

November 4, 2013 04:07 AM

# Android Working with Action Bar

59 Comments

Like  247

 +1

 Tweet  12

Android action bar was introduced to maintain a consistent navigation across the application. It has the powerful capabilities like adapting to screen configurations (landscape & portrait), prioritizing important actions, adding widgets to action bar (search, sharing etc.), providing navigation between screens (drop-down & tabbed navigation) and much more.

In this tutorial we are going to cover most of the action bar functionality. Some of the topics left for upcoming tutorials as they are pretty much lengthy.

advertise here

DOWNLOAD CODE

VIDEO DEMO



## Overview of Action Bar

Action bar mainly contains four functional areas. They are app icon, view control, action buttons and action overflow.

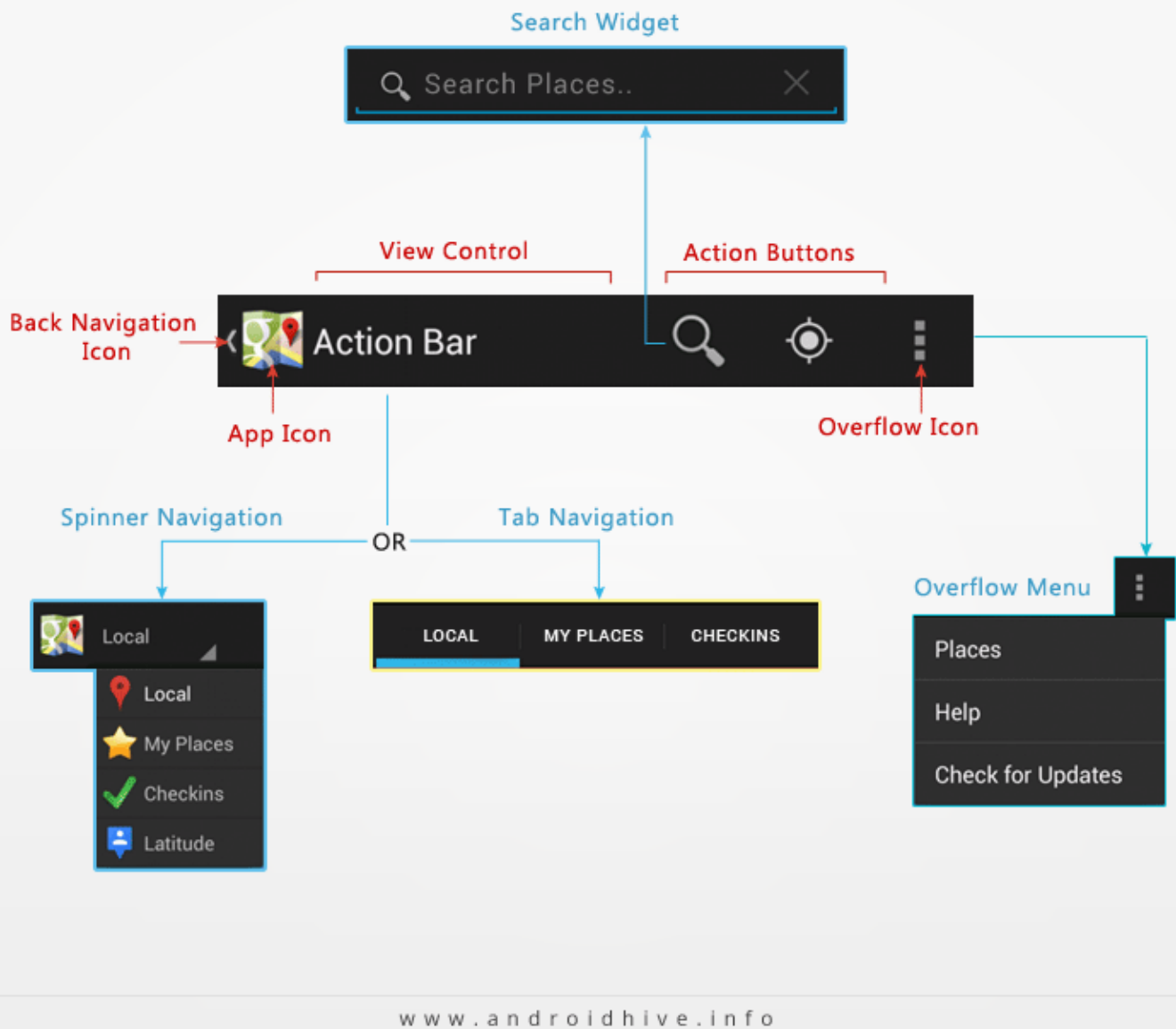
**App Icon** – App branding logo or icon will be displayed here.

**View Control** – A dedicated space to display app title. Also provides option to switch between views by adding spinner or tabbed navigation.

**Action Buttons** – Some important actions of the app can be added here.

**Action Overflow** – All unimportant action will be shown as a menu.

Check out the following diagram for complete overview about action bar.



