# Lesson 7

# Top Décor Elements: ActionBars, Menus, Toolbars

**Victor Matos**
Cleveland State University

# Android Design Strategies

## The ActionBar Design Model

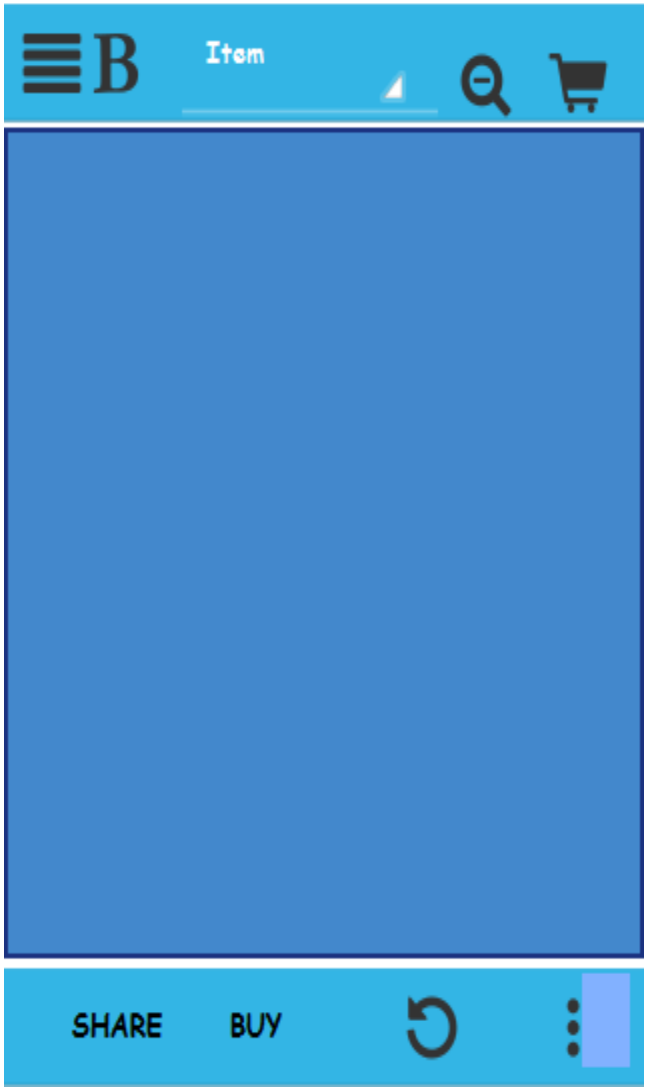
- **Successful UI Design** focuses on *gracefully* exposing all the capabilities of an app through individual elements of UI design such as headings, footers, search boxes, buttons, and so on to provide a harmonious, intuitive, pleasant, and rich user experience.

- Android -as a general purpose platform- benefits from repeated operational patterns that facilitate the process of 'naturally' interacting with their apps without an immerse training and/or reading of user guides.

- As you will discover, the **ActionBar** and **Toolbar** top décor elements are not only easy to use, but they also facilitate the exposing of additional custom actions, menus, and structural navigation in a simple and elegant way.

# 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.**

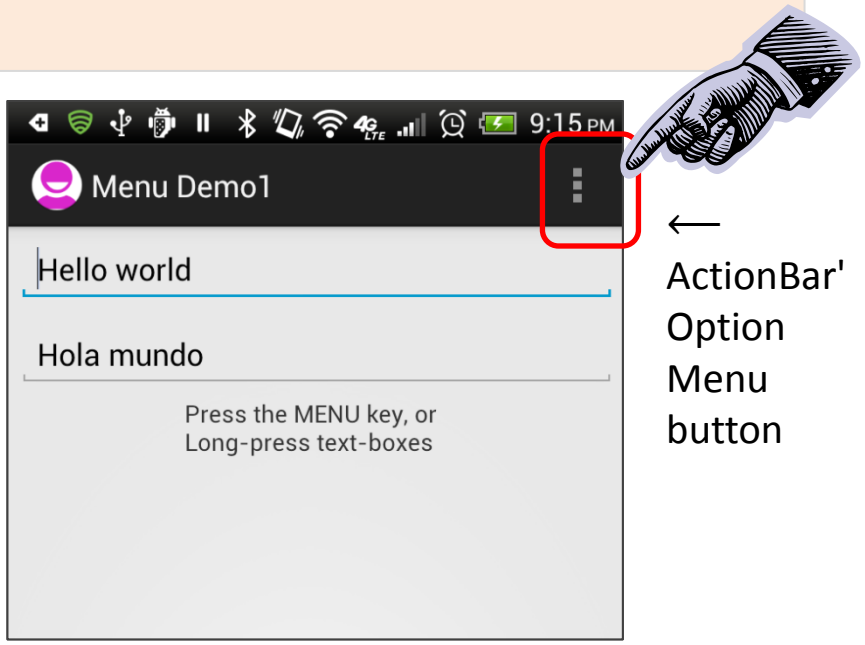
# Using Menus

## Menu Types

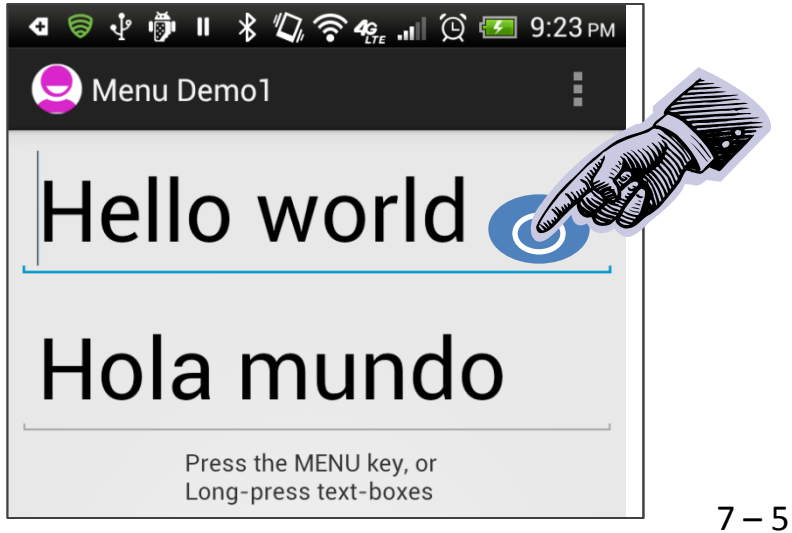
Android supports two types of menus: **Options Menu** and **Context Menu**.

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' Option Menu button



# 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*)

# ActionBar Design Strategy

The ActionBar is an important architectural element because

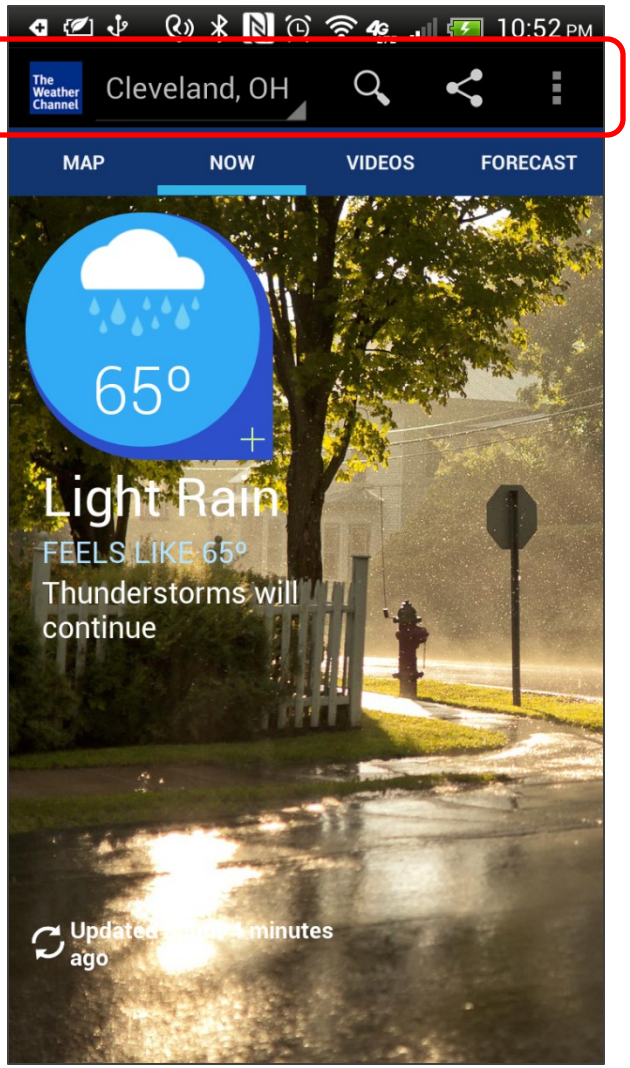
- Acts as a **condensed switchboard** intuitively exposing the app's main organization and functionality.
- It hosts on its rightmost position an unobtrusive **overflow menu** button '⋮' which could be customized for each UI.
- It normally appears on top of each screen, which conveys a feeling of **coherent design** where '*all important buttons are in the same place*', consequently easing the user's experience.
- Can be used to support various **navigation** patterns, for instance
  - (a) its "hamburger" button could display an overlapping DrawerView holding a list of important entry points in the app,
  - (b) embedded or neighboring horizontal tabs could be tapped to expose a selected view (perhaps a page from a ViewPager control)
  - Its **UP button** could be used to jump back to the previously visited screen or any higher place in the app's View-Hierarchy.

# 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 |

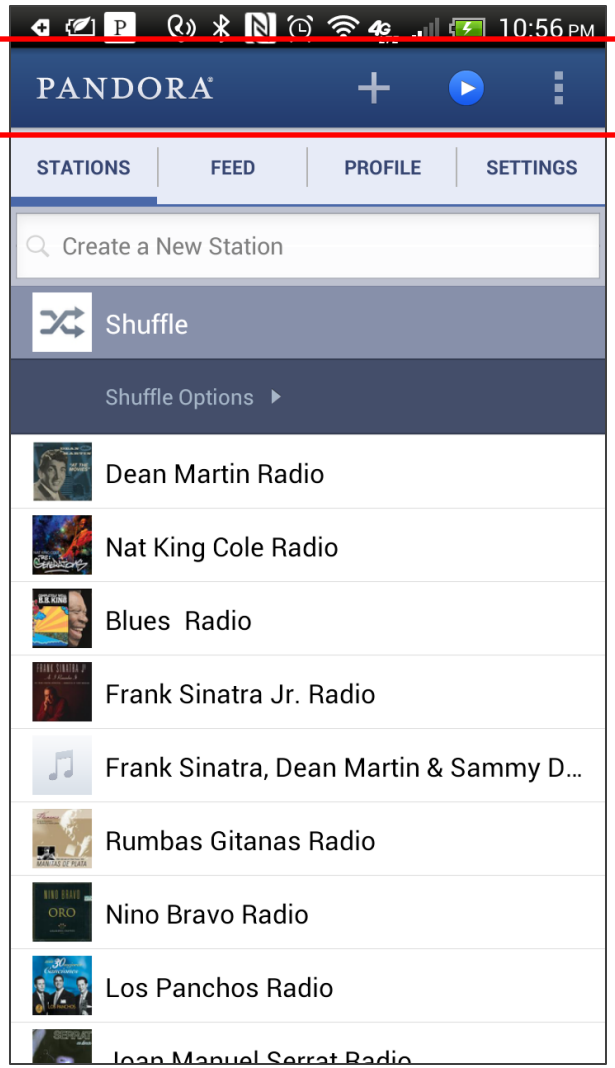
**Source**:
Google Play Store. Last visited: Feb 16, 2015. Link: https://play.google.com/store?hl=en

# 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 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.

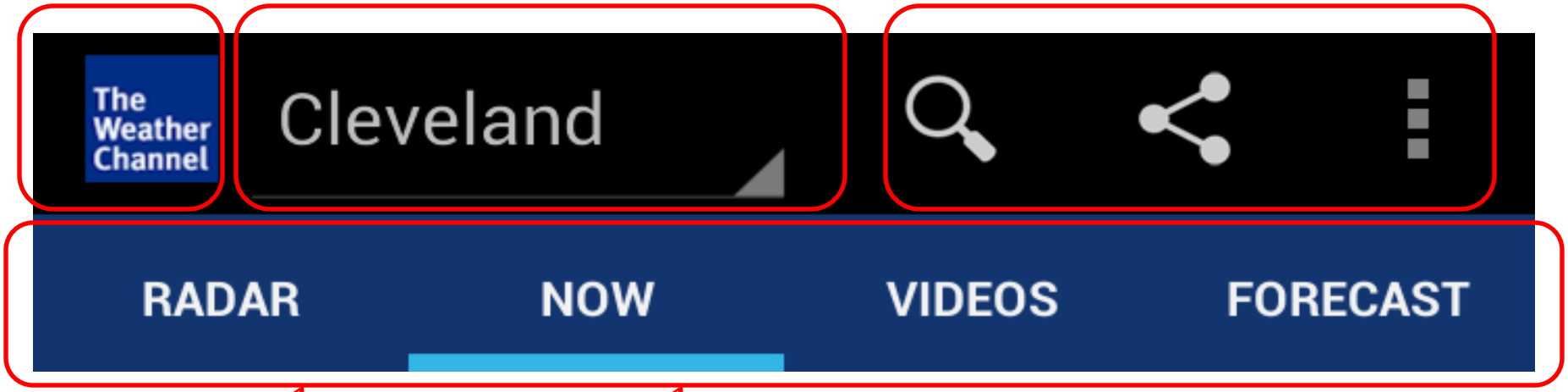
# Reverse Engineering an App's ActionBar

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 "OptionMenu" 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 buton. 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, disable, and changed.

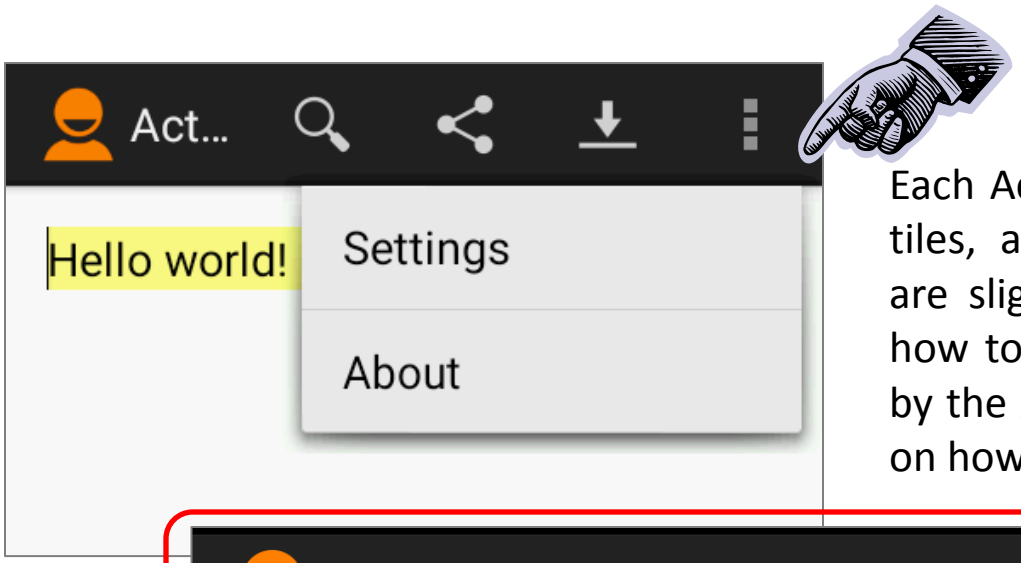
Two methods are responsible for most of the work related to interacting with the tiles on an ActionBar

**onCreateOptionsMenu( … )**
Inflates the XML specs defining each of the action-tiles.

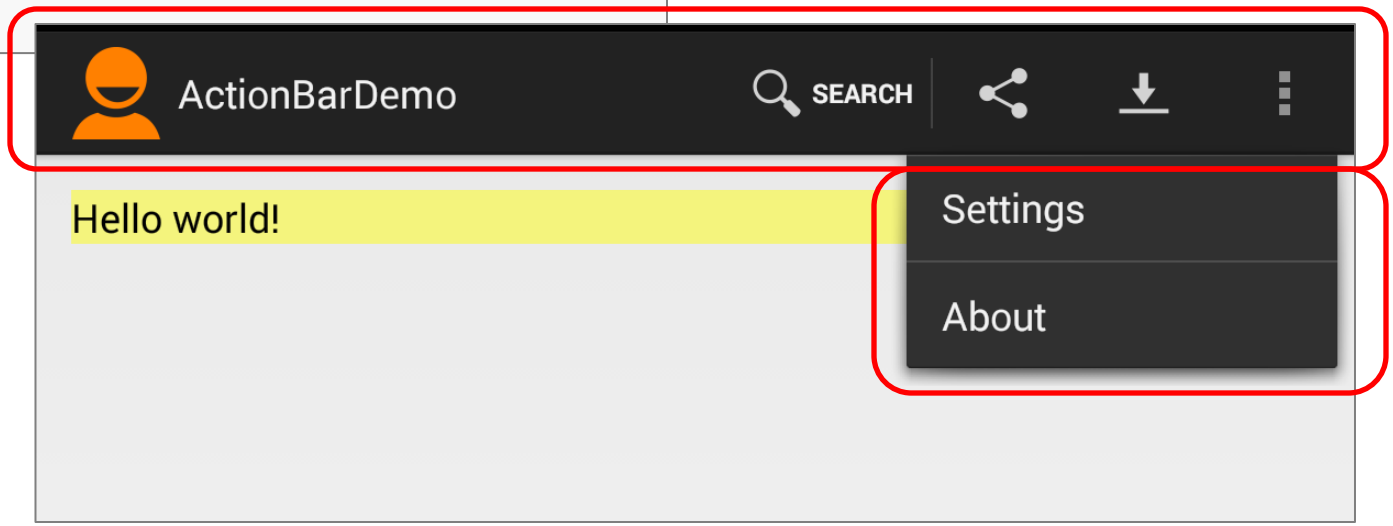
**onOptionsItemSelected( … )**
Captures the click-event on any tile and dispatches the proper response to the user's request.

# 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.




Each ActionBar contains: logo, title, action-tiles, and overflow menu button. Images are slightly different, the final decision of how to best display the ActionBar is made by the Android runtime environment based on how much screen space is available.



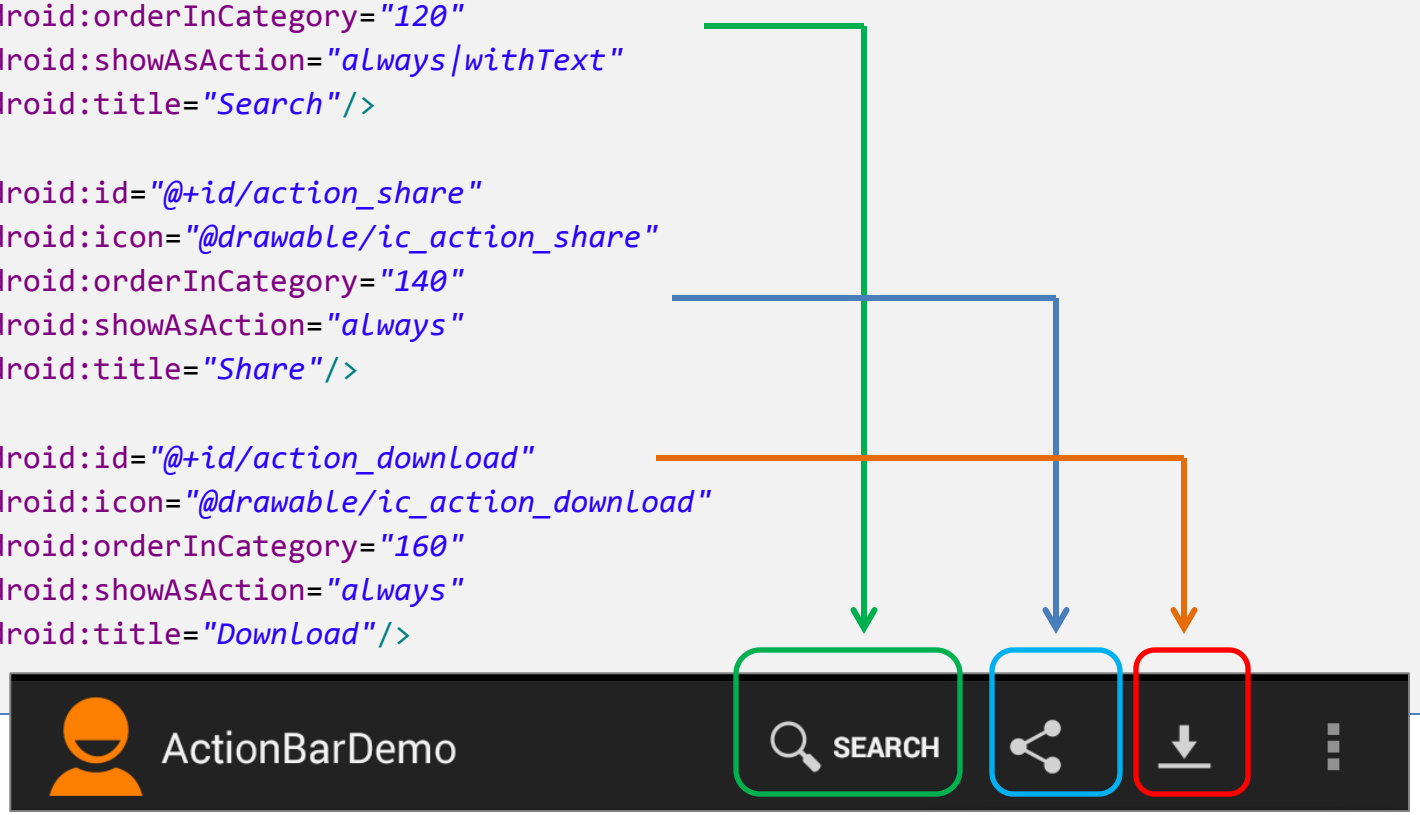

→ ActionBar

← Overflow Action Menu

# Example 1 – Creating a Simple ActionBar

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

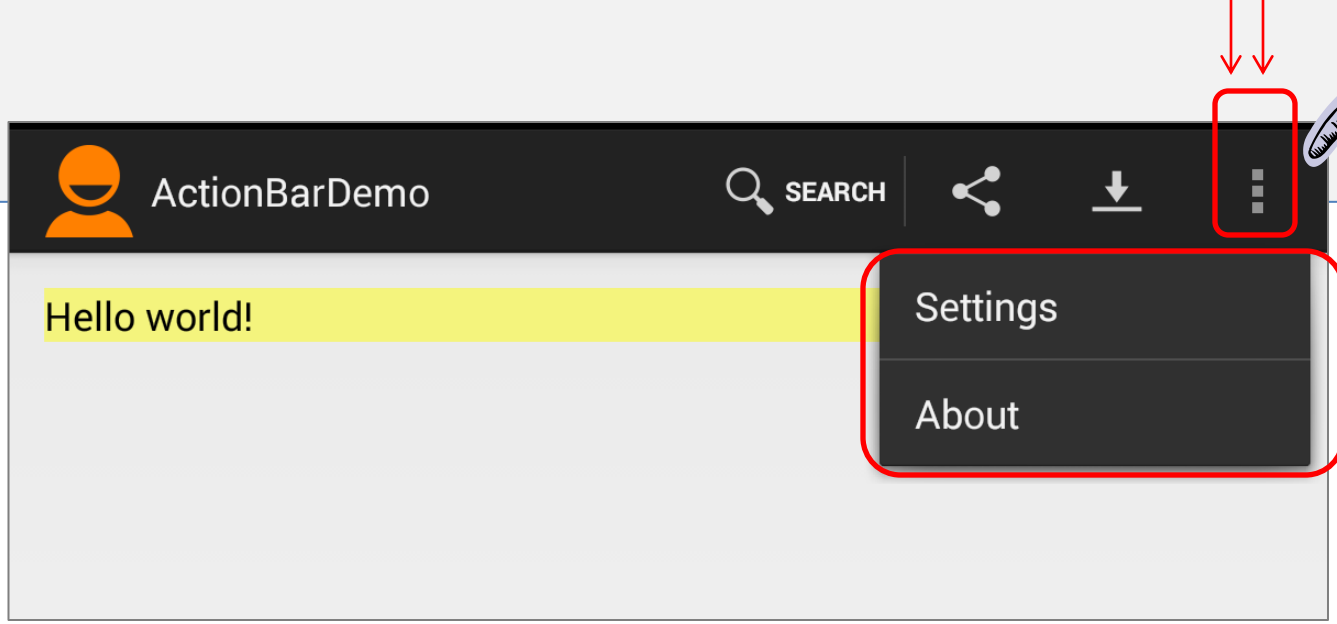
```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context="csu.matos.MainActivity" >

    <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"/>
```



