31. Creating and Managing Overflow Menus on Android

An area of user interface design that has not yet been covered in this book relates to the concept of menus within an Android application. Menus provide a mechanism for offering additional choices to the user beyond the view components that are present in the user interface layout. While there are a number of different menu systems available to the Android application developer, this chapter will focus on the more commonly used Overflow menu. The chapter will cover the creation of menus both manually via XML and visually using the Android Studio Layout Editor tool.

31.1 The Overflow Menu

The overflow menu (also referred to as the options menu) is a menu that is accessible to the user from the device display and allows the developer to include other application options beyond those included in the user interface of the application. The location of the overflow menu is dependent upon the version of Android that is running on the device. With the Android 4.0 release and later, the overflow menu button is located in the top right-hand corner (Figure 31-1) in the action toolbar represented by the stack of three squares:

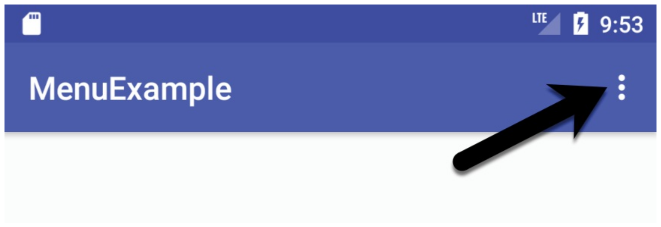


Figure 31-1

31.2 Creating an Overflow Menu

The items in a menu can be declared within an XML file, which is then inflated and displayed to the user on demand. This involves the use of the <menu> element, containing an <item> sub-element for each menu item. The following XML, for example, defines a menu consisting of two menu items relating to color choices:

```
<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=
           ".MenuExampleActivity" >
    <item
        android:id="@+id/menu_red"
        android:orderInCategory="1"
        app:showAsAction="never"
        android:title="@string/red_string"/>
         <item
        android:id="@+id/menu_green"
        android:orderInCategory="2"
        app:showAsAction="never"
        android:title="@string/green_string"/>
</menu>
```

In the above XML, the *android:orderInCategory* property dictates the order in which the menu items will appear within the menu when it is displayed. The *app:showAsAction* property, on the other hand, controls the conditions under which the corresponding item appears as an item within the action bar itself. If set to *if Room*, for example, the item will appear in the action bar if there is enough room. Figure 31-2 shows the effect of setting this property to *ifRoom* for both menu items:

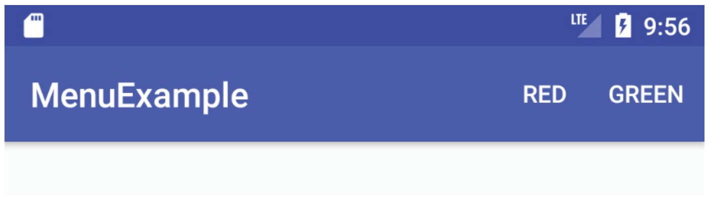


Figure 31-2

This property should be used sparingly to avoid over cluttering the action bar.

By default, a menu XML file is created by Android Studio when a new Android application project is created. This file is located in the *app -> res -> menu* project folder and contains a single menu item entitled “Settings”:

```
<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=".MainActivity">
       <item android:id="@+id/action_settings"
            android:title="@string/action_settings"
            android:orderInCategory="100"
            app:showAsAction="never" />
```

```
</menu>
```

This menu is already configured to be displayed when the user selects the overflow menu on the user interface when the app is running, so simply modify this one to meet your needs.

31.3 Displaying an Overflow Menu

An overflow menu is created by overriding the *onCreateOptionsMenu()* method of the corresponding activity and then inflating the menu's XML file. For example, the following code creates the menu contained within a menu XML file named *menu_menu_example*:

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

As with the menu XML file, Android Studio will already have overridden this method in the main activity of a newly created Android application project. In the event that an overflow menu is not required in your activity, either remove or comment out this method.