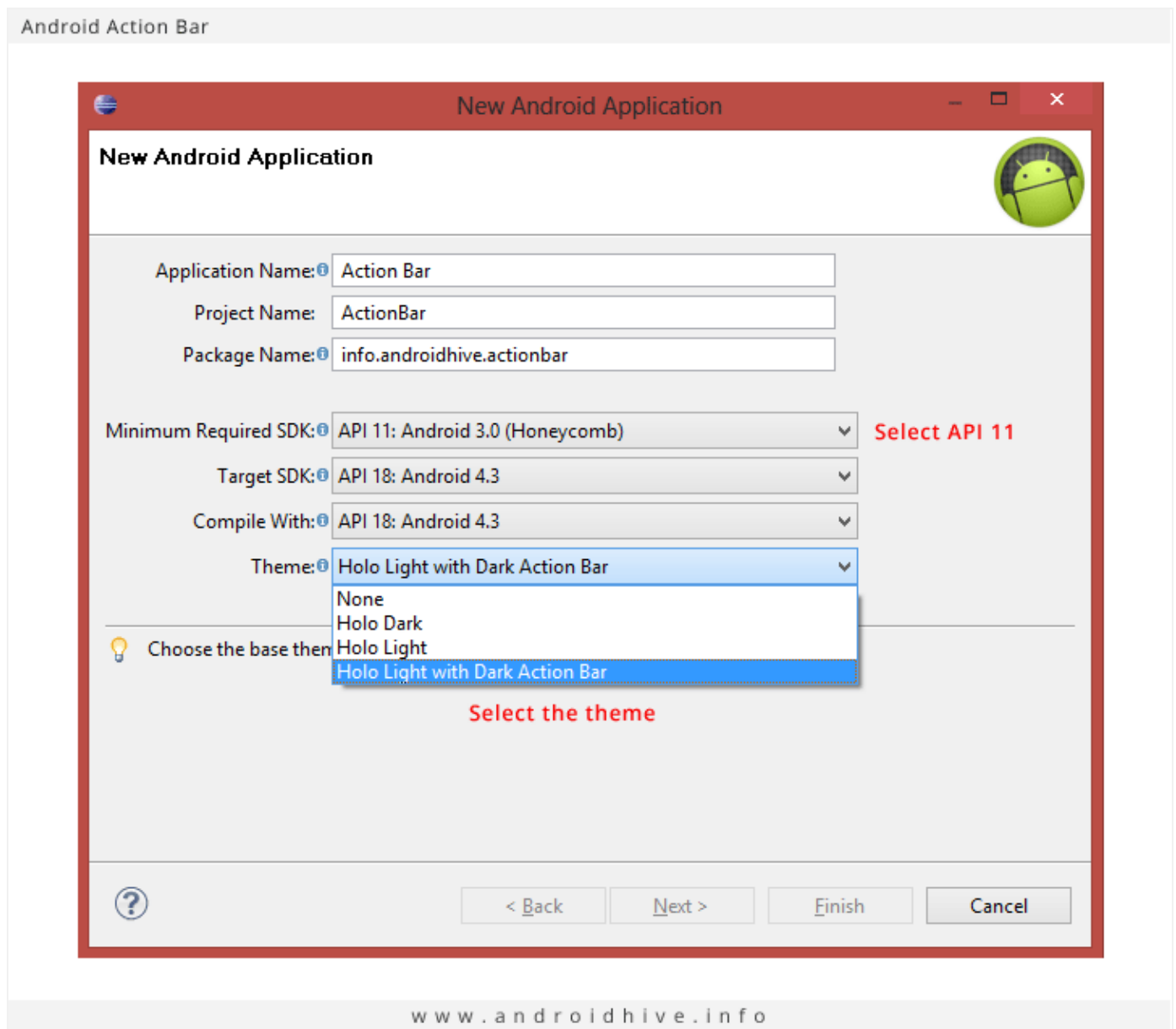
## Android version support below 3.0

Action bar is introduced in android 3.0 (API level 11), but if you want your app to support action bar in older versions too, then use [Support Library](#) to make it compatible with older versions (Android 2.1 and above)

## Starting new Project

1. Create a new project in Eclipse from **File** ⇒ **New** ⇒ **Android Application Project**. While creating the project

select Minimum SDK version to **API 11** and select a theme with action bar. (I left my main activity name as **MainActivity.java**)



2. Android.com provides some useful icons for action bar. Download the [Action Bar Icon Set](#) and select the required icons and add them to the project. Copy each icon with all resolutions (**xxhdpi**, **xhdpi**, **hdpi**, **mdpi**) into respected folders in Eclipse project under **res** ⇒ **drawable-** folders.

## Adding Action Bar Icons

Once you are done copying required icons, we will start adding the action items first. The action bar uses the same older menu method to show action items.

3. Create a new xml file under **res** ⇒ **menu** named **activity\_main\_actions.xml** and add the following code. Here each **<item>** indicates each action item.

```
activity_main_actions.xml

<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <!-- Search / will display always -->
    <item android:id="@+id/action_search"
        android:icon="@drawable/ic_action_search"
        android:title="@string/action_search"
        android:showAsAction="ifRoom"/>

    <!-- Location Found -->
    <item android:id="@+id/action_location_found"
        android:icon="@drawable/ic_action_location_found"
        android:title="@string/action_location_found"
        android:showAsAction="ifRoom" />

    <!-- Refresh -->
    <item android:id="@+id/action_refresh"
        android:icon="@drawable/ic_action_refresh"
        android:title="@string/action_refresh"
        android:showAsAction="ifRoom" />

    <!-- Help -->
    <item android:id="@+id/action_help"
        android:icon="@drawable/ic_action_help"
        android:title="@string/action_help"
        android:showAsAction="never"/>

    <!-- Check updates -->
    <item android:id="@+id/action_check_updates"
        android:icon="@drawable/ic_action_refresh"
        android:title="@string/action_check_updates"
        android:showAsAction="never" />
</menu>
```

Here the important xml attributes should be known are

**android:icon** – Defines the icon of the action item.

**android:title** – Title for the icon.

**android:showAsAction** – Defines the visibility of the action item. It accepts following values.

<b>ifRoom</b>	Displays the icon if there is space available on the screen
<b>never</b>	Never places this icon on the action bar
<b>always</b>	Forces to display the icon always irrespective of space available. This way is not suggested.
<b>withText</b>	Displays a text along with the icon. Normally the text value defined by android:title will be displayed
	Defines the action layout associated with it. This action view defined using

