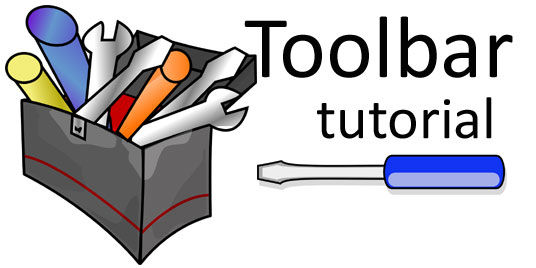
Using Toolbars in your apps

* Written by  Clive

Toolbars



A Toolbar is similar to an ActionBar.

It’s a ViewGroup so you can place it anywhere in your layout. You can even replace the ActionBar with a Toolbar.

Toolbar’s are also more flexible. You can modify its size, colour, position, etc. You can also add logos, labels, navigation icons and other views to it.

With the release of Android 5.0 and material design, Android has updated the AppCompat support libraries so that we can use Toolbars on devices running API Level 7 and up.

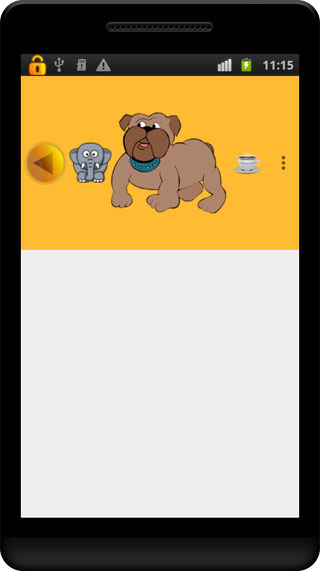
Our tutorial app

We’ll show you how to replace the Action Bar with the Toolbar. We also show you how to display a standalone Toolbar.

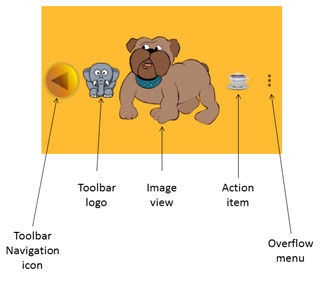
We’ve compiled our app using SDK 21. You should be able to build it with a minimum SDK of 7; we’ve set ours to 10.

We’re using the updated AppCompat support library (AppCompat v21) which gives us access to the Toolbar widget so that we can use it on pre-Lollipop devices (from API Level 7 and up).

Here’s a screenshot of our custom Toolbar in place of the ActionBar:



Our customized Toolbar replaces the activity’s ActionBar

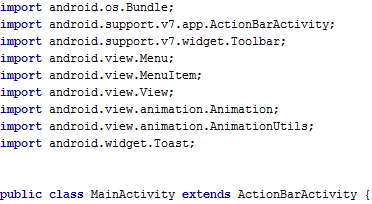


You have more control over the look and functionality of the Toolbar

You can do it your way but this is how we did it

Extend the ActionBarActivity class

Let your activity extend the ActionBarActivity class:



Let your Activity extend the ActionBarActivity class

Use the AppCompat theme

You must use the **Theme.AppCompat** as your base theme:

Android Toolbar extend ActionBarActivity theme

Use the Theme.AppCompat theme or any of its variants

We’ve used the *light* variation with no action bar as we’re replacing the ActionBar with the Toolbar.

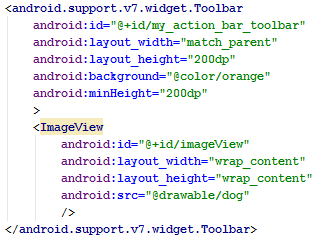
It’s your choice: Choose where you want to display the Toolbar

Take over the bar: Replacing the ActionBar with a Toolbar

Put it out of action: Disable the ActionBar

You need to disable the ActionBar if you want to replace it with a Toolbar. We’ve done this in our *styles.xml* file where we set our theme to use the *NoActionBar* variation of the AppCompat Light theme.

Get your toolbox: Create your Toolbar widget in an XML layout file



Define the Toolbar in a layout file

A quick explanation

Use the support library’s Toolbar widget element and give it an ID so that we can access it later in code.