31.4 Responding to Menu Item Selections

Once a menu has been implemented, the question arises as to how the application receives notification when the user makes menu item selections. All that an activity needs to do to receive menu selection notifications is to override the *onOptionsItemSelected()* method. Passed as an argument to this method is a reference to the selected menu item. The *getItemId()* method may then be called on the item to obtain the ID which may, in turn, be used to identify which item was selected. For example:

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

       switch (item.getItemId()) {
        case R.id.menu_red:
             // Red item was selected
             return true;
        case R.id.menu_green:
             // Green item was selected
             return true;
        default:
            return super.onOptionsItemSelected(item);
        }
}
```

31.5 Creating Checkable Item Groups

In addition to configuring independent menu items, it is also possible to create groups of menu items. This is of particular use when creating checkable menu items whereby only one out of a number of choices can be selected at any one time. Menu items can be assigned to a group by wrapping them in the *<group>* tag. The group is declared as checkable using the *android:checkableBehavior* property, setting the value to either *single, all or none*. The following XML declares that two menu items make up a group wherein only one item may be selected at any given time:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <group android:checkableBehavior="single">
        <item
            android:id="@+id/menu_red"
            android:title="@string/red_string"/>
        <item
            android:id="@+id/menu_green"
            android:title="@string/green_string"/>
    </group>
</menu>
```

When a menu group is configured to be checkable, a small circle appears next to the item in the menu as illustrated in Figure 31-3. It is important to be aware that the setting and unsetting of this indicator does not take place automatically. It is, therefore, the responsibility of the application to check and uncheck the menu item.

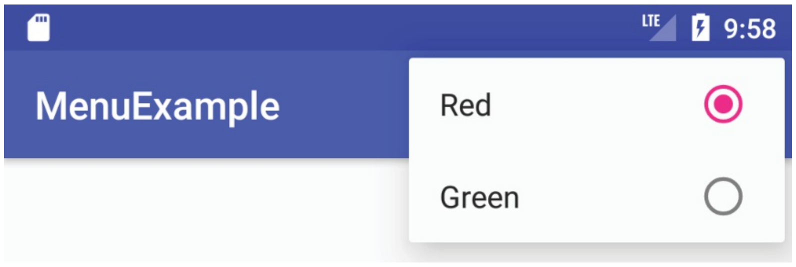


Figure 31-3

Continuing the color example used previously in this chapter, this would be implemented as follows:

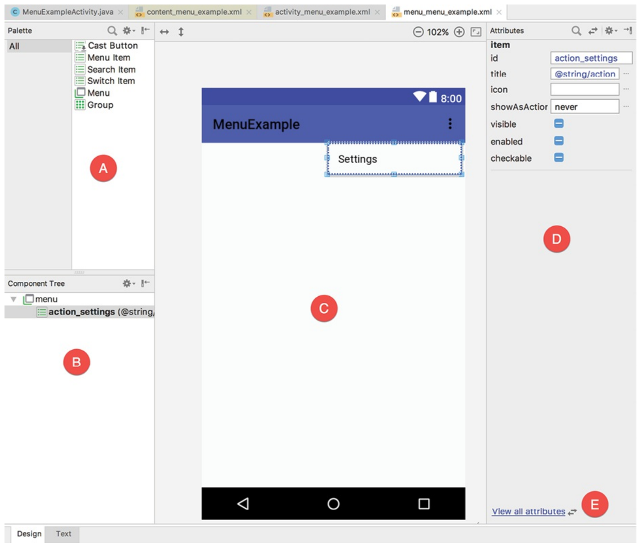
```
@Override
public boolean onOptionsItemSelected(MenuItem item) {

        switch (item.getItemId()) {
            case R.id.menu_red:
                if (item.isChecked()) item.setChecked(false);
                else item.setChecked(true);
                return true;
            case R.id.menu_green:
                if (item.isChecked()) item.setChecked(false);
                else item.setChecked(true);
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
}
```

31.6 Menus and the Android Studio Menu Editor

Android Studio allows menus to be designed visually simply by loading the menu resource file into the Menu Editor tool, dragging and dropping menu elements from a palette and setting properties. This considerably eases the menu design process, though it is important to be aware that it is still necessary to write the code in the *onOptionsItemSelected()* method to implement the menu behavior.