4. Now open your main activity class and do the following in **onCreateOptionsMenu()** method.

```
public class MainActivity extends Activity{

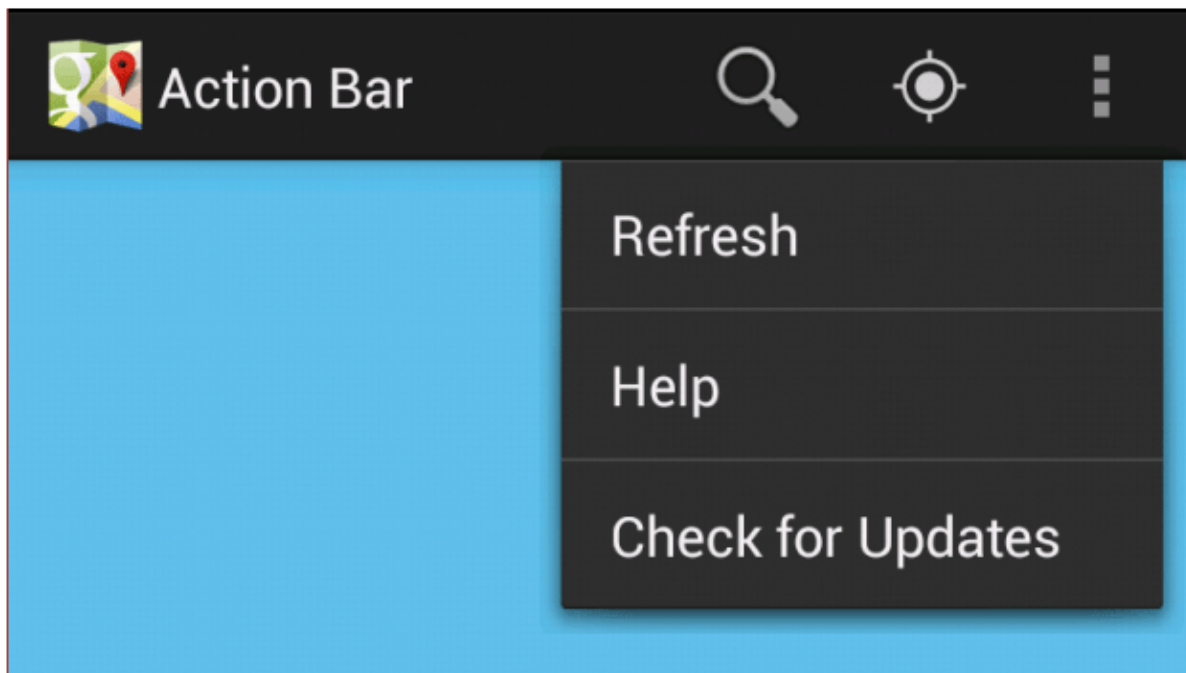
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.activity_main_actions, menu);

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

Now if you run the project, you can see the action bar with action icons. You can also notice that an overflow icon shown which opens the unimportant action items as a drop down menu.

#### Action Bar adding Action Items



# Handling Action Bar Icon Click Events

Until now we displayed action bar action items, but we haven't enabled the interaction with action items. Let's add click event listener to action items now.

5. Open your main activity and override **onOptionsItemSelected()** method. This method accepts menu item as a parameter. Selected action item can be identified by using its id. Normally a switch case statement is suggested for this purpose. You can perform appropriate action in the matched case block.

```
public class MainActivity extends Activity{
...
...

/**
 * On selecting action bar icons
 * */
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Take appropriate action for each action item click
    switch (item.getItemId()) {
        case R.id.action_search:
            // search action
            return true;
        case R.id.action_location_found:
            // location found
            LocationFound();
            return true;
        case R.id.action_refresh:
            // refresh
            return true;
        case R.id.action_help:
            // help action
            return true;
        case R.id.action_check_updates:
            // check for updates action
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}

/**
 * Launching new activity
 * */
private void LocationFound() {
    Intent i = new Intent(MainActivity.this, LocationFound.class);
    startActivity(i);
}
}
```

# Enabling Up / Back Navigation

Action bar also provides the feasibility of back navigation when the app involves hierarchical relationship between screens. Please note that this behavior is not the same as android system back button. A small back arrow icon will be displayed before action bar icon when up navigation is enabled. Follow these steps to enable up navigation.

Enabling up navigation involves two steps.

- > Defining the parent activity in the **AndroidManifest.xml** file
- > Enabling the up navigation in the activity class.

To test this scenario I have created another activity and enabled the up navigation in that activity. If you observe the above section, I have launched LocationFound activity in the menu selection block. So let's create required files for that activity.

7. Create the layout file for this activity named **activity\_location\_found.xml** under **res** ⇒ **layout** folder.

```
activity_location_found.xml<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="10dp">

    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Location is found. Drop a Message here"/>

</LinearLayout>
```

8. Create a new class under you main package named **LocationFound.java** and paste the following code.

Action bar Up navigation can be enabled by calling `setDisplayHomeAsUpEnabled(true)` method on to action bar. By calling this function a back arrow will be displayed on the action bar.

```
package info.androidhive.actionbar;

import android.app.ActionBar;
import android.app.Activity;
import android.os.Bundle;

public class LocationFound extends Activity {

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



```

// get action bar
ActionBar actionBar = getActionBar();

// Enabling Up / Back navigation
actionBar.setDisplayHomeAsUpEnabled(true);
    }
}

```

9. Also we need to add the new activity in **AndroidManifest.xml** file. Here we have to mention the parent activity to which the up navigation is pointing to by using `android:parentActivityName` property.

```

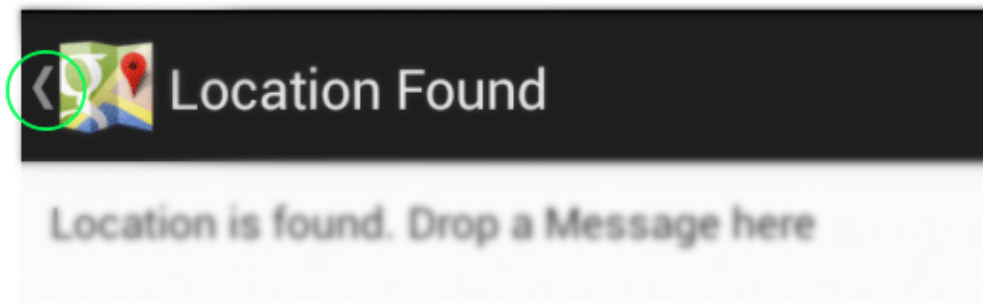
<!-- Location found activity -->
<activity
    android:name="info.androidhive.actionbar.LocationFound"
    android:label="@string/activity_new_message"
    android:parentActivityName="info.androidhive.actionbar.MainActivity"
</activity>

```

Run the project and check it once.

#### Action Bar Up / Back Navigation

Back Navigation



www.androidhive.info

## Hiding / Showing the Action Bar in Particular Activity

In some cases you might wanted to hide the action bar in particular activity. You can hide and show the action bar by calling `hide()` and `show()` methods.

```

ActionBar actionBar = getActionBar();
// hide the action bar
actionBar.hide();

// show the action bar
actionBar.show();

```

# Changing the Action Bar Icon

Action bar by default displays the application icon which was set using **android:icon** in the AndroidManifest.xml file. So if you want to change the action bar icon, you can do it by calling `setIcon(drawable)` on to action bar.

```
ActionBar actionBar = getActionBar();  
  
// set the icon  
actionBar.setIcon(R.drawable.ico_actionbar);
```

## Action Bar changing the Icon



Action Bar with application icon



Action Bar with custom icon

[www.androidhive.info](http://www.androidhive.info)

