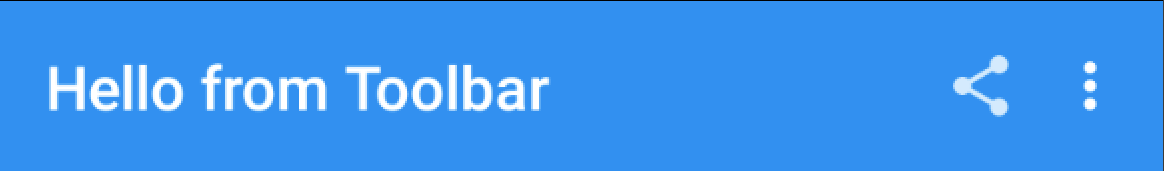
**Using the App ToolBar**

**Overview**

ToolBar was introduced in Android Lollipop, API 21 release and is the spiritual successor of the [ActionBar](http://guides.codepath.com/android/Defining-The-ActionBar). It's a ViewGroupthat can be placed anywhere in your XML layouts. ToolBar's appearance and behavior can be more easily customized than the ActionBar.



ToolBar works well with apps targeted to API 21 and above. However, Android has updated the AppCompat support libraries so the ToolBar can be used on lower Android OS devices as well. In AppCompat, ToolBar is implemented in theandroid.support.v7.widget.Toolbar class.

There are two ways to use Toolbar:

1. Use a Toolbar as an Action Bar when you want to use the existing ActionBar facilities (such as menu inflation and selection, ActionBarDrawerToggle, and so on) but want to have more control over its appearance.
2. Use a standalone Toolbar when you want to use the pattern in your app for situations that an Action Bar would not support; for example, showing multiple toolbars on the screen, spanning only part of the width, and so on.

**ToolBar vs ActionBar**

The ToolBar is a generalization of the [ActionBar system](http://guides.codepath.com/android/Defining-The-ActionBar). The key differences that distinguish the ToolBar from the ActionBarinclude:

* ToolBar is a View included in a layout like any other View
* As a regular View, the toolbar is easier to position, animate and control
* Multiple distinct ToolBar elements can be defined within a single activity

Keep in mind that you can also configure any ToolBar as an Activity’s ActionBar, meaning that your standard options menu actions will be display within.

Note that the ActionBar continues to work and if **all you need is a static bar at the top** that can host icons and a back button, then you can safely continue to use ActionBar.

**Using ToolBar as ActionBar**

To use Toolbar as an ActionBar, first ensure the AppCompat-v7 support library is added to your application build.gradle(Module:app) file:

dependencies **{**

**...**

compile 'com.android.support:appcompat-v7:23.1.0'

**}**

Second, let's disable the theme-provided ActionBar. The easiest way is to have your theme extend fromTheme.AppCompat.NoActionBar (or the light variant) within the res/styles.xml file:

<resources>

*<!-- Base application theme. -->*

<style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">

</style>

</resources>

Now you need to add a Toolbar to your Activity layout file. One of the biggest advantages of using the Toolbar widget is that you can place the view anywhere within your layout. Below we place the toolbar at the top of a LinearLayout like the standard ActionBar:

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:fitsSystemWindows="true"

android:orientation="vertical">

<android.support.v7.widget.Toolbar

android:id="@+id/toolbar"

android:minHeight="?attr/actionBarSize"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

app:titleTextColor="@android:color/white"

android:background="?attr/colorPrimary">

</android.support.v7.widget.Toolbar>

*<!-- Layout for content is here. This can be a RelativeLayout -->*

</LinearLayout>

**Note:** You'll want to add android:fitsSystemWindows="true" ([learn more](https://medium.com/google-developers/why-would-i-want-to-fitssystemwindows-4e26d9ce1eec)) to the parent layout of the Toolbar to ensure that the height of the activity is calculated correctly.

As Toolbar is just a ViewGroup and can be **styled and positioned like any other view**. Note that this means if you are in aRelativeLayout, you need to ensure that all other views are positioned below the toolbar explicitly. The toolbar is not given any special treatment as a view.

Next, in your Activity or Fragment, set the Toolbar to act as the ActionBar by calling the setSupportActionBar(Toolbar)method:

**Note:** When using the support library, make sure that you are importing android.support.v7.widget.Toolbar and notandroid.widget.Toolbar.

