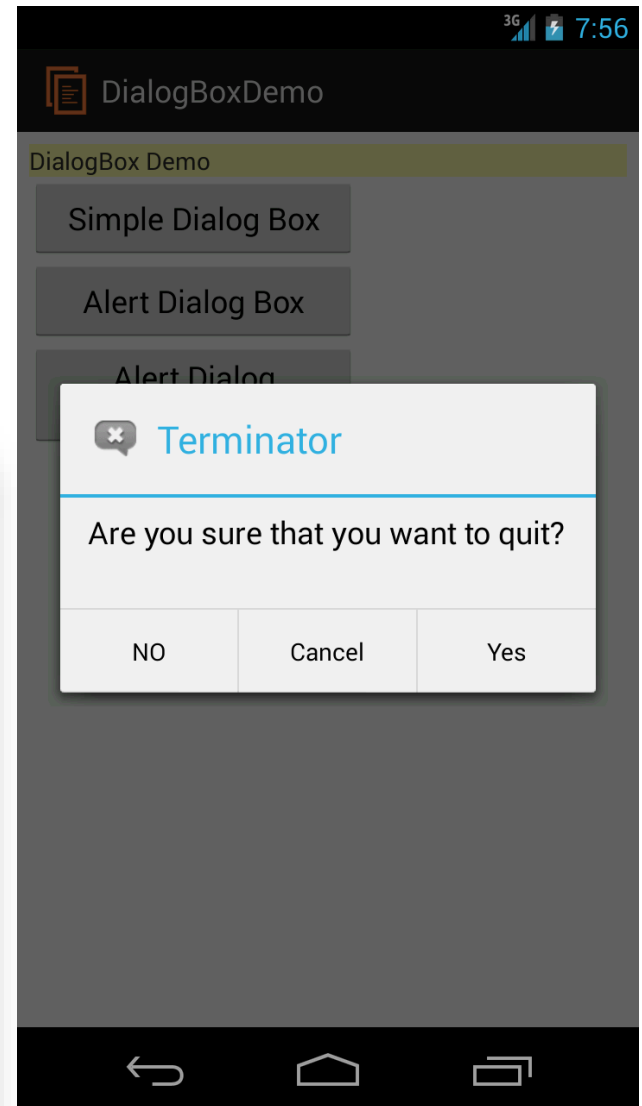
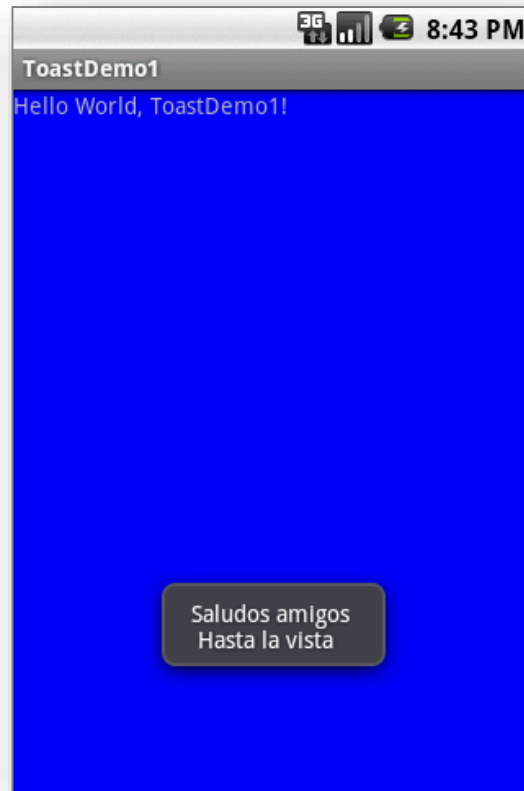


# Android DialogBoxes

Android provides two primitive forms of dialog boxes:

1. **AlertDialog** boxes, and
2. **Toast** views

Toasts are transitory boxes that –for a few seconds– flash a message on the screen, and then vanish without user intervention.



# The AlertDialog Box

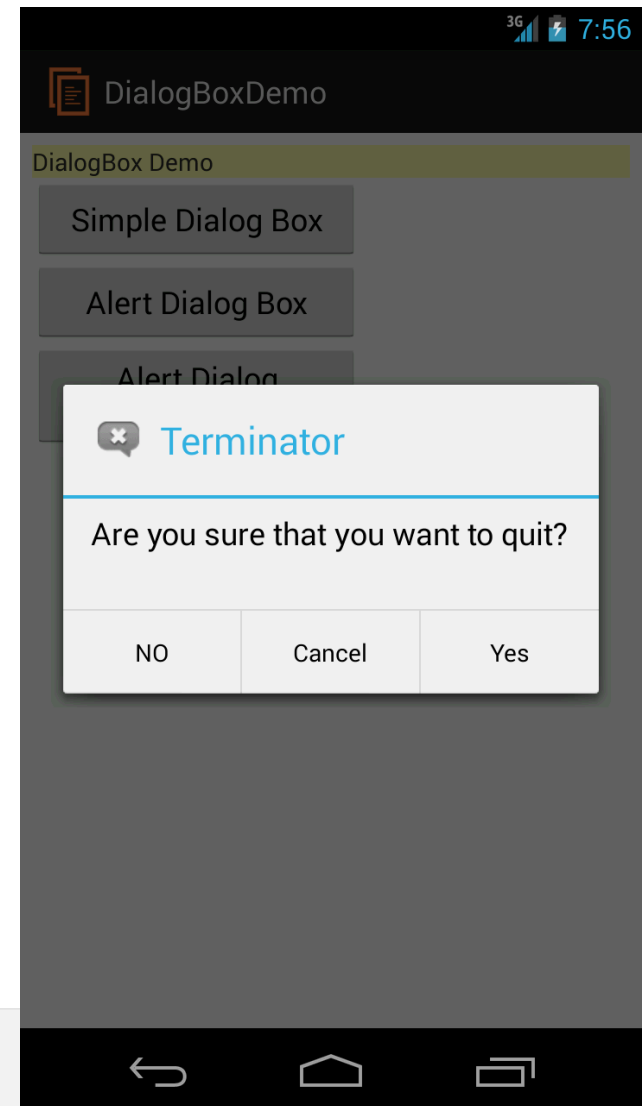
The **AlertDialog** is a message box that:

- (1) Displays as a small floating window on top of the (obscured) current UI.
- (2) The dialog window presents a message to the user as well as three optional buttons.
- (3) The box is dismissed by either clicking on the exposed buttons or touching any portion of the UI outside the borders of the DialogBox.

## Note:

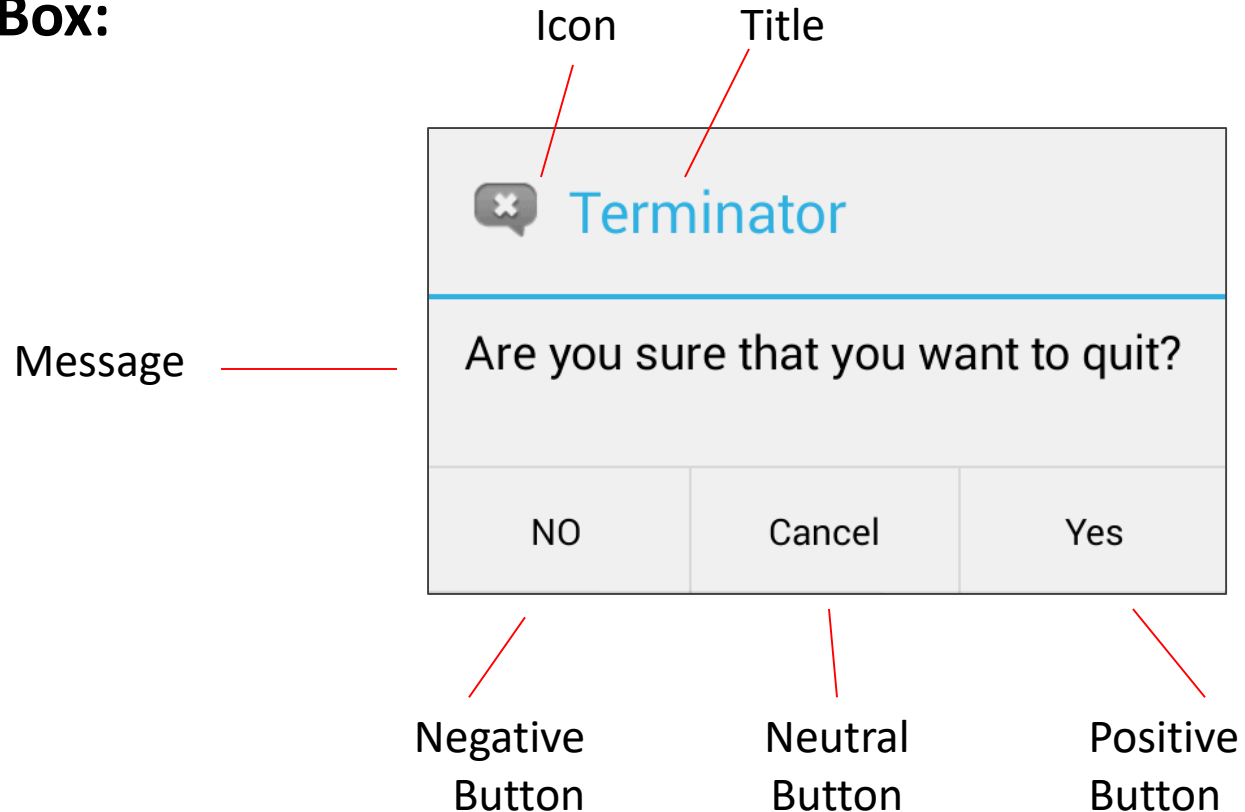
**Android's DialogBoxes are NOT modal views!**

A fully *modal* view remains on the screen waiting for user's input while *the rest of the application is on hold* (which is *not* the case of Android's DialogBoxes). A modal view (including Android's) has to be dismissed by an explicit user's action.