# 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>
```

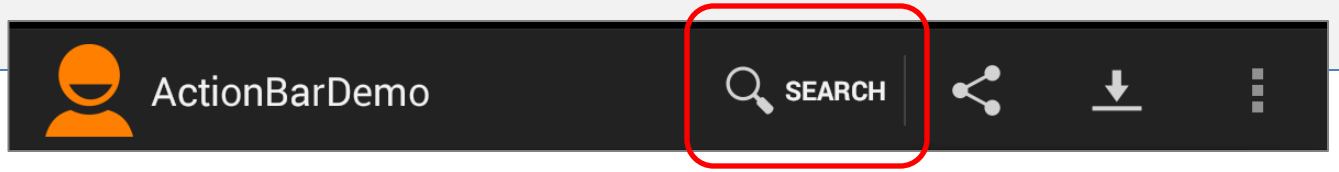



Overflow Menu

# Example 1 – Creating a Simple ActionBar

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

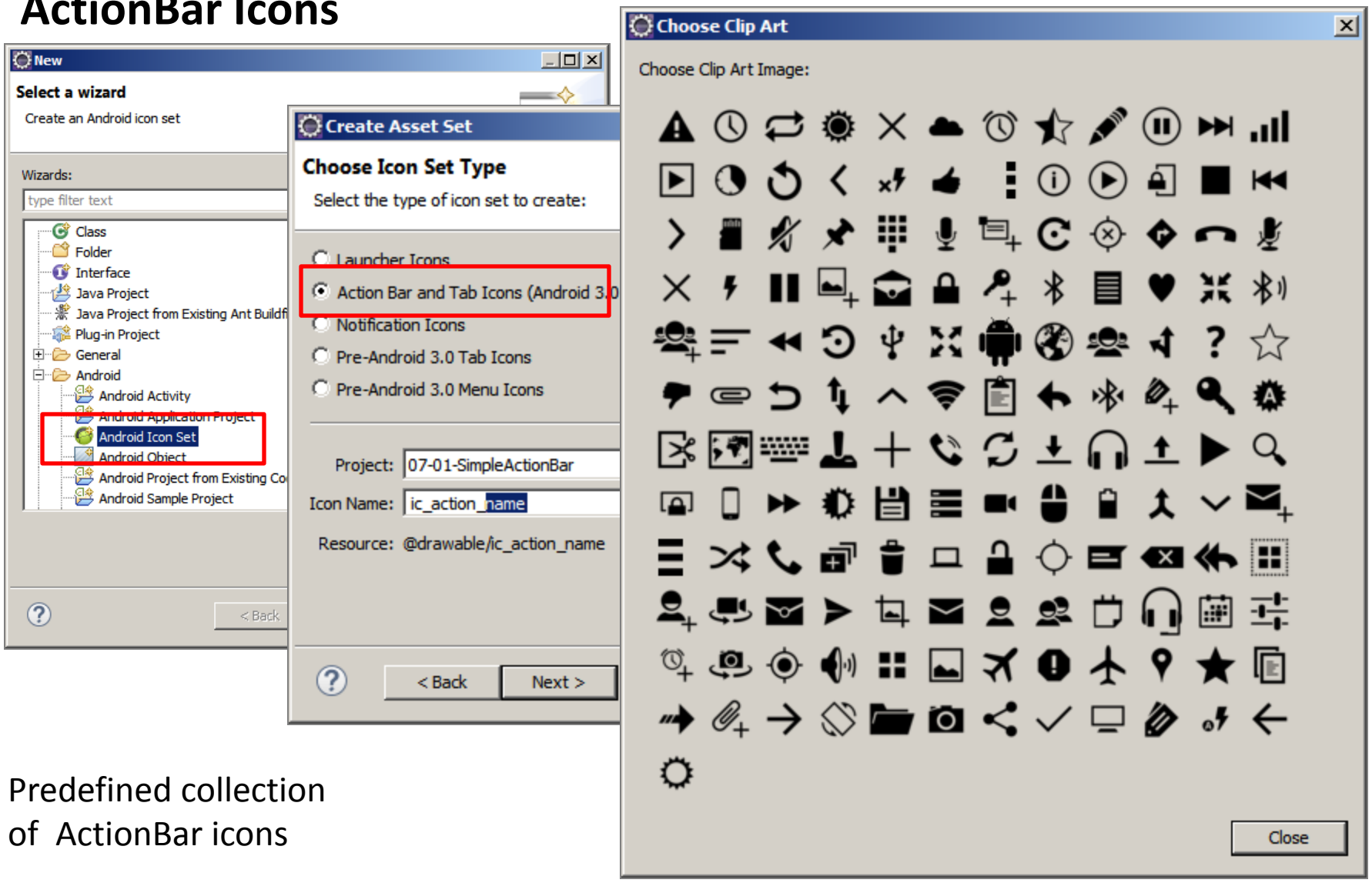
```
<item
        android:id="@+id/action_search"
        android:icon="@drawable/ic_action_search"
        android:orderInCategory="120"
        android:showAsAction="always|withText"
        android:title="Search"/>
```




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

# Example 1 – Creating a Simple ActionBar

## ActionBar Icons

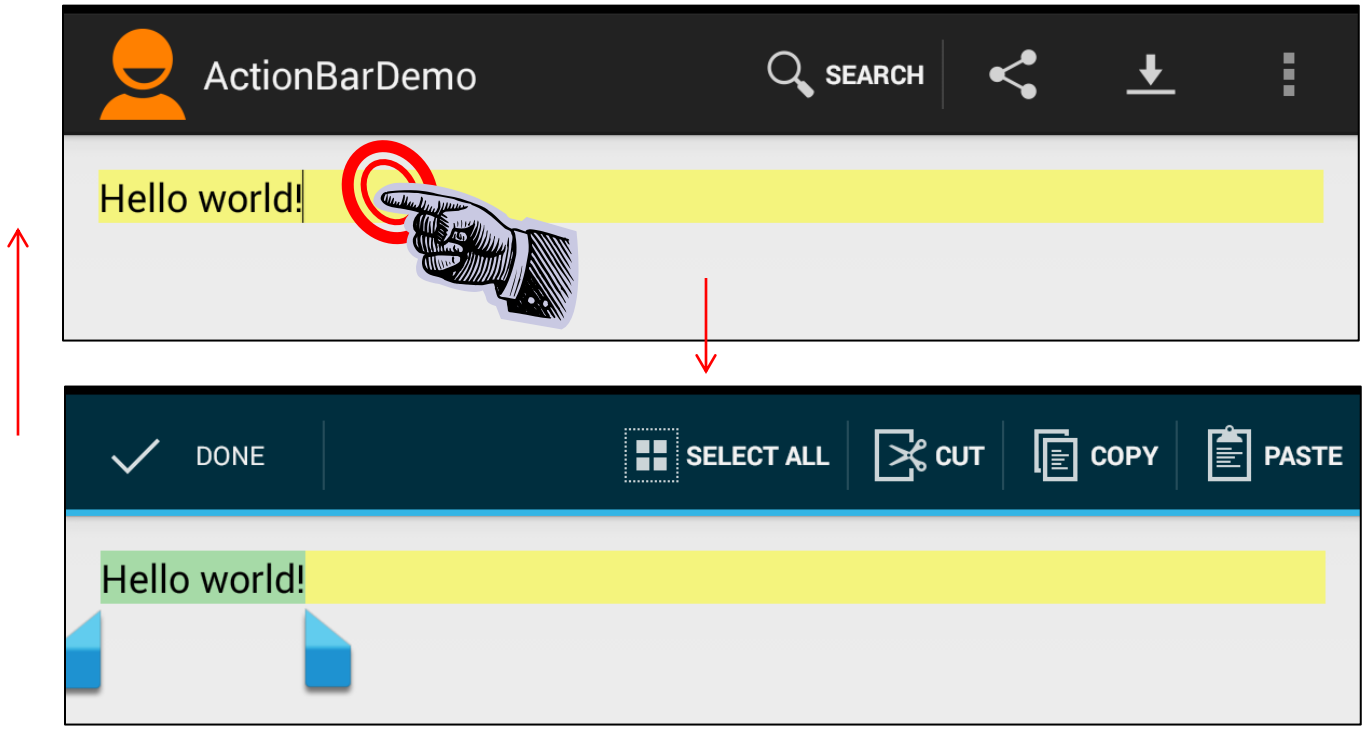



Predefined collection of ActionBar icons

# Example 1 – Creating a Simple ActionBar

**OBSERVATION:** Modify the app's **activity_main.xml** layout file. Change the "HelloWorld" **TextView** to an **EditText**. Run the application, apply a *long-tap* or *double-click* on the EditText field. The ActionBar changes to a default **Contextual ActionBar** (CAB) similar to the image shown below. The temporary CAB facilitates editing of the selected field. When finished just click "Done".

This example illustrates how CABs could be used to provide 'context-menu' capabilities to any chosen portion of the GUI (*we delay the discussion on how to create **custom CAB**s to a later point in the lesson*)




← **ActionBar**

← **Contextual ActionBar**

# 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);
1       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);
2       return true;
    }

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

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

# 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

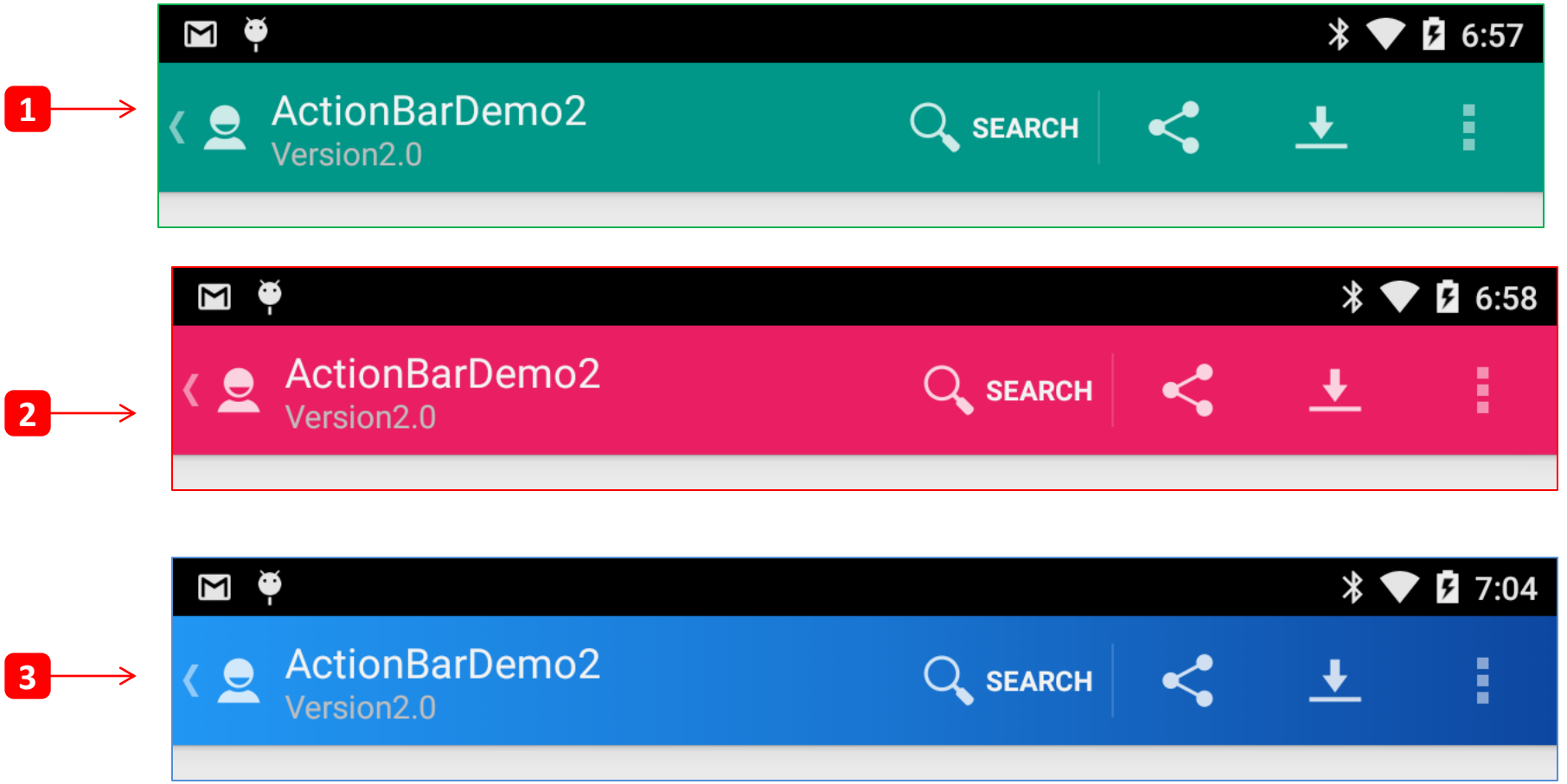
1. Plumbing operation. Establish access to the GUI's **EditText** field displaying the "Hello World" line.

2. The method `onCreateOptionsMenu()` is called to prepare the app's OptionMenu. 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
    actionBar = getActionBar();

    actionBar.setTitle("ActionBarDemo2");
    actionBar.setSubtitle("Version2.0");
    actionBar.setLogo(R.drawable.ic_action_logo);

    // choose one type of background
    actionBar.setBackgroundDrawable(getResources().getDrawable(R.drawable.mybackground0));
    actionBar.setBackgroundDrawable(getResources().getDrawable(R.drawable.mybackground1));
    actionBar.setBackgroundDrawable(getResources().getDrawable(R.drawable.mybackground2));

    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

 }
```

1 → `actionBar = getActionBar();`

2 → `actionBar.setSubtitle("Version2.0");`

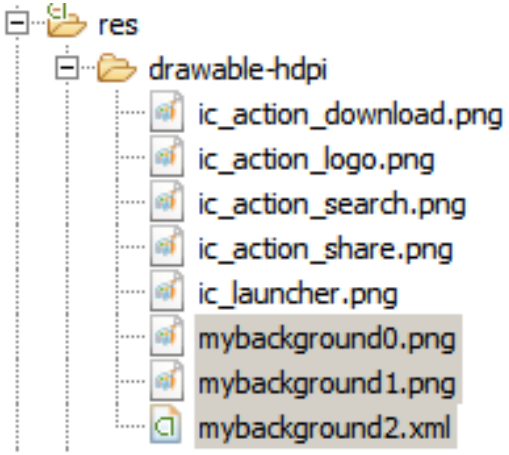
3 → `actionBar.setBackgroundDrawable(getResources().getDrawable(R.drawable.mybackground0));`

4 → `actionBar.setDisplayShowCustomEnabled(true);`

# 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**

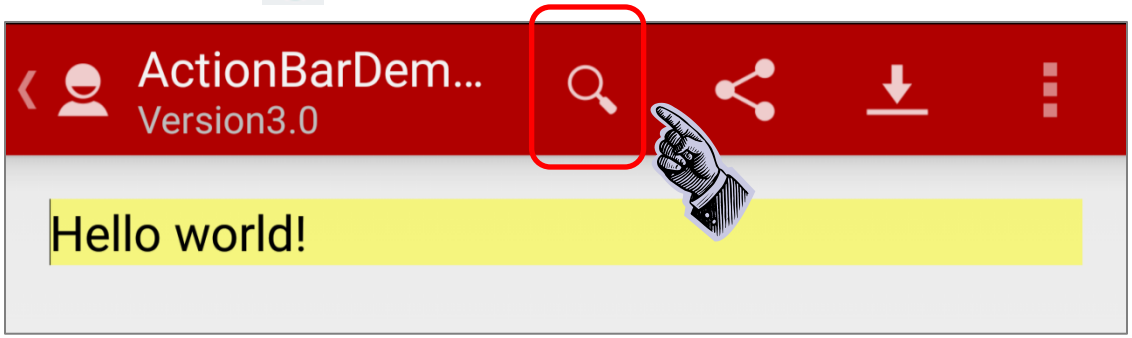
**Gradients** http://angrytools.com/gradient/

**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

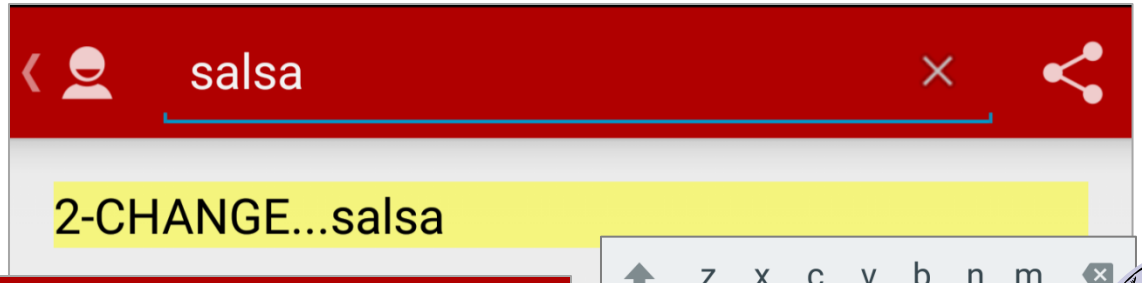
# Example 3 – Search ActionBar Tile

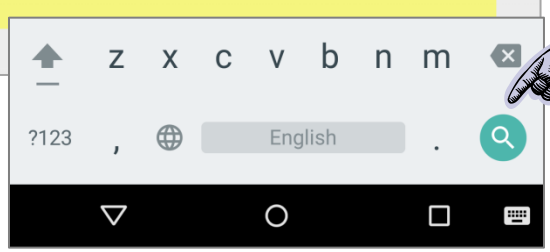
This is an extension of Example1. The ActionBar *search option* is implemented using a **SearchView** widget as a menu item. When the user finally taps on the keyboard's SEARCH key the text captured in the SeachView box is used to trigger an inquiry.

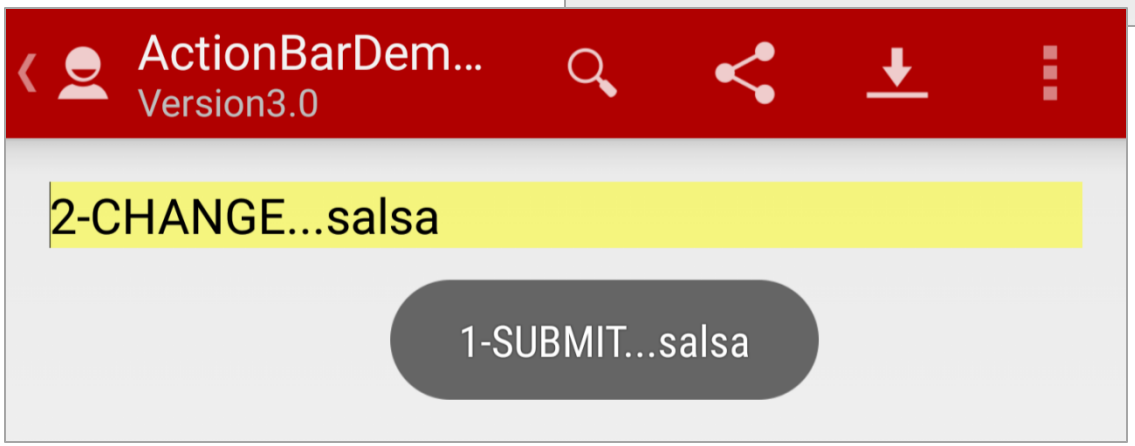



Press × to clear query string

A listener watches for each new character added to the query




Press to close SearchView and submit query

# Example 3 – Search ActionBar Tile



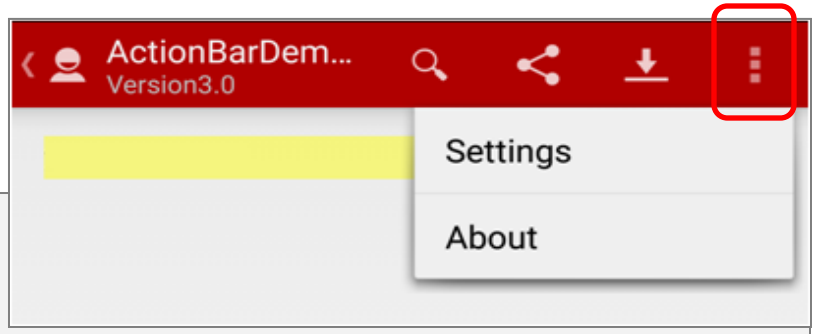
The **SearchView** box is defined as part of the ActionBar through an **XML MENU** file similar To the following code sample


```
<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"
1 →     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"/>
```

# Example 3 – Search ActionBar Tile



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.

# Example 3 – Search ActionBar Tile



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

```
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.setLogo(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

   }
```

# Example 3 – Search ActionBar Tile



```
@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
      public boolean onQueryTextSubmit(String query) {
2        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
      public boolean onQueryTextChange(String newText) {
3        // accept input one character at the time
         txtMsg.append("\n2-CHANGE..." + newText);
         return false;
      }
   });
   return true;
}
```

# Example 3 – Search ActionBar Tile



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
```

# Example 3 – Search ActionBar Tile

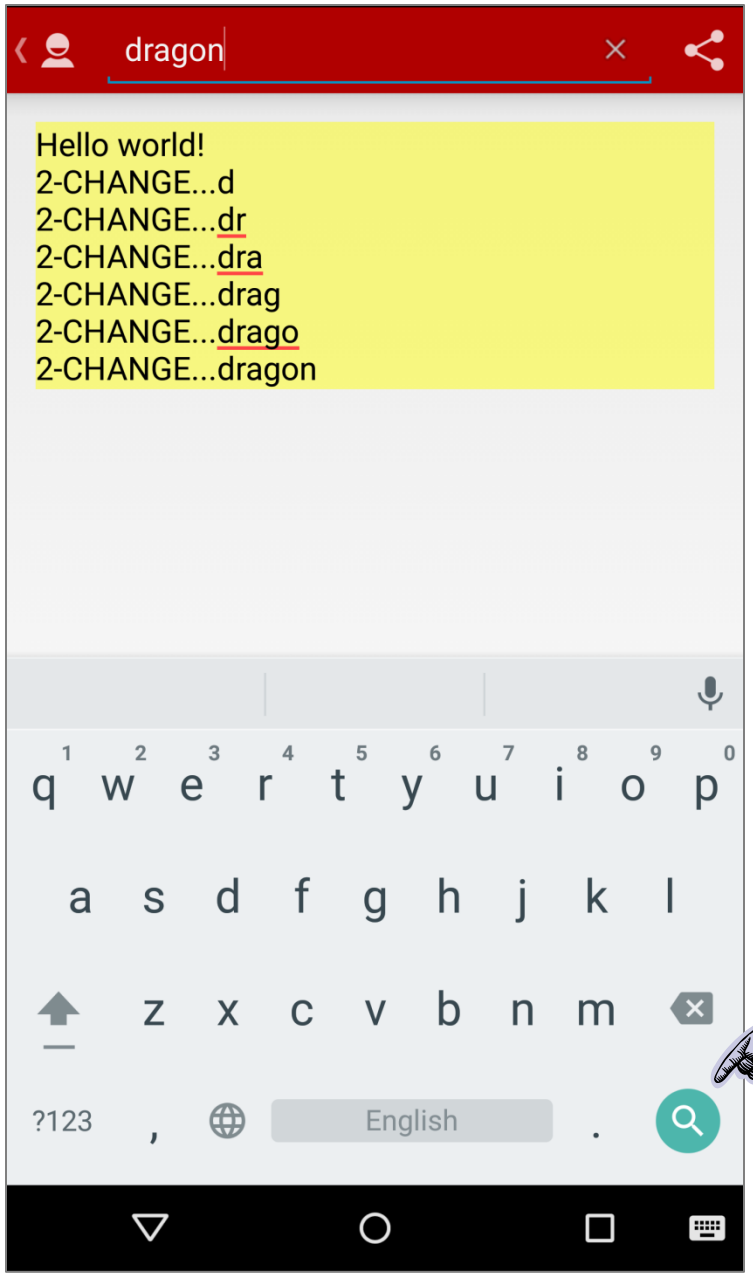
## Comments

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).

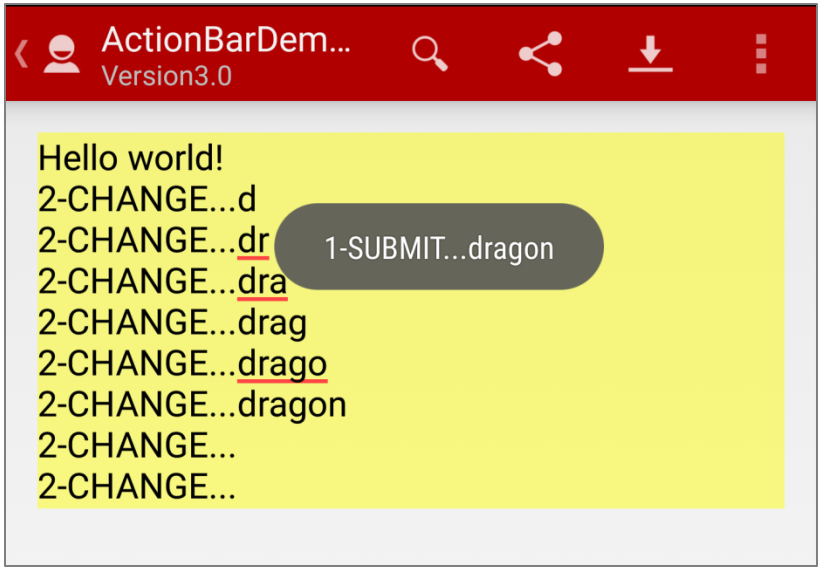
3. The second listening method **onQueryTextChange** is a text-watcher called after a new character is added to the SearchView. You may use this method to show suggestions progressively refined as more symbols are added to the query string.

# Example 3 – Search ActionBar Tile

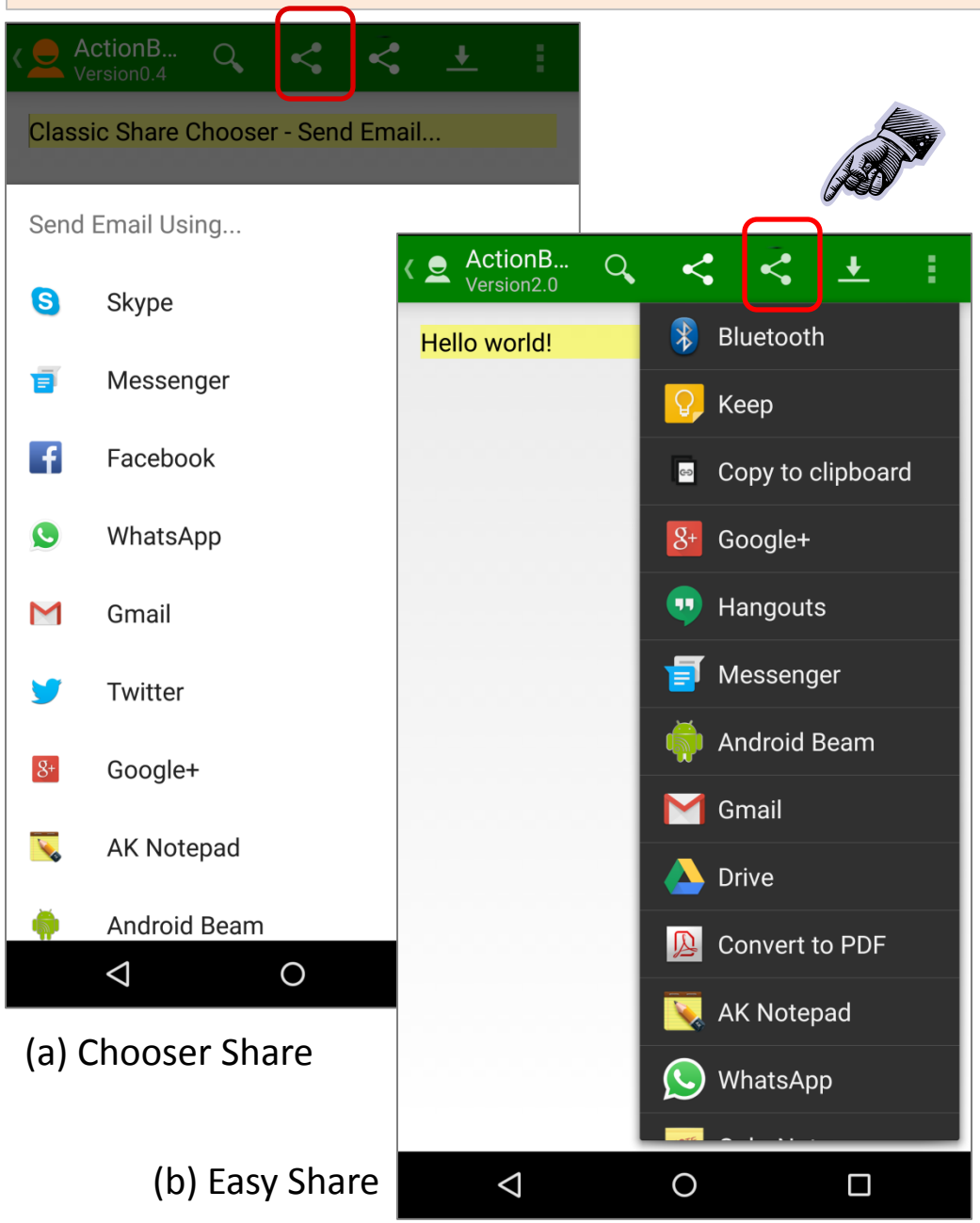



A sample screen illustrating the execution of Example3. Notice how the listener reacts to the typing of each new character into the query area of the SearchBox.

The toast-message appears after submitting the query string.



# Example 4 – Share ActionBar Tile




(a) Chooser Share

(b) Easy Share

In this example we enhance the app's functionality by adding **share capabilities** to the ActionBar.

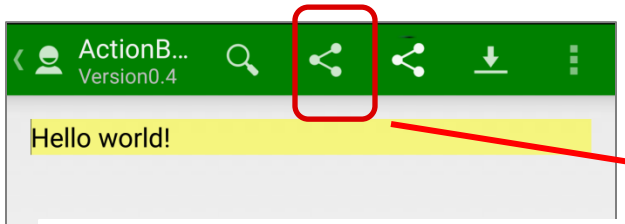
You choose what data-item(s) and channel of communication should be used for the transfer.

The example shows two styles:

(a) a classic **chooser dialog box** shows a list of providers from which the user selects a carrier,

(b) a contemporary '**Easy-Share**' action tile based on the *SharedActionProvider* widget introduced in API-14.

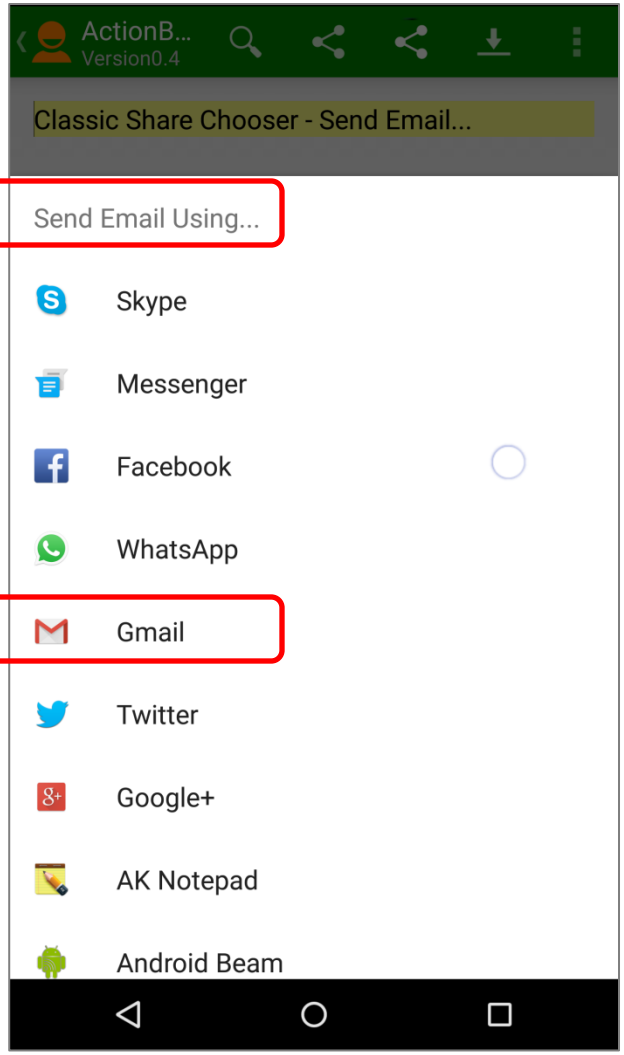
# Example 4 – Share ActionBar Tile

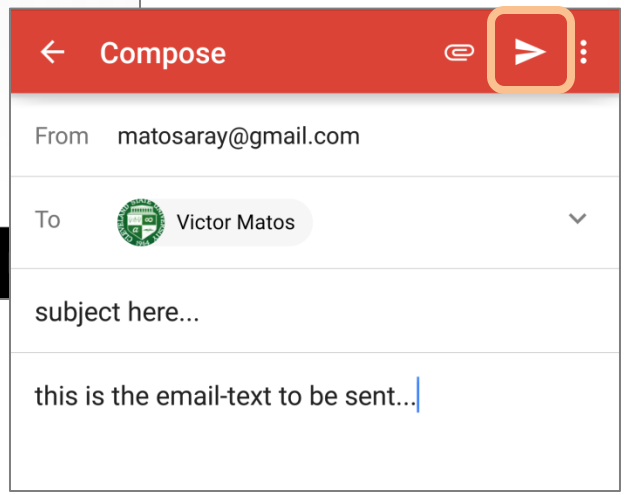



The statement