**import** android.support.v7.app.AppCompatActivity**;**

**import** android.support.v7.widget.Toolbar**;**

**public** **class** **MyActivity** **extends** AppCompatActivity **{**

@Override

**protected** **void** onCreate**(**Bundle savedInstanceState**)** **{**

**super.**onCreate**(**savedInstanceState**);**

setContentView**(**R**.**layout**.**activity\_my**);**

*// Find the toolbar view inside the activity layout*

Toolbar toolbar **=** **(**Toolbar**)** findViewById**(**R**.**id**.**toolbar**);**

*// Sets the Toolbar to act as the ActionBar for this Activity window.*

*// Make sure the toolbar exists in the activity and is not null*

setSupportActionBar**(**toolbar**);**

**}**

*// Menu icons are inflated just as they were with actionbar*

@Override

**public** **boolean** onCreateOptionsMenu**(**Menu menu**)** **{**

*// Inflate the menu; this adds items to the action bar if it is present.*

getMenuInflater**().**inflate**(**R**.**menu**.**menu\_main**,** menu**);**

**return** **true;**

**}**

**}**

Next, we need to make sure we have the action items listed within a menu resource file such as res/menu/menu\_main.xmlwhich is inflated above in onCreateOptionsMenu:

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

xmlns:app="http://schemas.android.com/apk/res-auto">

<item

android:id="@+id/miCompose"

android:icon="@drawable/ic\_compose"

app:showAsAction="ifRoom"

android:title="Compose">

</item>

<item

android:id="@+id/miProfile"

android:icon="@drawable/ic\_profile"

app:showAsAction="ifRoom|withText"

android:title="Profile">

</item>

</menu>

For more details about action items in the Toolbar including how to setup click handling, refer to our [ActionBar guide](http://guides.codepath.com/android/Defining-The-ActionBar#adding-action-items). The above code results in the toolbar fully replacing the ActionBar at the top:



From this point on, all menu items are displayed in your Toolbar, populated via the standard options menu callbacks.

**Reusing the Toolbar**

In many apps, the same toolbar can be used across multiple activities or in [alternative layout resources](http://guides.codepath.com/android/Understanding-App-Resources#providing-alternate-resources) for the same activity. In order to easily reuse the toolbar, we can leverage the [layout include tag](http://developer.android.com/training/improving-layouts/reusing-layouts.html) as follows. First, define your toolbar in a layout file in res/layout/toolbar\_main.xml:

<android.support.v7.widget.Toolbar

xmlns:android="http://schemas.android.com/apk/res/android"

android:id="@+id/toolbar"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:background="?attr/colorPrimary"/>

Next, we can use the <include /> tag to load the toolbar into our activity 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:fitsSystemWindows="true"

android:orientation="vertical">

*<!-- Load the toolbar here -->*

<include

layout="@layout/toolbar\_main"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"/>

*<!-- Rest of content for the activity -->*

</LinearLayout>

This allows us to create a consistent navigation experience across activities or configuration changes.

**Styling the Toolbar**

The Toolbar can be customized in many ways leveraging various style properties including android:theme,app:titleTextAppearance, app:popupTheme. Each of these can be mapped to a style. Start with:

<android.support.v7.widget.Toolbar

android:id="@+id/toolbar"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:minHeight="?attr/actionBarSize"

android:background="?attr/colorPrimary"

android:theme="@style/ToolbarTheme"

app:titleTextAppearance="@style/Toolbar.TitleText"

app:popupTheme="@style/ThemeOverlay.AppCompat.Light"

/>

Now, we need to create the custom styles in res/styles.xml with:

*<!-- Base application theme. -->*

<style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">

*<!-- Customize your theme here. -->*

<item name="colorPrimary">@color/colorPrimary</item>

<item name="colorPrimaryDark">@color/colorPrimaryDark</item>

<item name="colorAccent">@color/colorAccent</item>

</style>

<style name="ToolbarTheme" parent="@style/ThemeOverlay.AppCompat.Dark.ActionBar">

*<!-- android:textColorPrimary is the color of the title text in the Toolbar -->*

<item name="android:textColorPrimary">@android:color/holo\_blue\_light</item>

*<!-- actionMenuTextColor is the color of the text of action (menu) items -->*

<item name="actionMenuTextColor">@android:color/holo\_green\_light</item>

*<!-- Tints the input fields like checkboxes and text fields -->*

<item name="colorAccent">@color/cursorAccent</item>

*<!-- Applies to views in their normal state. -->*

<item name="colorControlNormal">@color/controlNormal</item>

*<!-- Applies to views in their activated state (i.e checked or switches) -->*

<item name="colorControlActivated">@color/controlActivated</item>

*<!-- Applied to framework control highlights (i.e ripples or list selectors) -->*

<item name="colorControlHighlight">@color/controlActivated</item>

*<!-- Enable these below if you want clicking icons to trigger a ripple effect -->*

