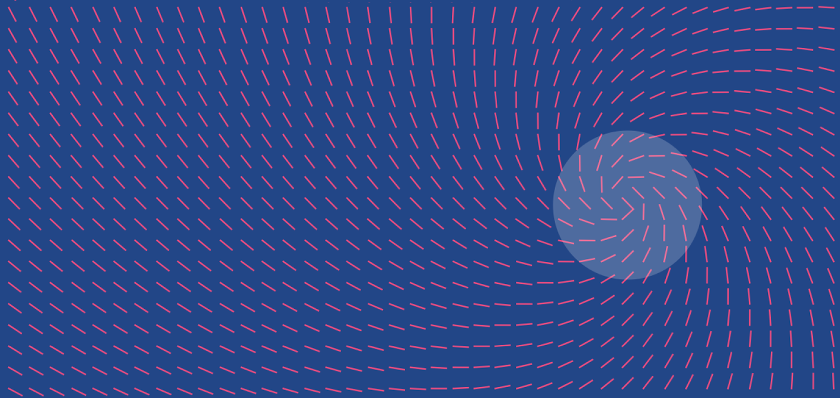
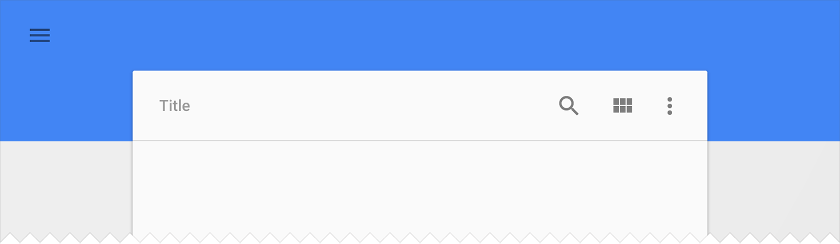
appcompat v21: material design for pre-Lollipop devices!

17 Oct 2014

The Android 5.0 SDK was released today and among the many tasty updates, includes new UI widgets and theming abilities to create material design style apps. To enable you to bring your updated designs to older platforms, we have also updated the support libraries including AppCompat. In this post I’ll outline what’s new in AppCompat and how you can use it in your app.

AppCompat (aka ActionBarCompat) started out as a backport of the Android 4.0 [ActionBar](https://developer.android.com/reference/android/app/ActionBar.html)API for devices running on Gingerbread, providing a common API layer on top of the backported implementation and framework implementation. The new v21 appcompat update provides the API and feature-set that is up to date with Android 5.0.

Android 5.0 introduces a new [Toolbar](https://developer.android.com/reference/android/widget/Toolbar.html) widget. This is a generalization of the ActionBar pattern but gives you **much** more control and flexibility in using it. Toolbar is a view in your hierarchy just like any other, making it easier to interleave with the rest of your views, animate, react to scroll events.



Setup

If you’re using Gradle then the first thing to do is to add appcompat as a dependency in your build.gradle:

dependencies {

...

compile "com.android.support:appcompat-v7:21.0.+"

}

New integration

If you are not currently using AppCompat, or you are starting from scratch, then here’s how to set it up:

* All of your Activities must extend from [ActionBarActivity](https://developer.android.com/reference/android/support/v7/app/ActionBarActivity.html). It extends from[FragmentActivity](https://developer.android.com/reference/android/support/v4/app/FragmentActivity.html) from the v4 support library, so you can continue to use fragments.
* All of your themes (that want an action bar/Toolbar) must inherit from Theme.AppCompat. There are variants available including Light and NoActionBar.
* When inflating anything to be displayed on the action bar (such as a [SpinnerAdapter](https://developer.android.com/reference/android/widget/SpinnerAdapter.html)for list navigation in the action bar), make sure you use the action bar’s themed context retrieved via getSupportActionBar().getThemedContext().
* You must use the static methods in [MenuItemCompat](https://developer.android.com/reference/android/support/v4/view/MenuItemCompat.html) for any action-related calls on a MenuItem.

The [Action Bar API guide](http://developer.android.com/guide/topics/ui/actionbar.html) is a comprehensive guide for AppCompat, although it needs to be updated for v21. Just ignore the ‘Example theme’ section at the bottom, where it mentions that you need to set ‘two version for each action bar style property’.

Migration from previous setup

If you already have an existing AppCompat setup then here’s how to update your theme to take advantage of the new capabilities:

We now use the support implementation of Toolbar/ActionBar on all platforms meaning that we no longer read any android: attributes related to the action bar.