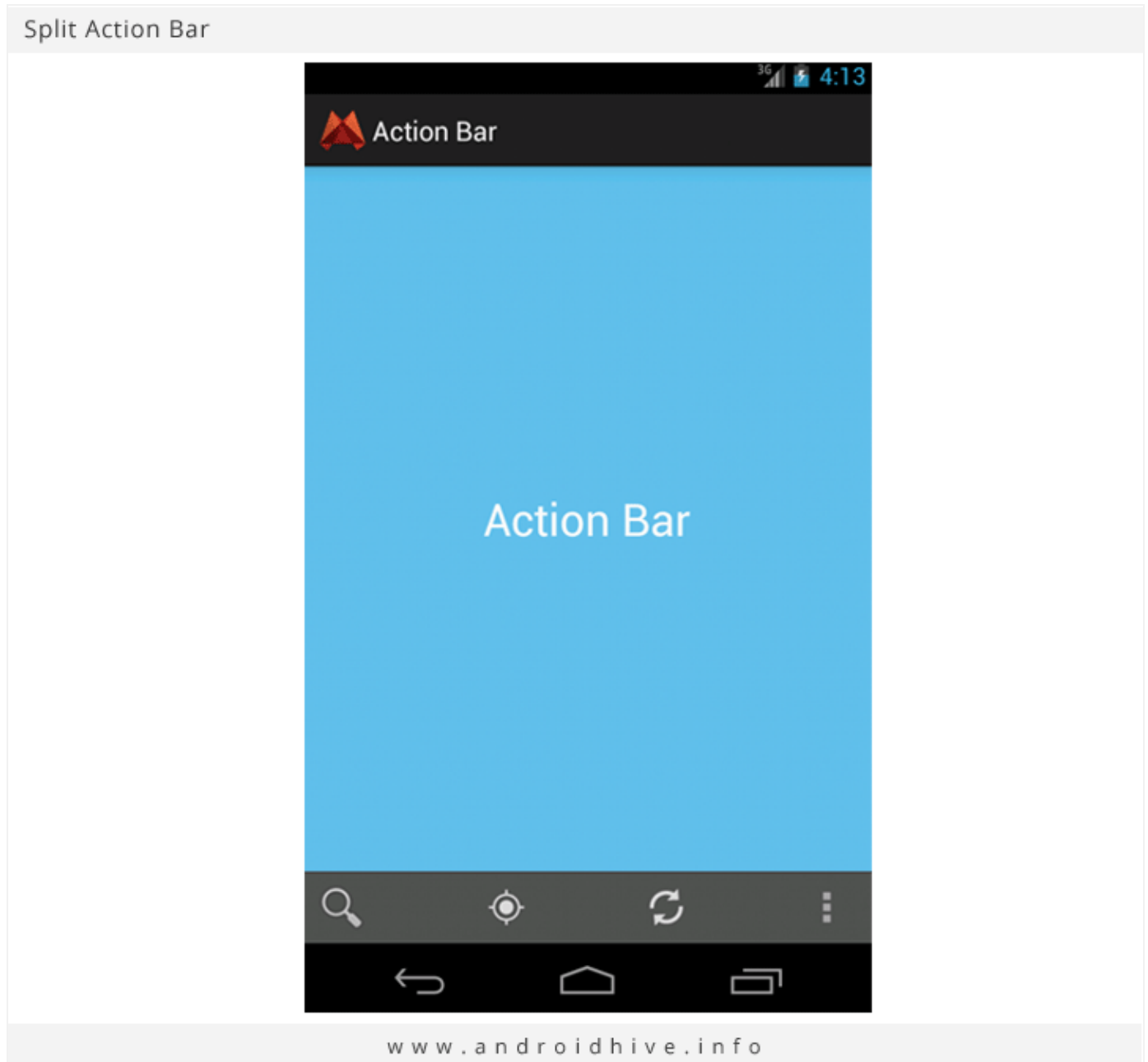
# Enabling Split Action Bars

Split action bar provides a separate bar at the bottom of the screen. This option will be useful when you want to display the action items at the bottom of the screen by leaving some space on the title bar.

To enable split action bar add `uiOptions="splitActionBarWhenNarrow"` to all the **<activity>** tags or to the **<application>** tag directly in **AndroidManifest.xml** file. Also we need to add **<meta-data>** with the value `android.support.UI_OPTIONS` to support older version below API level 14.

```
<!-- Location found activity -->
```

```
<activity
    android:name="info.androidhive.actionbar.LocationFound"
    android:label="@string/activity_location_found"
    android:parentActivityName="info.androidhive.actionbar.MainActivity" >
    <!-- To support below API Level 14 -->
    <meta-data android:name="android.support.UI_OPTIONS"
        android:value="splitActionBarWhenNarrow" />
</activity>
```



Once again run the project and check whether you are getting split action bar or not.

## Adding Search Widget to action bar

Another most useful feature of action bar is adding widgets to it. For example like adding search widget to action bar. Search widget will be useful when user wants to search for something in the app or across android OS.

Adding search widget involves these steps.

- > Adding the **Search Widget** to action bar action item
- > Defining the **searchable configuration** in the xml
- > Creating the activity to handle search query and display the results
- > Defining the **default searchable** activity and **SEARCH** intent filter in **AndroidManifest.xml** file

10. So first we add the search widget to action bar. Open your **activity\_main\_actions.xml** which is located under **menu** folder and add search widget to search action item as below.

```
android:actionViewClass="android.widget.SearchView"
```

**activity\_main\_actions.xml**

```
<!-- Search Widget -->
<item android:id="@+id/action_search"
      android:icon="@drawable/ic_action_search"
      android:title="@string/action_search"
      android:showAsAction="always"
      android:actionViewClass="android.widget.SearchView"/>
```

11. Create a searchable configuration file under **res** ⇒ **xml** folder named **searchable.xml** (If you don't see xml folder under res, create a new folder with the name xml). See list of [Searchable Configuration](#) options.