*<!--*

*<item name="selectableItemBackground">?android:selectableItemBackground</item>*

*<item name="selectableItemBackgroundBorderless">?android:selectableItemBackground</item>*

*-->*

</style>

*<!-- This configures the styles for the title within the Toolbar -->*

<style name="Toolbar.TitleText" parent="TextAppearance.Widget.AppCompat.Toolbar.Title">

<item name="android:textSize">21sp</item>

<item name="android:textStyle">italic</item>

</style>

This results in:



**Displaying an App Icon**

In certain situations, we might want to display an app icon within the Toolbar. This can be done by adding this code into theActivity

*// Find the toolbar view and set as ActionBar*

Toolbar toolbar **=** **(**Toolbar**)** findViewById**(**R**.**id**.**toolbar**);**

setSupportActionBar**(**toolbar**);**

*// ...*

*// Display icon in the toolbar*

getSupportActionBar**().**setDisplayShowHomeEnabled**(true);**

getSupportActionBar**().**setLogo**(**R**.**mipmap**.**ic\_launcher**);**

getSupportActionBar**().**setDisplayUseLogoEnabled**(true);**

Next, we need to remove the left inset margin that pushes the icon over too far to the left by adding app:contentInsetStart to the Toolbar:

<android.support.v7.widget.Toolbar

android:id="@+id/toolbar"

app:contentInsetLeft="0dp"

app:contentInsetStart="0dp"

...

>

</android.support.v7.widget.Toolbar>

With that the icon should properly display within the ToolBar as expected.

**Custom Title View**

A Toolbar is just a decorated ViewGroup and as a result, the title contained within can be completely customized by embedding a view within the Toolbar such as:

<android.support.v7.widget.Toolbar

android:id="@+id/toolbar"

android:minHeight="?attr/actionBarSize"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

app:titleTextColor="@android:color/white"

android:background="?attr/colorPrimary">

<TextView

android:id="@+id/toolbar\_title"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Toolbar Title"

android:textColor="@android:color/white"

style="@style/TextAppearance.AppCompat.Widget.ActionBar.Title"

android:layout\_gravity="center"

/>

</android.support.v7.widget.Toolbar>

This means that you can style the TextView like any other. You can access the TextView inside your activity with:

*/\* Inside the activity \*/*

*// Sets the Toolbar to act as the ActionBar for this Activity window.*

Toolbar toolbar **=** **(**Toolbar**)** findViewById**(**R**.**id**.**toolbar**);**

setSupportActionBar**(**toolbar**);**

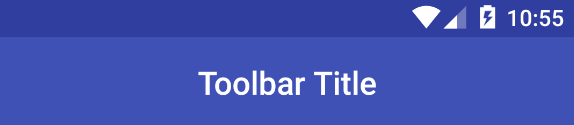
*// Remove default title text*

getSupportActionBar**().**setDisplayShowTitleEnabled**(false);**

*// Get access to the custom title view*

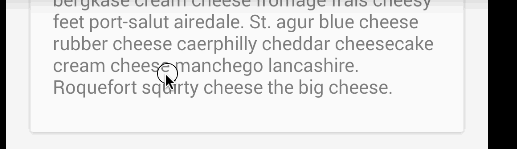
TextView mTitle **=** **(**TextView**)** toolbar**.**findViewById**(**R**.**id**.**toolbar\_title**);**

Note that you **must hide the default title using setDisplayShowTitleEnabled**. This results in:



**Reacting to Scroll**

We can configure the ToolBar to react and change as the page scrolls:



For example, we can have the toolbar hide when the user scrolls down on a list or expand as the user scrolls to the header. There are many effects that can be configured by using the [CoordinatorLayout](http://guides.codepath.com/android/Handling-Scrolls-with-CoordinatorLayout#expanding-and-collapsing-toolbars). First, we need to make sure we add the design support library to our app/build.gradle file:

dependencies **{**

*// ...*

compile 'com.android.support:appcompat-v7:23.1.0'

compile 'com.android.support:recyclerview-v7:23.1.0'

compile 'com.android.support:design:23.1.0'

**}**

Next, inside the activity layout XML such as res/layout/activity\_main.xml, we need to setup our coordinated layout with aToolbar and a scrolling container such as a RecyclerView:

*<!-- CoordinatorLayout is used to create scrolling and "floating" effects within a layout -->*

*<!-- This is typically the root layout which wraps the app bar and content -->*

<android.support.design.widget.CoordinatorLayout

android:id="@+id/main\_content"

xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

*<!-- AppBarLayout is a wrapper for a Toolbar in order to apply scrolling effects. -->*

