1 The Action Bar

1.1 What is the Action Bar?

The *action bar*, introduced in Honeycomb (API 11) is a title bar that includes:

- The application icon
- The activity title
- A set of user-selectable actions (optional)
- A set of user-selectable navigation modes (optional)

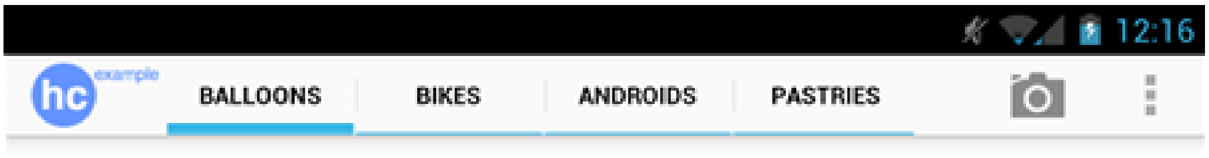

Figure 1: Action Bar

1.2 Enabling the Action Bar

Android automatically displays an action bar on an API 11+ system if the `<uses-sdk>` element of your applications manifest:

- Sets `minSdkVersion` to 11 or later, *or*
- Sets `targetSdkVersion` to 11 or later

Either or these settings enable the "holographic look and feel" introduced in Honeycomb, which includes action bar support.

- If neither `minSdkVersion` nor `targetSdkVersion` are set to 11+, then an API 11+ system renders the app in a legacy theme, without action bar support.

With the action bar enabled, legacy option menu items appear automatically in the action bar's *overflow menu*. You reveal the overflow menu with:

- The hardware Menu button (if present), *or*
- An additional button in the action bar (for devices without a hardware Menu button)

When the activity first starts, the system populates the action bar and overflow menu by calling `onCreateOptionsMenu()` for your activity.

1.3 Removing the Action Bar

Occasionally, you might want to disable the action bar for an activity.

- For example, you might want a game or movie player app to take over the entire screen.

To disable the action bar for an activity, either:

- Set the activity theme to `Theme.Holo.NoActionBar` in the manifest:

```
<activity android:theme="@android:style/Theme.Holo.NoActionBar">
```

- Programmatically disable the action bar:

```
ActionBar actionBar = getActionBar();
actionBar.hide();
```

1.4 Adding Action Items

To display an option menu item as an action item, in the menu resource file add `android:showAsAction="ifRoom"` to the `<item>` element.

- The device will display the item if there is room available in the action bar, otherwise the item appears in the overflow menu.
- Devices running API 10 or earlier ignore the `showAsAction` attribute.

If your menu item supplies both a title and an icon, the action item shows only the icon by default.

- To display the text title, add `withText` to the `android:showAsAction` attribute. For example:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_save"
          android:icon="@drawable/ic_menu_save"
          android:title="@string/menu_save"
          android:showAsAction="ifRoom|withText" />
</menu>
```

Note
The `withText` value is a hint to the action bar. The action bar will show the title if possible, but might not if an icon is available and the action bar is constrained for space.

You can also declare an item to "always" appear as an action item, instead of being placed in the overflow menu when space is limited. In most cases, you should not force an item to appear in the action bar by using the "always" value.

1.5 ICS Split Action Bars

API 14 introduced *split action bar* support.

- Enabling split action bar allows a device with a narrow screen to display a separate action bar at the bottom of the screen containing all action items.

You enable split action bar in the manifest add `uiOptions="splitActionBarWhenNarrow"`:

- To an `<activity>` element to enable split action bar for that activity, or
- To the `<application>` element to enable split action bar for all activities

Note
Devices on API 13 or earlier ignore the `uiOptions` attribute.

1.6 Using the App Icon for Navigation

You can enable the application icon — which appears in the action bar on the left side — to behave as an action item. If enabled, your app should respond by:

- Going to the application "home" activity, or
- Navigating "up" the application's structural hierarchy

When the user taps the app icon, the system calls your activity's `onOptionsItemSelected()` method. The `MenuItem` passed as an argument has an ID of `android.R.id.home`.

Note
The app icon was enabled by default in APIs 11-13. API 14 disables it be default. To enable it in API 14 or later:

```
if (Build.VERSION.SDK_INT > Build.VERSION_CODES.ICE_CREAM_SANDWICH) {
        getActionBar().setHomeButtonEnabled(true);
}
```

1.7 Implementing the "Go Home" App Icon Feature

If you decide to implement the "Go Home" feature for the app icon, Google recommends the following best practices.

- Include the `FLAG_ACTIVITY_CLEAR_TOP` flag in the `Intent` that starts the activity.

  If the home activity you're starting already exists in the current task, this flag destroys all activities on top of it and brings the home activity to the front. Without this flag, you could create a long stack of activities in the current task with multiple instances of the home activity.

- If the user can enter the current activity from another application, you might also want to add the `FLAG_ACTIVITY_NEW_TASK` flag.

  This prevents the new home activity from being added to the other application's task. The system either starts a new task with your new activity as the root activity or, if an existing task exists in the background with an instance of that activity, then that task is brought forward and the target activity receives `onNewIntent()`.

The following example incorporates both of these features:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case android.R.id.home:
            // app icon in action bar clicked; go home
            Intent intent = new Intent(this, HomeActivity.class);
            intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP | Intent.FLAG_ACTIVITY_NEW_TASK) ↩
                ;
            startActivity(intent);
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

1.8 Implementing "Up" Navigation with the App Icon

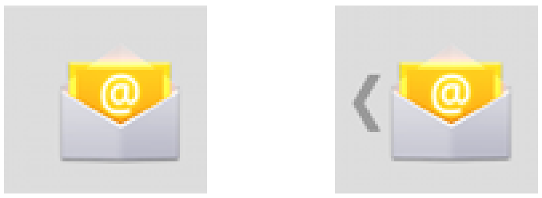

Figure 2: Adding an Up Indicator

You can enable the action bar icon to offer "up" navigation, which should take the user one step up in your application's structural hierarchy.

- "Back" navigation takes the user to the previous activity in the task — even if that is in a different app.

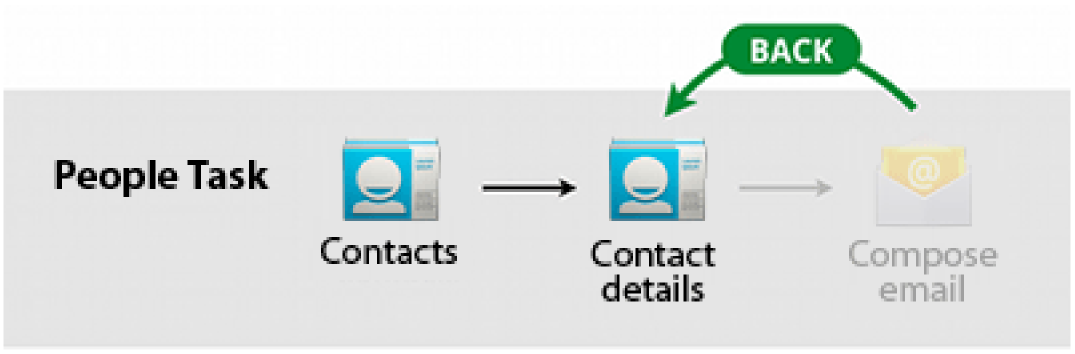

Figure 3: Back navigation

- Up navigation goes up the structural hierarchy of the current application.

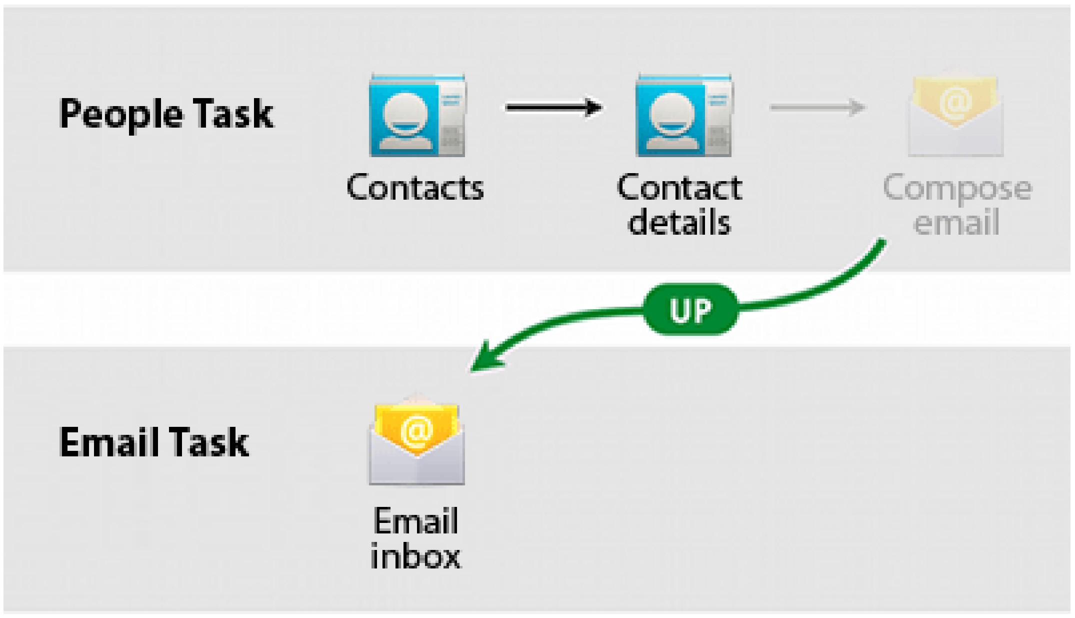

Figure 4: Up navigation

Tip
For more about how up and back navigation differ, read Android Design's Navigation guide.

To request Android to display an "up" indicator on the app icon:

```
ActionBar actionBar = getActionBar();
actionBar.setDisplayHomeAsUpEnabled(true);
```

When tapped, Android invokes your activity's `onOptionsItemSelected()` method with the `android.R.id.home` ID, exactly as discussed previous.

1.9 Adding Navigation Tabs

You can display a set of tabs in the action bar to enable navigation between fragments.

- The system adapts the action bar tabs for different screen sizes — placing them in the main action bar when the screen is sufficiently wide, or in a separate bar (known as the *stacked action bar*) when the screen is too narrow

Your layout must include a ViewGroup in which you place each Fragment associated with a tab

- You can supply a `ViewGroup` with a known ID in your activity's layout to host the fragments.
- Alternatively, you can have your fragments fill the entire activity layout by placing each fragment in the default root ViewGroup, which you can refer to with the `android.R.id.content` ID.

To add action bar tabs:

1. Set tab navigation mode on the action bar.

   ```
   ActionBar actionBar = getActionBar();
   actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);
   ```

2. Implement the `ActionBar.TabListener` interface.

   A `TabListener` receives event callbacks from the action bar as tabs are deselected, selected, and reselected. A `FragmentTransaction` is provided to each of these callbacks; if any operations are added to it, it will be committed at the end of the full tab switch operation. This lets tab switches be atomic without the app needing to track the interactions between different tabs.

   ```
   public interface TabListener {
       public void onTabSelected(Tab tab, FragmentTransaction ft);
       public void onTabUnselected(Tab tab, FragmentTransaction ft);
       public void onTabReselected(Tab tab, FragmentTransaction ft);
   }
   ```

3. For each tab you want to add, instantiate an `ActionBar.Tab`.

   - Set the `ActionBar.TabListener` by calling `setTabListener()`.
   - Set the tab's title and/or icon with `setText()` and/or `setIcon()`.

4. Add each tab to the action bar by calling `ActionBar.addTab()`.

See the Fragment Tabs demo in the Android APIs demo for an example of implementing tab navigation to the action bar.

1.10 Adding Drop-down Navigation

As another mode of navigation (or filtering) within your activity, the action bar offers a built in drop-down list.

- For example, the drop-down list can offer different modes by which content in the activity is sorted.

The basic procedure to enable drop-down navigation is:

1. Create a `SpinnerAdapter` that provides the list of selectable items for the drop-down and the layout to use when drawing each item in the list.

2. Implement `ActionBar.OnNavigationListener` to define the behavior that occurs when the user selects an item from the list.

   ```
   public interface OnNavigationListener {
       public boolean onNavigationItemSelected(int itemPosition, long itemId);
   }
   ```

3. Enable navigation mode for the action bar with `setNavigationMode()`. For example:

   ```
   ActionBar actionBar = getActionBar();
   actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_LIST);
   ```

   Note
   You should perform this during your activity's `onCreate()` method.

4. Set the callback for the drop-down list with `setListNavigationCallbacks()`. For example:

   ```
   actionBar.setListNavigationCallbacks(mSpinnerAdapter, mNavigationCallback);
   ```

   This method takes your `SpinnerAdapter` and `ActionBar.OnNavigationListener`.