# The AlertDialog

## Dissecting an AlertDialog Box:



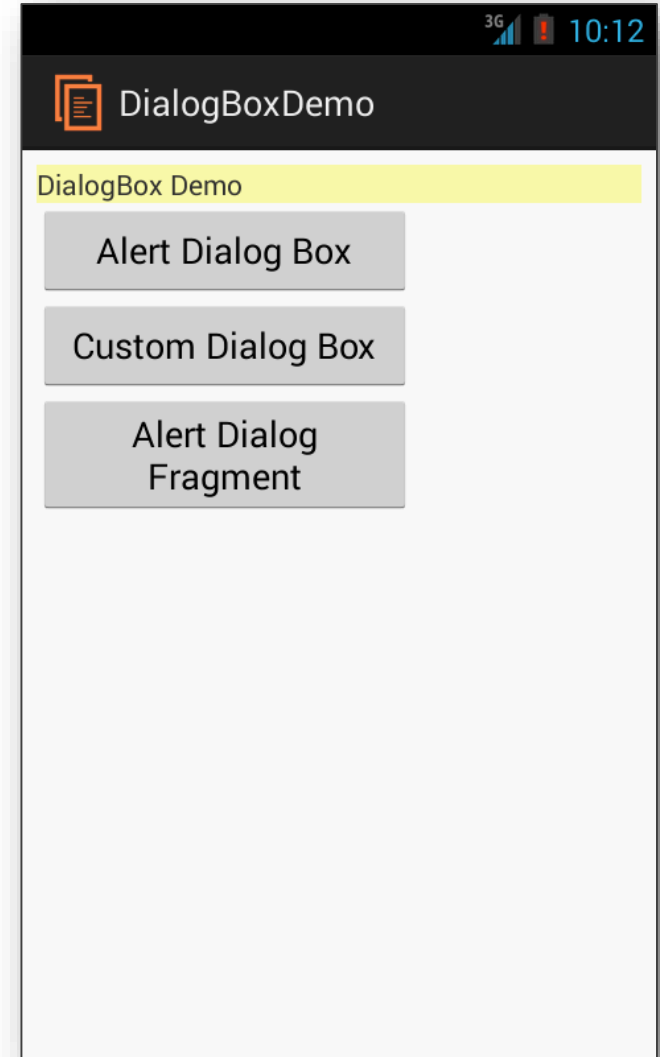
The image shown here uses:  
[Theme\\_Holo\\_Light\\_Dialog](#) and  
[STYLE\\_NORMAL](#)

# AlertDialog

## Example 1. AlertDialog Boxes

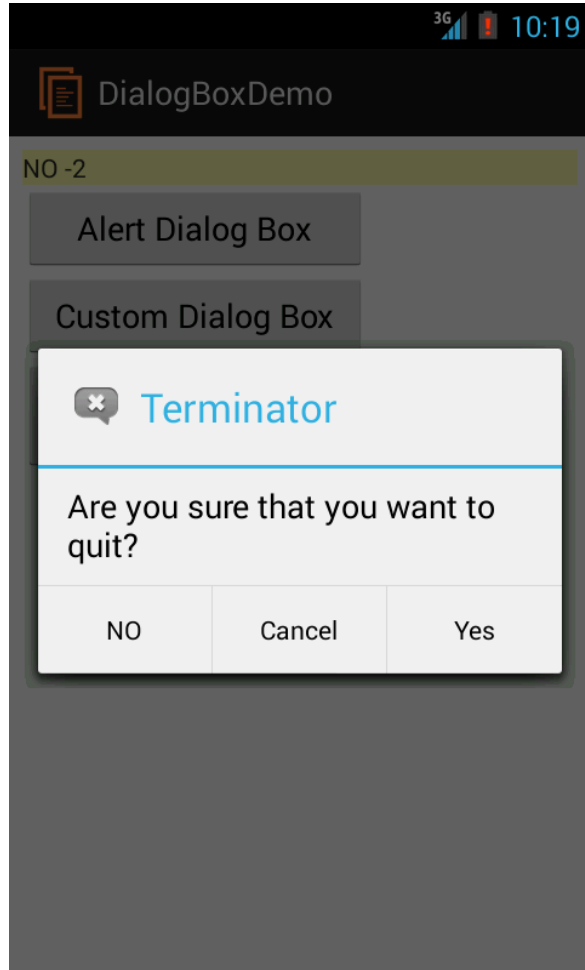
In this example the application's UI shows three buttons. When you click on them a different type of `AlertDialog` box is shown.

1. The first to be shown is a simple **AlertDialog** box with a message and buttons.
2. The second option is a **custom** `DialogBox` on which the user could type in a piece of data.
3. The last option shows a **DialogFragment** interacting with the main activity

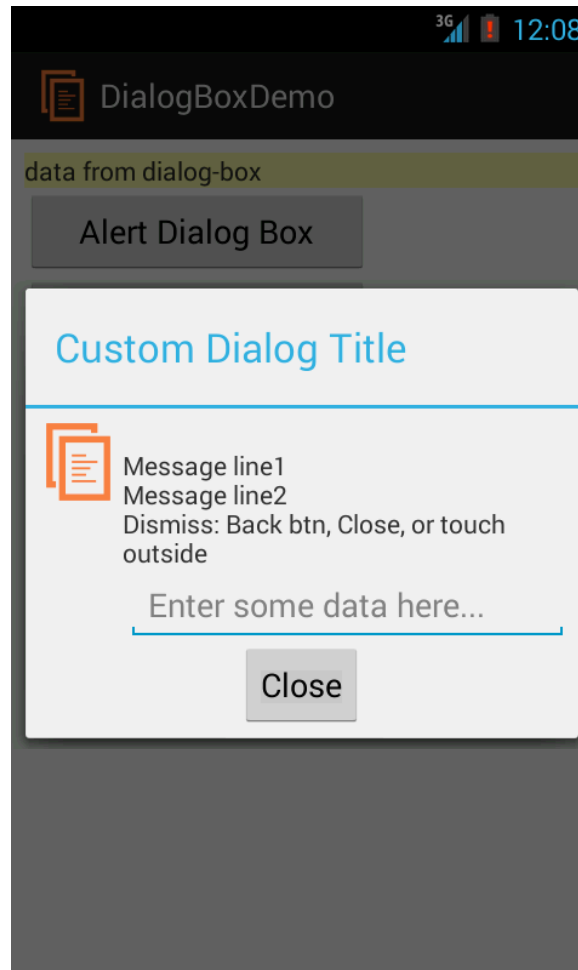


# AlertDialog

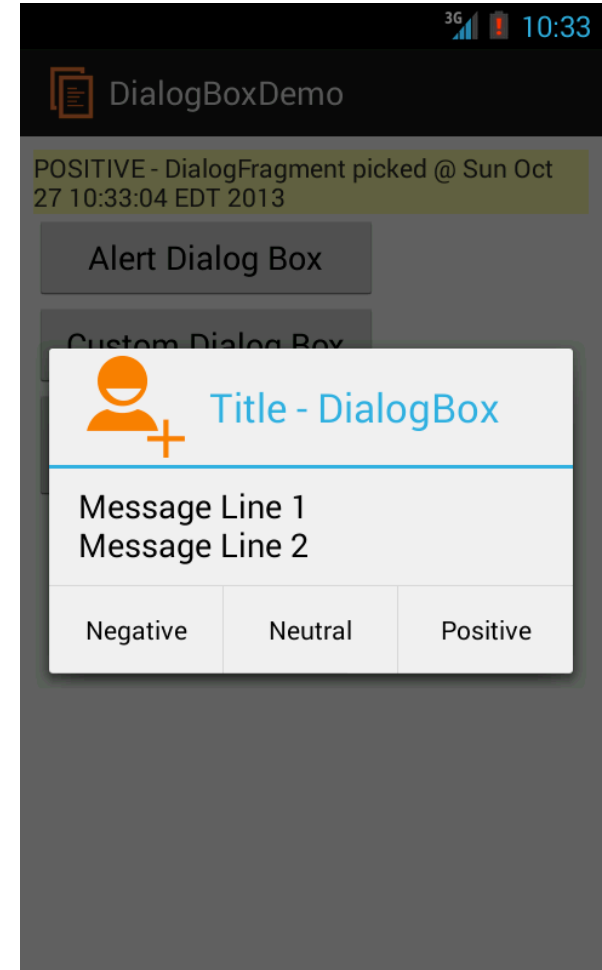
## Example 1. AlertDialog Boxes



A simple **AlertDialog** offering three choices.