```
startActivity(Intent.createChooser(
                emailIntent(),
                "Send EMAIL Using..."));
```

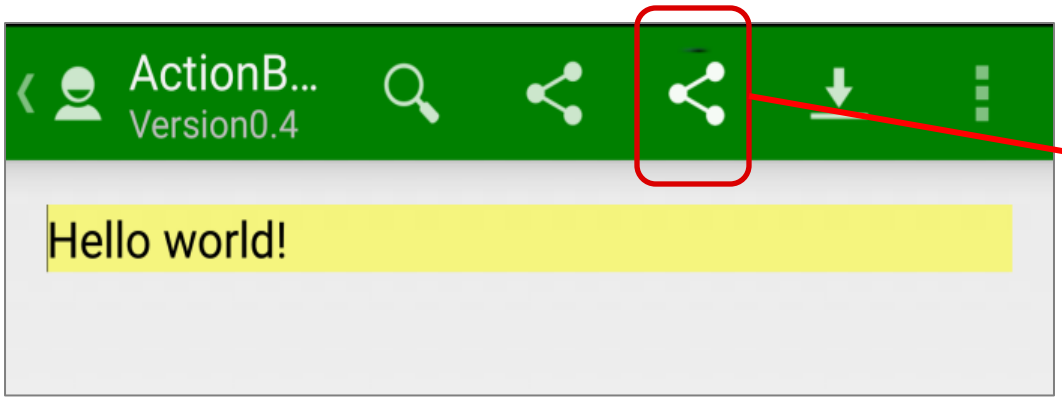
- displays a list of all possible apps registered to serve the requested intent (ACTION_SENDTO)
- The user chooses an entry from the list to complete the email intent.

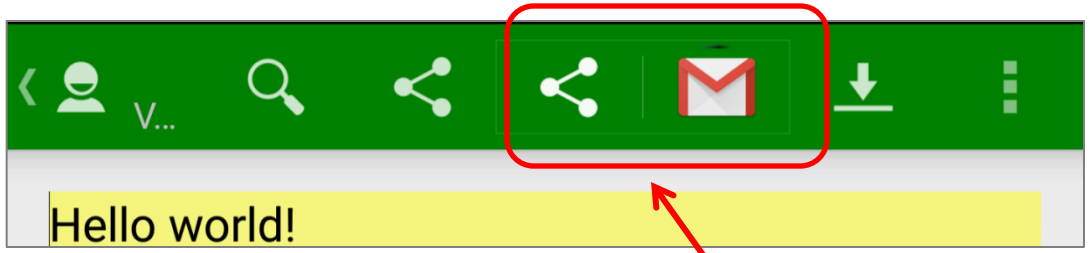


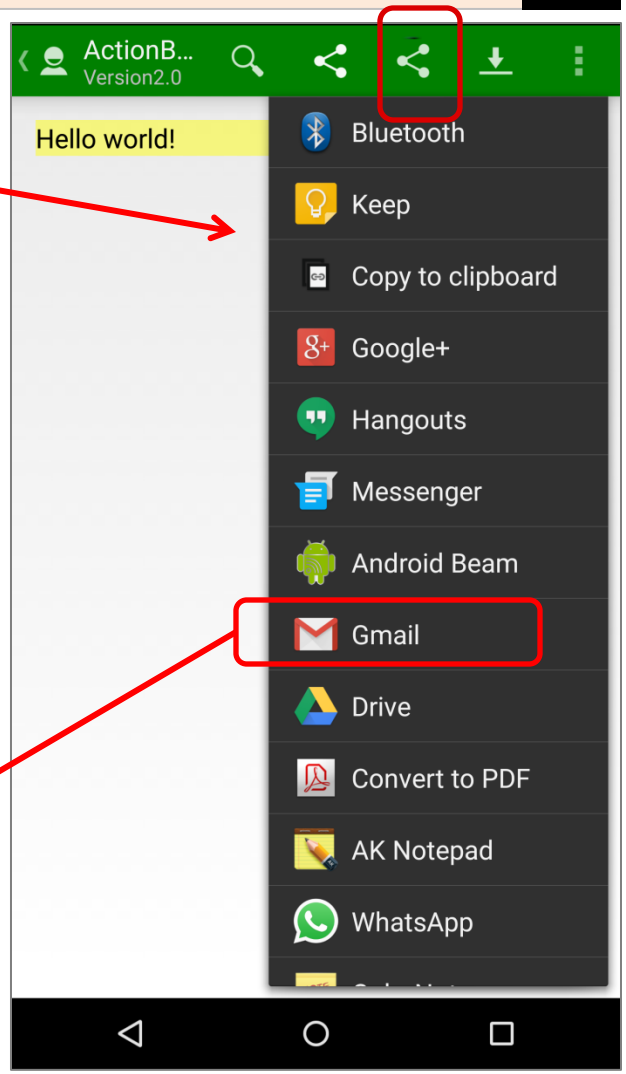



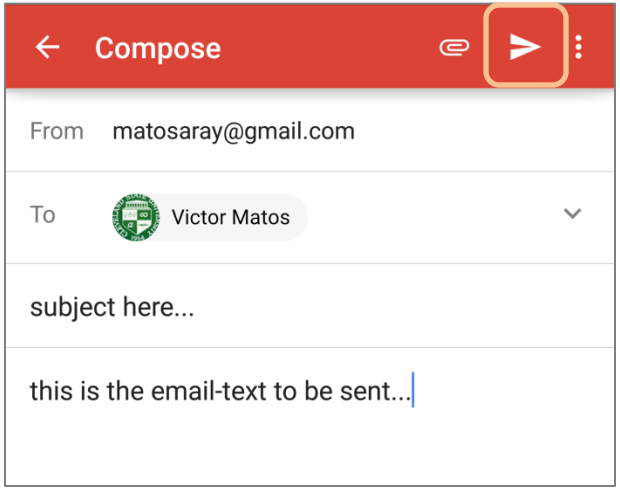

**(a) Chooser of ShareActionProvider**

# Example 4 – Share ActionBar Tile



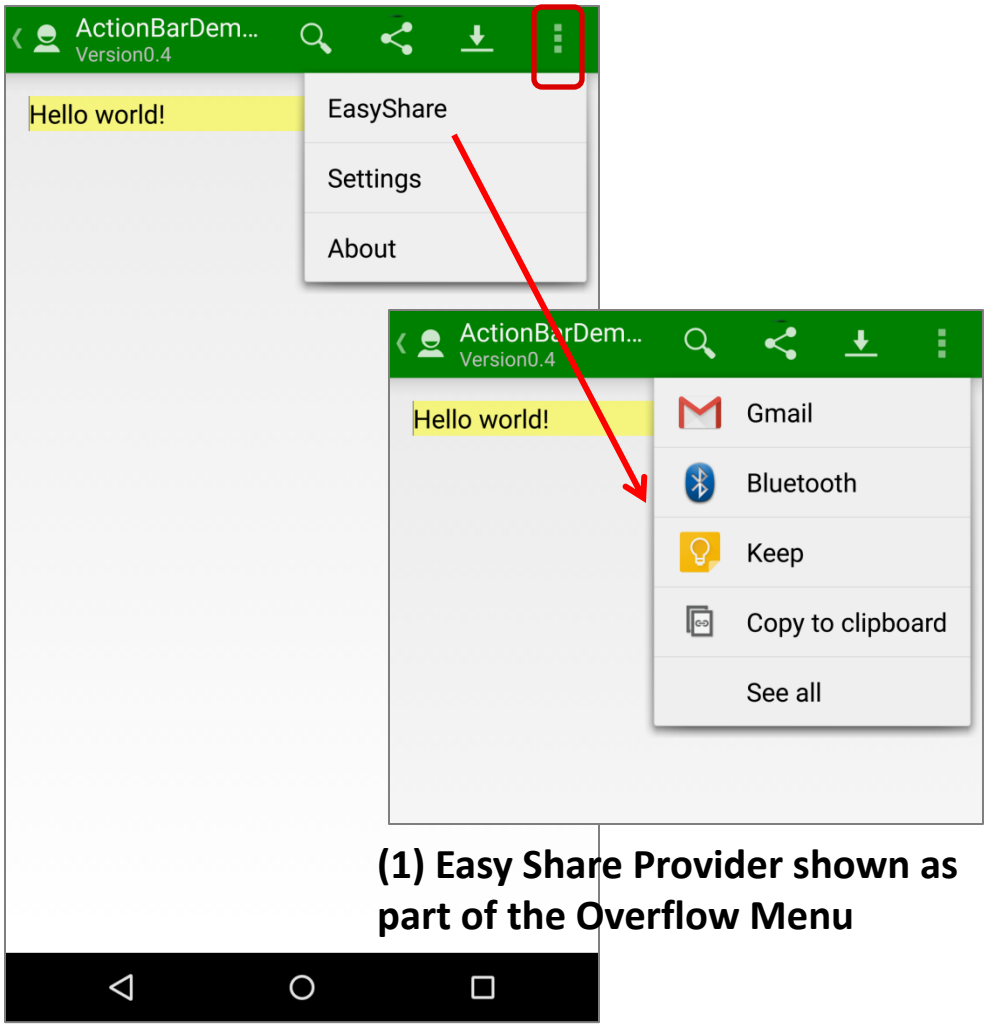

After an **EasyShare** provider is picked the ActionBar changes.

The selected option is added as a *default* for the next time the same type of intent is used.

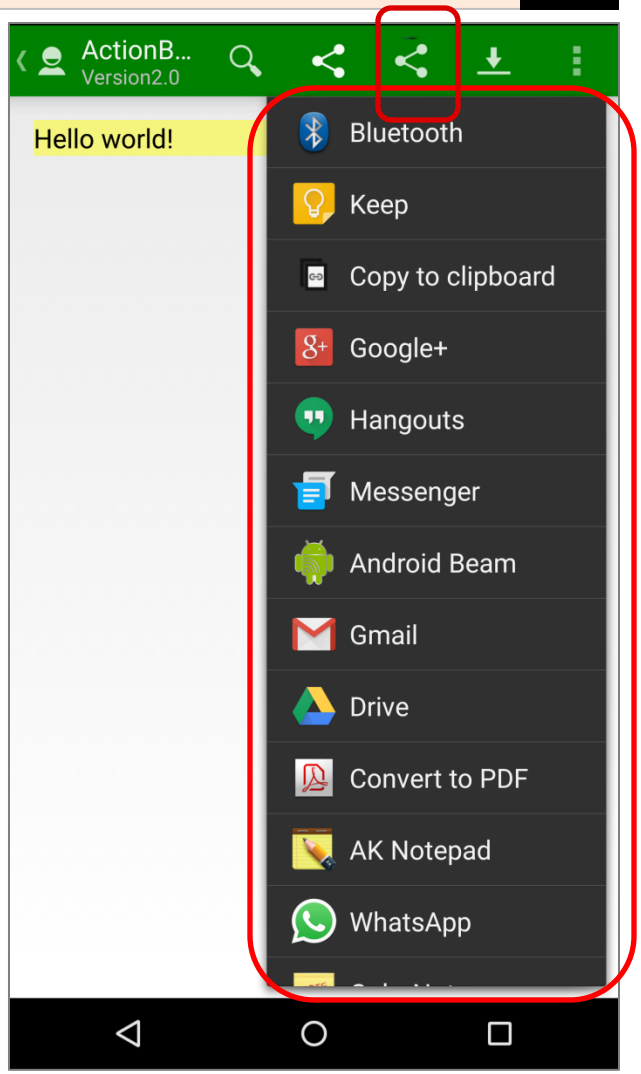
**(b) Easy Share Provider**

# Example 4 – Share ActionBar Tile

The **EasyShare** collapsible view can be displayed on the ActionBar as an Action-Tile or Overflow Menu entry.




**(1) Easy Share Provider shown as part of the Overflow Menu**




**(2) Easy Share Provider shown as part of the ActionBar**

# Example 4 – Share ActionBar Tile



The resource **MENU** file accommodating the **ShareActionProvider** is defined as follows

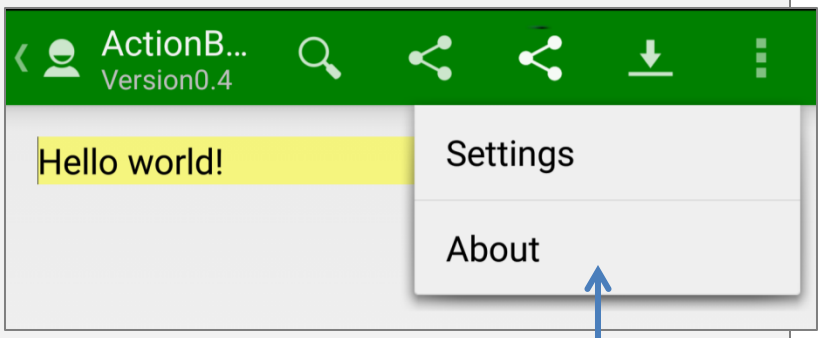
```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context="csu.matos.MainActivity" >

    <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_chooser"
        android:icon="@drawable/ic_action_share"
        android:orderInCategory="140"
        android:showAsAction="always"
        android:title="ShareChooser" />

     <item
        android:id="@+id/action_share_easy"
        android:orderInCategory="145"
        android:showAsAction="always"
        android:title="EasyShare"
        android:actionProviderClass="android.widget.ShareActionProvider" />
```

# Example 4 – Share ActionBar Tile



The resource **MENU** file accommodating the **ShareActionProvider** is defined as follows

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

    <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 4 – Share ActionBar Tile



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

```
public class MainActivity extends Activity {

    EditText txtMsg;
    ActionBar actionBar;
    ShareActionProvider shareActionProvider;

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

    txtMsg = (EditText)findViewById(R.id.txtMsg);

    // setup ActionBar
    actionBar = getActionBar();

    actionBar.setTitle("ActionBarDemo4");
    actionBar.setSubtitle("Version0.4");
    actionBar.setLogo(R.drawable.ic_action_logo);
    actionBar.setBackgroundDrawable(getResources()
              .getDrawable(R.drawable.mybackground0));

    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 4 – Share ActionBar Tile



```
@Override
public boolean onCreateOptionsMenu(Menu menu) {

    // Inflate the menu, add ShareChooser & EasyShare tiles
    getMenuInflater().inflate(R.menu.main, menu);

    // Locate MenuItem holding ShareActionProvider
1    MenuItem easySharedItem = menu.findItem(R.id.action_share_easy);

    // prepare EASY SHARE action tile:
    // Fetch and store a ShareActionProvider for future usage
    // an intent assembles the email(or SMS), you need only to select carrier
    shareActionProvider = (ShareActionProvider) easySharedItem.getActionProvider();

    // prepare an EMAIL
    shareActionProvider.setShareIntent( emailIntent() );

    // prepare an SMS - try this later...
    // shareActionProvider.setShareIntent( smsIntent() );

    return super.onCreateOptionsMenu(menu);
}
```

# Example 4 – Share ActionBar Tile



```
// return a SHARED intent to deliver an email
private Intent emailIntent() {

    Intent intent = new Intent(Intent.ACTION_SEND);
    intent.setType("text/plain");
    intent.putExtra(Intent.EXTRA_EMAIL, new String[] { "v.matos@csuohio.edu" });
    intent.putExtra(Intent.EXTRA_SUBJECT, "subject here...");
    intent.putExtra(Intent.EXTRA_TEXT, "this is the email-text to be sent...");

    return intent;

}


// return a SHARED intent to deliver an SMS text-message
private Intent smsIntent() {

    Intent intent = new Intent(Intent.ACTION_VIEW);
    String yourNumber="+ 1 216 555-4321";
    intent.setData( Uri.parse("sms:" + yourNumber) );
    intent.putExtra("sms_body", "Here goes my msg");

    return intent;

}
```

# Example 4 – Share ActionBar Tile



```
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. Observe EasyShare is NOT handled here!
        int id = item.getItemId();
        if (id == R.id.action_search) {
            txtMsg.setText("Search...");
            return true;
        }
        else if (id == R.id.action_share_chooser) {
            txtMsg.setText("Classic Share Chooser - Send Email...");

            startActivity( Intent.createChooser( emailIntent(), "Send EMAIL Using...") );
            //startActivity(Intent.createChooser(smsIntent(), "Send SMS Using..."));
            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;
    }
}//Activity
```

4

# Example 4 – Share ActionBar Tile

## Comments

1. The method **onCreateOptionsMenu** is used inflate the resource menu file. The item *@id+action_share_easy* hosting the *ShareAccessProvider* widget is bound to the object *easyShareProvider* . This will be used to control the **EasyShare** version of this example. Later we bind this object to the actual Intent that will be used to deliver the selected data items (ACTION_VIEW, and ACTION_SENDTO).

2. The method **`emailIntent()`** returns an intent in which a simple email message has been assembled. The message contains type, recipients, subject and body.

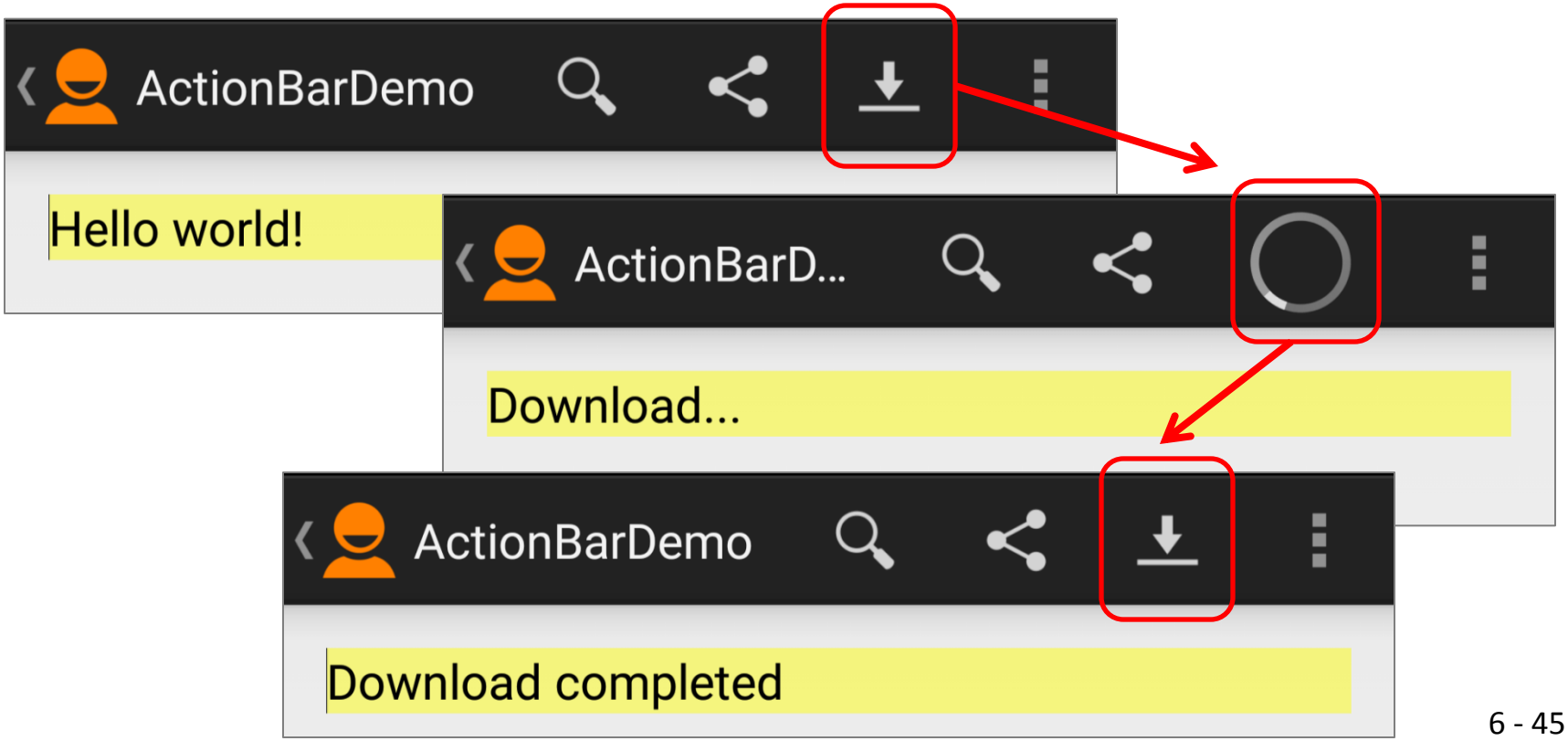
3. The method **`smsIntent()`** returns an intent in which a simple SMS message has been assembled. The message contains type, recipient, and body.

4. The method **`onOptionsItemSelected()`** is used to recognize the user's choice for the first style of sharing ( 'Share Chooser'). This in turn, invokes *Intent.createChooser*(…) to display a list of all apps registered to handle the requested service. Observe that EasyShare is NOT processed in this method. As indicated in (1), the EasyShare option is set in the onCreateOptionsMenu method.

# Example 5 – Download ActionBar Tile

In this example the ActionBar shows a **DownLoad** icon. When tapped, it invokes some arbitrary ***very slow user-define operation*** (such as a database query, or an Internet data transfer). Remember, slow tasks cannot be run by the main thread.

A parallel task is given the slow operation. The user is kept informed of the progress made by the background job by replacing the download icon with a custom circular progress bar. When the back worker finishes, it sends a message to the main thread, and the original Download icon is restored (Multitasking is discussed on Lesson11).

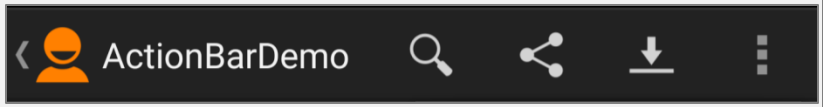


# Example 5 – Download ActionBar Tile



The resource **MENU** file accommodating the **Download** action is defined as follows

```
<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"/>
```

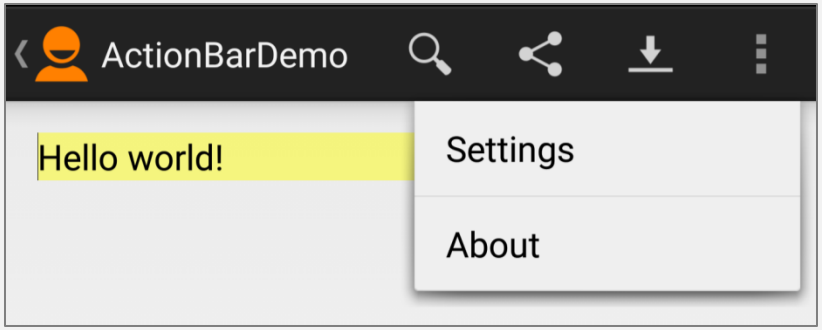


# Example 5 – Download ActionBar Tile



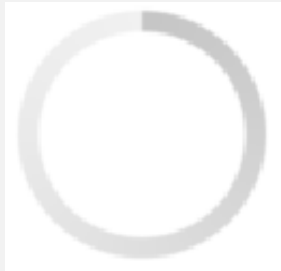
The resource **MENU** file accommodating the **Download** action is defined as follows

```
<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>
```




The file **res/layout/custom_view_download** defining the custom view showing a circular progress bar follows

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

<ProgressBar
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/customViewActionProgressBar"
    style="?android:attr/progressBarStyleLarge"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```



# Example 5 – Download ActionBar Tile



## MainActivity

```
public class MainActivity extends Activity {

   EditText txtMsg;
   ActionBar actionBar;

   @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.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
   }

   @Override
   public boolean onCreateOptionsMenu(Menu menu) {
      // Inflate the resource menu file: main.xml
      getMenuInflater().inflate(R.menu.main, menu);
      return true;

   }
```

# Example 5 – Download ActionBar Tile



```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
   // Handle clicking of ActionBar items here.
   int id = item.getItemId();

   if (id == R.id.action_search) {
      txtMsg.setText("Search...");
      return true;
   } else if (id == R.id.action_share) {
      txtMsg.setText("Share...");
      return true;

   } else if (id == R.id.action_download) {
      txtMsg.setText("Download...");
      // Temporarily replace the download action-tile with circular progress bar
      performSlowOperation(item);

      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;
}
```

1

# Example 5 – Download ActionBar Tile



```
   private void performSlowOperation(MenuItem item) {
      // temporally replace Download action-icon with a progress-bar
      final MenuItem downloadActionItem = item;
2 →   downloadActionItem.setActionView(R.layout.custom_view_download);
      downloadActionItem.expandActionView();
      // define an Android-Handler control to receive messages from a
      // background thread were the slow work is to be done
3 →   final Handler handler = new Handler() {
         @Override
         public void handleMessage(Message msg) {
            super.handleMessage(msg);
            txtMsg.setText("Download completed");      // announce completion of slow job
            downloadActionItem.collapseActionView();   // dismiss progress-bar
            downloadActionItem.setActionView(null);
         }
      };
      // a parallel Thread runs the slow-job and signals its termination
4 →   new Thread() {
         @Override
         public void run() {
            // real 'BUSY_WORK' goes here...(we fake 2 seconds)
            long endTime = SystemClock.uptimeMillis() + 2000; //now + 2 seconds
            handler.sendMessageAtTime(handler.obtainMessage(), endTime);
            super.run();
         }
      }.start();
   }

}//Activity
```

# Example 5 – Download ActionBar Tile

## Comments

1. After the user clicks on the *Download* action-tile the **onOptionsItemSelected ()** method recognizes the event and calls the background worker task to perform its slow job.