*<!-- Note that AppBarLayout expects to be the first child nested within a CoordinatorLayout -->*

<android.support.design.widget.AppBarLayout

android:id="@+id/appBar"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:theme="@style/ThemeOverlay.AppCompat.ActionBar">

*<!-- Toolbar is the actual app bar with text and the action items -->*

<android.support.v7.widget.Toolbar

android:id="@+id/toolbar"

android:layout\_width="match\_parent"

android:layout\_height="?attr/actionBarSize"

android:background="?attr/colorPrimary"

app:layout\_scrollFlags="scroll|enterAlways" />

</android.support.design.widget.AppBarLayout>

*<!-- This could also be included from another file using the include tag -->*

*<!-- i.e `res/layout/content\_main.xml` -->*

*<!-- `app:layout\_behavior` is set to a pre-defined standard scrolling behavior -->*

<android.support.v7.widget.RecyclerView

android:id="@+id/my\_recycler\_view"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:clipToPadding="false"

app:layout\_behavior="@string/appbar\_scrolling\_view\_behavior" />

</android.support.design.widget.CoordinatorLayout>

Of course, the RecyclerView could also be replaced with a FrameLayout which could then allow for fragments to be loaded instead:

<android.support.design.widget.CoordinatorLayout

android:id="@+id/main\_content"

xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

*<!-- AppBarLayout and Toolbar as outlined in previous snippet! -->*

*<!-- FrameLayout can be used to insert fragments to display the content of the screen -->*

*<!-- `app:layout\_behavior` is set to a pre-defined behavior for scrolling -->*

<FrameLayout

android:id="@+id/content"

android:layout\_width="match\_parent"

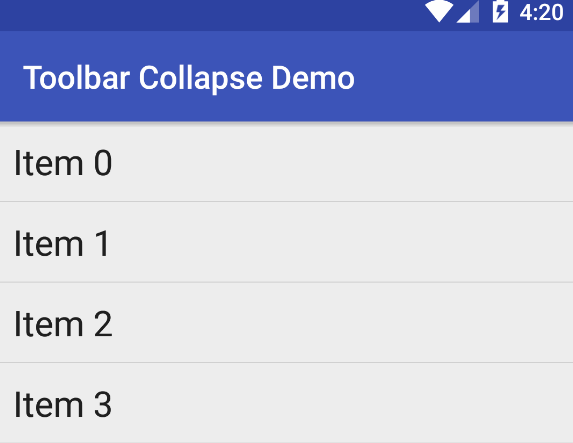
android:layout\_height="match\_parent"

app:layout\_behavior="@string/appbar\_scrolling\_view\_behavior"

/>

</android.support.design.widget.CoordinatorLayout>

This type of layout results in the following:



Refer to the [guide on CoordinatorLayout and AppBarLayout](http://guides.codepath.com/android/Handling-Scrolls-with-CoordinatorLayout#expanding-and-collapsing-toolbars) for additional explanation and specifics. For troubleshooting, refer to this [troubleshooting guide](http://guides.codepath.com/android/Handling-Scrolls-with-CoordinatorLayout#troubleshooting-coordinated-layouts).

**Advanced Scrolling Behavior for Toolbar**

The proper way of reacting to simple scroll behavior is leveraging the [CoordinatorLayout](http://guides.codepath.com/android/Handling-Scrolls-with-CoordinatorLayout#responding-to-scroll-events) built into the [Design Support Library](http://guides.codepath.com/android/Design-Support-Library)as shown in the previous section. However, there are a few other relevant resources around reacting to scrolling events with a more manual approach:

* [Hiding or Showing Toolbar on Scroll](http://rylexr.tinbytes.com/2015/04/27/how-to-hideshow-android-toolbar-when-scrolling-google-play-musics-behavior/) - Great guide on an alternate strategy not requiring the CoordinatorLayout to[replicate the behavior](https://www.youtube.com/watch?time_continue=25&v=z2JyRFpa0Xo) of the "Google Play Music" app. Sample code [can be found here](https://github.com/rylexr/android-show-hide-toolbar/tree/master/app/src/main/java/com/tinbytes/samples/showhidetoolbar).
* [Hiding or Showing Toolbar using CoordinatorLayout](https://mzgreen.github.io/2015/06/23/How-to-hideshow-Toolbar-when-list-is-scrolling%28part3%29/) - Great guide that outlines how to use CoordinatorLayout to hide the Toolbar and the [FAB](http://guides.codepath.com/android/Floating-Action-Buttons) when the user scrolls.

With these methods, your app can replicate any scrolling behaviors seen in common apps with varying levels of difficulty not captured with the method shown above.