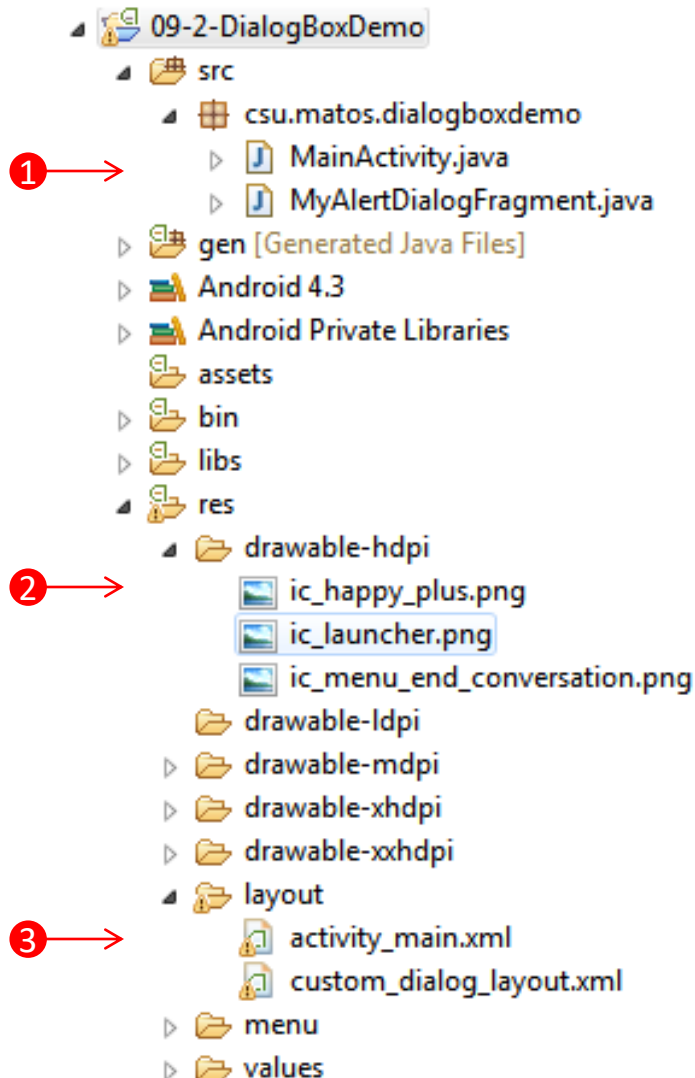
A **custom** AlertDialog allowing data to be typed.



A **DialogFragment** exposing three buttons.

# AlertDialog

## Example 1. App Structure



1. MainActivity shows main GUI and provides a frame for the DialogFragment to be displayed.
2. You want to enhance the appearance of dialog-boxes by adding meaningful icons. More details and tools at [Android Asset studio](http://j.mp/androidassetstudio) (<http://j.mp/androidassetstudio>)
3. Add your XML design indicating the way your custom AlertDialog looks like.

# AlertDialog

## Example 1. XML Layout – activity\_main.xml

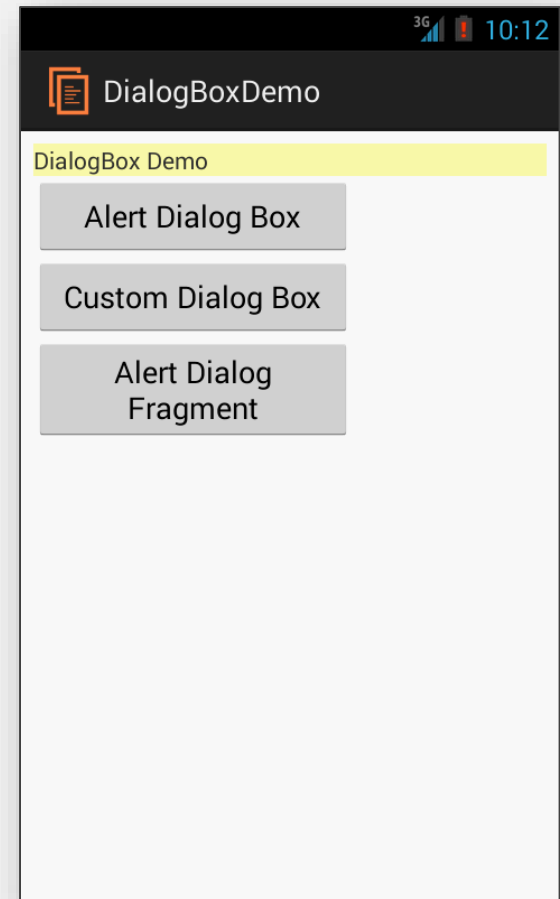
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"    android:layout_height="match_parent"
    android:orientation="vertical"        android:padding="7dp" >

    <TextView
        android:id="@+id/txtMsg"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#55ffff00"
        android:text="DialogBox Demo" />

    <Button
        android:id="@+id/btn_alert_dialog1"
        android:layout_width="190dp"
        android:layout_height="wrap_content"
        android:text="Alert Dialog Box" />

    <Button
        android:id="@+id/btn_custom_dialog"
        android:layout_width="190dp"
        android:layout_height="wrap_content"
        android:text="Custom Dialog Box" />

    <Button
        android:id="@+id/btn_alert_dialog2"
        android:layout_width="190dp"
        android:layout_height="wrap_content"
        android:text="Alert Dialog Fragment" />
</LinearLayout>
```





# AlertDialog

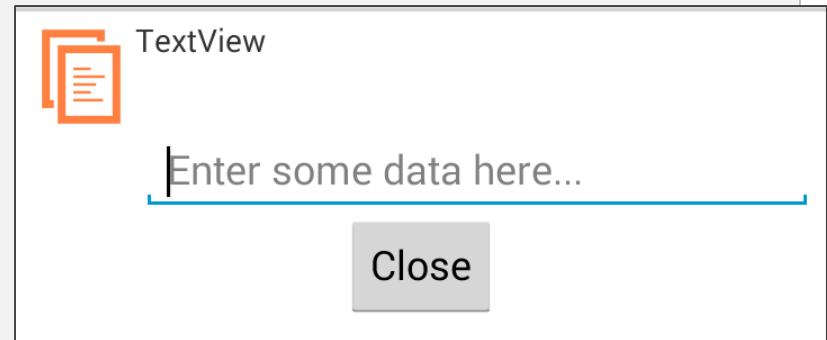
## Example 1. XML Layout – custom\_dialog\_layout.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:padding="5dp" >

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >

        <ImageView
            android:id="@+id/imageView1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@drawable/ic_launcher" />

        <TextView
            android:id="@+id/sd_textView1"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="TextView" />
    </LinearLayout>
</LinearLayout>
```



# AlertDialog

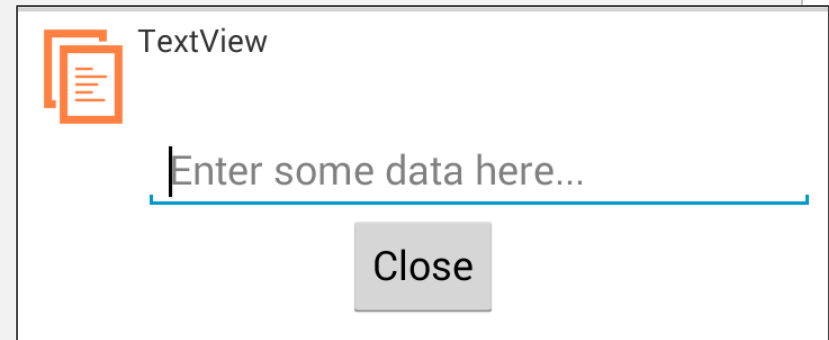
## Example 1. XML Layout – custom\_dialog\_layout.xml cont. 1

```
<EditText
    android:id="@+id/sd_editText1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="50dp"
    android:ems="15"
    android:hint="Enter some data here..." >

    <requestFocus />
</EditText>

<Button
    android:id="@+id/sd_btnClose"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:text="Close" />

</LinearLayout>
```