To visually design a menu, locate the menu resource file and double-click on it to load it into the Menu Editor tool. Figure 31-4, for example, shows the default menu resource file for a basic activity loaded into the Menu Editor:



Attributes

item

id: action_settings

title: @string/action

icon:

showAsActior: never

visible

enabled

checkable

Figure 31-4

The palette (A) contains items that can be added to the menu contained in the design area (C). The Component Tree (B) is a useful tool for identifying the hierarchical structure of the menu. The Attributes panel (D) contains a subset of common attributes for the currently selected item. The view all attributes link (E) may be used to access the full list of attributes.

New elements may be added to the menu by dragging and dropping objects either onto the layout canvas or the Component Tree. When working with menus in the Layout Editor tool, it will sometimes be easier to drop the items onto the Component Tree since this provides greater control over where the item is placed within the tree. This is of particular use, for example, when adding items to a group.

Although the Menu Editor provides a visual approach to constructing menus, the underlying menu is still stored in XML format which may be viewed and edited manually by switching from Design to Text mode using the tab marked F in the above figure.

31.7 Creating the Example Project

To see the overflow menu in action, create a new project in Android Studio, entering *MenuExample* into the Application name field and *ebookfrenzy.com* as the Company Domain setting before clicking on the *Next* button.

On the form factors screen, enable the *Phone and Tablet* option and set the minimum SDK setting to API 14: Android 4.0 (IceCreamSandwich). Continue to proceed through the screens, requesting the creation of a basic activity named *MenuExampleActivity* with a corresponding layout file named *activity_menu_example.*

When the project has been created, navigate to the *app* -> *res* -> *layout* folder in the Project tool window and double-click on the *content_menu_example.xml* file to load it into the Android Studio Menu Editor tool. Switch the tool to Design mode, select the ConstraintLayout from the Component Tree panel and enter *layoutView* into the ID field of the Attributes panel.

31.8 Designing the Menu

Within the Project tool window, locate the project's *app* -> *res* -> *menu* -> *menu_menu_example.xml* file and double-click on it to load it into the Layout Editor tool. Select and delete the default Settings menu item added by Android Studio so that the menu currently has no items.

From the palette, click and drag a menu *group* object onto the title bar of the layout canvas as highlighted in Figure 31-5:

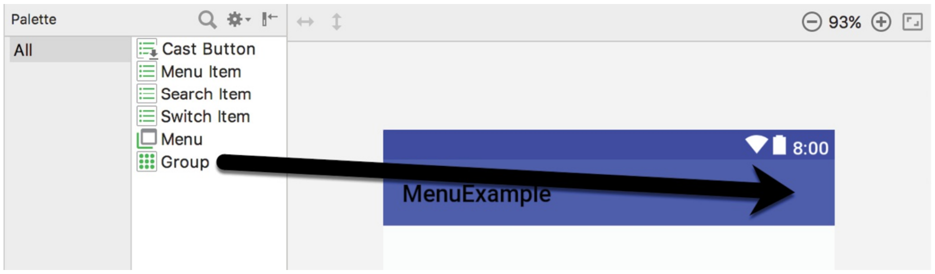


Figure 31-5

Although the group item has been added, it will be invisible within the layout. To verify the presence of the element, refer to the Component Tree panel where the group will be listed as a child of the menu:

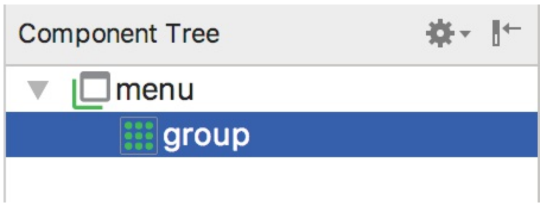
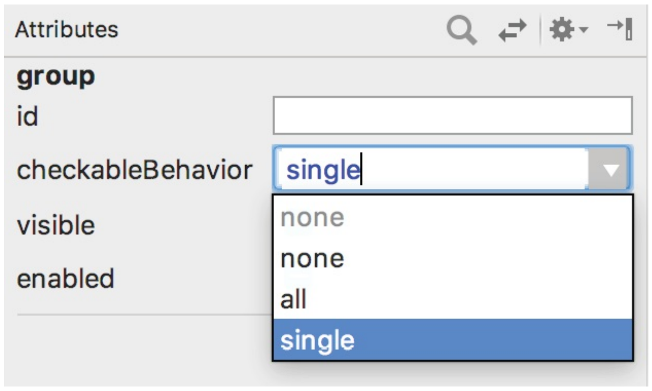