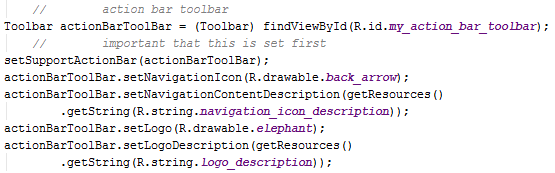
The width is set to match the parent’s so that it will span the width of the screen. The height is set to 200dp to accommodate the *dog* image which is 200dp high.

We’ve set the background colour to *orange*.

The minimum height is set to 200dp. This ensures that all the displayed items are vertically centred.

Finally we include an image view referencing a drawable resource for the image to display.

Get to work: Set up the Toolbar as the ActionBar in code



Get a reference to the Toolbar and then set the icon and logo

A quick explanation

We get a reference to the Toolbar and then call *setSupportActionBar()* to set the Toolbar as the ActionBar. Note that you should call this first if you’re going to be adding logos and navigation icons else they won’t show.

Then we call *setNavigationIcon()* to display our *back arrow image* which we’ll use to navigate back. You must include a *description* for the navigation icon which will be used to describe the logo to visually impaired users

We also include a *logo* for the Toolbar by calling *setLogo()*. We also include a description for it.

Use the *onOptionsItemSelected()* method to handle the options menu item selections.

You can also use the *setNavigationOnClickListener()* to listen for when the navigation icon is pressed, using its onClick() method to deal with the navigation. We’ll show you how when we discuss the standalone Toolbar later.

Call *setTitle(null)* in the *onCreate()* method if the title shows and you don't want it to.

Including a menu in the Toolbar

As this Toolbar is replacing our ActionBar, we chose to include a menu action item as well as the menu’s overflow button.

We use the *onCreateOptionsMenu()* method to inflate the menu resource.

Handling the Toolbar’s item selections

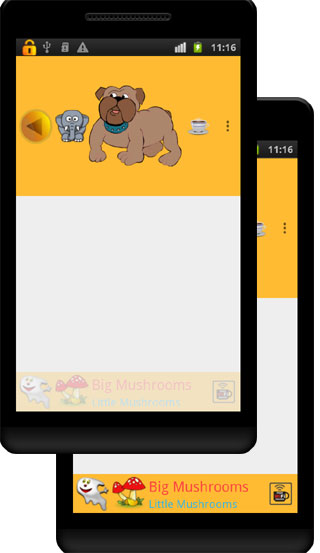
We use the *onOptionsItemSelected()* method to handle the menu item selections.

We display an appropriate Toast message if either of the menu items is selected.

Selecting the action item makes the standalone Toolbar visible. We then use an animation to fade it in.

Going solo: Displaying a standalone Toolbar

You can display a standalone Toolbar anywhere on the screen. We show ours at the bottom. Here’s a screenshot:

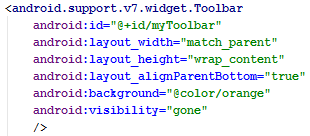


Pressing the Action item fades in a standalone Toolbar at the bottom of the screen. Pressing the Ghost navigation icon on the left sets the visibility of the Toolbar to GONE

You don’t have to disable the **ActionBar** for the standalone option.

Create your Toolbar widget in an XML layout file

Define the Toolbar in a layout file:



We define the standalone Toolbar in our activity\_main.xml layout file

A quick explanation

Use the support library’s Toolbar widget element and give it an ID so that we can access it later in code.

The width is set to match the parent’s so that it will span the width of the screen. The height is set to *wrap\_content* so it will only be as high as the largest object it contains.

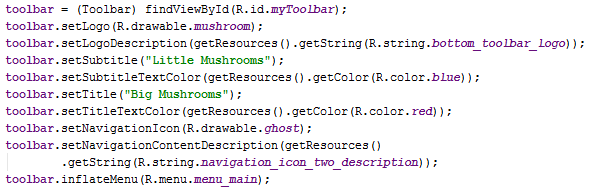
We use the *layout\_alignParentBottom* attribute to display the **Toolbar** at the bottom of the screen.

