Android Themes

Introduction

Android allows you to set the appearance of your application using themes. A theme specifies default colors, fonts, and images, for an Android activity or an entire application. Your application can use the device's built-in themes, or include custom themes. Titanium's root splash screen activity uses `"Theme.Titanium"` by default. All other activity windows will use `"Theme.MaterialComponents.Bridge"` as of Titanium 9.3.0. Older Titanium versions will use "Theme.AppCompat" for all activity windows.

To update the look of your application, you can either:

- Use a different built-in Android theme
- Use a Titanium-defined theme
- Create a custom theme and define the elements you want to change
- Use the Android Material Theme to quickly customize the theme's color palette
- Use third-party tools to help you generate a custom theme

You can also assign a unique theme to an activity instead which would override the application's assigned theme.

Android themes

Android provides some built-in themes to easily change the overall appearance of your application. To use a built-in theme, you need to create a theme XML file for your project, specify the built-in theme you want to use, and reference it in the Android manifest section of your `tiapp.xml` fil e.

First, create a theme XML file in `./platform/android/res/values`. For Titanium SDK 8.x.x and older, do **NOT** name the file `theme.xml` sin ce it will overwrite Titanium's built-in `theme.xml` file. This is not an issue with Titanium 9.0.0 and higher.

In the theme XML file, add the theme you want to use. Themes defined by the Android system, excluding the AppCompat ones, are prefaced with @android:style (for example, @android:style/Theme.Translucent). Custom themes defined by the application are prefaced with @style (for example, @style/Theme.MyTheme). For example, the file below adds support for some of the common built-in Android themes.

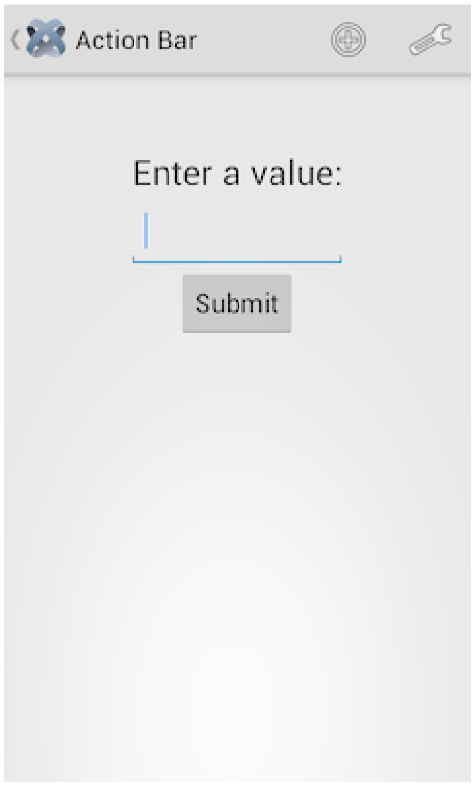
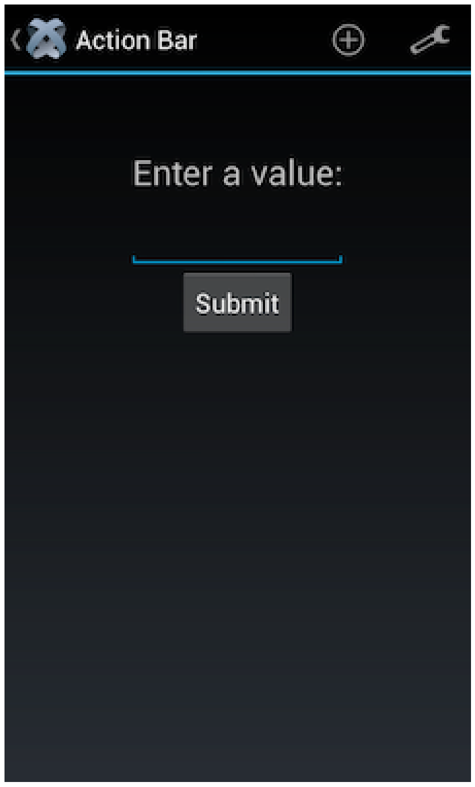
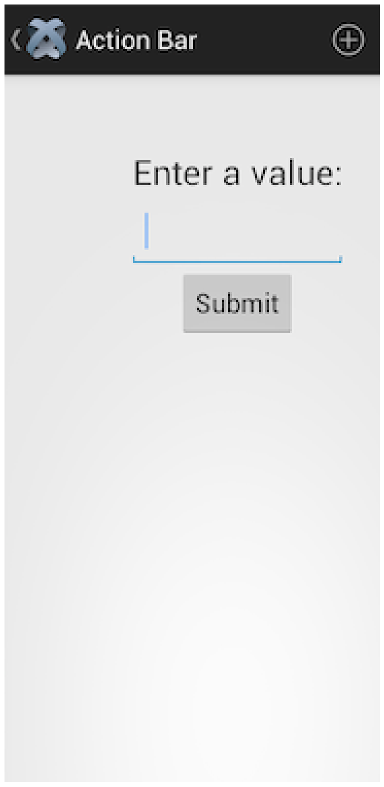
platform/android/res/values/builtin_themes.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources xmlns:android="http://schemas.android.com/apk/res/android">
 <style name="LightDarkBar" parent="Theme.AppCompat.Light.DarkActionBar"/>
 <style name="Light" parent="Theme.AppCompat.Light"/>
 <style name="Dark" parent="Theme.AppCompat"/>
</resources>
```

Finally, to use a theme in your application, modify the Android section of your `tiapp.xml` file to reference the style name you want to use:

```
<android xmlns:android="http://schemas.android.com/apk/res/android">
    <manifest>
        <application android:theme="@style/LightDarkBar"/>
    </manifest>
</android>
```

The screenshots below show the difference between the various built-in themes:

AppCompat.Light	AppCompat/Holo	AppCompat.Light.DarkActionB
		

Titanium themes

The Titanium SDK includes the below predefined themes. You can use these themes instead of creating your own.

NOTE: Do not create a theme with the same name as a predefined theme.

Theme Name	SDK Version	Description
Theme.Titanium	*	Theme applied to the root splash screen activity only. Does not show a top action bar. As of Titanium 9.3.0, this theme is based on `Theme.MaterialComponents.Bridge`. Prior to 9.3.0, this theme is based on `Theme.AppCompat`.
Theme.AppCompat.Translucent	3.4.0	Based on `Theme.AppCompat`. Has a transparent background.
Theme.AppCompat.Translucent.NoTitleBar	3.4.0	Based on `Theme.AppCompat`. Has a transparent background and no action bar.
Theme.AppCompat.Translucent.NoTitleBar.Fullscreen	3.4.0	Based on `Theme.AppCompat`. Has a transparent background. Has no action bar or status bar.

Theme.AppCompat.Fullscreen	3.4.0	Based on `Theme.AppCompat`. Has no action or status bar.
Theme.AppCompat.NoTitleBar	4.2.0	Based on `Theme.AppCompat`. Has no action bar.
Theme.AppCompat.NoTitleBar.Fullscreen	4.2.0	Exactly the same as `Theme.AppCompat.Fullscreen` above.
Theme.MaterialComponents.Fullscreen.Bridge	9.3.0	Based on `Theme.MaterialComponents.Bridge`. Has no action bar or status bar.
Theme.Titanium.NoTitleBar	9.3.0	Based on the application's assigned theme, which uses `Theme.MaterialComponents.Bridge` by default. Has no action bar. Can only be applied to activities and not the application.
Theme.Titanium.Fullscreen	9.3.0	Based on the application's assigned theme, which uses `Theme.MaterialComponents.Bridge` by default. Has no action bar or status bar. Can only be applied to activities and not the application.
Theme.Titanium.Translucent.NoTitleBar	9.3.0	Based on the application's assigned theme, which uses `Theme.MaterialComponents.Bridge` by default. Has a transparent background and no action bar. Can only be applied to activities and not the application.
Theme.Titanium.Translucent.Fullscreen	9.3.0	Based on the application's assigned theme, which uses `Theme.MaterialComponents.Bridge` by default. Has a transparent background. Has no action bar or status bar. Can only be applied to activities and not the application.

To apply a theme globally, in the `tiapp.xml` file, set the `android:theme` attribute to the theme name in the `<application/>` element of the Android manifest section. As of Titanium 9.0.0, you can use a `Theme.MaterialComponents.*` based theme to support Google's material design.

```
<android xmlns:android="http://schemas.android.com/apk/res/android">
    <manifest>
        <!-- For Titanium 8.x.x and older, use "Theme.AppCompat.NoTitleBar" instead.
-->
        <application
android:theme="@style/Theme.MaterialComponents.NoActionBar.Bridge"/>
    </manifest>
</android>
```

To change the theme on a per-window basis, set the theme name to a Window's `theme` property. For Titanium 9.3.0 and higher, you should use a `Theme.Titanium.*` based theme since they are based on the theme assigned to `<application/>`, making it look consistent with other windows. For Titanium versions older than 9.3.0, you should use a `Theme.AppCompat.*` based theme.