# AlertDialog

## Example 1. MainActivity.java

```
// example adapted from:  
// http://developer.android.com/reference/android/app/DialogFragment.html  
  
public class MainActivity extends Activity implements OnClickListener {  
    TextView txtMsg;  
    Button btnCustomDialog;  
    Button btnAlertDialog;  
    Button btnDialogFragment;  
    Context activityContext;  
    String msg = "";  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        activityContext = this;  
  
1 → txtMsg = (TextView) findViewById(R.id.txtMsg);  
    btnAlertDialog = (Button) findViewById(R.id.btn_alert_dialog1);  
    btnCustomDialog = (Button) findViewById(R.id.btn_custom_dialog);  
    btnDialogFragment = (Button) findViewById(R.id.btn_alert_dialog2);  
  
    btnCustomDialog.setOnClickListener(this);  
    btnAlertDialog.setOnClickListener(this);  
    btnDialogFragment.setOnClickListener(this);  
    }
```

# AlertDialog

## Example 1. MainActivity.java cont. 1

```
@Override
public void onClick(View v) {
    2 → if (v.getId() == btnAlertDialog.getId()) {
        showMyAlertDialog(this);
    }
    if (v.getId() == btnCustomDialog.getId()) {
        showCustomDialogBox();
    }
    if (v.getId() == btnDialogFragment.getId()) {
        showMyAlertDialogFragment(this);
    }
} // onClick

private void showMyAlertDialog(MainActivity mainActivity) {
    3 → new AlertDialog.Builder(mainActivity)
        .setTitle("Terminator")
        .setMessage("Are you sure that you want to quit?")
        .setIcon(R.drawable.ic_menu_end_conversation)

        // set three option buttons
        .setPositiveButton("Yes",
            new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int whichButton) {
                    // actions serving "YES" button go here
                    msg = "YES " + Integer.toString(whichButton);
                    txtMsg.setText(msg);
                }
            }) // setPositiveButton
```

# AlertDialog

## Example 1. MainActivity.java cont. 2

```
.setNeutralButton("Cancel",
    new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog,
            int whichButton) {
            // actions serving "CANCEL" button go here
            msg = "CANCEL " + Integer.toString(whichButton);
            txtMsg.setText(msg);
        } // onClick
    }) // setNeutralButton

.setNegativeButton("NO", new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int whichButton) {
        // actions serving "NO" button go here
        msg = "NO " + Integer.toString(whichButton);
        txtMsg.setText(msg);
    }
}) // setNegativeButton

.create()
.show();

} // showMyAlertDialog
```

# AlertDialog

## Example 1. MainActivity.java cont. 3

```
private void showCustomDialogBox() {  
    final Dialog customDialog = new Dialog(activityContext);  
    customDialog.setTitle("Custom Dialog Title");  
    // match customDialog with custom dialog layout  
    customDialog setContentView(R.layout.custom_dialog_layout);  
    ((TextView) customDialog.findViewById(R.id.sd_textView1))  
        .setText("\nMessage line1\nMessage line2\n"  
        + "Dismiss: Back btn, Close, or touch outside");  
    final EditText sd_txtInputData = (EditText) customDialog  
        .findViewById(R.id.sd_editText1);  
    ((Button) customDialog.findViewById(R.id.sd_btnClose))  
        .setOnClickListener(new OnClickListener() {  
            @Override  
            public void onClick(View v) {  
                txtMsg.setText(sd_txtInputData.getText().toString());  
                customDialog.dismiss();  
            }  
        });  
    customDialog.show();  
}
```

# AlertDialog

## Example 1. MainActivity.java cont. 4

```
private void showMyAlertDialogFragment(MainActivity mainActivity) {  
5 →   DialogFragment dialogFragment = MyAlertDialogFragment  
        .newInstance(R.string.title);  
  
    dialogFragment.show(getFragmentManager(), "TAG_MYDIALOGFRAGMENT1");  
}  
  
6 → public void doPositiveClick(Date time) {  
    txtMsg.setText("POSITIVE - DialogFragment picked @ " + time);  
}  
  
public void doNegativeClick(Date time) {  
    txtMsg.setText("NEGATIVE - DialogFragment picked @ " + time);  
}  
  
public void doNeutralClick(Date time) {  
    txtMsg.setText("NEUTRAL - DialogFragment picked @ " + time);  
}  
}
```

# AlertDialog

## Example 1. MainActivity.java

### Comments

1. The main UI shows three buttons and a TextView on which data coming from the executing dialog-boxes is to be written.
2. When a button is clicked the proper DialogBox is shown.
3. **showMyAlertDialog** uses a builder class to create a new AlertDialog adding to it a title, icon, message and three action buttons. Each action button has an onClick() method responsible for services to be rendered on behalf of the selection. We update the main UI's top TextView with the button's id.
4. The **custom** dialog-box is *personalized* when the `.setContentView(R.layout.custom_dialog_layout)` method is executed. Later, its "Close" button is given a listener, so the data entered in the dialog's EditText view could be sent to the UI's top TextView and, the box is finally dismissed.
5. A **DialogFragment** is instantiated. It's title is supplied as an argument to be 'bundled' when the fragment is created. Later the dialog will be show on top of the containing activity.
6. **Callback** methods (doPositive(), doNegative()...) are provided to empower the DialogFragment to pass data (a timestamp) back to the main activity.



# AlertDialog

## Example 1. MyAlertDialogFragment.java

```
public class MyAlertDialogFragment extends DialogFragment {  
    1 → public static MyAlertDialogFragment newInstance(int title) {  
        MyAlertDialogFragment frag = new MyAlertDialogFragment();  
  
        Bundle args = new Bundle();  
        args.putInt("title", title);  
        args.putString("message", "Message Line 1\nMessage Line 2");  
        args.putInt("icon", R.drawable.ic_happy_plus);  
  
        frag.setArguments(args);  
        return frag;  
    }  
  
    @Override  
    2 → public Dialog onCreateDialog(Bundle savedInstanceState) {  
        int title = getArguments().getInt("title");  
        int icon = getArguments().getInt("icon");  
        String message = getArguments().getString("message");  
  
        return new AlertDialog.Builder(getActivity())  
            .setIcon(icon)  
            .setTitle(title)  
            .setMessage(message)  
    }  
}
```



# AlertDialog

## Example 1. MyAlertDialogFragment.java cont. 1

```
3 → .setPositiveButton("Positive",
    new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog,
            int whichButton) {
            ((MainActivity) getActivity())
                .doPositiveClick(new Date());
        }
    })
.setNegativeButton("Negative",
    new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog,
            int whichButton) {
            ((MainActivity) getActivity())
                .doNegativeClick(new Date());
        }
    })
.setNeutralButton("Neutral",
    new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog,
            int whichButton) {
            ((MainActivity) getActivity())
                .doNeutralClick(new Date());
        }
    })
    .create();
}
```



# AlertDialog

## Example 1. MyAlertDialogFragment.java

### Comments

1. The class extends **DialogFragment**. It's *instantiator* accepts a title, message and icon arguments. As customary with fragments, the arguments are placed into a single bundle which is then associated to the fragment.
2. The **onCreateDialog** method extracts the arguments (title, icon, and message) from the DialogFragment's bundle. A common AlertDialog builder is called to prepare the dialog box using the supplied arguments.
3. Three option buttons are added to the DialogFragment. Each has a listener that when activated, makes its onClick method interact with a callback method in the MainActivity. To illustrate that data from the fragment could be passed from the dialog-box, a timestamp is supplied to the callbacks.