2. The background task begins by replacing the ActionBar's Download item with a custom view showing a circular progress bar (.**setActionView(…)** and **expandActionView()** ).

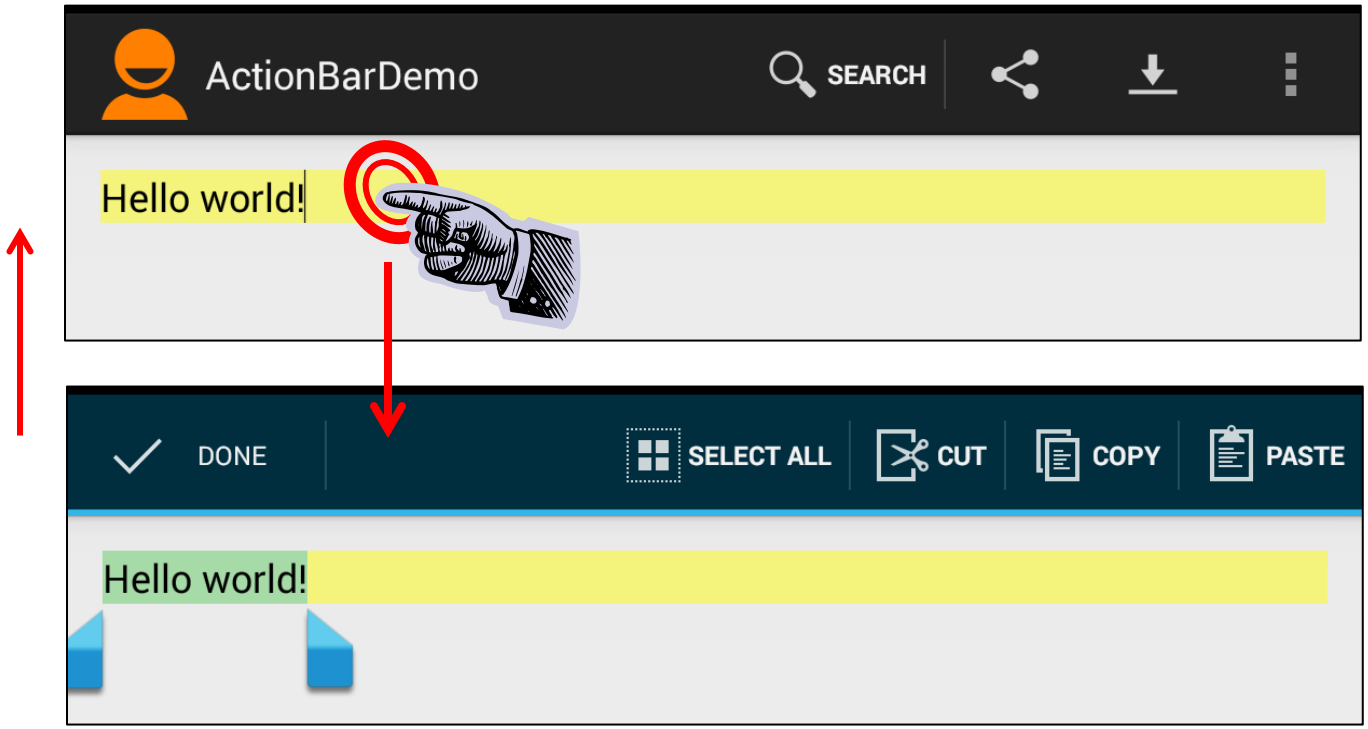
3. A Handler object is created in the main thread to receive asynchronous messages posted by the background thread. When finally the 'job done' message arrives, the handler removes the progress bar and restores the original app's ActionBar look ( .**collapseActionView()** and **.setActionView(null**) ).

4. A parallel **Java Thread** is created to wrap the slow job. In our case we simulate an isolated 5 seconds operation (a lot of time in Android's life). By doing this, the main thread continues to expose a highly responsive UI which can do other tasks for the user while the background job completes its assignment. When the slow thread finishes working, it posts a message into the handler's message queue signaling its completion. [*Multithreading is discussed in Lesson 11*]

# Example 6 – Contextual ActionBar (CAB)

An app's conventional ActionBar décor could be temporarily replaced by another ActionBar to provide menu-like support to any widget selected through a *long-tap* interaction. This temporary top décor is called a **Contextual ActionBar** (or **CAB**). To some degree, a CAB is a replacement of the legacy "Context Menu" strategy of early SDKs (see **Appendix XYZ**).

Example 1, provided a glimpse to default CABs, as illustrated below



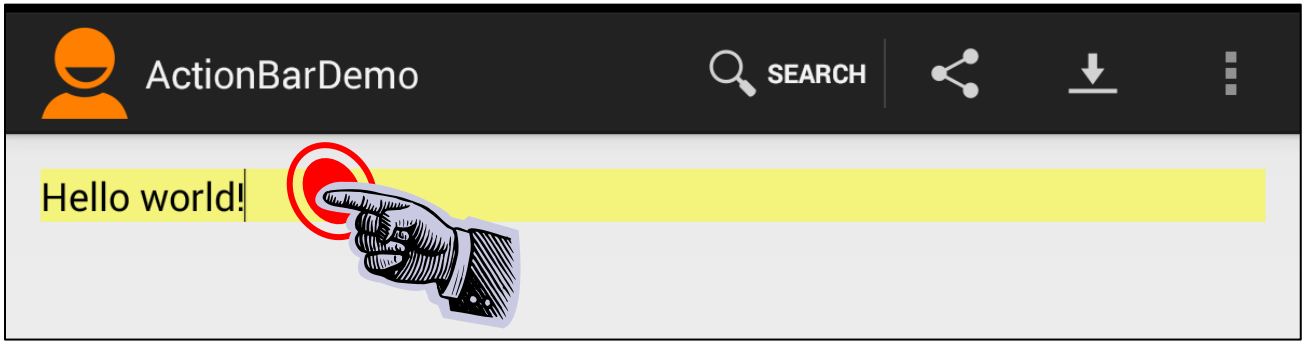

← **ActionBar**

← **Default Contextual ActionBar CAB**

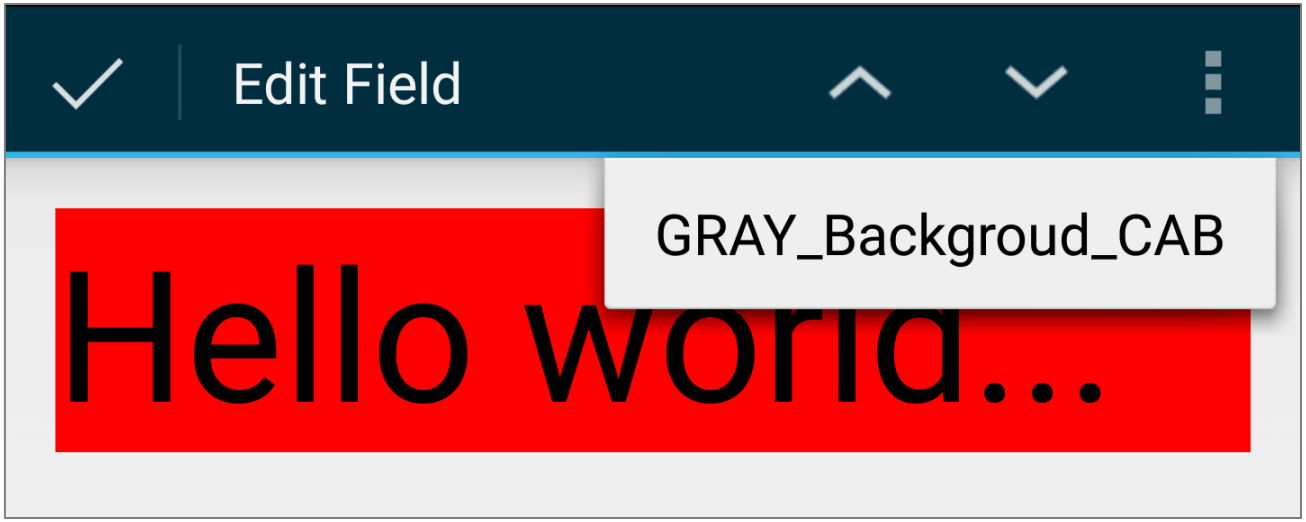
# Example 6 – Contextual ActionBar (CAB)

You may create custom CABs to enhance your app's functionality. To create a CAB, the developer must supplied a **temporary menu** to be inflated on the ActionBar. The supplier must also implement the **ActionMode.Callback** interface to indicate how to respond to each selection called from the CAB.

The CAB is disabled and removed when the user deselects all items, presses the BACK button, or selects the *Done* action on the left side of the bar.



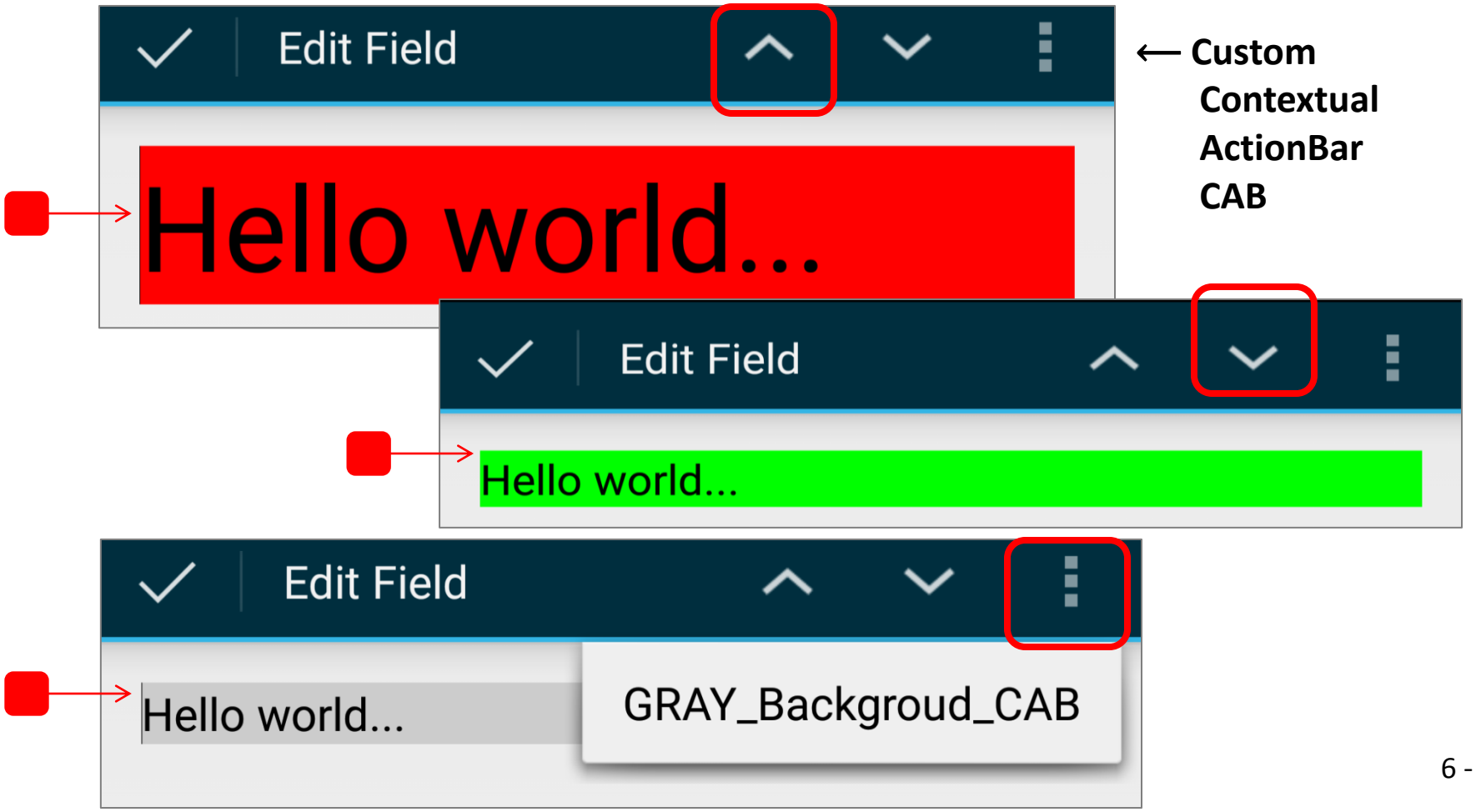

← **ActionBar**




← **Custom Contextual ActionBar CAB**

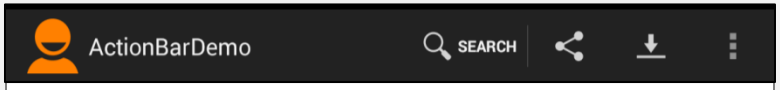
# Example 6 – Contextual ActionBar (CAB)

In our example a long-tap on the EditText view displaying the message “Hello world…” invokes a custom CAB. The CAB offers two action tiles and one entry in the overflow menu. The first action-tile ( Λ) increases the text-size and changes the background to red, the second action-tile (V) decreases the text-size and sets the background color to green, the final entry in the overflow menu changes the background to gray.



# Example 6 – Contextual ActionBar (CAB)

```
<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"/>
    <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>
```

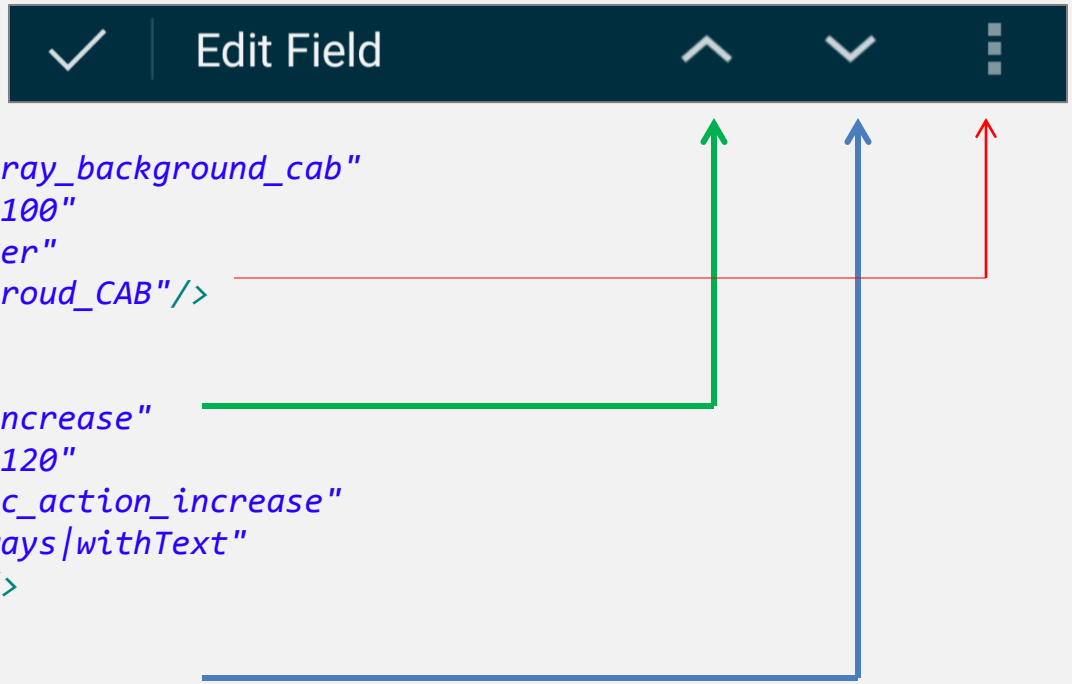



**res/menu/main.xlm**

First, let's show the app's global menu xml specification (same as Example1)

# Example 6 – Contextual ActionBar (CAB)

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context="com.example.x3.MainActivity" >

    <item
        android:id="@+id/action_gray_background_cab"
        android:orderInCategory="100"
        android:showAsAction="never"
        android:title="GRAY_Backgroud_CAB"/>

    <item
        android:id="@+id/action_increase"
        android:orderInCategory="120"
        android:icon="@drawable/ic_action_increase"
        android:showAsAction="always|withText"
        android:title="Increase"/>

     <item
        android:id="@+id/action_decrease"
        android:orderInCategory="160"
        android:icon="@drawable/ic_action_decrease"
        android:showAsAction="always|withText"
        android:title="Decrease"/>

</menu>
```




**res/menu/ txtmsg_cab_menu.xlm**

In addition to the apps common menu items a second xml file must be supplied to define the CAB.

# Example 6 – Contextual ActionBar (CAB)



## MainActivity

```
public class MainActivity extends Activity {

   EditText txtMsg;
   public ActionMode actionmode;
   private TxtMsgCallbacks txtMsgCallbacks;

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

      txtMsg = (EditText) findViewById(R.id.txtMsg);
      txtMsg.setOnLongClickListener(new OnLongClickListener() {

         @Override
         public boolean onLongClick(View v) {

            if (actionmode == null) {
               txtMsgCallbacks = new TxtMsgCallbacks(MainActivity.this, txtMsg);
            } else {
               Toast.makeText(getApplicationContext(), "REUSING",
                              Toast.LENGTH_LONG).show();
            }
            actionmode = startActionMode(txtMsgCallbacks);
            actionmode.setTitle("Edit Field");
            return true;
         }
      });

   } //onCreate
```

1 → `txtMsg = (EditText) findViewById(R.id.txtMsg);`

2 → `txtMsgCallbacks = new TxtMsgCallbacks(MainActivity.this, txtMsg);`

3 → `actionmode = startActionMode(txtMsgCallbacks);`

# Example 6 – Contextual ActionBar (CAB)



```
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        int id = item.getItemId();

        if (id == R.id.action_search) {
            txtMsg.setText("Search...");
            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
```

# Example 6 – Contextual ActionBar (CAB)

## Comments – MainActivity

1. A **LongClickListener** is set on top of the UI's EditText box. The listener is responsible for monitoring the custom CAB and allow custom editing.

2. When a *LongClick* event is detected, an **ActionMode** object is created to manage the CAB. Action modes are used to provide alternative interaction modes and temporarily replace parts of the normal UI. In our example the ActionMode overlays the global app's ActionBar. We supply a caption ("Edit Field"), a reference to the monitored EditText view, and a reference to the application's context.

3. A **TxtMsgCallback** object is created or reused, then it is bound to the ActionMode control to configure and handle events raised by the user's interaction with the action mode

# Example 6 – Contextual ActionBar (CAB)



## TxtMsgCallbacks

```
public class TxtMsgCallbacks  implements ActionMode.Callback {
   // This class handles the aActions shown by the custom CAB
   MainActivity mainContext;
   TextView txtMsg;

   public TxtMsgCallbacks(MainActivity mainContext, TextView txtMsg) {
      // this is the EditText view controlled by the CAB
      this.txtMsg = txtMsg;
      this.mainContext = mainContext;
   }

   @Override
   public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
      //set value of 10px (regardless of screen density)
      float tenPixels = TypedValue.applyDimension( TypedValue.COMPLEX_UNIT_DIP, 10,
                                    mainContext.getResources().getDisplayMetrics());
      //detect current text-size
      float oldSize = txtMsg.getTextSize();

      //increase by 10px text-size with red background
      if (item.getItemId() == R.id.action_increase) {
         txtMsg.setTextSize(TypedValue.COMPLEX_UNIT_PX, oldSize + tenPixels);
         txtMsg.setBackgroundColor(Color.RED);

      //decrease by 10px text-size with green background
      } else if (item.getItemId() == R.id.action_decrease) {
         txtMsg.setTextSize(TypedValue.COMPLEX_UNIT_PX, oldSize - tenPixels);
         txtMsg.setBackgroundColor(Color.GREEN);
```

1 → 2 → 3

# Example 6 – Contextual ActionBar (CAB)



## TxtMsgCallbacks

4

5

```
        //set background to gray
        } else if (item.getItemId() == R.id.action_gray_background_cab) {
            txtMsg.setBackgroundColor(Color.LTGRAY);
        }

        return false;
    }

    // showing other states from the  ActionMode life-cycle
    @Override
    public boolean onCreateActionMode(ActionMode mode, Menu menu) {
        mode.getMenuInflater().inflate(R.menu.txtmsg_cab_menu, menu);
        return true;
    }

    @Override
    public void onDestroyActionMode(ActionMode mode) {
        Toast.makeText(mainContext, "Destroy CAB", Toast.LENGTH_LONG).show();
    }

    @Override
    public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
        Toast.makeText(mainContext, "Prepare CAB", Toast.LENGTH_LONG).show();
        return false;
    }
}
```

# Example 6 – Contextual ActionBar (CAB)

## Comments – TxtMsgCallbacks

The **ActionMode** class is responsible for showing the custom CAB. The **TxtMsgCallbacks** class inflates the custom CAB menu (**txtmsg_cab_menu.xml**) that covers the app's global ActionBar and monitors the tapping of action-tiles shown on the custom CAB.

1. The method **`onActionItemClicked()`** is called when the user clicks on a CAB's action tile. Its first step is to calculate what 10 pixels measure on the current device (regardless of the actual screen density).

2. The user's first action-choice is to increase the UI's EditText view (txtMsg) by 10 px, in addition the view's background color is set to red.

3. The user's second action-choice is to decrease the UI's EditText view (txtMsg) by 10 px, the view's background color is change to green.

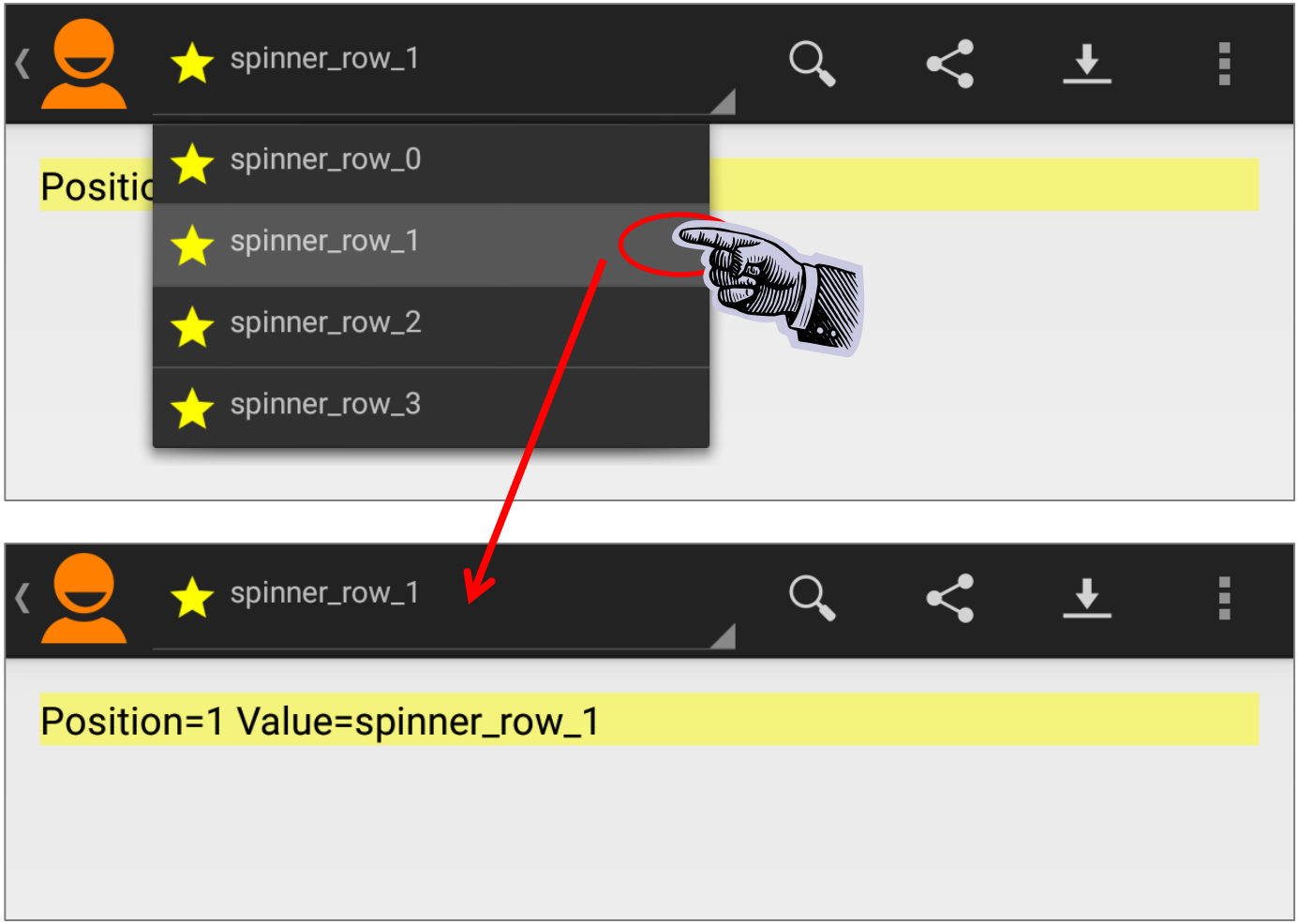
4. The user's third action-choice (part of the overflow menu) is to change the txtMsg background color to gray.

5. A Toast-Message is displayed when visiting the other methods in the ActionMode's life-cycle sequence.

# Example 7 – 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).

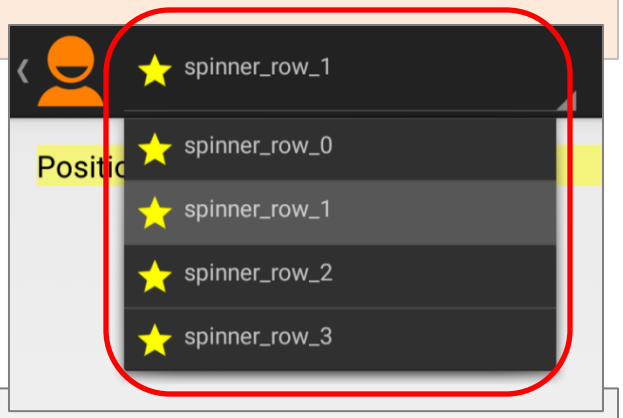
For example, The **Weather Channel** app presents in its ActionBar a **spinner** widget to set location.



# Example 7 – 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**

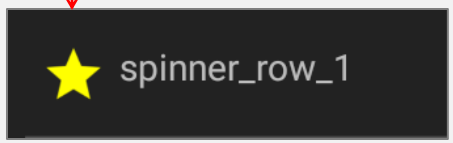