For apps which already have existing appcompat setups, this means that you should remove your v14+ themes which re-set the same values in the android namespace. Please note, this is ONLY applicable for styles/widgets which affect the action bar.

For most apps, you now only need one theme declaration, in values/:

**values/themes.xml**:

<style name="Theme.MyTheme" parent="Theme.AppCompat.Light">

<!-- Here we setting appcompat’s actionBarStyle -->

<item name="actionBarStyle">@style/MyActionBarStyle</item>

<!-- ...and here we setting appcompat’s color theming attrs -->

<item name="colorPrimary">@color/my\_awesome\_red</item>

<item name="colorPrimaryDark">@color/my\_awesome\_darker\_red</item>

<!-- The rest of your attributes -->

</style>

Theming

AppCompat has support for the new [color palette](http://developer.android.com/training/material/theme.html#ColorPalette) theme attributes which allow you to easily customise your theme to fit your brand with a [primary and accent colour](http://www.google.com/design/spec/style/color.html#color-ui-color-application). Here’s a quick example:

**values/themes.xml:**

<style name="Theme.MyTheme" parent="Theme.AppCompat.Light">

<!-- colorPrimary is used for the default action bar background -->

<item name="colorPrimary">@color/my\_awesome\_color</item>

<!-- colorPrimaryDark is used for the status bar -->

<item name="colorPrimaryDark">@color/my\_awesome\_darker\_color</item>

<!-- colorAccent is used as the default value for colorControlActivated

which is used to tint widgets -->

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

<!-- You can also set colorControlNormal, colorControlActivated

colorControlHighlight & colorSwitchThumbNormal. -->

</style>

By setting these, AppCompat will automatically propagate the values to the framework attributes on API 21+. This automatically colours the status bar and recents task entry.

On older platforms, AppCompat emulates the color theming where possible. At the moment this is limited to coloring the action bar and some widgets.

Widget Tinting

When running on an Android 5.0 device, all of the widgets will be tinted using the color theme attributes we just talked about. There are two main features which allow this on Lollipop: drawable tinting, and referencing theme attributes (of the form ?attr/foo) in drawables.

AppCompat provides similar behaviour on older devices for a subset of UI widgets:

* Everything provided by AppCompat’s toolbar (action modes, etc)
* EditText
* Spinner
* CheckBox
* RadioButton
* Switch (use the new [android.support.v7.widget.SwitchCompat](https://developer.android.com/reference/android/support/v7/widget/SwitchCompat.html))
* CheckedTextView

You do not need to do anything special to make these work, just use these controls in your layouts as usual and AppCompat will do the rest (with some caveats; see the FAQ below).

Widgets

Toolbar

Firstly, Toolbar is fully supported in AppCompat and has feature and API parity with the framework widget. To use it, you need to use the class [android.support.v7.widget.Toolbar](https://developer.android.com/reference/android/support/v7/widget/Toolbar.html). There are two ways to use Toolbar:

* Use a Toolbar as an action bar when you want to use the existing ActionBar facilities (such as menu inflation and selection, [ActionBarDrawerToggle](https://developer.android.com/reference/android/support/v7/app/ActionBarDrawerToggle.html) etc.) but want to have more control over it’s appearance.
* Use a standalone Toolbar when you want to use the pattern in your app in situations the Action Bar would not support e.g. showing multiple toolbars on the screen, spanning only part of the width etc.

Action Bar

To use Toolbar as an action bar, the first thing you need to do is disable the decor provided action bar. The easiest way is to have your theme extend from Theme.AppCompat.NoActionBar (or the light variant).

Second, you need to create a Toolbar instance, usually via your layout XML:

<android.support.v7.widget.Toolbar

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

android:id="@+id/my\_awesome\_toolbar"

android:layout\_height="wrap\_content"

android:layout\_width="match\_parent"

android:minHeight="?attr/actionBarSize"

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

The height, width, background, etc are totally up to you, these are just good examples. As Toolbar is just a ViewGroup, you can style and position it however you want.

Then in your Activity or Fragment, you set the Toolbar to act as your action bar:

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.my\_layout);

Toolbar toolbar = (Toolbar) findViewById(R.id.my\_awesome\_toolbar);

setSupportActionBar(toolbar);

}

From that point on, all menu items will be displayed in your Toolbar, populated via the standard options menu callbacks.