**searchable.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
      android:hint="@string/search_hint"
      android:label="@string/app_name" />
```

It is suggested that always assign the values of **android:hint** and **android:label** from strings.xml file only instead of writing the values directly.

Now open the main activity and modify the code in the **onCreateOptionsMenu()** method.

**MainActivity.java**

```
@Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.activity_main_actions, menu);

        // Associate searchable configuration with the SearchView
        SearchManager searchManager = (SearchManager) getSystemService(Context.S
        SearchView searchView = (SearchView) menu.findItem(R.id.action_search)
```

```

        .getActionView();
searchView.setSearchableInfo(searchManager
        .getSearchableInfo(getComponentName()));

    return super.onCreateOptionsMenu(menu);
}

```

12. We need another activity to handle search query and results. Create a new class named **SearchResultsActivity.java** and paste following code. Also create layout file too named **activity\_search\_results.xml**.

Here I just passed the search query to another activity. You have to take appropriate action to display the search results using the query either from SQLite database or making a request to server and getting the results or some other way.

#### activity\_search\_results.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=".MainActivity" >

    <TextView
        android:id="@+id/txtQuery"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</RelativeLayout>

```

#### SearchResultsActivity.java

```

package info.androidhive.actionbar;

import android.app.ActionBar;
import android.app.Activity;
import android.app.SearchManager;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;

public class SearchResultsActivity extends Activity {

    private TextView txtQuery;

    @Override

```

```

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_search_results);

    // get the action bar
    ActionBar actionBar = getActionBar();

    // Enabling Back navigation on Action Bar icon
    actionBar.setDisplayHomeAsUpEnabled(true);

    txtQuery = (TextView) findViewById(R.id.txtQuery);

    handleIntent(getIntent());
}

@Override
protected void onNewIntent(Intent intent) {
    setIntent(intent);
    handleIntent(intent);
}

/**
 * Handling intent data
 */
private void handleIntent(Intent intent) {
    if (Intent.ACTION_SEARCH.equals(intent.getAction())) {
        String query = intent.getStringExtra(SearchManager.QUERY);

        /**
         * Use this query to display search results like
         * 1. Getting the data from SQLite and showing in listview
         * 2. Making webrequest and displaying the data
         * For now we just display the query only
         */
        txtQuery.setText("Search Query: " + query);
    }
}
}

```

13. Finally in the **AndroidManifest.xml** file define the searchable configuration, default searchable activity and the activity performing the search.

**android.app.default\_searchable** – Defines the default searchable activity handle search. You can add this block anywhere in the manifest file either inside **<application>** tag or **<activity>** tag.

**android.app.searchable** – Defines the searchable configuration which was written in **searchable.xml** file

**android.intent.action.SEARCH** – Should be defined as a intent filter for the activity which receives the search query.

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="info.androidhive.actionbar"
    android:versionCode="1"

```

```

android:versionName="1.0" >

<uses-sdk
    android:minSdkVersion="11"
    android:targetSdkVersion="18" />

<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name="info.androidhive.actionbar.MainActivity"
        android:label="@string/app_name">
        <meta-data
            android:name="android.app.default_searchable"
            android:value=".SearchResultsActivity" />

        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>

    <!-- Search results activity -->
    <activity android:name=".SearchResultsActivity"
        android:parentActivityName="info.androidhive.actionbar.MainActivity"
        <intent-filter>
            <action android:name="android.intent.action.SEARCH" />
        </intent-filter>

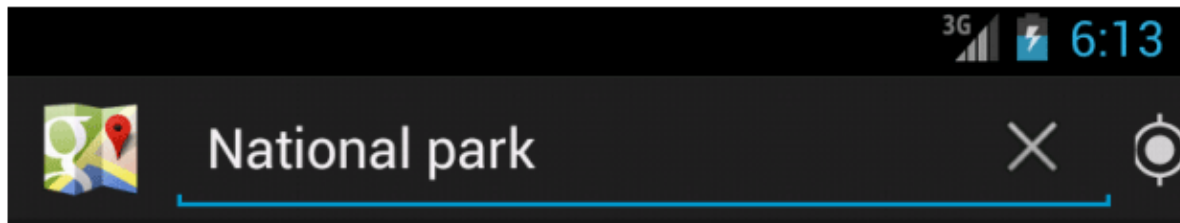
        <meta-data
            android:name="android.app.searchable"
            android:resource="@xml/searchable" />
    </activity>

    <!-- Location found activity -->
    <activity
        android:name="info.androidhive.actionbar.LocationFound"
        android:label="@string/activity_location_found"
        android:parentActivityName="info.androidhive.actionbar.MainActivity"
    </activity>
</application>

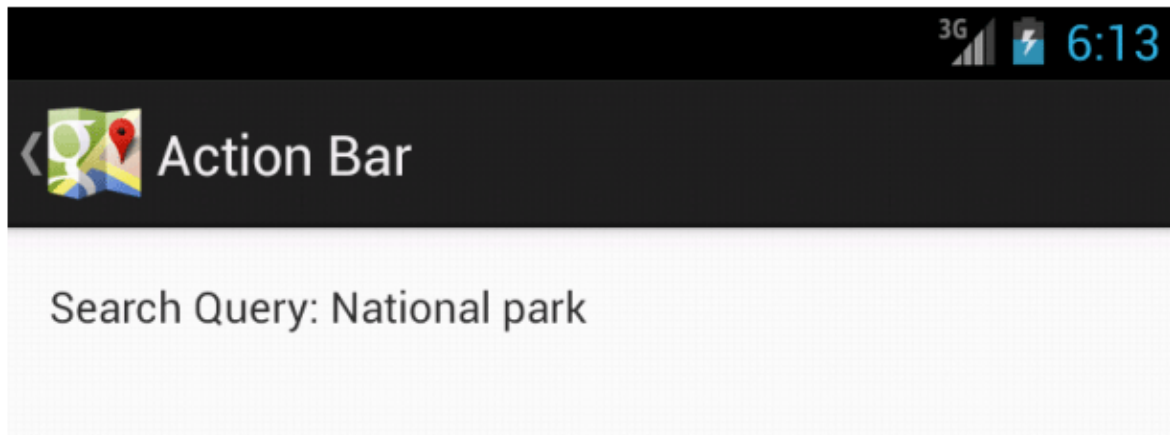
</manifest>

```

## Action Bar adding Search Widget



Action Bar Search Widget



Receiving the search query

[www.androidhive.info](http://www.androidhive.info)

Action bar provides two kinds of navigational options to switch between views. One is Tabbed navigation and other is Spinner Drop down navigation.

## Adding Spinner Drop-down Navigation

Action bar has the inbuilt capability of adding spinner drop down to switch b/w views. This navigation is suggested when your app view switching is not much frequent.

The spinner I am adding here is using a custom list adapter where the list item will have a icon and text like Google Maps App. I gathered all the required icons for the list and placed them under drawable folders with different resolutions.

14. Create a new package named **info.androidhive.actionbar.model** to store the model classes. For spinner list item create a model named **SpinnerNavItem.java** under this package. This model class contains two elements, title and image for the list item.



SpinnerNavItem.java