```
<Spinner
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/spinner_data_row"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

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 7 – 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.setOnItemSelectedListener(this);
     customSpinner.setSelection(selectedSpinnerRow);

}//onResume
```

# Example 7 – CustomView - Spinner



**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 7 – CustomView - Spinner



**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 7 – CustomView - Spinner



**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 7 – CustomView - Spinner

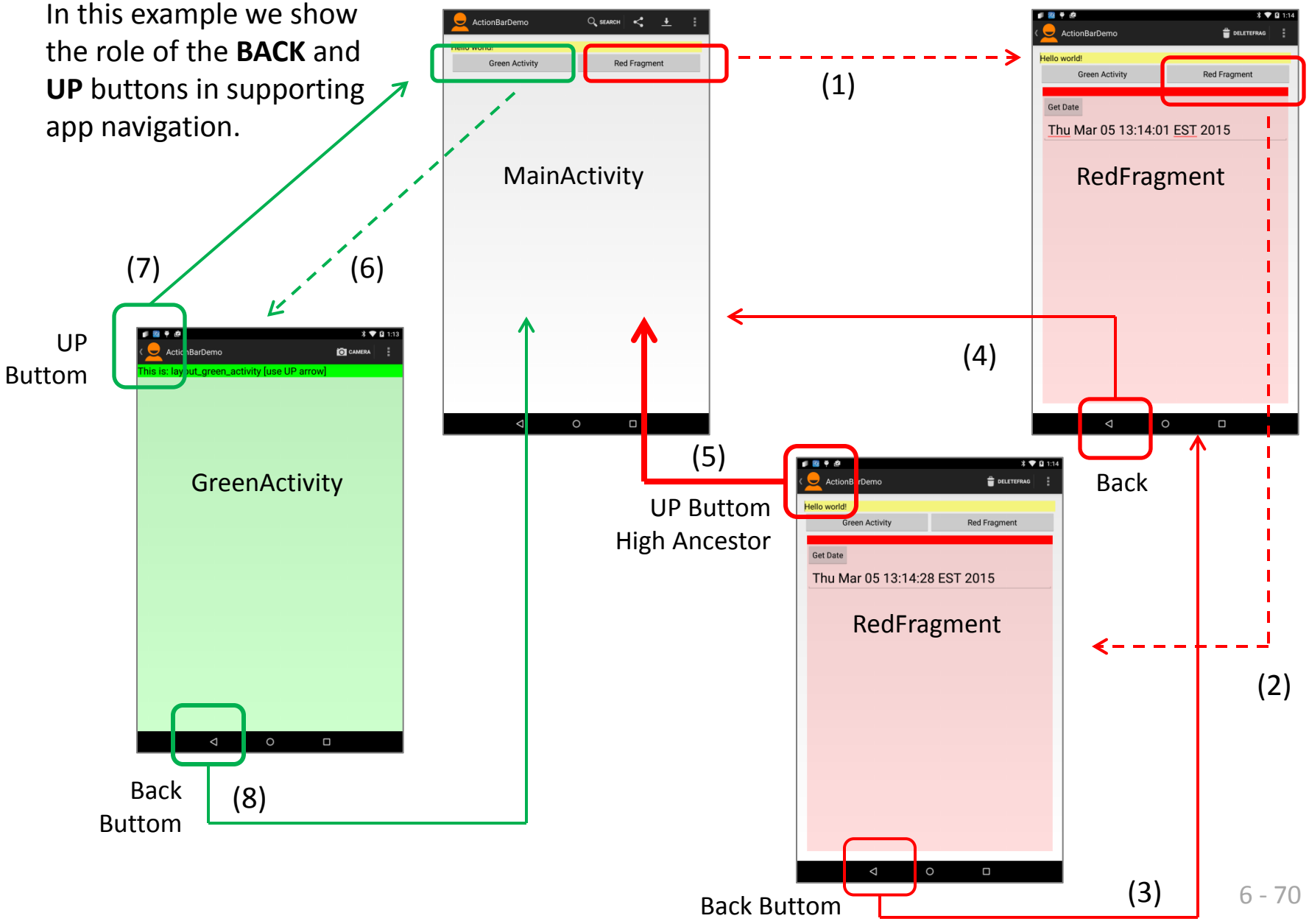


**Dummy Data (SpinnerDummyContent.java).** Use the following code fragment to generate spinner's data.

```
   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 8 – UP-KEY and BACK-KEY Navigation

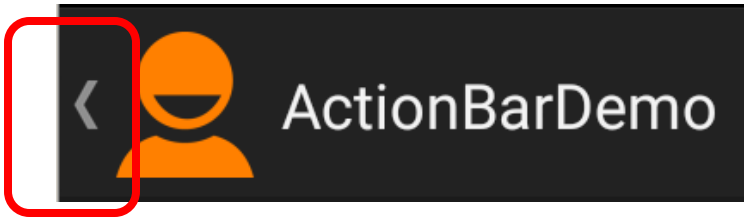
In this example we show the role of the **BACK** and **UP** buttons in supporting app navigation.



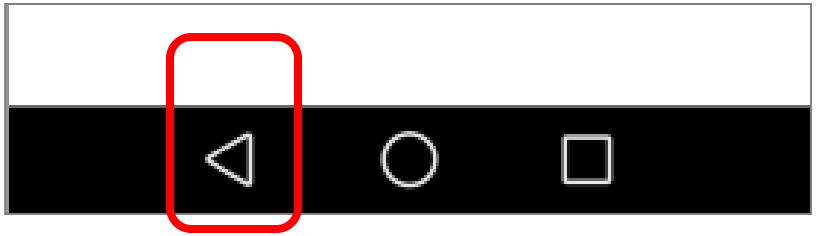
# Example 8 – UP-KEY and BACK-KEY Navigation

In our example, the **MainActivity**'s UI can be temporarily replaced by a secondary Activity (**GreenActivity**) or could be partially overlapped with views produced by **RedFragments**.

Navigation between MainActivity, GreenActivity and redFragments is accomplished using the **BACK-KEY** and **UP-KEY.**




**UP-KEY** (High-Ancestor Navigation)



**BACK-KEY** (Historical Navigation)

The **UP-KEY** is found on the left-upper corner of the ActionBar as an extension of the HOME button. The **BACK-KEY** is at the bottom of the device.

We use the **UP-KEY** to redirect the app to an arbitrary *High-Ancestor* in the calling hierarchy. Similarly, we use the **BACK-KEY** to provide *historical navigation* .

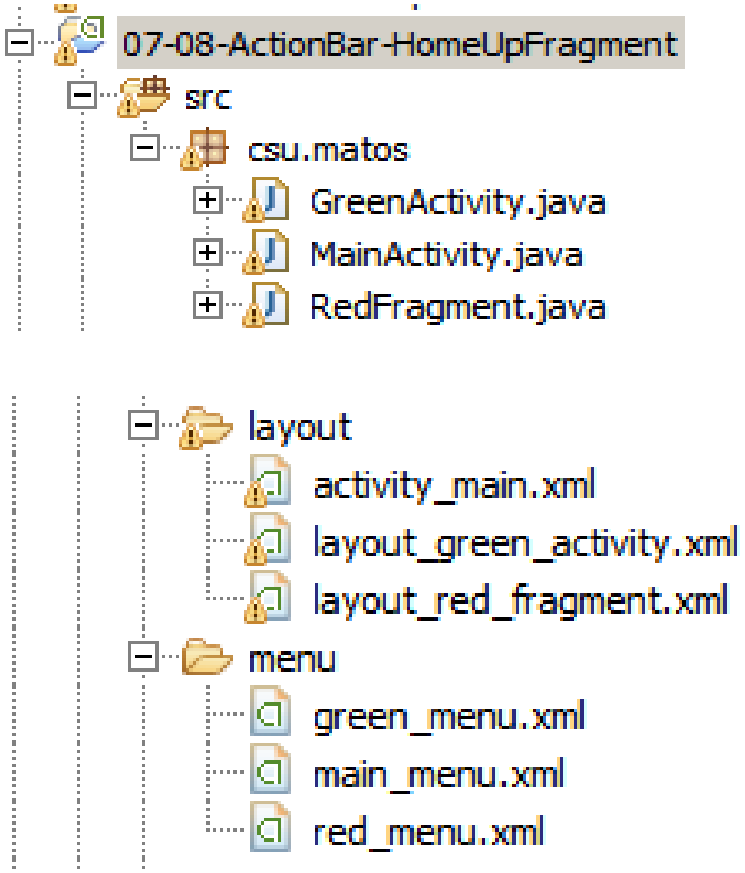
# Example 8 – UP-KEY and BACK-KEY Navigation

**STEPS SHOWN IN THE EXAMPLE**

1. After tapping the button labeled ‘Red Fragment’ the MainActivity shows a view produced by a RedFragment (now, you should click on the ‘Get Date’ button to change the state data shown by this instance of RedFragment).
2. A second redFragment is requested and placed on top of the first.
3. Pressing BACK-KEY removes the current view (second redFragment) returning the app to the UI shown by the first redFragment.
4. Pressing BACK-KEY on first redFragment takes us to previous UI (MainActivity).
5. Pressing UP-KEY makes the app jump to a parent UI. Observe it does NOT need to be the immediate parent but any on the view-hierarchy (MainActivity in our example).
6. An Intent is used to call GreeActivity which becomes active and visible.
7. The AndroidManifest has an entry indicating the MainActivity is the “ParentActivity” for the GreenActivity. Pressing UP-KEY navigates to the designaated ‘parentActivity’.
8. Tapping the BACK-KEY provides a way for the GreenActivity to return to its immediate ancestor (MainActivity).

# Example 8 – UP-KEY and BACK-KEY Navigation

The app's **MainActivity** will either (1) invoke the supporting **GreenActivity** using an Intent, or (2) show on its host screen overlapping instances of **RedFragments**



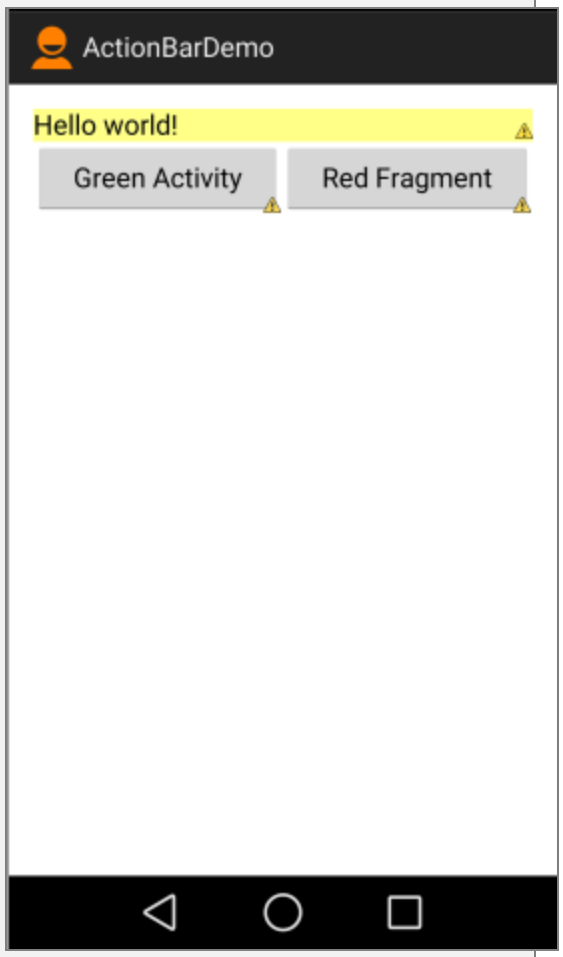

Layouts for the MainActivity, GreenActivity and RedFragment

Each of the above components will present its own customized menu on the ActionBar décor.

# Example 8 – LAYOUT: activity_main.xml



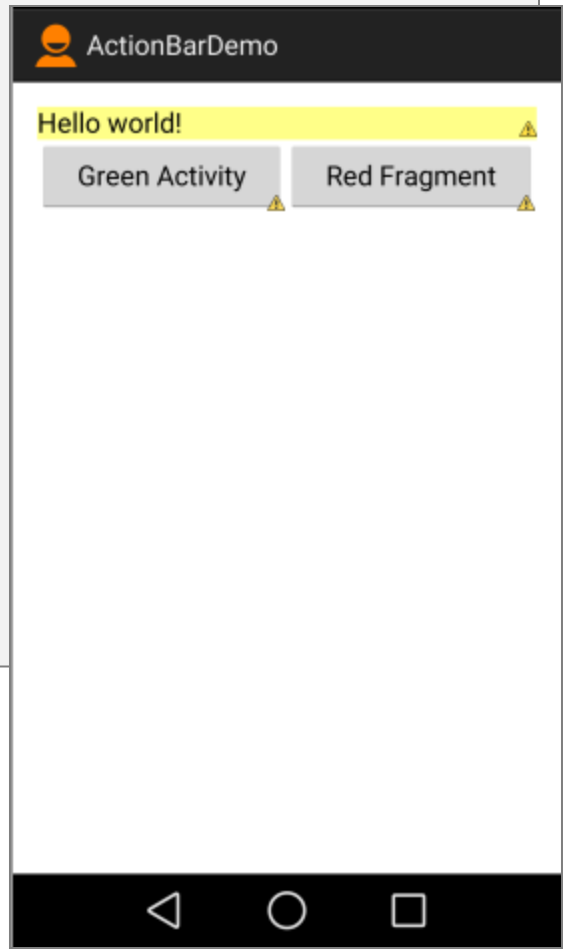
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="csu.matos.MainActivity" >

    <EditText
        android:id="@+id/txtMsg"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#77ffff00"
        android:text="@string/hello_world" />

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

        <Button
            android:id="@+id/btnGreenActivity"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Green Activity" />
```



# Example 8 – LAYOUT: activity_main.xml

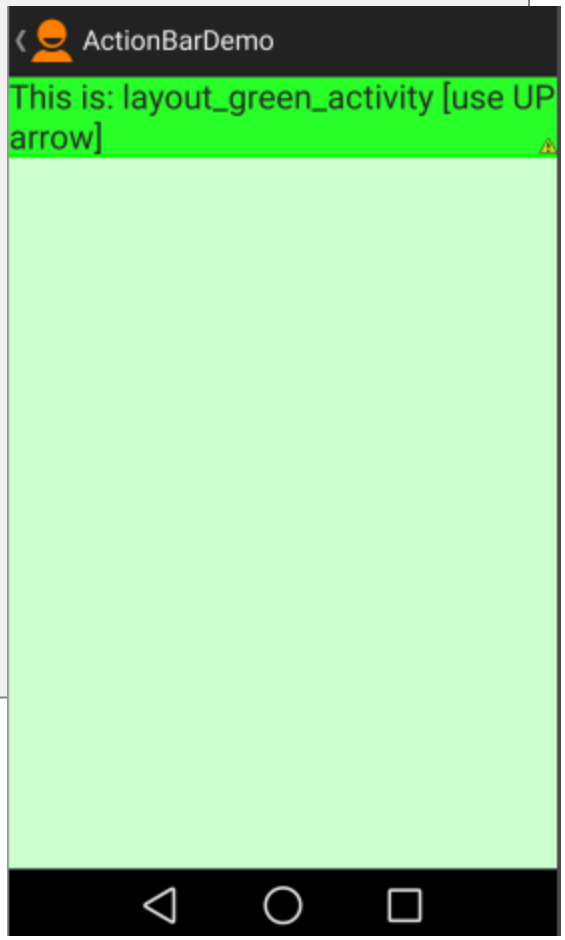


```
        <Button
            android:id="@+id/btnRedFragment"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Red Fragment" />
    </LinearLayout>

    <FrameLayout
        android:id="@+id/frag_container_inside_activity_main"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_margin="6dp"
        android:layout_weight="2" />

</LinearLayout>
```



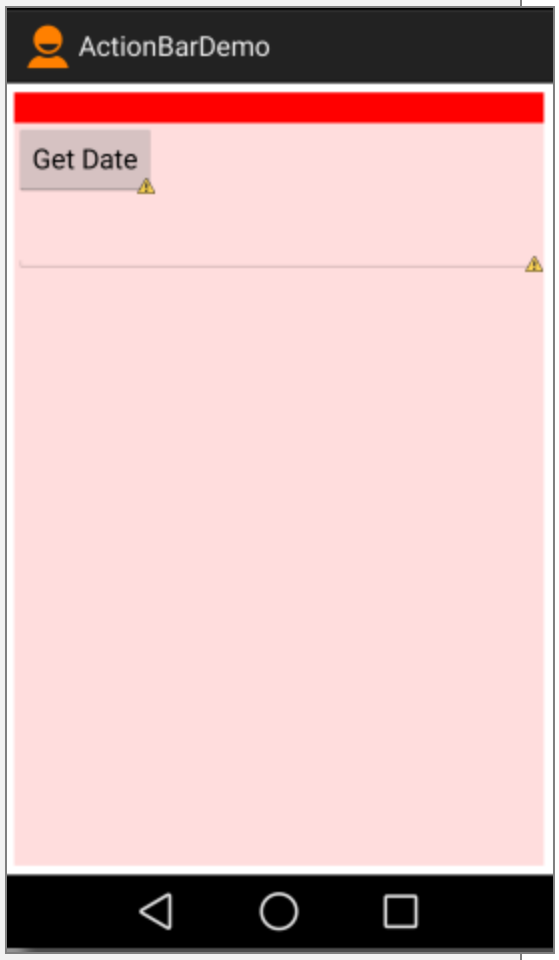
# Example 8 – LAYOUT: layout_green_activity.xml

```
<?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:background="#3300ff00"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/textViewGreen1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#ff00ff00"
        android:text=
          "This is: layout_green_activity [use UP arrow]"
        android:textAppearance=
          "?android:attr/textAppearanceLarge" />

</LinearLayout>
```



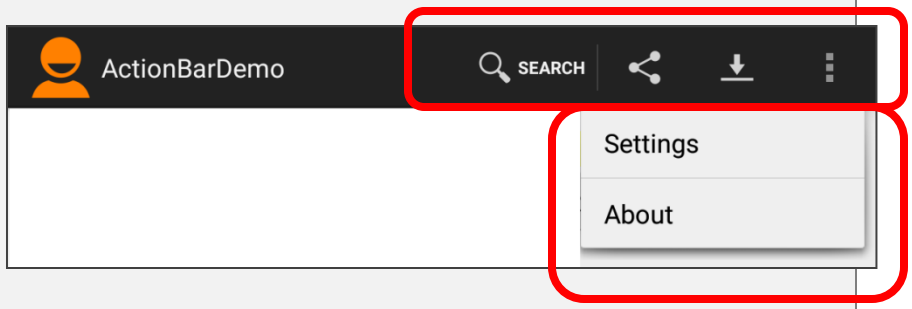
# Example 8 – LAYOUT: layout_red_fragment.xml

```
<?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:background="#22ff0000"
    android:layout_margin="6dp"
    android:orientation="vertical" >

     <View
         android:background="#ffff0000"
         android:layout_width="match_parent"
         android:layout_height="20dp" />
    <Button
         android:id="@+id/btn_date_red_fragment"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="Get Date" />

    <EditText
         android:id="@+id/txtMsgFragmentRed"
         android:layout_width="match_parent"
         android:layout_height="wrap_content"
         android:textSize="30sp"
         android:ems="10"  />

</LinearLayout>
```



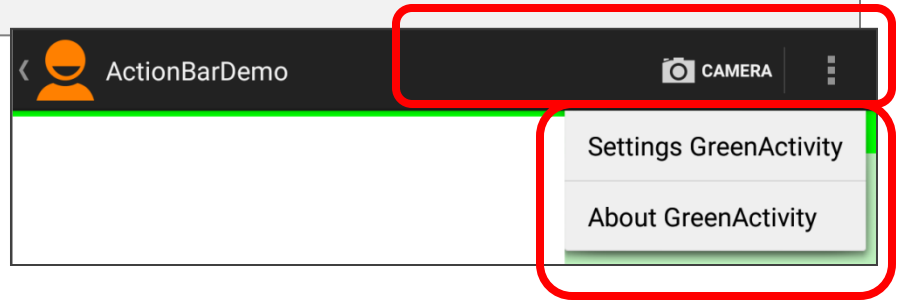
# Example 8 – MENU: main_menu.xml

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <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"/>
    <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 8 – MENU: green_menu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:id="@+id/action_camera_green_activity"
        android:icon="@drawable/ic_action_camera"
        android:orderInCategory="100"
        android:showAsAction="ifRoom|withText"
        android:title="Camera"/>
    <item
        android:id="@+id/action_settings_green_activity"
        android:icon="@drawable/ic_launcher"
        android:orderInCategory="120"
        android:showAsAction="never"
        android:title="Settings GreenActivity"/>
    <item
        android:id="@+id/action_about_green_activity"
        android:icon="@drawable/ic_launcher"
        android:orderInCategory="140"
        android:showAsAction="never"
        android:title="About GreenActivity"/>
</menu>
```



# Example 8 – MENU: red_menu.xml

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

    <item
        android:id="@+id/action_delete_red_frag"
        android:icon="@drawable/ic_action_delete"
        android:orderInCategory="100"
        android:showAsAction="ifRoom|withText"
        android:title="DeleteFrag"/>

    <item
        android:id="@+id/settings_red_frag"
        android:orderInCategory="120"
        android:showAsAction="never"
        android:title="Settings_Red_Frag"/>

    <item
        android:id="@+id/about_red_frag"
        android:orderInCategory="140"
        android:showAsAction="never"
        android:title="About_Red_Frag"/>

</menu>
```

# Example 8 – AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="csu.matos" android:versionCode="1“ android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="16"
        android:targetSdkVersion="21" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <activity
            android:name=".MainActivity"
            android:label="@string/app_name"
            android:windowSoftInputMode="stateHidden" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity
            android:name=".GreenActivity"
            android:label="@string/app_name"
            android:parentActivityName="MainActivity" >
        </activity>

    </application>

</manifest>
```

# Example 8 – MainActivity.java



```
public class MainActivity extends Activity implements OnClickListener {

   EditText txtMsg;
   Button btnGreenActivity;
   Button btnRedFragment;

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

      txtMsg = (EditText) findViewById(R.id.txtMsg);
      btnGreenActivity = (Button) findViewById(R.id.btnGreenActivity);
      btnRedFragment = (Button) findViewById(R.id.btnRedFragment);
      btnGreenActivity.setOnClickListener(this);
      btnRedFragment.setOnClickListener(this);

   }

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

# Example 8 – MainActivity.java



1

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {

    // Handle action bar item clicks here.
    int id = item.getItemId();

    if (id == R.id.action_search) {
        txtMsg.setText("Search...");
        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;
}
```

# Example 8 – MainActivity.java



```
  // "GreenActivity" button used to invoke a supporting Activity called via
  // Intent while "RedFragment" button replaces the MainActivity's UI with
  // a new instance of a "FragmentRed"
  @Override
  public void onClick(View v) {

    if (v.getId() == R.id.btnGreenActivity) {
      Intent greenActivityIntent = new Intent(MainActivity.this,
                                              GreenActivity.class);
      // if needed put data items inside the bundle
      Bundle datainfo = new Bundle();
      greenActivityIntent.putExtra("data", datainfo);
      startActivityForResult(greenActivityIntent, 0);
    } else

    if( v.getId() == R.id.btnRedFragment ){
      // create a new RED fragment - show it
      FragmentTransaction ft = getFragmentManager().beginTransaction();
      RedFragment fragmentRed = RedFragment.newInstance("new-red-frag-arg1");
      ft.replace(R.id.frag_container_inside_activity_main,fragmentRed,"red_frag");
      ft.addToBackStack("red_tran");  //allows BackButton pop-navigation
      ft.commit();
    }

  }//onClick

}//MainActivity
```

2

3

# Example 8 – MainActivity.java

**Comments**

1. In our example each Activity and Fragment has its own menu. Here we begin by allowing the MainActivity to deploy in its ActionBar the options defined in the **main_menu.xml** file. Among other, the menu includes entries to *Search*, *Share*, and *Download*.

2. When the user clicks on the 'Green Activity' button, an Intent is assembled to invoke a GreenActivity. Any optional data to be supplied is placed in the intent's bundle. The method '*startActivityForResults*(…)' is used to allow for possible output values to be returned.

3. Tapping on the 'Red Fragment' button creates a new instance of a RedFragment. The .*replace*(…) method removes any previous view shown on the host area of the MainActivity with the newly created RedFragment UI. The enclosing transaction is pushed on the **BackStack** for possible future use. This will result in a faster and more efficient app.

# Example 8 – GreenActivity.java



```
public class GreenActivity extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.layout_green_activity);

    // enable the home button
    getActionBar().setDisplayHomeAsUpEnabled(true);
    getActionBar().setHomeButtonEnabled(true);

  }//onCreate


    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
      getMenuInflater().inflate(R.menu.green_menu, menu);
      return true;

    }
```

# Example 8 – GreenActivity.java



```
@Override
public boolean onOptionsItemSelected(MenuItem item) {

    switch (item.getItemId()) {

        case android.R.id.home:
        // the user pressed the UP button - where to go now? look for
        // Manifest's entry: android:parentActivityName="MainActivity"
            if (getParentActivityIntent() == null) {
                Log.i("ActivityGreen",
                      "Fix Manifest to indicate the parentActivityName");
                onBackPressed(); //terminate the app

            } else {
                NavUtils.navigateUpFromSameTask(this); //back to parent activity
            }
            return true;

        default:
            return super.onOptionsItemSelected(item);
    }
 }
}
```

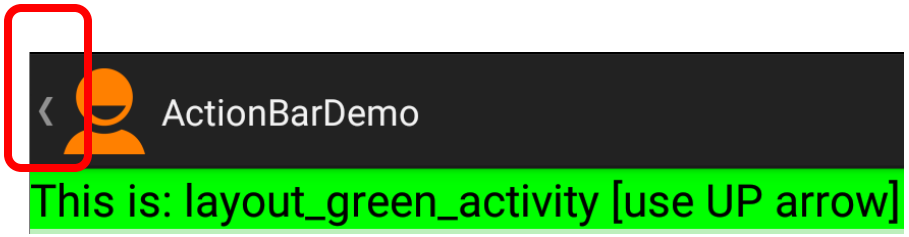
1

# Example 8 – GreenActivity.java

**Comments**

1. In this example the GreenActivity has identified MainActivity as its HighAncestor. This connection is defined in the XML manifest as follows:

```
…
<activity
     android:name=".GreenActivity“
     android:label="@string/app_name“
     android:parentActivityName="MainActivity" >
</activity>
…
```




The MenuItem **`android.R.id.home`** is used to recognize the UP-KEY tapping event. In our example, first we check the Manifest contains a link to the parent. If the link is not defined the app executes *onBackPressed*() to exit. Otherwise, the **NavUtils** class is used to transfer to the declared parent (which is part of the same task).