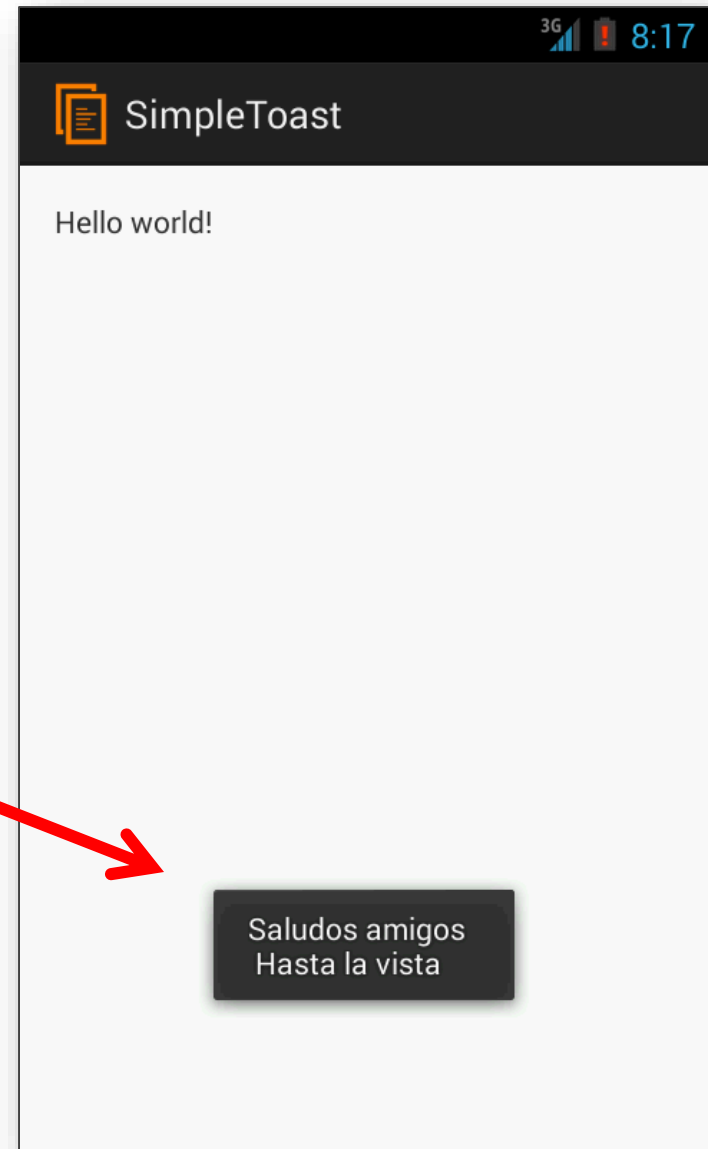
# The Toast Widget

**Toasts** are very simple one-way message boxes.

Typically they are used in situations in which a **brief message** should be flashed to the user.

A toast is shown as a semi-opaque floating view over the application's UI. It's lifetime is between 2-4 sec.

Notoriously, *Toasts never receive focus !*



# The Toast Widget

## Example 2. Toast's Syntax

```
Toast.makeText ( context, message, duration ).show();
```

*Context:* A reference to the view's environment (where am I, what is around me...)

*Message:* The message you want to show

*Duration:* Toast.LENGTH\_SHORT (0) about 2 sec  
Toast.LENGTH\_LONG (1) about 3.5 sec

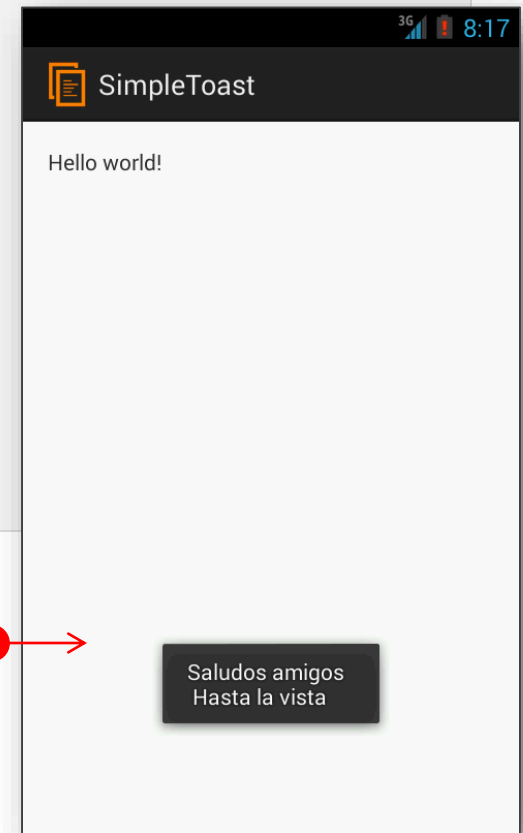
The Toast class has only a few methods including: *makeText*, *show*, *setGravity*, and *setMargin*.

# The Toast Widget

## Example 2. A Simple Toast

```
public class MainActivity extends Activity {  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
  
        Toast.makeText( getApplicationContext(),  
                        "Saludos amigos \n Hasta la vista",  
                        Toast.LENGTH_LONG).show();  
    }  
}
```

In this simple application, passing the **context** variable could be done using: `getApplicationContext()`, `MainActivity.this`, or simply using **this**.



# The Toast Widget

## Example 3. Re-positioning a Toast View



- By **default** Toast views are displayed at the **center-bottom** of the screen.
- However the user may change the placement of a Toast view by using either of the following methods:

```
void setGravity (int gravity, int xOffset, int yOffset)
```

```
void setMargin (float horizontalMargin, float verticalMargin)
```

# The Toast Widget

## Example 3. Re-positioning a Toast View



### Method 1

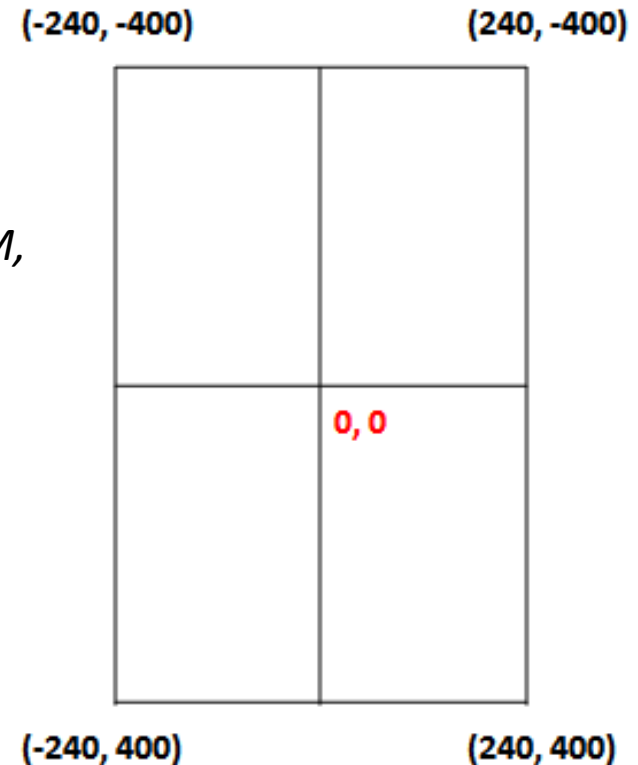
```
void setGravity (int gravity, int xOffset, int yOffset)
```

( Assume the phone has a **480x800** screen density)

**gravity:** Overall placement. Typical values include:  
*Gravity.CENTER, Gravity.TOP, Gravity.BOTTOM,*  
(see *Appendix B*)

**xOffset:** The *xOffset* range is -240,...,0,...240  
left, center, right

**yOffset:** The *yOffset* range is: -400,...,0,...400  
top, center, bottom





# The Toast Widget

## Example 3. Re-positioning a Toast View



### Method 2

- The (0,0) point – *Center of the screen* – occurs where horizontal and vertical center lines cross each other.
- There is 50% of the screen to each side of that center point
- Margins are expressed as a percent value between: -50,..., 0, ..., 50.

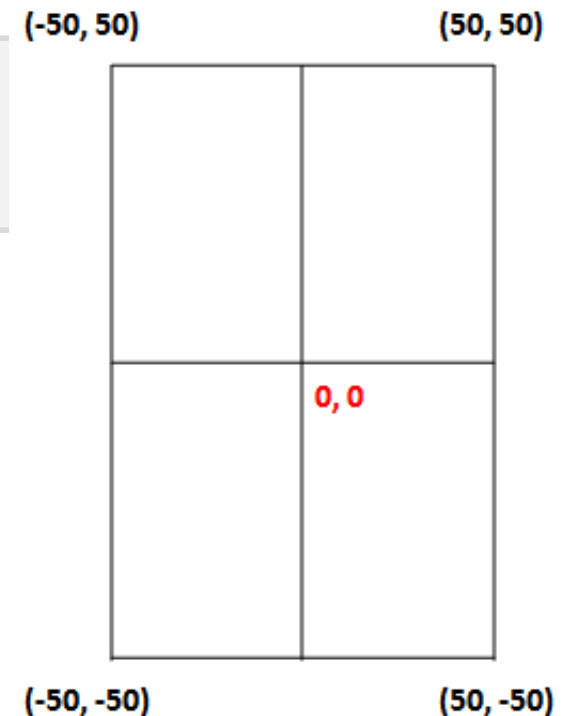
```
void setMargin (float horizontalMargin,  
               float verticalMargin)
```