```
package info.androidhive.actionbar.model;

public class SpinnerNavItem {

    private String title;
    private int icon;

    public SpinnerNavItem(String title, int icon){
        this.title = title;
        this.icon = icon;
    }

}

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

}
```



Search ..

15. Each list item needs a layout file which contains a image and text. Create a xml layout file under **res** ⇒ **layout** folder named **list\_item\_title\_navigation.xml**

list\_item\_title\_navigation.xml

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

    <ImageView android:id="@+id/imgIcon"
        android:layout_width="25dp"
        android:layout_height="25dp"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:src="@drawable/ic_launcher"
        android:layout_marginRight="5dp"
        />

    <TextView android:id="@+id/txtTitle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerVertical="true"
        android:layout_toRightOf="@id/imgIcon"/>

</RelativeLayout>
```

16. Now we need another package to keep our adapter classes. Create a new package with the name **info.androidhive.info.actionbar.adapter**. The adapter class for spinner list I am creating is

## TitleNavigationAdapter.java

TitleNavigationAdapter.java

```
package info.androidhive.info.actionbar.adapter;

import info.androidhive.actionbar.R;
import info.androidhive.actionbar.model.SpinnerNavItem;

import java.util.ArrayList;

import android.app.Activity;
import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.ImageView;
import android.widget.TextView;

public class TitleNavigationAdapter extends BaseAdapter {

    private ImageView imgIcon;
    private TextView txtTitle;
    private ArrayList<SpinnerNavItem> spinnerNavItem;
    private Context context;

    public TitleNavigationAdapter(Context context,
        ArrayList<SpinnerNavItem> spinnerNavItem) {
        this.spinnerNavItem = spinnerNavItem;
        this.context = context;
    }

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

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

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

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        if (convertView == null) {
            LayoutInflater mInflater = (LayoutInflater)
                context.getSystemService(Activity.LAYOUT_INFLATER_SERVICE);
            convertView = mInflater.inflate(R.layout.list_item_title_navigation,
                null);

            imgIcon = (ImageView) convertView.findViewById(R.id.imgIcon);
            txtTitle = (TextView) convertView.findViewById(R.id.txtTitle);
        }
    }
}
```

```

        imgIcon.setImageResource(spinnerNavItem.get(position).getIcon());
        imgIcon.setVisibility(View.GONE);
        txtTitle.setText(spinnerNavItem.get(position).getTitle());
        return convertView;
    }

    @Override
    public View getDropDownView(int position, View convertView, ViewGroup parent) {
        if (convertView == null) {
            LayoutInflater mInflater = (LayoutInflater)
                context.getSystemService(Activity.LAYOUT_INFLATER_SERVICE);
            convertView = mInflater.inflate(R.layout.list_item_title_navigation,
                null);
        }

        imgIcon = (ImageView) convertView.findViewById(R.id.imgIcon);
        txtTitle = (TextView) convertView.findViewById(R.id.txtTitle);

        imgIcon.setImageResource(spinnerNavItem.get(position).getIcon());
        txtTitle.setText(spinnerNavItem.get(position).getTitle());
        return convertView;
    }
}

```

17. Now open the main activity and implement the class from `ActionBar.OnNavigationListener` to handle the spinner select listener.

```

class MainActivity extends Activity implements ActionBar.OnNavigationListener{

```

18. Adding spinner drop down to action bar involves few steps.

- > First hide the action bar title using **setDisplayShowTitleEnabled(false)**
- > Second enable the action bar spinner navigation mode using **setNavigationMode(ActionBar.NAVIGATION\_MODE\_LIST)**
- > Then create a adapter and assign it to action bar using **setListNavigationCallbacks()**

Following is the complete code to enable to action bar spinner navigation

```

MainActivity.java
package info.androidhive.actionbar;

import info.androidhive.actionbar.model.SpinnerNavItem;
import info.androidhive.info.actionbar.adapter.TitleNavigationAdapter;

import java.util.ArrayList;

import android.app.ActionBar;

```

```

import android.app.Activity;
import android.app.SearchManager;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.widget.SearchView;

public class MainActivity extends Activity implements ActionBar.OnNavigationList

    // action bar
    private ActionBar actionBar;

    // Title navigation Spinner data
    private ArrayList<SpinnerNavItem> navSpinner;

    // Navigation adapter
    private TitleNavigationAdapter adapter;

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

        actionBar = getActionBar();

        // Hide the action bar title
        actionBar.setDisplayShowTitleEnabled(false);

        // Enabling Spinner dropdown navigation
        actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_LIST);

        // Spinner title navigation data
        navSpinner = new ArrayList<SpinnerNavItem>();
        navSpinner.add(new SpinnerNavItem("Local", R.drawable.ic_location));
        navSpinner.add(new SpinnerNavItem("My Places", R.drawable.ic_my_places));
        navSpinner.add(new SpinnerNavItem("Checkins", R.drawable.ic_checkin));
        navSpinner.add(new SpinnerNavItem("Latitude", R.drawable.ic_latitude));

        // title drop down adapter
        adapter = new TitleNavigationAdapter(getApplicationContext(), navSpinner

        // assigning the spinner navigation
        actionBar.setListNavigationCallbacks(adapter, this);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        ....
    }

    /**
     * On selecting action bar icons
     * */
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        ...
    }

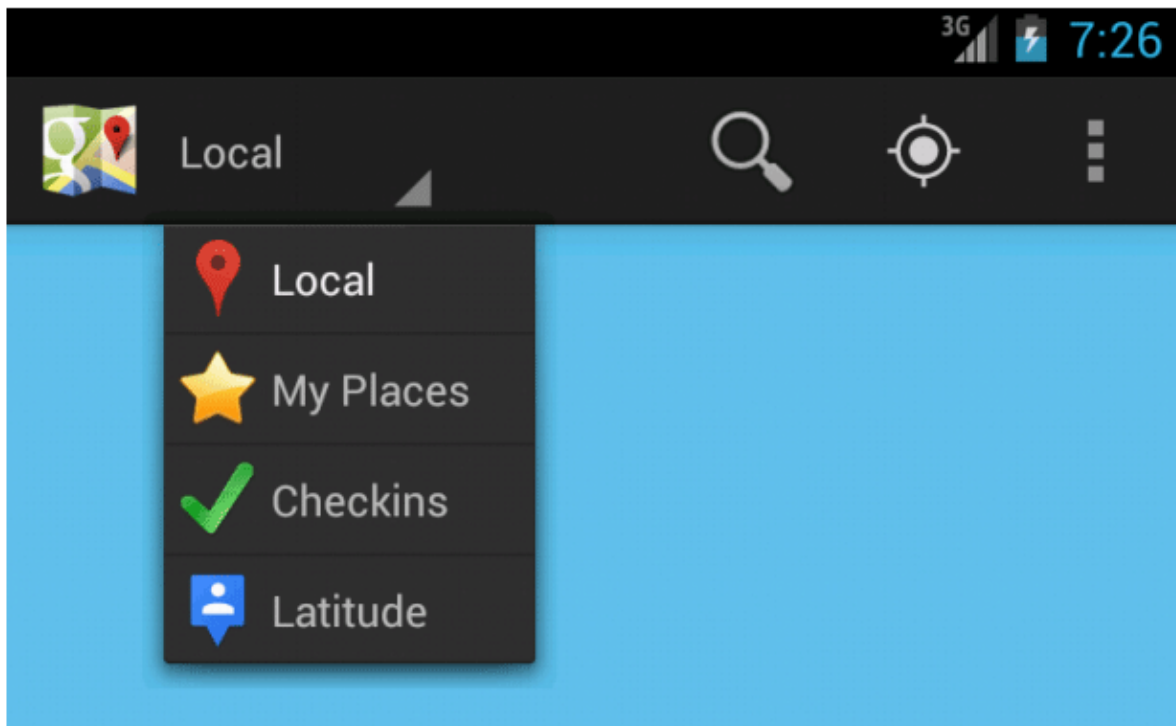
```

```

/**
 * ActionBar navigation item select listener
 * */
@Override
public boolean onNavigationItemSelected(int itemPosition, long itemId) {
    // Action to be taken after selecting a spinner item
    return false;
}
}