# Example 8 – RedFragment.java



```
public class RedFragment extends Fragment {

   TextView txtMsgFragmentRed =  null;

   public static RedFragment newInstance(String strArg) {
      RedFragment fragmentRed = new RedFragment();
      Bundle args = new Bundle();
      args.putString("strArg1", strArg);
      fragmentRed.setArguments(args);
      return fragmentRed;
   }

   @Override
   public void onCreate(Bundle arg0) {
      super.onCreate(arg0);
      setHasOptionsMenu(true);

      // enable the home button
      getActivity().getActionBar().setDisplayHomeAsUpEnabled(true);
      getActivity().getActionBar().setHomeButtonEnabled(true);
   }

   @Override
   public View onCreateView(LayoutInflater inflater,
                     ViewGroup container,
                     Bundle savedInstanceState) {

      // inflate layout_blue.xml holding a TextView and a ListView
      LinearLayout redLayout = (LinearLayout) inflater.inflate(
                                 R.layout.layout_red_fragment, null );
```

1

# Example 8 – RedFragment.java



```
        // plumbing - get a reference to textview and listview
        txtMsgFragmentRed = (TextView) redLayout.findViewById(R.id.txtMsgFragmentRed);

        final Button button = (Button)redLayout.findViewById(R.id.btn_date_red_fragment);
        button.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                String text = (new Date()).toString();
                txtMsgFragmentRed.setText(text);
            }
        });
        return redLayout;
    }

    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
        menu.clear();
        inflater.inflate(R.menu.red_menu, menu);

    }
     @Override
     public boolean onOptionsItemSelected(MenuItem item) {
         // Handle action bar item clicks here.
         int id = item.getItemId();

         if (id == android.R.id.home) {
2 →        clearBackStack();          //TRY: jump to MainActivity (HigAncestor)
          //showPreviousRedScreen(); //TRY: same as pressing Back button
         }else {
```

# Example 8 – RedFragment.java



```
            //for now -just show the action-id
            txtMsgFragmentRed.setText("ACTION_ID="+id);
           }

           return super.onOptionsItemSelected(item);
        }

    private void clearBackStack() {
       try {
          FragmentTransaction ft = getFragmentManager().beginTransaction();

          android.app.FragmentManager fragmentManager = getFragmentManager();

          fragmentManager.popBackStackImmediate(null,
                          FragmentManager.POP_BACK_STACK_INCLUSIVE);

           ft.commit();

       } catch (Exception e) {
          Log.e("CLEAR-STACK>>> ", e.getMessage() );

       }

    }//clearBackStack()
```

3

# Example 8 – RedFragment.java



```
    private void showPreviousRedScreen() {
      try {
        FragmentTransaction ft = getFragmentManager().beginTransaction();
        android.app.FragmentManager fragmentManager = getFragmentManager();

        //determine the size n of the BackStack (0,1,..n-1)
        int bsCount = fragmentManager.getBackStackEntryCount();

        //see (without removing) the stack's top entry
        BackStackEntry topEntry = fragmentManager.getBackStackEntryAt(bsCount-1);

        //obtain the numeric ID and name tag of the top entry
        String tag = topEntry.getName();
        int id = topEntry.getId();
        Log.e("RED Top Fragment name: ", "" + tag + " " + id);

        //pop the top entry (until matching id) and reset UI with its state data
        //fragmentManager.popBackStackImmediate(id, 1);
        fragmentManager.popBackStackImmediate();

        ft.commit();

      } catch (Exception e) {
          Log.e("REMOVE>>> ", e.getMessage() );
      }

    }//showPreviousRedScreen

}
```

4

# Example 8 – UP-KEY and BACK-KEY Navigation 1 of 2

**Comments**

1. A RedFragment begins its execution enabling the use of the UP-KEY.

2. For illustration purposes our example allows two course of actions on the tapping of the UP-KEY. The options provide *High-Ancestor* and *Historical* navigation.

3. **High-Ancestor Navigation**. Our custom method ***clearBackStack***() is called to remove all entries pushed by RedFragments on the BackStack. Its execution will result in a return to a 'clean' MainActivity.

   Invoking .*popBackStackImmediate*(*tag, flag*) pops all back stack states up to the one with the given identifier.

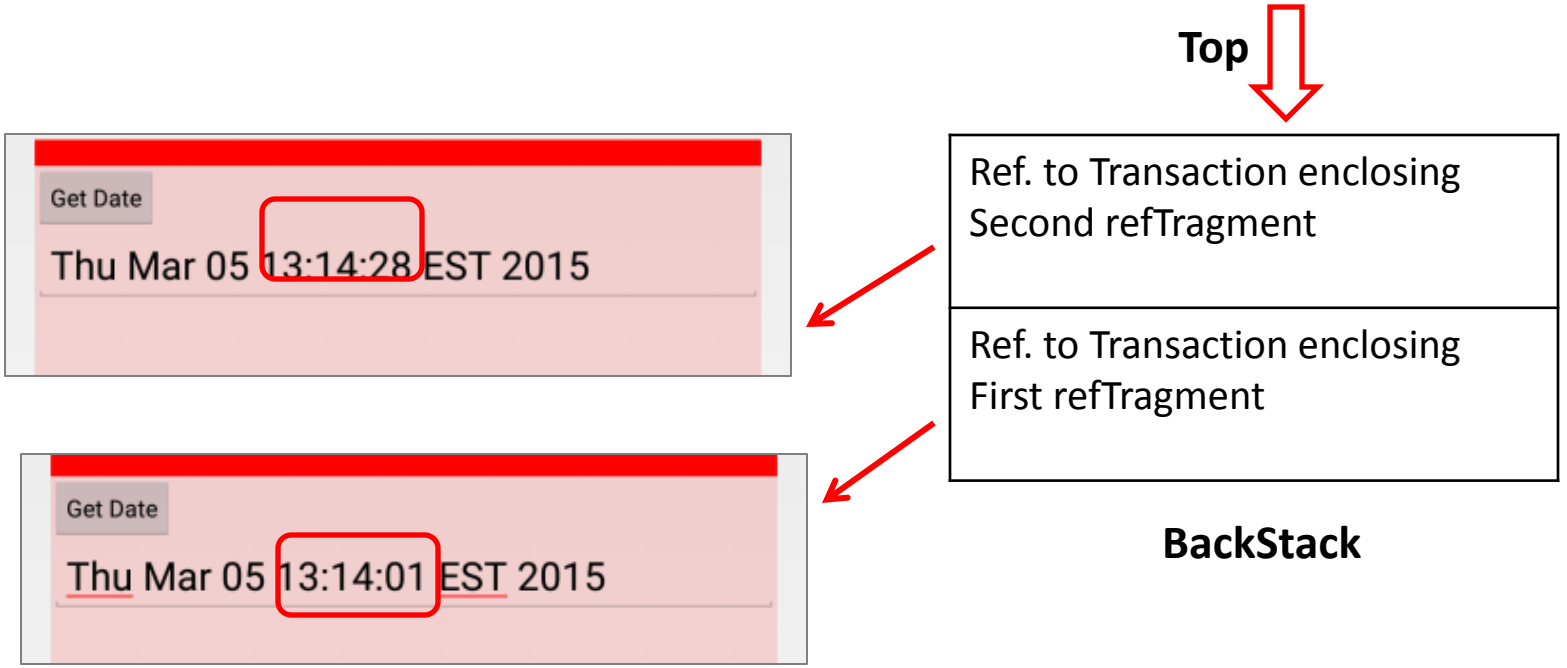
   If *flag* is set to **POP_BACK_STACK_INCLUSIVE** then all matching entries will be consumed until one that doesn't match is found or the bottom of the stack is reached. Otherwise, all entries up to but not including that entry will be removed.

# Example 8 – UP-KEY and BACK-KEY Navigation 2 of 2

**Comments**

4. **Historical Navigation.** Our custom method *showPreviousRedScreen*() illustrates how we could force the tapping of UP-KEY to the behave as a Historical-Navigation.

   The transition to the preceding RedFragment (if any) is made by calling **`.popBackStackImmediate()`** which retrieves and displays the last state saved by a RedFragment in the BackStack.



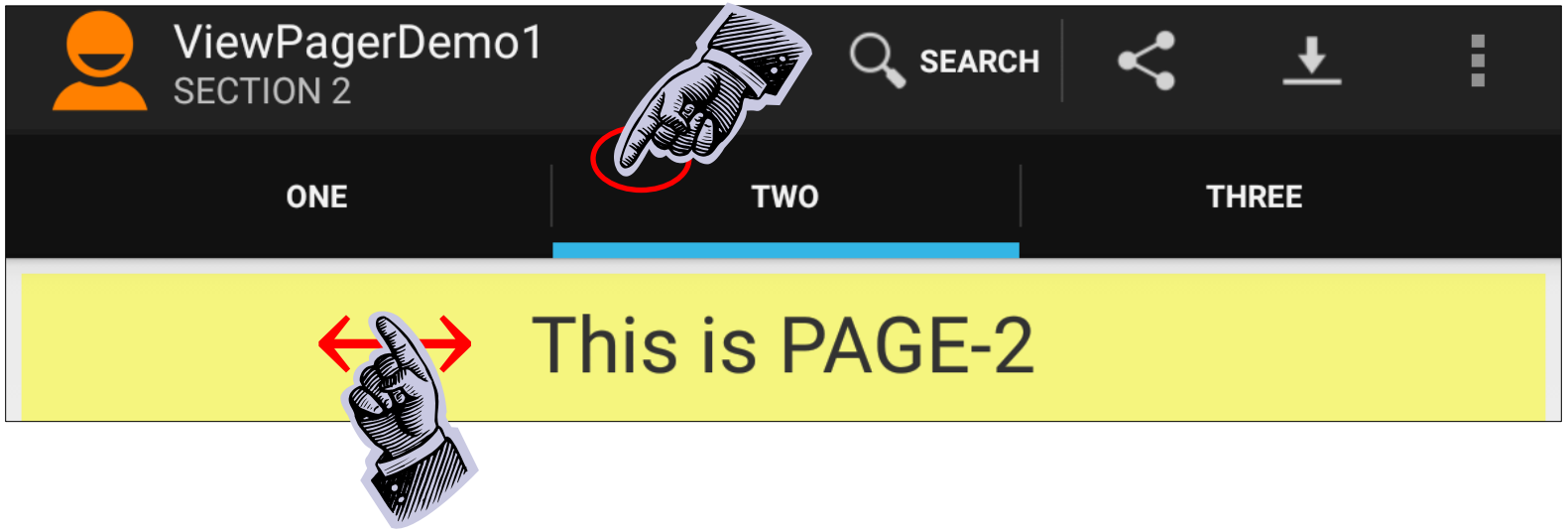
# Example 9 – Tab Navigation: ViewPager

Swiping (or flipping) pages is a **lateral navigation** strategy in which you bring to focus new pages by dragging them from the sides of the screen. The **ViewPager** widget is one component that could be used to produce the horizontal scrolling movement that characterizes lateral navigation..

Pages – commonly represented by Fragments- are generally generated by a custom **PagerAdapter**.

The ViewPager's functionality is usually combined with the app's **ActionBar** or a supporting **TabStrip** to provide an additional **TAB** oriented navigation.

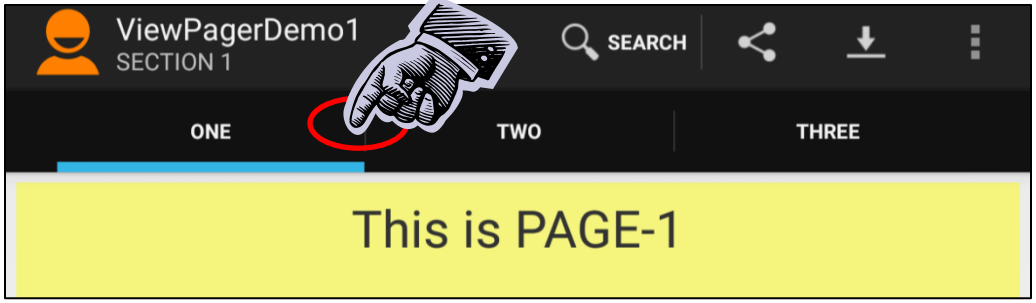
When combined with TAB-navigation, flipping pages updates the tab-marker, and clicking a tab-marker scrolls pages until the selected is shown.

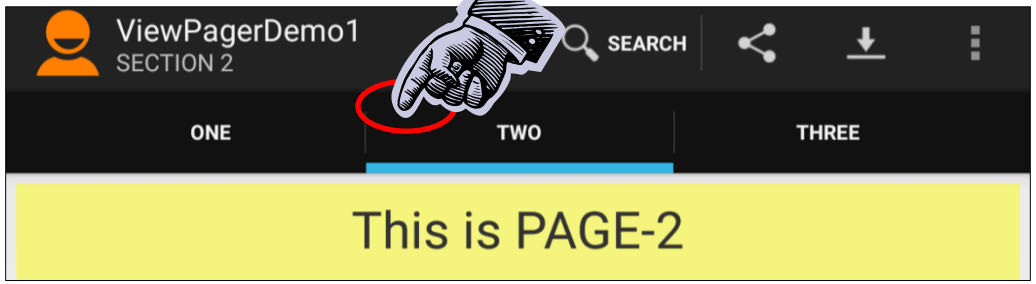


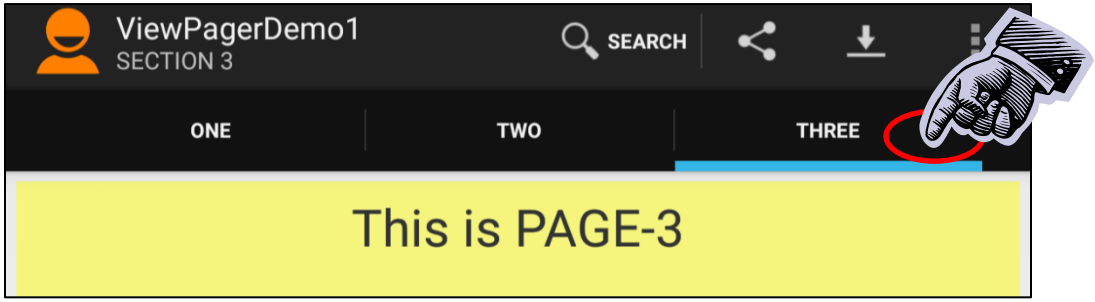
# Example 9 – ViewPager & ActionBar Tabs

In this example we will explore the use of the **ViewPager** widget for lateral-navigation in combination with **Tabs** added to the app's **ActionBar** décor.

Please observe this practice has been ***deprecated*** since API-21 (that is, if TABS are used, they should be placed on other controls but not the ActionBar)



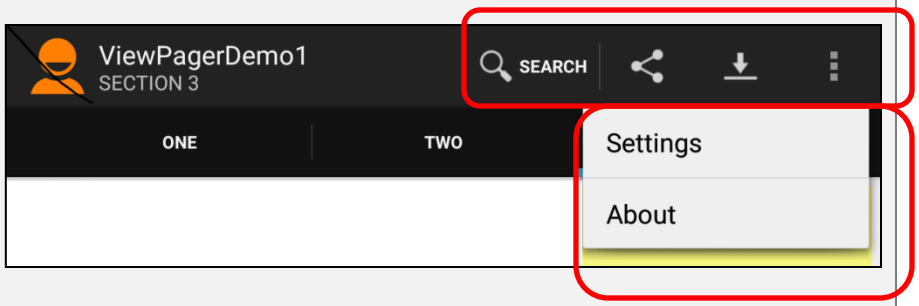




TABS are part of the **ActionBar**.

Clicking on a TAB triggers an animated horizontal scrolling effect bringing into focus the corresponding page

# Example 9 – MENU: main_menu.xml

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <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"/>
    <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 9 – LAYOUTS

## activity_main.xml

```
<android.support.v4.view.ViewPager
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/pager"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="6dp"
    tools:context="csu.matos.MainActivity" />
```

## fragment_page_layout.xml

```
<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:padding="6dp"
    android:background="#77ffff00"
    tools:context="csu.matos.MainActivity$PlaceholderFragment" >

    <TextView
        android:id="@+id/section_label"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:text="Hola"
        android:textSize="30sp" />

</LinearLayout>
```

# Example 9 – MainActivity.java



```
public class MainActivity extends Activity implements TabListener {
  SectionsPagerAdapter mSectionsPagerAdapter;
  ViewPager mViewPager;
  ActionBar actionBar;
  Context context;
  int duration = Toast.LENGTH_SHORT;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    context = MainActivity.this;

    // adapter returns fragments representing pages added to the ViewPager
    mSectionsPagerAdapter = new SectionsPagerAdapter(getFragmentManager());
1 → mViewPager = (ViewPager) findViewById(R.id.pager);
    mViewPager.setAdapter(mSectionsPagerAdapter);

    actionBar = getActionBar();
2 → actionBar.addTab(actionBar.newTab().setText("ONE").setTabListener(this));
    actionBar.addTab(actionBar.newTab().setText("TWO").setTabListener(this));
    actionBar.addTab(actionBar.newTab().setText("THREE").setTabListener(this));
    actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);

    actionBar.setDisplayShowHomeEnabled(true);
    actionBar.setDisplayShowTitleEnabled(true);
```

# Example 9 – MainActivity.java



3 →

```
    mViewPager.setOnPageChangeListener(new OnPageChangeListener() {
    //this listener reacts to the changing of pages
      @Override
      public void onPageSelected(int position) {
        actionBar.setSelectedNavigationItem(position);
        actionBar.setSubtitle(mSectionsPagerAdapter.getPageTitle(position));
      }

      @Override
      public void onPageScrolled(int position, float positionOffset,
                                 int positionOffsetPixels) {
      }

      @Override
      public void onPageScrollStateChanged(int state) {
      }
    });
  }//onCreate

  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
  }
```

# Example 9 – MainActivity.java



```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
  // Handle action bar item clicks here.
  int id = item.getItemId();

  if (id == R.id.action_search) {
    Toast.makeText(context, "Search...", duration).show();
    return true;
  }
  else if (id == R.id.action_share) {
    Toast.makeText(context, "Share...", duration).show();
    return true;
  }
  else if (id == R.id.action_download) {
    Toast.makeText(context, "Download...", duration).show();
    return true;
  }
  else if (id == R.id.action_about) {
    Toast.makeText(context, "About...", duration).show();
    return true;
  }
  else if (id == R.id.action_settings) {
    Toast.makeText(context, "Settings...", duration).show();
    return true;
  }
  return false;
}
```

4

# Example 9 – MainActivity.java



```
public class SectionsPagerAdapter extends FragmentPagerAdapter {

  public SectionsPagerAdapter(FragmentManager fm) {
    super(fm);
  }

  @Override
  public Fragment getItem(int position) {
    return PlaceholderFragment.newInstance(position + 1); //return a fragment
  }

  @Override
  public int getCount() {
    return 3; // Show 3 total pages.
  }

  @Override
  public CharSequence getPageTitle(int position) {
    Locale locale = Locale.getDefault();
    switch (position) {
    case 0:   return getString(R.string.title_section1).toUpperCase(locale);
    case 1:   return getString(R.string.title_section2).toUpperCase(locale);
    case 2:   return getString(R.string.title_section3).toUpperCase(locale);
    }
    return null;
  }
}//SectionsPagerAdapter
```

5

# Example 9 – MainActivity.java



```
public static class PlaceholderFragment extends Fragment {

  private static final String ARG_SECTION_NUMBER = "section_number";

  public static PlaceholderFragment newInstance(int sectionNumber) {
    PlaceholderFragment fragment = new PlaceholderFragment();
    Bundle args = new Bundle();
    args.putInt(ARG_SECTION_NUMBER, sectionNumber);
    fragment.setArguments(args);
    return fragment;
  }

  public PlaceholderFragment() { }

  @Override
  public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
    int pagePosition = getArguments().getInt(ARG_SECTION_NUMBER, -1);
    View rootView = inflater.inflate( R.layout.fragment_main, container, false);
    TextView txtMsg = (TextView) rootView.findViewById(R.id.section_label);
    String text = "This is PAGE-" + pagePosition;
    txtMsg.setText(text);

    return rootView;
  }
}//PlaceHolderFragment
```

6

7

# Example 9 – MainActivity.java



```
  // Implementing ActionBar TAB listener
  @Override
  public void onTabSelected(Tab tab, FragmentTransaction ft) {
    // move to page selected by clicking a TAB
    mViewPager.setCurrentItem(tab.getPosition());
  }

  @Override
  public void onTabUnselected(Tab tab, FragmentTransaction ft) {
    // TODO: nothing - needed by the interface

  }

  @Override
  public void onTabReselected(Tab tab, FragmentTransaction ft) {
    // TODO: nothing - needed by the interface


  }

}
```

8

# Example 9 – ViewPager & ActionBar Tabs 

**Comments**

1. The ViewPager widget defined in the **activity_main.xml** layout as **pager**, will occupy the entire app's screen. A custom **SectionsPagerAdapter** must is defined to create the individual pages to be shown inside the **pager** container.

2. After gaining access to the app's **ActionBar**, we proceed adding three tabs labeled "ONE", "TWO", and "THREE". The clause `setNavigationMode()` is used to activate TAB navigation mode (tabs are not shown if the clause is omitted);

3. The `onPageChangeListener` reacts to the flipping/swiping of pages. When the user moves to a new page the corresponding ActionBar tab is set to reflect the selection.

4. In addition to the navigation tabs, the ActionBar shows the inflate items defined in the **res/menu/main.xml** file. The method `onOptionsMenuItemSelected` is responsible for servicing a requested menu action.

5. Every time a new page *position* on the ViewPager is reached, a fragment is created to fill up its corresponding UI. The method `getItem()` returns the appropriate fragment for the given position.

# Example 9 – ViewPager & ActionBar Tabs 

**Comments**

6. In this example only one kind of fragment is created. For a realistic app, each page may need its own Fragment class. A good practice for defining Fragments is to include a *NewInstace* constructor-style. You accept any supplied arguments, place them in a Bundle, bind to the fragment, and return the newly created fragment instance.

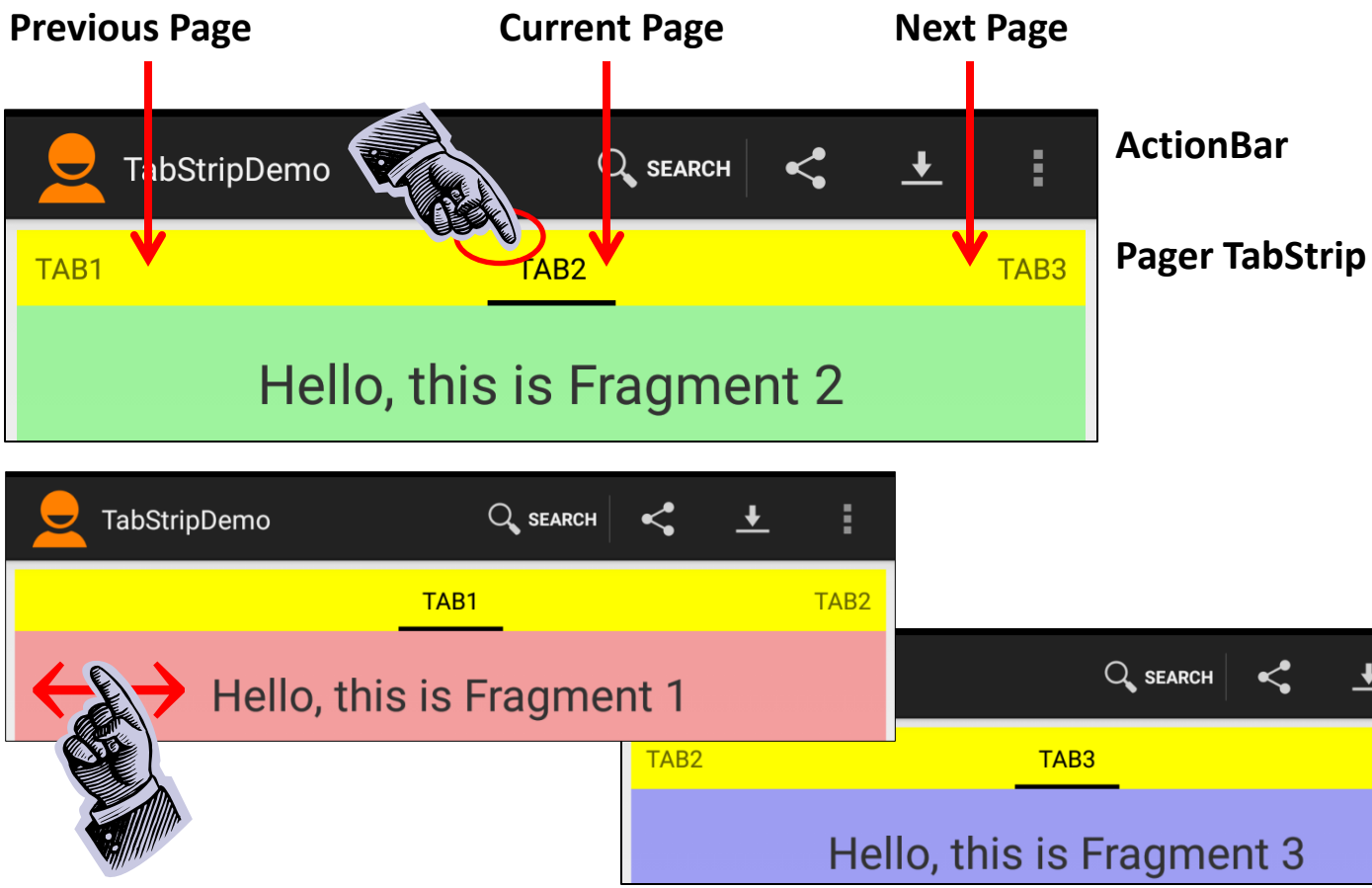
7. All the work of inflating individual fragment layouts, wiring their UI components, defining their listeners etc., is done inside the `onCreateView()` method.

8. The ActionBar's tab-listener detects the clicking of a tab. When that happens it updates the ViewPager so the corresponding page could be shown. The method `setCurrentItem()` flips the pages inside the swipper stopping on the selected one.

# Example 10 – ViewPager & TabStrip

The **PagerTabStrip** is a horizontal band exposing up to *three* tabs. The tabs correspond to the *current, next*, and *previous* pages of a ViewPager. Each tab consists of a *caption* and an *underscore* marker. The marker appears below the selected tab.

The PagerTabStrip and ViewPager work in harmony, swiping pages on the ViewPager produces a similar flipping effect on the TabStrip (and the other way around).



# Example 10 – ViewPager & TabStrip

**LAYOUT:** **activity_main.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v4.view.ViewPager
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/viewpager"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="6dp" >

    <android.support.v4.view.PagerTabStrip
        android:id="@+id/pager_tab_strip"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="top"
        android:background="#FFFFFF00"
        android:padding="10dp" />

</android.support.v4.view.ViewPager>
```

- **ViewPager** & its top **TabStrip** occupy the *entire* screen.

- Individual pages are created with fragments. Each fragment usually inflates an associated layout. When this view becomes visible it replaces the current page.

# Example 10 – ViewPager & TabStrip

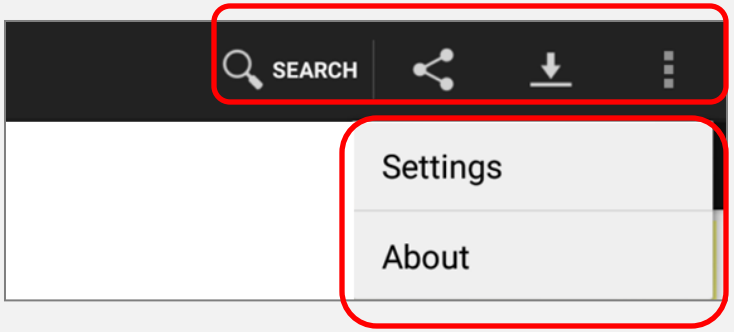
**LAYOUT:** **fragment_page1.xml**

```
<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:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="22dp"
        android:textSize="30sp"
        android:text="Hello, this is Fragment 1" />

</RelativeLayout>
```

The other two layouts: **fragment_page2** and **fragment_page3** are similar, just different caption and background color.

# Example 10 – MENU: main_menu.xml

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <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"/>
    <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>
```




The app's menu is placed on the ActionBar. Keep in mind that the Tabs shown by the PagerTabStrips widget are NOT connected with the ActionBar.