**Note:** The pair of margins:

(-50, -50) represent the lower-left corner of the screen,

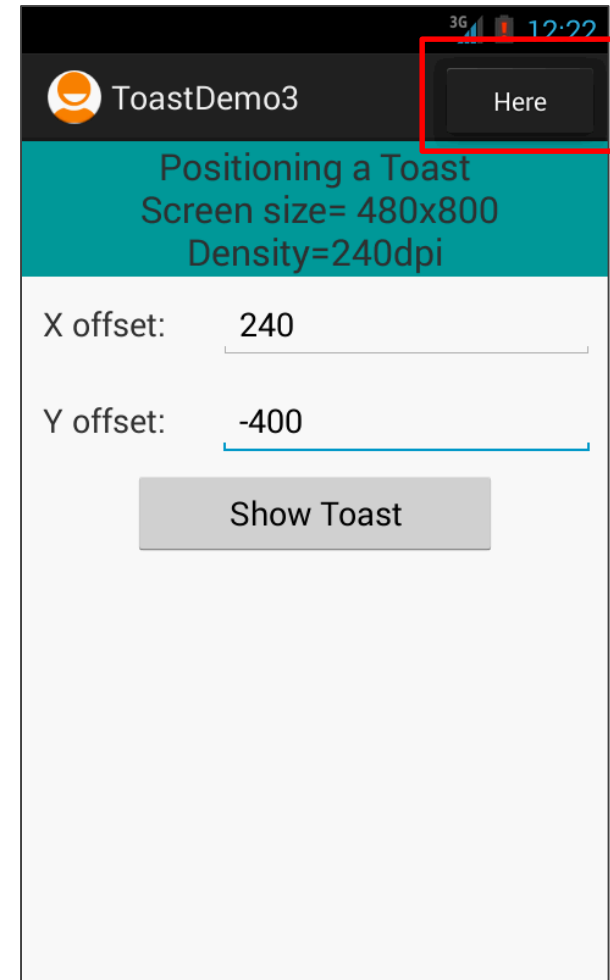
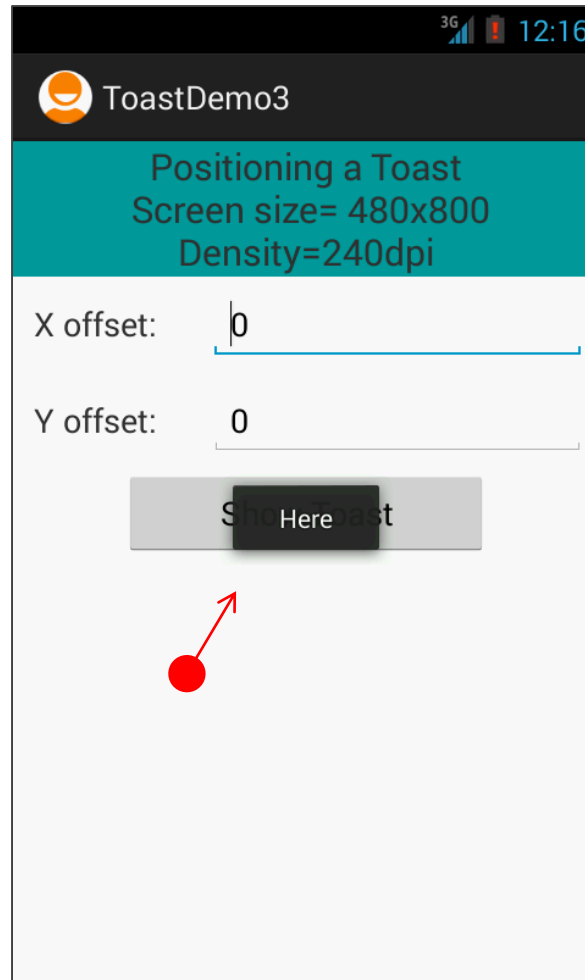
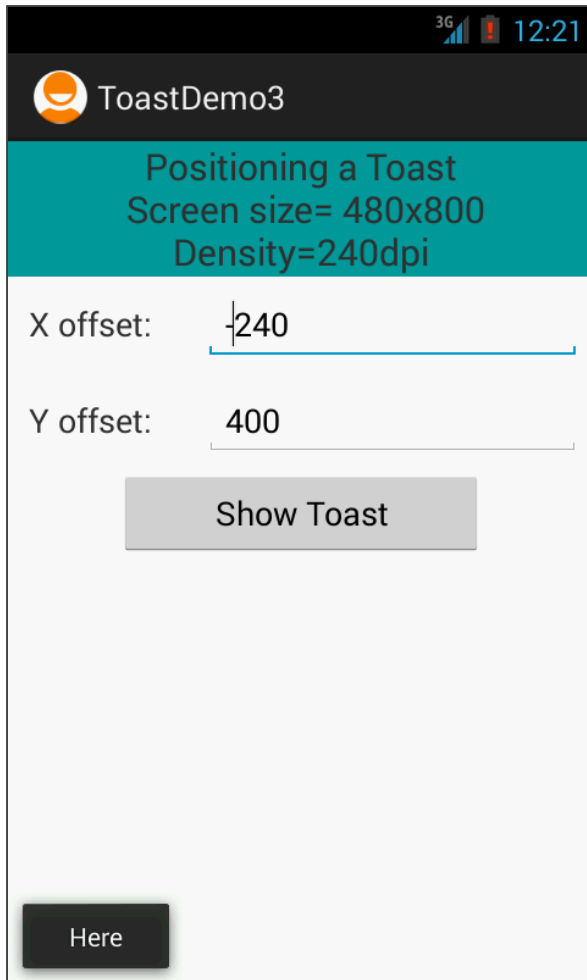
( 0, 0) is the center, and

(50, 50) the upper-right corner.



# The Toast Widget

## Example 3. Re-positioning a Toast View



# The Toast Widget

## Example 3. XML Layout: activity\_main.xml

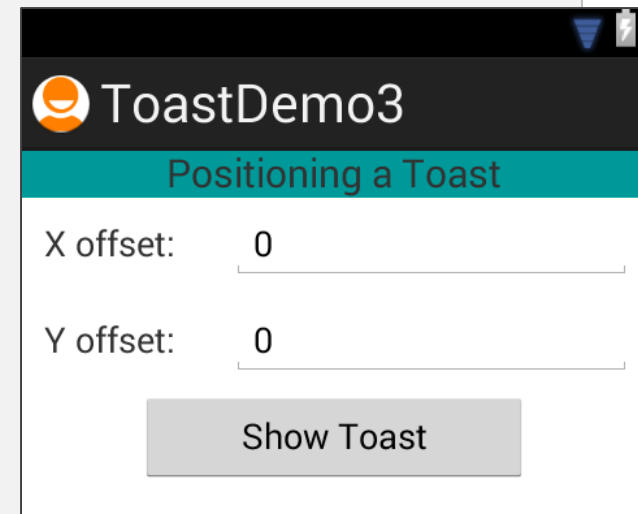
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/txtCaption"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#ff009999"
        android:gravity="center"
        android:text="Positioning a Toast"
        android:textSize="20sp" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="10dp" >

        <TextView
            android:layout_width="100dp"
            android:layout_height="wrap_content"
            android:text=" X offset: "
            android:textSize="18sp" />

        <EditText
            android:id="@+id/txtXCoordinate"
```



# The Toast Widget

## Example 3. XML Layout: activity\_main.xml

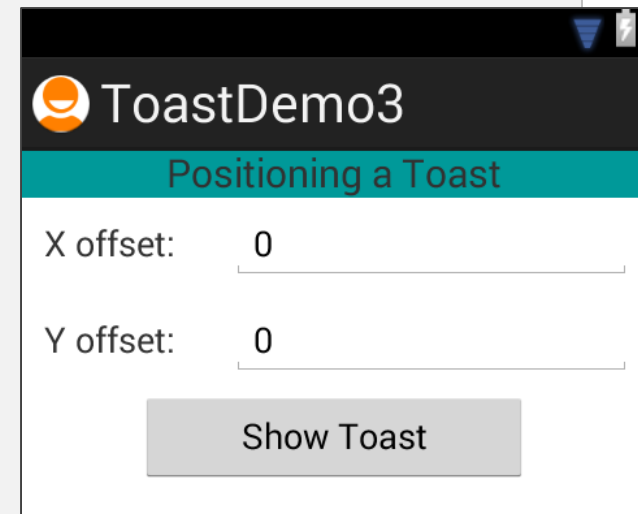
cont. 1

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="2"
        android:inputType="numberSigned"
        android:text="0"
        android:textSize="18sp" />
</LinearLayout>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:padding="10dp" >

    <TextView
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:text=" Y offset: "
        android:textSize="18sp" />

    <EditText
        android:id="@+id/txtYCoordinate"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="2"
        android:inputType="numberSigned"
        android:text="0"
        android:textSize="18sp" />
</LinearLayout>
```