```

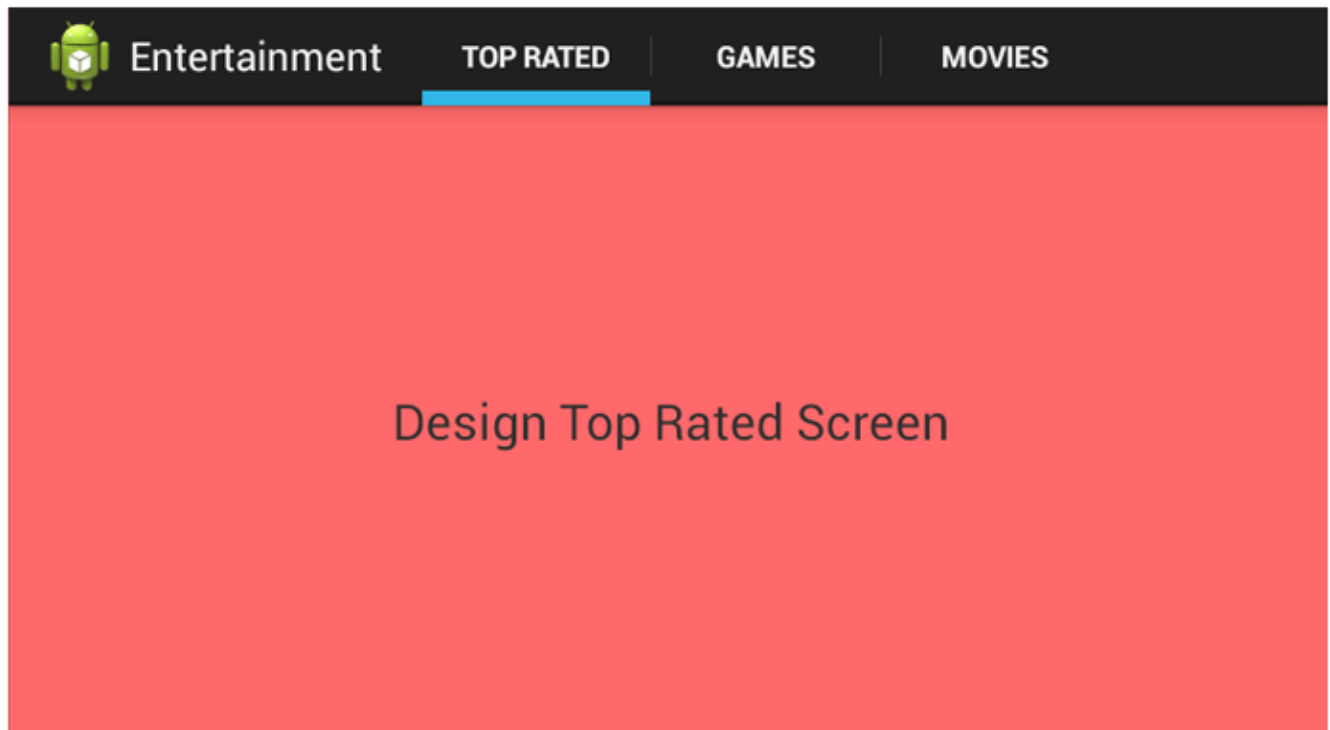
Action Bar adding spinner navigation



www.androidhive.info

## Adding Tab Navigation

Another navigation action bar provides is tabbed navigation. This way is suggested when the navigation is too frequent between views. I already explained tab navigation in my previous article [Android Tab Layout with Swipeable Views](#) in a detailed manner. Follow the tutorial to enable the tabbed navigation with swipe gestures.



## Adding Custom Views – Refresh and Loading

Action bar also provides an option to add a custom view apart from widgets. You can see lot of apps uses refresh icon and progress loader views while making a HTTP requests to sync the data. That can be done in a easy way.

**19.** First we will create a custom view which we want to show in the action bar. We will create a simple layout with progress bar. Create a new xml layout file under **res** ⇒ **layout** folder named **action\_progressbar.xml**

```
action_progressbar.xml
<?xml version="1.0" encoding="utf-8"?>
<ProgressBar xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/progressBar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">

</ProgressBar>
```

20. As you can see we have kept a refresh action item in our action bar. Open your main activity class and do the following changes.

What we are doing here is, when user selects the refresh action item we need to get the data from server using Async task. Before sending a request, using **onPreExecute()** method we set a custom layout in refresh action item. In **doInBackground()** method we send the request to server and once we get the result from server, in **onPostExecute()** method we remove the custom progress layout.