# Example 10 – MainActivity.java



```
public class MainActivity extends FragmentActivity {
   Context context;
   int duration = Toast.LENGTH_SHORT;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      // Get the view from activity_main.xml
      setContentView(R.layout.activity_main);
      context = getApplication();

      // Locate the viewpager in activity_main.xml
      ViewPager viewPager = (ViewPager) findViewById(R.id.viewpager);

1 →   // Set the ViewPagerAdapter into ViewPager
      viewPager.setAdapter(new MyViewPagerAdapter(getSupportFragmentManager()));

   }//onCreate

   @Override
   public boolean onCreateOptionsMenu(Menu menu) {
      // Inflate the menu; this adds items to the action bar if it is present.
      getMenuInflater().inflate(R.menu.main, menu);
      return true;
   }
```

# Example 10 – MainActivity.java



2

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
   int id = item.getItemId();

   if (id == R.id.action_search) {
      Toast.makeText(context, "Search...", duration).show();
      return true;
   }
   else if (id == R.id.action_share) {
      Toast.makeText(context, "Share...", duration).show();
      return true;
   }
   else if (id == R.id.action_download) {
      Toast.makeText(context, "Download...", duration).show();
      return true;
   }
   else if (id == R.id.action_about) {
      Toast.makeText(context, "About...", duration).show();
      return true;
   }
   else if (id == R.id.action_settings) {
      Toast.makeText(context, "Settings...", duration).show();
      return true;
   }
   return false;
}
```

# Example 10 – FragmentPage1.java



```
public class FragmentPage1 extends Fragment {

   @Override
   public View onCreateView(LayoutInflater inflater, ViewGroup container,
                            Bundle savedInstanceState) {

      // Get the view design from file: res/layout/fragment_page1.xml
      View view = inflater.inflate(R.layout.fragment_page1, container, false);

      view.setBackgroundColor(Color.parseColor("#55FF0000"));

      return view;
   }

}
```

3

The other two fragments: **FragmentPage2** and **FragmentPage3** are similar.

# Example 10 – MiViewPagerAdapter.java

```
public class MyViewPagerAdapter extends FragmentPagerAdapter {
   // Tab Captions
   private String tabCaption[] = new String[] { "TAB1", "TAB2", "TAB3" };

   public MyViewPagerAdapter(FragmentManager fragmentManager) {
      super(fragmentManager);
   }

   @Override
   public int getCount() {
      return tabCaption.length;  // return 3 (numbers of tabs in the example)
   }
3 →
   @Override
   public Fragment getItem(int position) {
      switch (position) {
      case 0:  return new FragmentPage1();
      case 1:  return new FragmentPage2();
      case 2:  return new FragmentPage3();
      }
      return null;
   }

   @Override
   public CharSequence getPageTitle(int position) {
      return tabCaption[position];    // return tab caption
   }
}
```

# Example 10 – ViewPager & TabStrip

## Comments

1. Our app shows three independent horizontal bands: the ActionBar on top, followed by the PagerTabStrip, and finally a ViewPager. The ViewPager control takes most of the UI space. It requires an adapter to make the pages to be shown in any given position of the UI. Pages are implemented through fragments. In our example, three different kind of pages will be returned by **MyViewPagerAdapter.**

2. The app's ActionBar inflates the items defined in the **res/menu/main.xml** file. The **onOptionsMenuItemSelected** lstener attends click events requesting those actions to be executed. Please notice the ActionBar and the PagerTabStrips are two separated controls.

3. **MyViewPageAdapter** is responsible for producing the body and caption of ViewPagerviews. There are two important methods in the class: `getItem()` and `getPageTitle()`. Both receive a particular position (0, 1, 2,…) and return the corresponding page and caption respectively.

# Example 11 – Toolbar

The **Toolbar** is a new (and still evolving) control introduced in SDK5.0. It is functionally equivalent to the ActionBar and offers a few more features not found in the 'older' ActionBar.

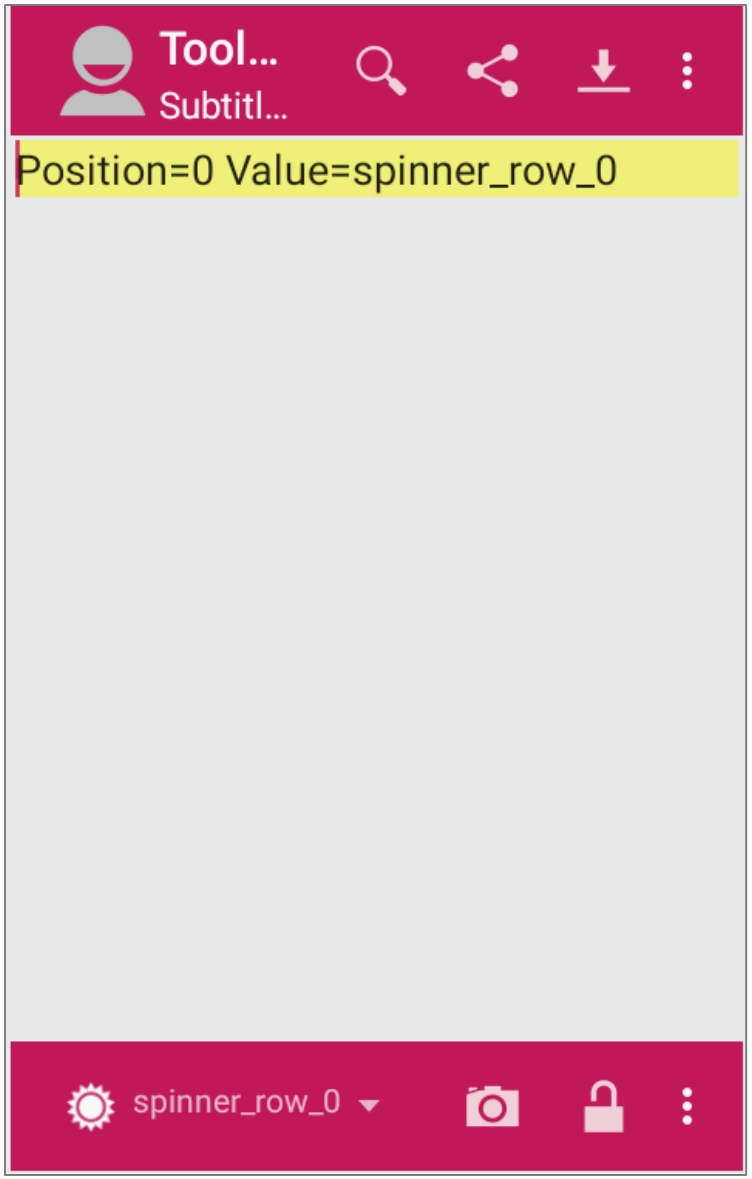
In a manner similar to ActionBars, a Toolbar can include a navigation button, identity logo, title, subtitle, action menu items, and an optional custom view.

Unlike ActionBars -which appear *only* on top of the screen- a Toolbar can be placed anywhere in the user interface. You may have more than one Toolbar on the app's UI each presenting its own theme (e.g. one could be pink the other transparent).


*You can not distinguish between the views produced by a Toolbar and an ActionBar*

# Example 11 – Toolbar

**Toolbar** on top




**Toolbar** at the bottom

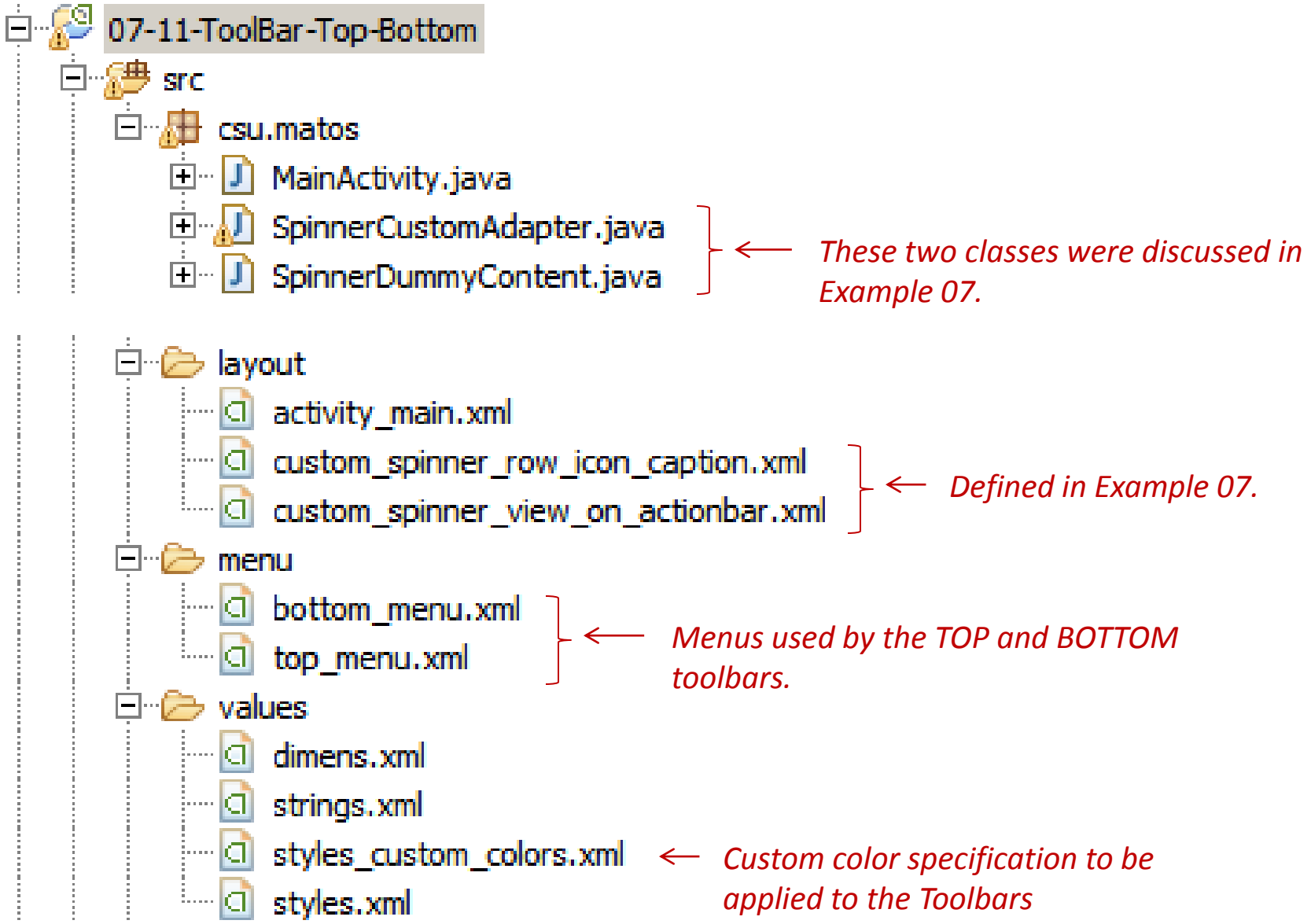
In this example the app has two independent Toolbars.

Toolbars are customizable. The TOP view includes: logo, title, subtitle, menu items, overflow button.

The BOTTOM Toolbar includes a custom view (hosting a Spinner), and additional action buttons.

Each Toolbar may use its own color scheme. In this example we use a custom **Theme** to define colors to be used by the status bar, app bar, and other widgets.

# Example 11 – Toolbar Example Structure

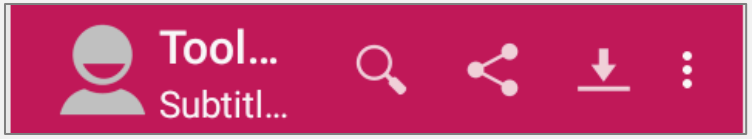


# Example 11 – Layout: activity_main.xml



```
<LinearLayout  xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="?android:attr/colorBackground"
    android:orientation="vertical"
    android:padding="2dp" >

    <Toolbar
        android:id="@+id/Toolbar_top"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="?android:attr/colorPrimary"
        android:minHeight="?android:attr/actionBarSize"
        android:popupTheme="@android:style/ThemeOverlay.Material.Light"
        android:theme="@android:style/ThemeOverlay.Material.Dark.ActionBar" />
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="2"
        android:orientation="vertical" >
        <EditText
            android:id="@+id/txtMsg"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="2dp"
            android:background="#77ffff00"
            android:text="@string/hello_world" />
    </LinearLayout>
```

Toolbar on top of the UI



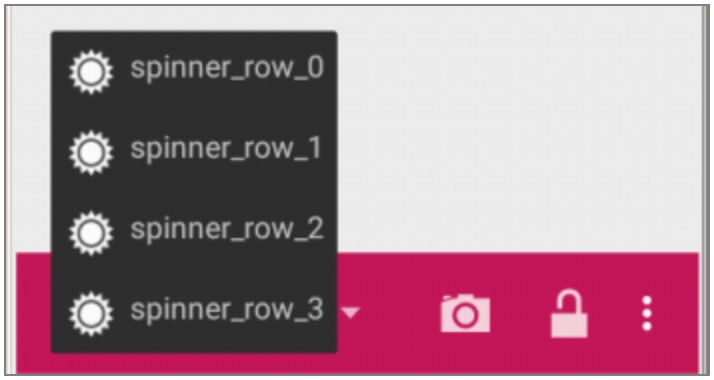
# Example 11 – Layout: activity_main.xml



Toolbar at the bottom of the UI

```
    <Toolbar
        android:id="@+id/Toolbar_bottom"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="?android:attr/colorPrimary"
        android:minHeight="?android:attr/actionBarSize"
        android:popupTheme="@android:style/ThemeOverlay.Material.Light"
        android:theme="@android:style/ThemeOverlay.Material.Dark.ActionBar" />

</LinearLayout>
```

*Later, a custom view holding a Spinner will be added to the bottom Toolbar.*

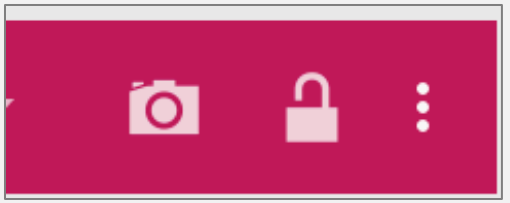
# Example 11 – MENU: top_menu.xml

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <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"/>
    <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>
```


*Portion of the menu shown by the top Toolbar*

# Example 11 – MENU: bottom_menu.xml

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

    <item
        android:id="@+id/action_camera_bottom"
        android:icon="@drawable/ic_action_camera"
        android:orderInCategory="100"
        android:showAsAction="always"
        android:title="Camera"/>
    <item
        android:id="@+id/action_unlock_bottom"
        android:icon="@drawable/ic_action_unlock"
        android:orderInCategory="110"
        android:showAsAction="always"
        android:title="Unlock"/>
    <item
        android:id="@+id/action_settings_bottom"
        android:icon="@drawable/ic_launcher"
        android:orderInCategory="120"
        android:showAsAction="never"
        android:title="Settings Bottom "/>

</menu>
```



*Portion of the menu shown by the bottom Toolbar*

# Example 11 – STYLE: style_custom_colors.xml

You need to modify the **AndroidManifest** to inform of your choice of color themes to be applied to the toolbar. Add the following clause to the manifest's **<application>** element:

**android:theme="@style/MyPinkTheme**

```
<resources>
<style name="MyPinkTheme" parent="@android:style/Theme.Material.Light.DarkActionBar">
    <!-- Customizing the COLOR PALETTE                                          -->
    <!-- Using PINK theme colors from suggested Android palette. Visit link:    -->
    <!-- http://www.google.com/design/spec/style/color.html#color-color-palette -->
    <!-- Elements of material design described at link:                         -->
    <!-- https://developer.android.com/training/material/theme.html             -->

    <!-- do not show app title       -->
    <item name="android:windowNoTitle">true</item>
    <!-- do not show ActionBar       -->
    <item name="android:windowActionBar">false</item>

    <!-- colorPrimary: PINK700 (Toolbar background) -->
    <item name="android:colorPrimary">#C2185B</item>
    <!-- colorPrimaryDark: PINK900 (status bar)     -->
    <item name="android:colorPrimaryDark">#880E4F</item>
    <!-- colorAccent: PINK700 (UI widgets like: checkboxes, text fields )  -->
    <item name="android:colorAccent">#C2185B</item>

    <!-- colorBackground: PINK100 (window background)  -->
    <item name="android:colorBackground">#F8BBD0</item>

  </style>
</resources>
```

Put this definition in the file **res/values/styles_custom_colors.xml**

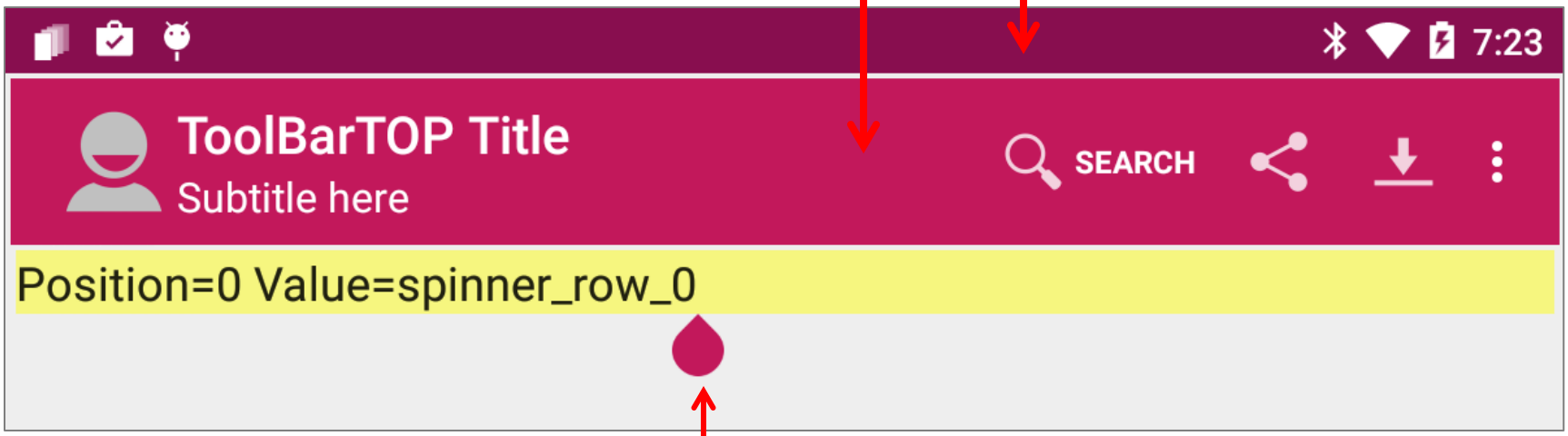
# Example 11 – Toolbar

The following color pattern is defined in the XML file: **styles_custom_colors.xml**

**android:colorPrimaryDark**
Defined in the custom style specification.
To be applied to the *Status Bar*

**Toolbar** on top

**android:colorPrimary**
To be used by the *app's bar*




**android:colorAccent**
To be applied to UI widgets such as text fields , checkboxes, etc.

# Example 11 – MainActivity.java



```
public class MainActivity extends    Activity
             implements OnMenuItemClickListener, OnItemSelectedListener {
   EditText txtMsg;
   Toolbar toolbarTop;
   Toolbar toolbarBottom;
   int selectedSpinnerRow = 0;

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

      //TOP toolbar ------------------------------------------------------
      toolbarTop = (Toolbar) findViewById(R.id.toolbar_top);
      toolbarTop.setTitle("ToolBarTOP Title");
      toolbarTop.setSubtitle("Subtitle here");
      toolbarTop.inflateMenu(R.menu.top_menu);
      toolbarTop.setLogo(R.drawable.ic_action_app_logo);
      toolbarTop.setOnMenuItemClickListener(this);
      //BOTTOM toolbar ---------------------------------------------------
      toolbarBottom = (Toolbar) findViewById(R.id.toolbar_bottom);
      //toolbarBottom.setTitle("ToolBarBOTTOM Title");
      //toolbarBottom.setSubtitle("Subtitle here");
      //toolbarBottom.setLogo(R.drawable.ic_action_app_logo);
      toolbarBottom.inflateMenu(R.menu.bottom_menu);
      //adding a custom-view
      prepareCustomView(toolbarBottom);
      toolbarBottom.setOnMenuItemClickListener(this);
   }
```

1 →

2 →

# Example 11 – MainActivity.java



3

```
@Override
public boolean onMenuItemClick(MenuItem item) {
    int id = item.getItemId();

    // Handle TOP action bar item clicks here.
    if (id == R.id.action_search) {
        txtMsg.setText("Search...");      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;
    }

    // Handle BOTTOM action bar item
    if (id == R.id.action_camera_bottom) {
        txtMsg.setText("Camera...");      return true;
    } else if (id == R.id.action_unlock_bottom) {
        txtMsg.setText("Unlock...");      return true;
    } else if (id == R.id.action_settings_bottom) {
        txtMsg.setText("Settings-Bottom..."); return true;
    }

    return false;
}
```

# Example 11 – MainActivity.java



4

```
private void prepareCustomView(Toolbar toolbar) {

  // setup the BOTTOM toolbar
  LayoutInflater inflater = (LayoutInflater) getApplication()
                            .getSystemService(Context.LAYOUT_INFLATER_SERVICE);

  toolbar.addView(inflater.inflate(R.layout.custom_spinner_view_on_actionbar, null));

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

  // plumbing - get access to the spinner widget shown on the actionBar
  Spinner customSpinner = (Spinner) toolbar.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.setOnItemSelectedListener(this);

  customSpinner.setSelection(selectedSpinnerRow);

}//prepareCustomView
```

# Example 11 – MainActivity.java



5

```
  @Override
  public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {

    selectedSpinnerRow = position;

    TextView caption = (TextView) toolbarBottom.findViewById(
                                  R.id.txtSpinnerRowCaption);

    txtMsg.setText( "Position=" + position + " Value=" + caption.getText());

  }

  @Override
  public void onNothingSelected(AdapterView<?> parent) {
    // TODO nothing to do here...