We’ve set the background colour to *orange* using the *background* attribute.

Finally we set the Toolbar’s visibility to GONE. This ensures that it’s not visible and nor does it take up any screen space.

Putting it together: Set up the standalone Toolbar in code

We first declare a Toolbar field, *toolbar*. Then we get a reference to the Toolbar widget and apply our custom settings:



Get a reference to the Toolbar and then set the logo, title, subtitle, navigation icon and inflate the menu

A quick explanation

We set the logo to display, passing a drawable reference as a parameter. We also include a description for the logo which will be used to describe the logo to visually impaired users.

We then set the title and subtitle as well as their text colours.

We set the navigation icon, referencing a drawable resource to display. We also include a description for the icon which will be used to describe the icon to visually impaired users.

Finally, we call *inflateMenu()* to inflate a menu resource file. This will display the action item in the *Toolbar*.

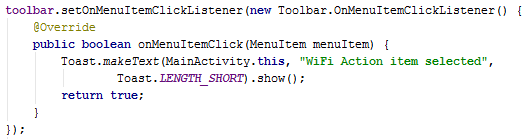
Lend me your ears: Listen for the action

We include two listeners:

* One listens for the menu item selections
* One listens for the navigation icon selections

Listening for menu item selections

We need to include the Toolbar class’s *setOnMenuItemClickListener()* method to listen for when the menu’s Action item is selected:



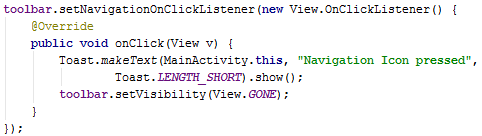
Use the setOnMenuItemClickListener() method to listen for menu item selections

A quick explanation

We attach the listener to our standalone Toolbar. It listens for when a menu item is selected, triggering its *onClick()* method. Here we include a Toast message to display when the Action item is selected.

Listening for when the navigation item is selected

We include the Toolbar class’s *setNavigationOnClickListener()* method to listen for when the navigation icon is selected:



Use the setNavigationOnClickListener () method to listen for when the navigation icon is selected

A quick explanation

We attach the listener to our standalone Toolbar. It listens for when the navigation icon is selected, triggering its *onClick()* method. Here we include a Toast message which is displayed when the navigation icon is selected.

We also set the Toolbar’s visibility to GONE, making it invisible, and ensuring that it doesn’t take up any screen space.

Check out [Chris Bane’s blog post on using AppCompat v21 and the Toolbar](http://android-developers.blogspot.com/2014/10/appcompat-v21-material-design-for-pre.html)

You may also be interested in our tutorials:

* [Using menus in your apps](http://101apps.co.za/index.php/articles/using-menus-in-your-apps-a-tutorial.html)
* [Using Android’s Action Bar](http://101apps.co.za/index.php/articles/using-android-s-action-bar.html)

I hope that you have found this tutorial helpful.

This project was created using [Android Studio](http://www.amazon.com/Android-Studio-How-guide-tutorial-ebook/dp/B00HZ1O78S/ref=sr_1_2?ie=UTF8&qid=1393066467&sr=8-2&keywords=android+studio).

You can download the project files here [Download icon](http://101apps.co.za/downloads.html)

Are you using Eclipse or another IDE? [Here's how you can use this project's Android Studio files](http://101apps.co.za/articles/importing-android-studio-projects-into-eclipse.html).

Related items

* [Open Images Gold: Version 6 help](http://101apps.co.za/index.php/item/184-open-images-gold-version-6-help.html)
* [Free Clip Art Gold, version 2.1 upgrade](http://101apps.co.za/index.php/item/183-free-clip-art-gold-version-2-1-upgrade.html)
* [Android Image editing app](http://101apps.co.za/index.php/applications/android-image-editing-app.html)
* [Smart Photo editor: The free, professional image editor](http://101apps.co.za/index.php/articles/smart-photo-editor-the-free-professional-image-editor.html)
* [Smart Photo Editor Help](http://101apps.co.za/index.php/articles/smart-photo-editor-help.html)