#### MainActivity.java

```
package info.androidhive.actionbar;

import info.androidhive.actionbar.model.SpinnerNavItem;
import info.androidhive.info.actionbar.adapter.TitleNavigationAdapter;

import java.util.ArrayList;

import android.app.ActionBar;
import android.app.Activity;
import android.app.SearchManager;
import android.content.Context;
import android.content.Intent;
import android.os.AsyncTask;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.widget.SearchView;

public class MainActivity extends Activity implements
    ActionBar.OnNavigationListener {

    // action bar
    private ActionBar actionBar;

    // Refresh menu item
    private MenuItem refreshMenuItem;

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

        actionBar = getActionBar();
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        ....
    }

    /**
     * On selecting action bar icons
     * */
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
```

```

// Take appropriate action for each action item click
switch (item.getItemId()) {
    ...
    case R.id.action_refresh:
        // refresh
        refreshMenuItem = item;
        // load the data from server
        new SyncData().execute();
        return true;
    default:
        return super.onOptionsItemSelected(item);
}

}

/**
 * Async task to load the data from server
 * **/
private class SyncData extends AsyncTask<String, Void, String> {
    @Override
    protected void onPreExecute() {
        // set the progress bar view
        refreshMenuItem.setActionView(R.layout.action_progressbar);

        refreshMenuItem.expandActionView();
    }

    @Override
    protected String doInBackground(String... params) {
        // not making real request in this demo
        // for now we use a timer to wait for sometime
        try {
            Thread.sleep(3000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        return null;
    }

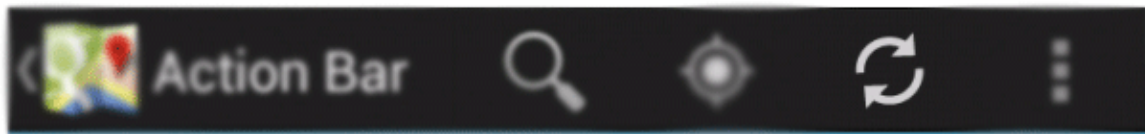
    @Override
    protected void onPostExecute(String result) {
        refreshMenuItem.collapseActionView();
        // remove the progress bar view
        refreshMenuItem.setActionView(null);
    }
};
}

```

Run the project and test the progress loader once. If you can't see the refresh icon, rotate the emulator into landscape mode. (Use **Ctrl + F11** to rotate the emulator)



## Action Bar adding custom layout - Refresh / Progress bar



www.androidhive.info

There is a lot more customization can be done to action bar. I'll cover those topics in my upcoming tutorials.

Share this article on

247

Like

12

Tweet

81

g+1

1

### Report a Bug in this article

(If you find any error either in code or content please help me in improvising the content)

Email address / Full name

Give brief description about the bug...

Report

---

Follow @RaviTamada 2,444 followers

Like 15k Tweet 440 g+1 1.1k

Subscribe for latest updates

143658

Enter your email here

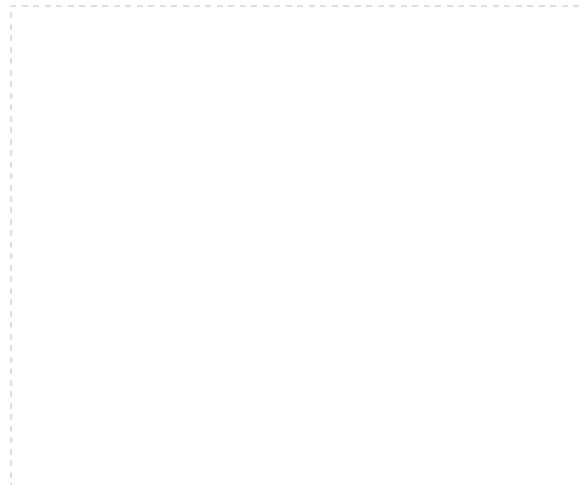
SUBSCRIBE

Advertise



Advertise Here

Advertise Here



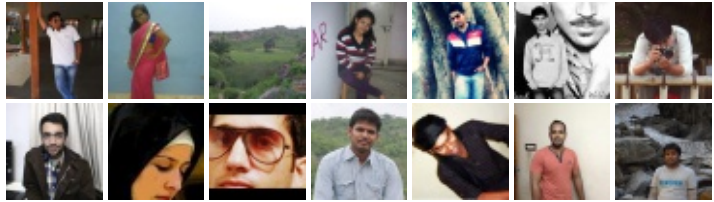
advertise here



AndroidHive

Like

15,232 people like [AndroidHive](#).




Facebook social plugin

## Tag Cloud

Action Bar   Adapter   Animation   API   Apps  
Async   Beginner   Camera   Dashboard  
Database   facebook   Fragments   GCM  
Gestures   Google   Google Plus   GPS   Grid  
HTTP   Intermediate   json   List View   Maps  
MySQL   Navigation Drawer   PHP   Pinch  
Quick Tips   REST   sessions   Slim   Spinner  
SQLite   Swipe   Tab View   Twitter   UI  
Video   View Pager   xml

Ravi Tamada

google.com/+RaviTamada

 Follow

2,896 followers

## MOST POPULAR

- 1 Android SQLite Database Tutorial - 646,484 views
- 2 Android Custom ListView with Image and Text - 523,883 views
- 3 Android JSON Parsing Tutorial - 514,244 views
- 4 How to connect Android with PHP, MySQL - 498,778 views
- 5 Android Push Notifications using

## NETWORK

### DESIGN

[design.androidhive.info](http://design.androidhive.info)

### TIPS

[tips.androidhive.info](http://tips.androidhive.info)

### ERRORS

[errors.androidhive.info](http://errors.androidhive.info)

Google Cloud Messaging (GCM),  
PHP and MySQL - 404,964  
views

6 Android Login and Registration  
with PHP, MySQL and SQLite -  
354,775 views

7 Android Tab Layout Tutorial -  
342,271 views

8 Android XML Parsing Tutorial -  
266,060 views

9 Android Login and Registration  
Screen Design - 261,254 views

10 Android GPS, Location Manager  
Tutorial - 241,388 views

DOWNLOAD  
download.androidhive.in

143658 11909  
subscribers downloads

REQUEST TUTORIAL

Email / Name \*

Your email id or name

Description \*

Brief description about your  
thought. (Please don't spam)

SEND



Creative Commons Attribution 3.0 Unported  
License