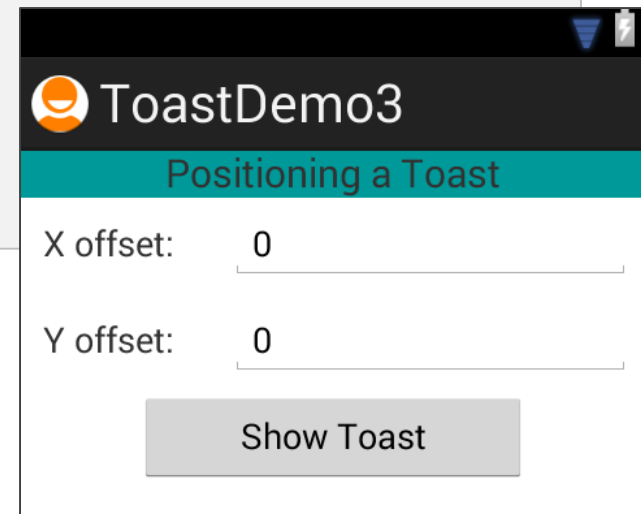


# The Toast Widget

## Example 3. XML Layout: activity\_main.xml

cont. 2

```
<Button
    android:id="@+id/btnShowToast"
    android:layout_width="200dp"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:text=" Show Toast " >
</Button>
</LinearLayout>
```



# The Toast Widget

## Example 3. MainActivity: ToastDemo3.java

```
public class ToastDemo3 extends Activity {
    EditText txtXCoordinate;
    EditText txtYCoordinate;
    TextView txtCaption;

    Button btnShowToast;

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

        // bind GUI and Java controls
        txtCaption = (TextView) findViewById(R.id.txtCaption);
        txtXCoordinate = (EditText) findViewById(R.id.txtXCoordinate);
        txtYCoordinate = (EditText) findViewById(R.id.txtYCoordinate);
        btnShowToast = (Button) findViewById(R.id.btnShowToast);

        // find screen-size and density(dpi)
        int dpi = Resources.getSystem().getDisplayMetrics().densityDpi;
        int width= Resources.getSystem().getDisplayMetrics().widthPixels;
        int height = Resources.getSystem().getDisplayMetrics().heightPixels;
        txtCaption.append("\n Screen size= " + width + "x" + height
            + " Density=" + dpi + "dpi");
    }
}
```

# The Toast Widget

### Example 3. MainActivity: ToastDemo3.java

## cont. 1

```
// show toast centered around selected X,Y coordinates
btnShowToast.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        try {
            Toast myToast = Toast.makeText(getApplicationContext(),
                "Here", Toast.LENGTH_LONG);

            myToast.setGravity(
                Gravity.CENTER,
                Integer.valueOf(txtXCoordinate.getText().toString()),
                Integer.valueOf(txtYCoordinate.getText().toString()));

            myToast.show();

        } catch (Exception e) {
            Toast.makeText(getApplicationContext(), e.getMessage(),
                Toast.LENGTH_LONG).show();
        }
    }
});

// onCreate

class
```

# The Toast Widget

## Example 3. MainActivity: ToastDemo3.java

### Comments

1. Plumbing. GUI objects are bound to their corresponding Java controls. When the button is clicked a Toast is to be shown.
2. The call `Resources.getSystem().getDisplayMetrics()` is used to determine the screen size (Height, Width) in pixels, as well as its density in dip units.
3. An instance of a Toast is created with the *makeText* method. The call to `setGravity` is used to indicate the (X,Y) coordinates where the toast message is to be displayed. X and Y refer to the actual horizontal/vertical pixels of a device's screen.



# The Toast Widget

## Example 4. A Custom-Made Toast View

Toasts could be modified to display a custom combination of color, shape, text, image, and background.

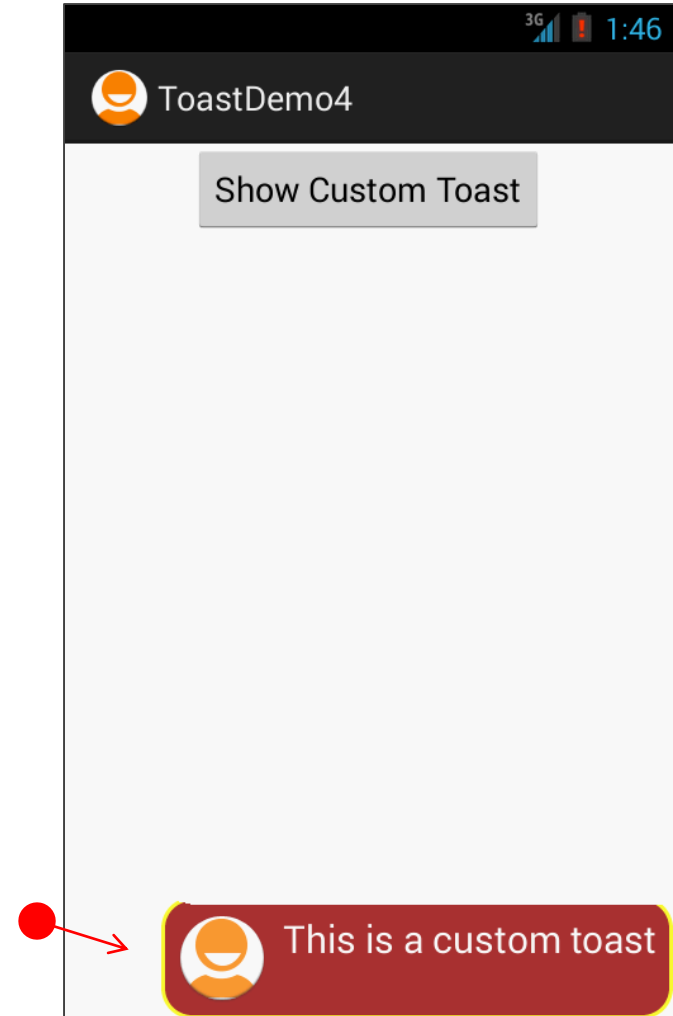
### Steps

To create a custom Toast do this:

1. Define the XML layout you wish to apply to the custom toasts.
2. In addition to a TextView where the toast's message will be shown, you may add other UI elements such as an image, background, shape, etc.
3. Inflate the XML layout. Attach the new view to the toast using the `setView()` method.

Example based on:

<http://hustleplay.wordpress.com/2009/07/23/replicating-default-android-toast/>  
<http://developer.android.com/guide/topics/ui/notifiers/toasts.html>



# The Toast Widget

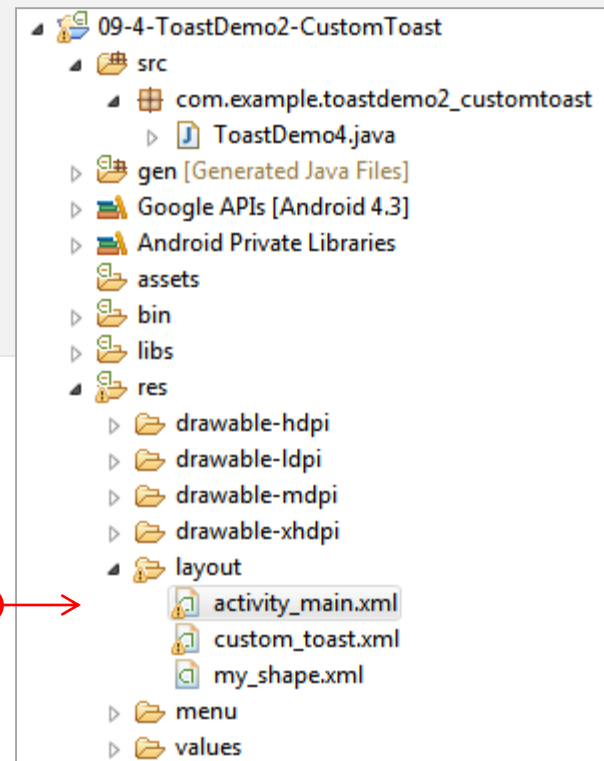
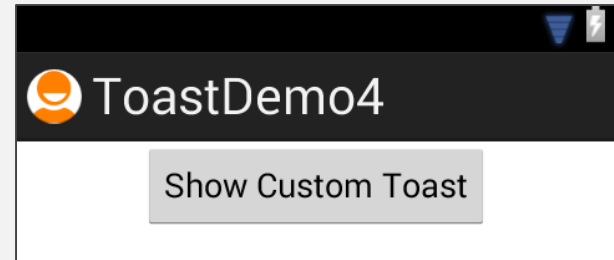
## Example 4. XML Layout - activity\_main.xml

```
<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" >
```

```
<Button
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="showCustomToast"
    android:text="Show Custom Toast"
    android:layout_gravity="center"
    tools:context=".ToastDemo4" />
```

```
</LinearLayout>
```