Figure 31-6

Select the *group* entry in the Component Tree and, referring to the Attributes panel, set the *checkableBehavior* property to *single* so that only one group menu item can be selected at any one time:


Attributes

group

id

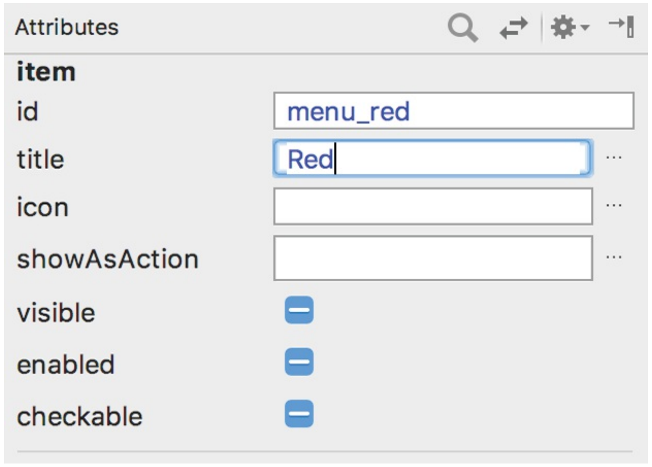
checkableBehavior single

visible

enabled

Figure 31-7

Next, drag four *Menu Item* elements from the palette and drop them onto the *group* element in the Component Tree. Select the first item and use the Attributes panel to change the title to "Red" and the ID to *menu_red*:


Attributes

item

id menu_red

title Red

icon

showAsAction

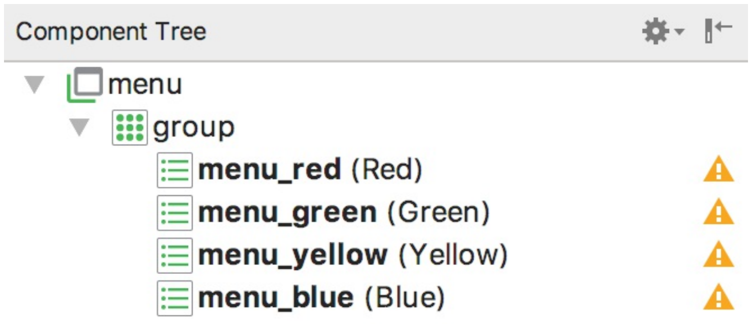
visible

enabled

checkable

Figure 31-8

Repeat these steps for the remaining three menu items setting the titles to "Green", "Yellow" and "Blue" with matching IDs of *menu_green*, *menu_yellow* and *menu_blue*. Use the warning buttons to the right of the menu items in the Component Tree panel to extract the strings to resources:


Component Tree

menu

group

menu_red (Red)

menu_green (Green)

menu_yellow (Yellow)

menu_blue (Blue)

Figure 31-9

On completion of these steps, the menu layout should match that shown in Figure 31-10 below:

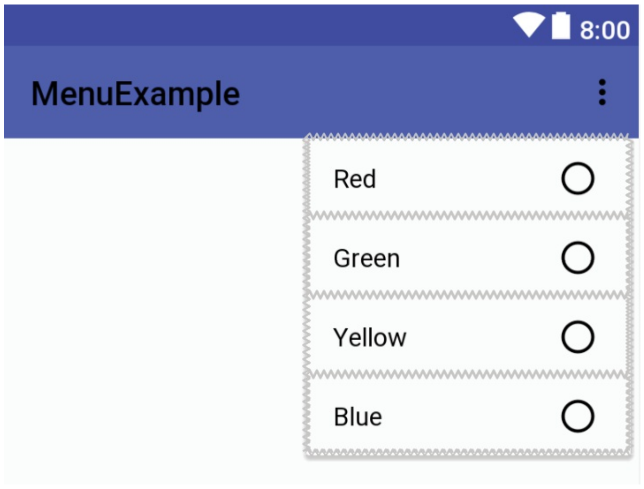


Figure 31-10

Switch the Layout Editor tool to Text mode and review the XML representation of the menu which should match the following listing:

```
<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="com.ebookfrenzy.menuexample.MenuExampleActivity">

    <group android:checkableBehavior="single">
        <item android:title="@string/red_string"
            android:id="@+id/menu_red" />
        <item android:title="@string/green_string"
            android:id="@+id/menu_green" />
        <item android:title="@string/yellow_string"
            android:id="@+id/menu_yellow" />
        <item android:title="@string/blue_string"
            android:id="@+id/menu_blue" />
    </group>
</menu>
```

31.9 Modifying the onOptionsItemSelected() Method

When items are selected from the menu, the overridden *onOptionsItemSelected()* method of the application's activity will be called. The role of this method will be to identify which item was selected and change the background color of the layout view to the corresponding color.

Locate and double-click on the *app -> java -> com.ebookfrenzy.menuexample -> MenuExampleActivity* file and modify the method as follows:

```
package com.ebookfrenzy.menuexample;

import android.os.Bundle;
import android.support.design.widget.FloatingActionButton;
import android.support.design.widget.Snackbar;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.View;
import android.view.Menu;
import android.view.MenuItem;
import android.support.constraint.ConstraintLayout;

public class MenuExampleActivity extends AppCompatActivity {
.
.
.
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {

        ConstraintLayout mainLayout =
                (ConstraintLayout) findViewById(R.id.layoutView);

        switch (item.getItemId()) {
            case R.id.menu_red:
                if (item.isChecked()) item.setChecked(false);
                else item.setChecked(true);
                mainLayout.setBackgroundColor(android.graphics.Color.
                return true;
            case R.id.menu_green:
                if (item.isChecked()) item.setChecked(false);
                else item.setChecked(true);
                mainLayout.setBackgroundColor(android.graphics.Color.
                return true;
            case R.id.menu_yellow:
                if (item.isChecked()) item.setChecked(false);
                else item.setChecked(true);
                mainLayout.setBackgroundColor(android.graphics.Color.
                return true;
            case R.id.menu_blue:
                if (item.isChecked()) item.setChecked(false);
                else item.setChecked(true);
                mainLayout.setBackgroundColor(android.graphics.Color.I
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }
.
.
}
```

31.10 Testing the Application

Build and run the application on either an emulator or physical Android device. Using the overflow menu, select menu items and verify that the layout background color changes appropriately. Note that the currently selected color is displayed as the checked item in the menu.

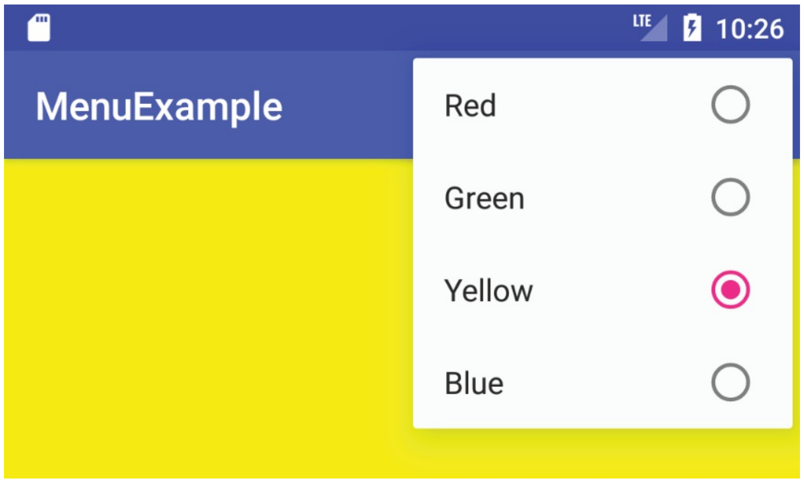


Figure 31-11

31.11 Summary

The Android overflow menu is accessed from the far right of the actions toolbar at the top of the display of the running app. This menu provides a location for applications to provide additional options to the user.

The structure of the menu is most easily defined within an XML file and the application activity receives notifications of menu item selections by overriding and implementing the *onOptionsItemSelected()* method.