Standalone

Very similar to above, but in this mode you do **not** set the Toolbar to act as your action bar. For this reason, you can use any AppCompat theme and you do not need to disable the decor-provided action bar.

When used by itself, you need to populate the Toolbar with content/actions manually. For instance, if you want it to display actions, you need to inflate a menu into it:

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.my\_layout);

Toolbar toolbar = (Toolbar) findViewById(R.id.my\_awesome\_toolbar);

// Set an OnMenuItemClickListener to handle menu item clicks

toolbar.setOnMenuItemClickListener(

new Toolbar.OnMenuItemClickListener() {

@Override

public boolean onMenuItemClick(MenuItem item) {

// Handle the menu item

return true;

}

});

// Inflate a menu to be displayed in the toolbar

toolbar.inflateMenu(R.menu.your\_toolbar\_menu);

}

There are a number of other things you can do with Toolbar. See the Toolbar [javadoc](https://developer.android.com/reference/android/support/v7/widget/Toolbar.html) for more information.

Styling

Styling of Toolbar is done differently to the standard action bar, and is set directly onto the view.

Here’s a basic style you should be using when you’re using a Toolbar as your action bar:

<android.support.v7.widget.Toolbar

android:layout\_height="wrap\_content"

android:layout\_width="match\_parent"

android:minHeight="?attr/actionBarSize"

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

The app:theme declaration will make sure that your text and items are using solid colors (i.e 100% opacity white).

DarkActionBar

A common question I was asked during development was how to achieve a Toolbar which looks like ‘DarkActionBar’ (dark content, light overflow menu). This is achieved by providing the theme and popupTheme attributes:

<android.support.v7.widget.Toolbar

android:layout\_height="wrap\_content"

android:layout\_width="match\_parent"

android:minHeight="@dimen/triple\_height\_toolbar"

app:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"

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

Obviously the background, height, etc can be anything you want, the above are just suggestions.

SearchView

AppCompat offer’s Lollipop’s updated SearchView API with is far more customizable and styleable (queue the applause). Due to this, we are now using the L style structure instead of the old searchView\* theme attributes.

So here is how you style SearchView:

**values/themes.xml:**

<style name="Theme.MyTheme" parent="Theme.AppCompat">

<item name="searchViewStyle">@style/MySearchViewStyle</item>

</style>

<style name="MySearchViewStyle" parent="Widget.AppCompat.SearchView">

<!-- The layout for the search view. Be careful. -->

<item name="layout">...</item>

<!-- Background for the search query section (e.g. EditText) -->

<item name="queryBackground">...</item>

<!-- Background for the actions section (e.g. voice, submit) -->

<item name="submitBackground">...</item>

<!-- Close button icon -->

<item name="closeIcon">...</item>

<!-- Search button icon -->

<item name="searchIcon">...</item>

<!-- Go/commit button icon -->

<item name="goIcon">...</item>

<!-- Voice search button icon -->

<item name="voiceIcon">...</item>

<!-- Commit icon shown in the query suggestion row -->

<item name="commitIcon">...</item>

<!-- Layout for query suggestion rows -->

<item name="suggestionRowLayout">...</item>

</style>

Obviously you do not need to set all (or any) of these, the defaults will work for the majority of apps.

Toolbar is coming

Hopefully this post will help you get up and running with AppCompat and let you create some awesome, material apps. Let me know in the comments/[G+](https://plus.google.com/u/0/+ChrisBanes/posts/L4Y9TDndifL)/[Twitter](https://twitter.com/chrisbanes/status/523158184283078657) if you’re having trouble with AppCompat or any of the support libraries, or where we could provide more documentation.

FAQ

**Why is my EditText (or other widget listed above) not being tinted correctly on my pre-Lollipop device?**

The widget tinting in appcompat works by intercepting any layout inflation and inserting a special tint-aware version of the widget in its place. For most people this will work fine but I can think of a few scenarios where this won’t work including:

* You have your own custom version of the widget (i.e. you’ve extended EditText)
* You are creating the EditText without a LayoutInflater (i.e. calling new EditText()).
* You are hooking up to the [LayoutInflater](https://developer.android.com/reference/android/view/LayoutInflater.html)’s [Factory](https://developer.android.com/reference/android/view/LayoutInflater.Factory.html).

The special tint aware widgets are currently hidden because they’re an unfinished implementation detail.

Related Posts