# The Toast Widget

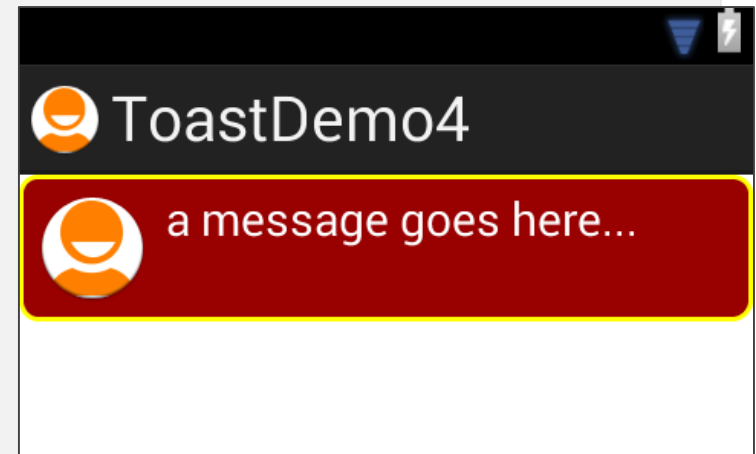
## Example 4. XML Layout - custom\_toast.xml

```
<?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:background="@layout/my_shape"
    android:orientation="horizontal"
    android:padding="8dp" >

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginRight="8dp"
        android:src="@drawable/ic_launcher" />

    <TextView
        android:id="@+id/toast_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="a message goes here..."
        android:textColor="#ffffffff"
        android:textSize="20sp" />

</LinearLayout>
```



# The Toast Widget

## Example 4. XML Layout - my\_shape.xml

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

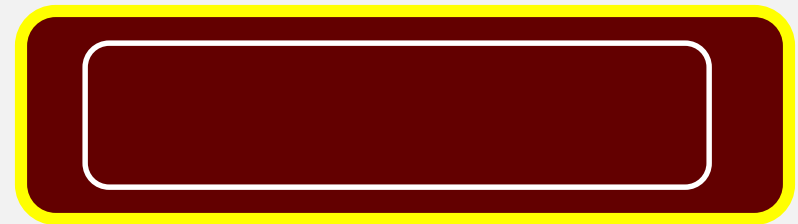
    <stroke
        android:width="2dp"
        android:color="#ffffff00" />

    <solid android:color="#ff990000" />

    <padding
        android:bottom="4dp"
        android:left="10dp"
        android:right="10dp"
        android:top="4dp" />

    <corners android:radius="15dp" />

</shape>
```



**Note:** A basic shape is a drawable such as a rectangle or oval. Defining attributes are stroke(border) , solid(interior part of the shape), corners, padding, margins, etc. Save this file in the **res/layout** folder. For more information see **Appendix A**.

# The Toast Widget

## Example 4. MainActivity - ToastDemo4.java

```
public class ToastDemo4 extends Activity {

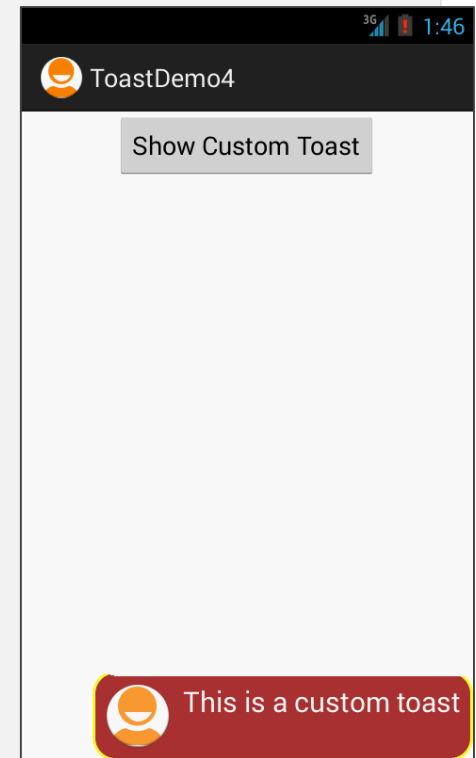
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

    } // onCreate

    public void showCustomToast(View v){
        // //////////////////////////////////////
        // this fragment creates a custom Toast showing
        // image + text + shaped_background
        // triggered by XML button's android:onClick=...

        Toast customToast = makeCustomToast(this);

        customToast.show();
    }
}
```



# The Toast Widget

## Example 4. MainActivity - ToastDemo4.java

cont. 1

```
protected Toast makeCustomToast(Context context) {  
    // Reference:  
    // http://developer.android.com/guide/topics/ui/notifiers/toasts.html  
  
    1 → LayoutInflater inflater = getLayoutInflater();  
  
    View layout = inflater.inflate( R.layout.custom_toast, null);  
  
    TextView text = (TextView) layout.findViewById(R.id.toast_text);  
    text.setText("This is a custom toast");  
  
    2 → Toast toast = new Toast(context);  
    toast.setMargin(50,-50); //lower-right corner  
    toast.setDuration	Toast.LENGTH_LONG);  
  
    3 → toast.setView(layout);  
  
    return toast;  
  
} //makeCustomToast  
  
} //ToastDemo2
```

# The Toast Widget

## Example 4. MainActivity - ToastDemo4.java

### Comments

1. After the custom toast layout is inflated, you gain control to its TextView in which the user's message will be held.
2. The toast is positioned using the `setMargin()` method to the lower right corner of the screen (50, -50)
3. The inflated view is attached to the newly created Toast object using the `.setView()` method.

# ProgressDialog

## Example 5. ProgressDialog

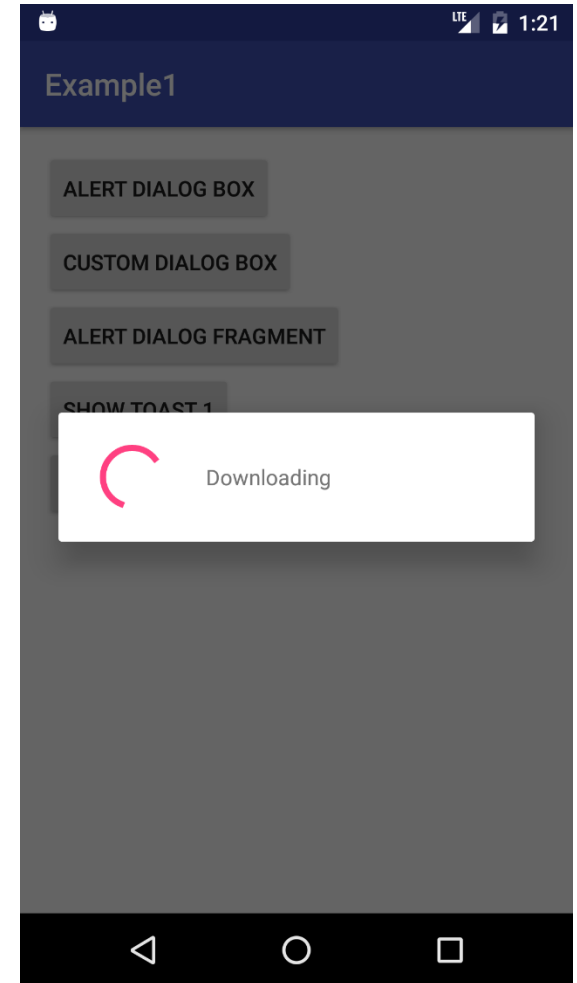
```
private class DownloadTask extends AsyncTask<Void, Integer, Boolean> {

    ProgressDialog progressDialog;

    @Override
    protected Boolean doInBackground(Void... voids) {
        try {
            Thread.sleep(5000);
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return true;
    }

    @Override
    protected void onPreExecute() {
        progressDialog = new ProgressDialog(MainActivity.this);
        progressDialog.setMessage("Downloading");
        progressDialog.setCanceledOnTouchOutside(false);
        progressDialog.show();
    }

    @Override
    protected void onPostExecute(Boolean aBoolean) {
        if (progressDialog.isShowing())
            progressDialog.dismiss();
    }
}
```





# ProgressDialog

## Example 5. ProgressDialog

```
private class DownloadTask extends AsyncTask<Void, Integer, Boolean> {

    ProgressDialog progressDialog;

    @Override
    protected Boolean doInBackground(Void... voids) {
        try {
            for (int i = 0; i < 10; i++) {
                Thread.sleep(500);
                publishProgress(i * 10);
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return true;
    }

    @Override
    protected void onPreExecute() {
        progressDialog = new ProgressDialog(MainActivity.this);
        progressDialog.setMessage("Downloading");
        progressDialog.setCanceledOnTouchOutside(false);
        progressDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
        progressDialog.setMax(100);
        progressDialog.show();
    }

    @Override
    protected void onPostExecute(Boolean aBoolean) {
        if (progressDialog.isShowing())
            progressDialog.dismiss();
    }

    @Override
    protected void onProgressUpdate(Integer... values) {
        progressDialog.setProgress(values[0]);
    }
}
```