  }

}
```

# Example 11 – Toolbar

## Comments

1. We begin by setting the top toolbar adding a logo, title, subtitle, and option-menu items.

2. The bottom toolbar includes a custom-view spinner, and option-menu items.

3. The method `onMenuItemClick()` is shared by the top and bottom toolbars, and it processes the clicking event on menu action tiles.

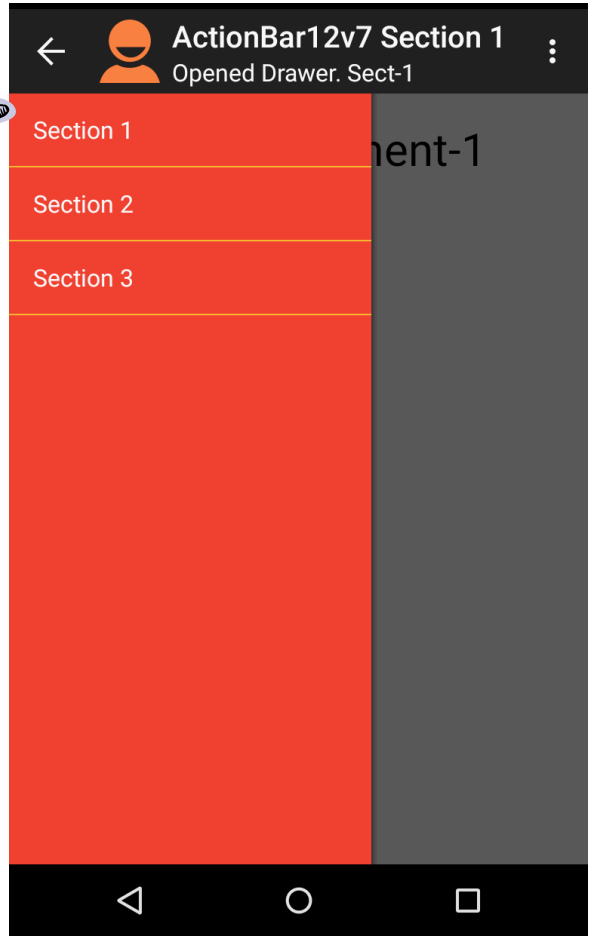
4. The method `prepareCustomView()` inflates the spinner layout. Each line in the list includes an icon and a caption (defined in `custom_spinner_row_icon_caption.xml`). A custom adapter –similar to the one in Example7- is responsible for creating each view added to to the spinner.

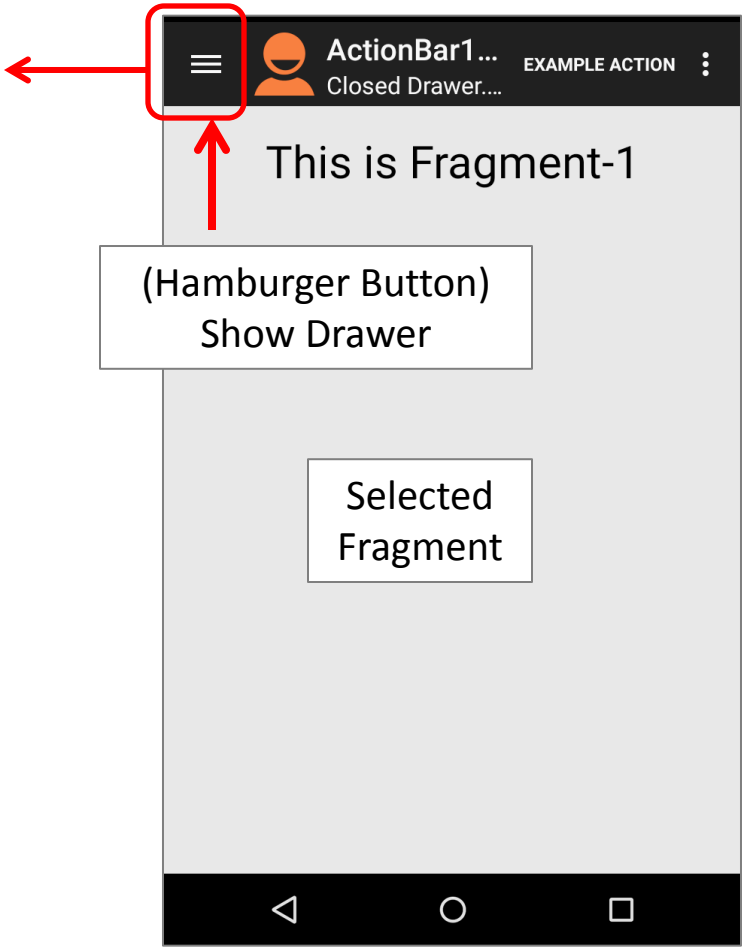
5. When the user taps on a spinner's row, the listener method `onItemSelected()` is called to process the selection and remember the location of the selected row.

# Example 12 – Drawer Layout

- The ActionBar may be modified to include a **HAMBURGER** button on its top-left corner.
- When clicked, you may temporarily overlap a floating UI panel called **DrawerLayout**.
- The Drawer typically presents a list of options (also called Sections, Pages, or Fragments).
- After selecting one of the options, the drawer fades and the UI is updated showing the view associated to the user's chosen section. This *list-based* strategy is called **Drawer-Navigation Pattern**.

Overlapping Drawer Navigation Panel



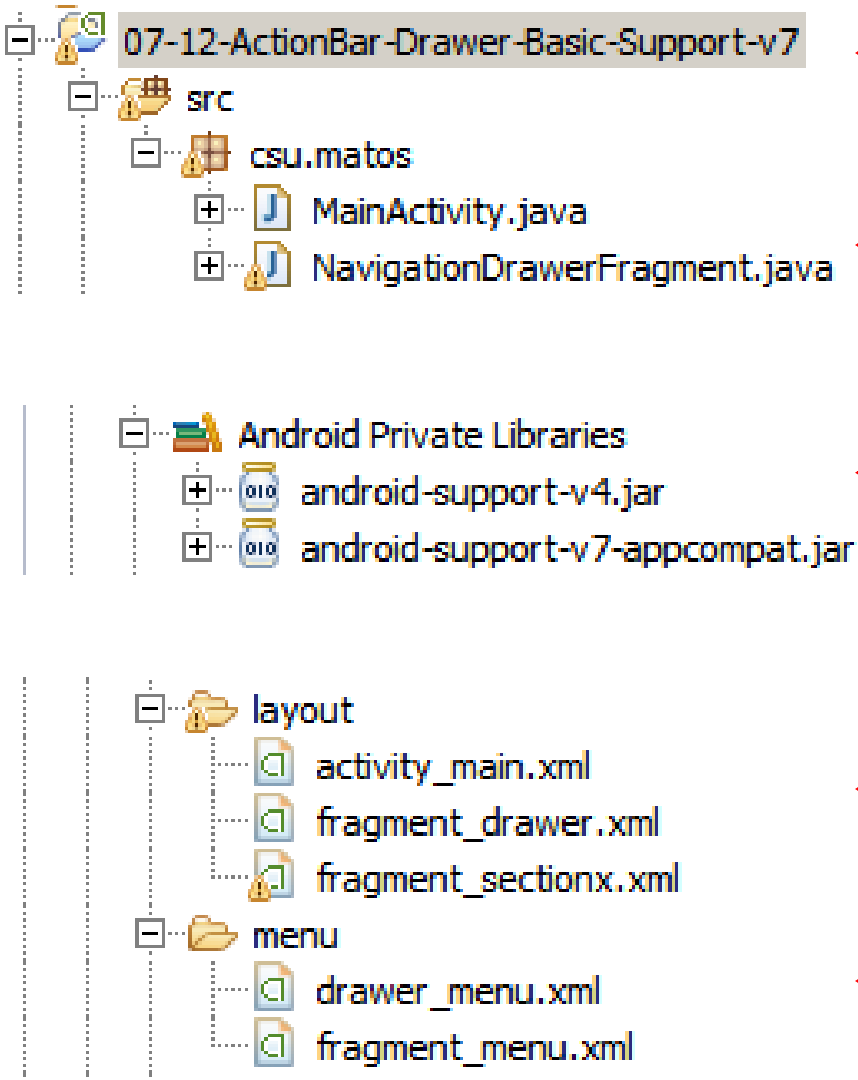




(Hamburger Button) Show Drawer

Selected Fragment

# Example 12 – Drawer Layout

The image below shows various important pieces of the example's architecture.




**This solution is a slightly modified version of the code produced by the Android's 'Navigation Drawer Activity' wizard.**

Define how the floating drawer panel should create a list of options for the user to choose an entry

Use **appcompat-v4** for Fragment management, and **appcompat-v7** for ActionBar support

Layouts for: MainActivity, selected Sections, and Navigation Drawer.

You may show a menu for each section/fragment as well as the navigation drawer's view.

# Example 12 – LAYOUT: activity_main.xml

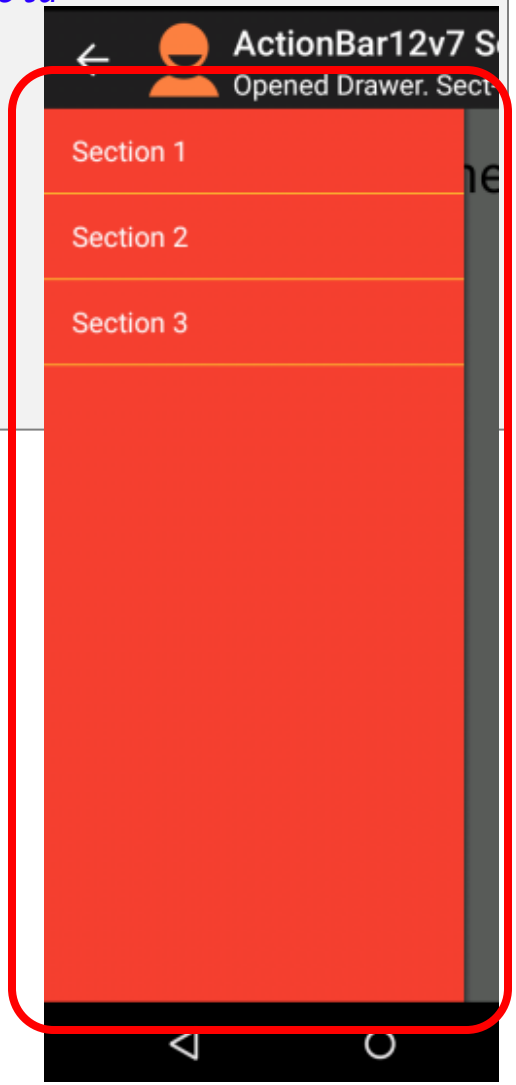
```
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/drawer_layout_activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="csu.matos.MainActivity" >

    <!-- main content view, consumes the entire screen -->
    <FrameLayout
        android:id="@+id/main_fragment_frameLayout"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <!-- the navigation drawer will partially cover the screen -->
    <fragment
        android:id="@+id/navigationDrawerFragment_host_container"
        android:name="csu.matos.NavigationDrawerFragment"
        android:layout_width="@dimen/navigation_drawer_width"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        tools:layout="@layout/fragment_navigation_drawer" />

</android.support.v4.widget.DrawerLayout>
```

Sections/Pages to be made by selected Fragments will occupy this space

Drawer panel will be shown here

# Example 12 – LAYOUT: fragment_drawer.xml

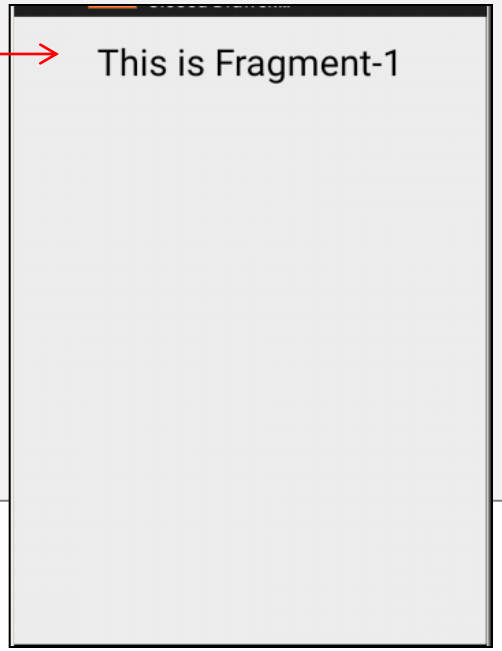
```
<ListView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#F44336"
    android:choiceMode="singleChoice"
    android:divider="@android:color/holo_orange_light"
    android:dividerHeight="1dp"
    tools:context="csu.matos.NavigationDrawerFragment" />
```




*A simple list is shown. You may customize the appearance of the rows by means of a custom list adapter*

# Example 12 – LAYOUT: fragment_sectionx.xml

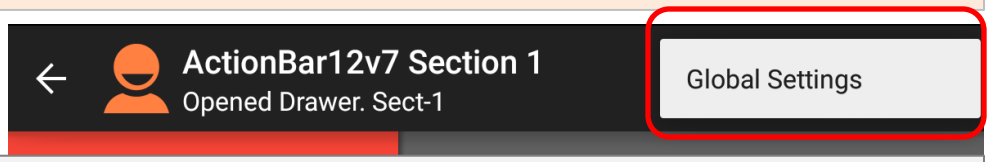
```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="csu.matos.MainActivity$PlaceholderFragment" >

    <TextView
        android:id="@+id/txt_section_label"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:text="This is fragment-n"
        android:textColor="#ff000000"
        android:textSize="30sp" />

</RelativeLayout>
```




*You will need a layout for each Section offered in the option's drawer. For simplicity this example only shows one type of choice.*

# Example 12 – MENUS: fragment_main.xml

## res/menu/drawer_menu.xml




```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto" >
    <item
        android:id="@+id/action_global_settings"
        android:orderInCategory="100"
        android:title="Global Settings"
        app:showAsAction="never"/>
</menu>
```

## res/menu/fragment_menu.xml


```
<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" >
    <item
        android:id="@+id/action_example"
        android:title="@string/action_example"
        app:showAsAction="withText|ifRoom"/>
    <item
        android:id="@+id/action_settings"
        android:orderInCategory="100"
        android:title="@string/action_settings"
        app:showAsAction="never"/>
</menu>
```

# Example 12 – MainActivity.java



```
import ...
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
import android.support.v7.app.ActionBar;
import android.support.v7.app.ActionBarActivity;

public class MainActivity extends ActionBarActivity
             implements NavigationDrawerFragment.NavigationDrawerCallbacks {

   private NavigationDrawerFragment mNavigationDrawerFragment;
   private CharSequence mTitle;
   ActionBar actionBar;

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

      // this is the navigation fragment
      mNavigationDrawerFragment = (NavigationDrawerFragment)
                                  getSupportFragmentManager().findFragmentById(
                                  R.id.navigationDrawerFragment_host_container);

      // link navigation drawerFragment to main_activity layout
      mNavigationDrawerFragment.setUp(
                               R.id.navigationDrawerFragment_host_container,
                               R.id.drawer_layout_activity_main );
      prepareActionBar();
   }
```

1

2

# Example 12 – MainActivity.java



```
@Override
public void onNavigationDrawerItemSelected(int position) {
   // update the main GUI content with selected fragment(SectionX)
   FragmentManager fragmentManager = getSupportFragmentManager();
   fragmentManager
         .beginTransaction()
         .replace(R.id.main_fragment_frameLayout,
                  PlaceholderFragment.newInstance(position + 1))
         .commit();
}

public void onSectionAttached(int number) {
   switch (number) {
   case 1:
      mTitle = getString(R.string.title_section1);
      break;
   case 2:
      mTitle = getString(R.string.title_section2);
      break;
   case 3:
      mTitle = getString(R.string.title_section3);
      break;
   }

}
```

3 →

4 →

# Example 12 – MainActivity.java



```
public void prepareActionBar() {
   actionBar = getSupportActionBar();
5  actionBar.setDisplayShowTitleEnabled(true);
   actionBar.setDisplayShowHomeEnabled(true);
   actionBar.setLogo(R.drawable.ic_launcher);
   actionBar.setDisplayUseLogoEnabled(true);
   actionBar.setTitle(getTitle() + " " + mTitle);
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
   if (!mNavigationDrawerFragment.isDrawerOpen()) {
      getMenuInflater().inflate(R.menu.fragment_menu, menu);
6     prepareActionBar();
      return true;
   }
   return super.onCreateOptionsMenu(menu);
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
   // Handle action bar item clicks here.
7  int id = item.getItemId();
   if (id == R.id.action_settings) {
      return true;
   }
   return super.onOptionsItemSelected(item);
}
```

# Example 12 – MainActivity.java



```
public static class PlaceholderFragment extends Fragment {

    private static final String ARG_SECTION_NUMBER = "section_number";

8   public static PlaceholderFragment newInstance(int sectionNumber) {
        PlaceholderFragment fragment = new PlaceholderFragment();
        Bundle args = new Bundle();
        args.putInt(ARG_SECTION_NUMBER, sectionNumber);
        fragment.setArguments(args);
        return fragment;
    }

    public PlaceholderFragment() { }

    @Override
9   public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View fragmentView = inflater.inflate(R.layout.fragment_sectionx,
                                             container, false);
        TextView txtCaptionFragment = (TextView)fragmentView.findViewById(
                                                 R.id.txt_section_label);
        txtCaptionFragment.setText("This is Fragment-"
                        + getArguments().getInt(ARG_SECTION_NUMBER, 1));

        return fragmentView;

    }
```

# Example 12 – MainActivity.java

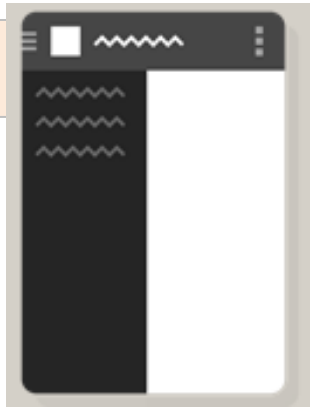


A

```
    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        ((MainActivity) activity).onSectionAttached(getArguments()
                                    .getInt(ARG_SECTION_NUMBER));
    }

}//PlaceHolderFragment

}
```

# Example 12 – MainActivity.java





**Comments**

This example is the result of a series of very minor modifications made to the code automatically generated by the ADT wizard : '**Drawer Navigation Activity**'.

*The app manages two type of fragments; one is dedicated to providing navigation options to the drawer panel. We will call this* one *fragment the* ***drawer-fragment*** *. The other fragments represent full windows shown on the main screen after the user selects a section or topic from the option-list; we will call them* ***section-fragments****.*

1. The MainActivity extends **ActionBarActivity** which is the base class for activities that use the support library action bar features (including methods such as `getSupportFragmentManager(), getSupportActionBar(),` etc). Use it for API level 7 or higher, set the activity theme to *Theme.AppCompat* (or a similar theme). Observe that in this example we employ appcompat-v7 to support ActionBar operations. Drawer actions (open, close, etc.) are managed by appcompat-v4.

# Example 12 – MainActivity.java



2. The resource file **activity_main.xml** defines a `<fragment>` element where the overlapping drawer panel is to be contained. Bullet-2 points to the moment in which the *drawer-fragment* is created and bound to the app's <fragment> component. The drawer will appear on the screen when the user either taps on the **Hamburger** button (top left element of the ActionBar) or, she swipes the screen from side-to-side.

3. The method **`onNavigationDrawerItemSelected`** is an implementation of the **Callback** interface aimed at linking the *drawer-fragment* and the MainActivity. When executed, it creates a *section-fragment* based on the selection made by the user. Its input represents the position of the selected item (or section) picked up from the drawer panel.

4. When the section-fragment is attached to the app's window (usually replacing a previous view) the method **`onSectionAttached`** is called. Here we update the global class variable **`mTitle`** with the section's name.

5. The ActionBar is obtained and setup to include title, logo and Home button.

# Example 12 – MainActivity.java



6. Each section-fragment is allowed to display its own menu. The method **`onCreateOptionsMenu`** inflates the corresponding menu and updated the ActionBar.

7. When a menu item shown on the ActionBar is clicked, the listening method **`onOptionsItemSelected`** is called to processes the user's request.

8. For simplicity, this example creates only one kind of *section-fragment* object. The class **`PlaceholderFragment`** represents the app's response to the selection made by the user.

9. Although all *section-fragments* in this example are alike, each will distinguish itself with its numeric identifier ('This is fragment-1', 'This is fragment-2, …). The **`onCreateView`** method is called each time a selection is made by tapping an option on the drawer panel. It returns a full window, usually made by inflating a layout and setting all necessary listeners on the view.

A. This is an utility method, it accepts the position of the selected drawer-panel option and returns the title of the corresponding *section-fragment*.

# Example 12 – NavigationDrawerFragment.java



This code manages the Navigation Drawer Fragment

```
import ...
import android.support.v4.app.Fragment;
import android.support.v4.view.GravityCompat;
import android.support.v4.widget.DrawerLayout;
import android.support.v7.app.ActionBar;
import android.support.v7.app.ActionBarActivity;
import android.support.v7.app.ActionBarDrawerToggle;

1 →
public class NavigationDrawerFragment extends Fragment {

   private static final String STATE_SELECTED_POSITION ="chosen_drawer_position";
   private static final String PREF_USER_LEARNED_DRAWER = "drawer_learned";

   private NavigationDrawerCallbacks mCallbacks;
   private ActionBarDrawerToggle mDrawerToggle;
   private DrawerLayout mainActivityLayout;
   private ListView mDrawerListView;
   private View mDrawerFragmentHost;
   private ActionBar actionBar;

   private int mCurrentSelectedPosition = 0;
   private boolean mFromSavedInstanceState;
   private boolean mUserLearnedDrawer;

   public NavigationDrawerFragment() {
   }
```

# Example 12 – NavigationDrawerFragment.java



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

   // Read in the flag indicating whether or not the user has demonstrated
   // awareness of the drawer. See PREF_USER_LEARNED_DRAWER for details.
   SharedPreferences sp = PreferenceManager
                          .getDefaultSharedPreferences(getActivity());
   mUserLearnedDrawer = sp.getBoolean(PREF_USER_LEARNED_DRAWER, false);

   if (savedInstanceState != null) {
      mCurrentSelectedPosition = savedInstanceState
                                 .getInt(STATE_SELECTED_POSITION);
      mFromSavedInstanceState = true;
   }

   // Select either the default item (0) or the last selected item.
   selectItem(mCurrentSelectedPosition);

}

@Override
public void onActivityCreated(Bundle savedInstanceState) {
   super.onActivityCreated(savedInstanceState);
   // this fragment may alter the action bar.
   setHasOptionsMenu(true);
}
```

2

3

# Example 12 – NavigationDrawerFragment.java 

```
@Override
public View onCreateView( LayoutInflater inflater, ViewGroup container,
                          Bundle savedInstanceState) {

  mDrawerListView = (ListView)  inflater.inflate(
                                R.layout.fragment_drawer, container, false);

  mDrawerListView.setOnItemClickListener(new AdapterView.OnItemClickListener(){
       @Override
       public void onItemClick(AdapterView<?> parent, View view,
                               int position, long id) {
4 ->     selectItem(position);
       }
     });

  mDrawerListView.setAdapter(new ArrayAdapter<String>(
         getActionBar().getThemedContext(),
         android.R.layout.simple_list_item_1,
         android.R.id.text1,
         new String[] { getString(R.string.title_section1),
                        getString(R.string.title_section2),
                        getString(R.string.title_section3), } ) );
5 ->

  mDrawerListView.setItemChecked(mCurrentSelectedPosition, true);
  return mDrawerListView;

}
```

# Example 12 – NavigationDrawerFragment.java



```
public boolean isDrawerOpen() {
   return mainActivityLayout != null
         && mainActivityLayout.isDrawerOpen(mDrawerFragmentHost);
}

public void setUp(int drawerFragmentId, int activity_main_Id) {
   //find the place in the main layout where drawer fragment is to be shown
   mDrawerFragmentHost = getActivity().findViewById(drawerFragmentId);
   //access the full activity_main layout (DrawerView)
   mainActivityLayout =  (DrawerLayout) getActivity().findViewById(
                                                      activity_main_Id);
   // set custom shadow to overlay main content when the drawer opens
   mainActivityLayout.setDrawerShadow(R.drawable.drawer_shadow,
                                      GravityCompat.START);
   // set up the drawer's list view with items and click listener
   actionBar = getActionBar();
   actionBar.setDisplayHomeAsUpEnabled(true);
   actionBar.setHomeButtonEnabled(true);

   try {
      mDrawerToggle = new CustomActionBarDrawerToggle(mainActivityLayout);
      mainActivityLayout.setDrawerListener(mDrawerToggle);

   } catch (RuntimeException e) {
      Toast.makeText(getActivity(), "Error:"+e.getMessage(),
                     Toast.LENGTH_SHORT).show();
   }
```

6

# Example 12 – NavigationDrawerFragment.java



```
        //the unseen drawer should be shown to the user (once at launch time)
        if (!mUserLearnedDrawer && !mFromSavedInstanceState) {
            mainActivityLayout.openDrawer(mDrawerFragmentHost);
        }

        // Defer code dependent on restoration of previous instance state.
        mainActivityLayout.post(new Runnable() {
            @Override
            public void run() {
                mDrawerToggle.syncState();
            }
        });

    }

    private void selectItem(int position) {
        mCurrentSelectedPosition = position;
7       if (mDrawerListView != null) { //user is learning - section checked!
            mDrawerListView.setItemChecked(position, true);
        }
        if (mainActivityLayout != null) {
            mainActivityLayout.closeDrawer(mDrawerFragmentHost);
        }
        if (mCallbacks != null) {
            mCallbacks.onNavigationDrawerItemSelected(position);
        }
    }
```

# Example 12 – NavigationDrawerFragment.java



```
@Override
public void onAttach(Activity activity) {
    super.onAttach(activity);
    try {
8       mCallbacks = (NavigationDrawerCallbacks) activity;
    } catch (ClassCastException e) {
        throw new ClassCastException("Main must implement Nav.DrawerCallbacks.");
    }
}

@Override
public void onDetach() {
    super.onDetach();
    mCallbacks = null;
}

@Override
public void onSaveInstanceState(Bundle outState) {
9   super.onSaveInstanceState(outState);
    outState.putInt(STATE_SELECTED_POSITION, mCurrentSelectedPosition);
}

@Override
public void onConfigurationChanged(Configuration newConfig) {
    super.onConfigurationChanged(newConfig);
    // Forward the new configuration the drawer toggle component.
    mDrawerToggle.onConfigurationChanged(newConfig);
}
```

# Example 12 – NavigationDrawerFragment.java



```
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
   // (Opened Drawer) show the global app actions in the action bar.
   if (mainActivityLayout != null && isDrawerOpen()) {
      inflater.inflate(R.menu.drawer_menu, menu);

   }
   super.onCreateOptionsMenu(menu, inflater);
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
   if (mDrawerToggle.onOptionsItemSelected(item)) {
      return true;
   }

   if (item.getItemId() == R.id.action_example) {
      Toast.makeText(getActivity(),"Ex. action.",Toast.LENGTH_SHORT).show();
      return true;
   }

   return super.onOptionsItemSelected(item);
}

private ActionBar getActionBar() {
   return ((ActionBarActivity) getActivity()).getSupportActionBar();
}
```

A

# Example 12 – NavigationDrawerFragment.java



B

```
public static interface NavigationDrawerCallbacks {

  void onNavigationDrawerItemSelected(int position);
}
```

C

```
private class CustomActionBarDrawerToggle extends ActionBarDrawerToggle {

  public CustomActionBarDrawerToggle(DrawerLayout mainActivityLayout) {
    super(getActivity(), mainActivityLayout,
          R.string.navigation_drawer_open,R.string.navigation_drawer_close );
  }

  @Override
  public void onDrawerClosed(View view) {
    actionBar.setSubtitle("Closed Drawer. Sect"+(mCurrentSelectedPosition + 1));
    getActivity().invalidateOptionsMenu(); //next is onPrepareOptionsMenu()

  }
```

# Example 12 – NavigationDrawerFragment.java



D

```
    @Override
    public void onDrawerOpened(View drawerView) {

      if (!isAdded()) {
        return;
      }

      if (!mUserLearnedDrawer) {
        // The user manually opened the drawer; store this flag to
        // prevent auto-showing the navig.drawer in the future.
        mUserLearnedDrawer = true;

        SharedPreferences sp = PreferenceManager
                               .getDefaultSharedPreferences(getActivity());

        sp.edit().putBoolean(PREF_USER_LEARNED_DRAWER, true).apply();
      }

      actionBar.setSubtitle("Open Drawer. Sect-"+(mCurrentSelectedPosition + 1));
      getActivity().invalidateOptionsMenu(); // next is onPrepareOptionsMenu()
    }

  }//CustomActionBarDrawerToggle

}// Fragment
```

# Example 12 – NavigationDrawerFragment.java 1 of 4

## Comments

1. The **`NavigationDrawerFragment`** class is responsible for managing the actions of the *drawer-fragment*. The drawer implements a navigation strategy where important sections of the app's architecture are accessible to the user from a custom list of selected options or topics. When the user taps on a section, a fragment representing this selection is shown on the app's window space.

2. When the app launches for the first time it will find there is no state data saved (i.e. no previous section visited, no experience using the drawer). Therefore, the drawer panel will be exposed on top of the main screen. After the user has explored in one session all the options listed on the drawer, the boolean flag `mUserLearnedDrawer` will be set to remember this event. On subsequent sessions of the app, that boolean value will be used to decide whether to show or hide the drawer at the opening stage of the app.

3. The method `setHasOptionsMenu()` is called to let the system know this fragment wants to render its own menu (it will happen as soon as `onCreateOptionsMenu` is executed)

# Example 12 – NavigationDrawerFragment.java 

4. A call to `onCreateView()` results in the inflating of the *drawer-fragment's* layout. This is followed by the setting of a click-listener that captures the position of the selected row shown by the option list. In our example we present a simple text list, however you may show a very elaborated list with custom rows showing any combination of text and images.

5. The example uses a simple `ArrayAdapter<String>` to fill-up the individual drawer's rows. For more elaborated row formats, you need to provide a custom ListAdapter (see Lesson 5).

6. The overlapping Drawer panel is prepared by the `setUp()` method. It first locates the area in the main screen were the *drawer-fragment* is to be placed. Next, it decorates the drawer with a shadow on its right edge. Then it makes sure the action bar will show on the HomeButtom (top-left corner) a **Hamburger** icon  to inform the user of the presence of a navigation panel. Tapping on this icon opens the side menu. The final task of this method is to create a custom toggle control to tell the drawer what to do when it opens and closes.

# Example 12 – NavigationDrawerFragment.java 3 of 4

7. When the user makes a selection from the drawer's list, the position of the selected row is recorded, the list item is checked as 'Visited' , the drawer is closed, and the MainActivity is informed though a callback method of the item's position.

8. The *drawer-fragment* must make sure the MainActivity has a way of listening to its messages. The `onAttach()` method verifies that the MainActivity indeed implements the Callback interface.

9. Before the app terminates, the *position* of the last selected drawer's option is saved in a persistent bundle. The next session will begin showing that choice.

A. The *drawer-fragment* inflates its option menu. It is show on top of the ActionBar.

B. The callback interface to be implemented by the MainActivity is defined. In our example it consists of a single method `onNavigationDrawerItemSelected()` accepting an integer that represents the position of the selected list-item.

# Example 12 – NavigationDrawerFragment.java 4 of 4

C. The `CustomActionBarDrawerToggle` class tells what actions apply when the drawer opens and closes. Here the `onDrawerClosed()` method updates the ActionBar's title to the selected section's name. Finally, the ActionBar is asked to drop the current drawer's menu and replaced it with the MainActivity's menu (it will occur on the next call to onCreateOptionsMenu() ).

   Observe that we are supporting ActionBar operations with the `appcompat-v7` library. The conversion of the top-left corner Navigation-Arrow ( ⟵ ) to a Hamburger icon is done through a pleasant animation. The arrow rotates while the drawer panel is been pulled to the side. Finally the arrow becomes a Hamburger.

D. The first time that `onDrawerOpenned()` is executed the *drawer-fragment* is not added yet to the host activity, and consequently the side-menu will be shown. A record of this visit is kept in persistent storage . The method then updates the ActionBar's title (first time: **null** , after that it holds the last requested title) and invalidates the incoming menu so its own could be shown on top of the ActionBar.

# Lesson 7
# Top Décor Elements:
# ActionBars, Menus, Toolbars

**QUESTIONS?**