```
var win = Ti.UI.createWindow({
 theme: "Theme.Titanium.Fullscreen"
});
```

Custom themes

To define custom themes, place the theme XML files with your custom styles in the `platform/android/res/values` folder. For Titanium SDK 8.x.x and older, do **NOT** name the file `theme.xml` since it will overwrite Titanium's built-in `theme.xml` file. This is not an issue with Titanium 9.0.0 and higher.

Note that you can also define version-specific themes by adding a `values-v<version>` folder. For example a theme defined under a `values-v23` folder will be used for API Level 23 (Android 6) and above. A theme defined under the `values-v29` folder will be used for API Level 29 (Android 10) and above.

For example, if you want your theme to be based on the Light theme, create the following theme file:

platform/android/res/values/mytheme.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- Define a theme using the AppCompat.Light theme as a base theme -->
    <!-- Note: For Titanium 8.x.x and older, use "Theme.AppCompat.Light" instead. -->
    <style name="Theme.MyTheme" parent="Theme.MaterialComponents.Light.Bridge">
        <!-- Overrides the background color-->
        <item name="android:windowBackground">@drawable/example</item>
        <!-- Overrides the default text color -->
        <item name="android:textColorPrimary">@color/myTextColor</item>
        <!-- Overrides the default cursor color -->
        <item name="android:textCursorDrawable">@null</item>
        <!-- Overrides style for a component with custom one -->
        <item name="android:buttonStyle">@style/myButtonStyle</item>
    </style>
    <!-- Define custom style for Buttons. -->
    <style name="myButtonStyle" parent="Widget.AppCompat.Button">
        <item name="android:minHeight">@dimen/myMinButtonHeight</item>
        <item name="android:minWidth">@dimen/myMinButtonWidth</item>
    </style>
</resources>
```

platform/android/res/values/colors.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="myBackground">#EE5678</color>
    <color name="myButton">#39FF21</color>
    <color name="myTextColor">#0000FF</color>
    <color name="myButtonTextColor">#FFFF00</color>
</resources>
```

platform/android/res/values/dimens.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
 <dimen name="myMinButtonWidth">250dp</dimen>
 <dimen name="myMinButtonHeight">125dp</dimen>
</resources>
```

The theme file above contains two custom style items:

1. The first item defines the "Theme.MyTheme" based on the Android Light theme. The style sets the window background using the image located at `platform/android/res/drawable/example.png`.
2. The second item overrides the default text color used in the theme. For convenience all colors can be defined in a separate file located at `platform/android/res/values/colors.xml`.
3. The third item overrides the Light theme's default cursor color for text fields and uses the TextField's `color` property as the cursor color.
4. The last item overrides the whole style used for buttons. The custom style is defined in the theme's file as a child of `Widget.AppCompat.Button`. In the custom style every property of it can be overriden. In this example the minimum width and height for every button are set to custom values. For convenience dimensions can be defined in a separate file located at `platform/android/res/values/dimens.xml`

To use the "Theme.MyTheme" theme in your application, modify the Android section of your `tiapp.xml` file:

```
<android xmlns:android="http://schemas.android.com/apk/res/android">
    <manifest>
        <application android:theme="@style/Theme.MyTheme"/>
    </manifest>
</android>
```

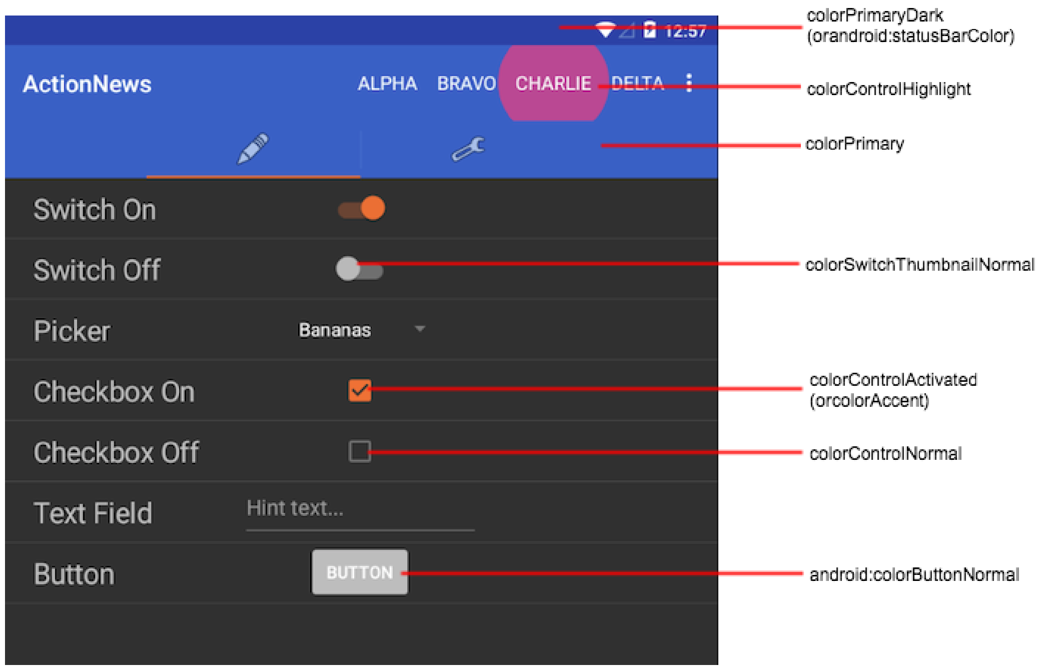
Refer to Android Developers: Styles and Themes for detailed information on customizing items.

Material theme

As of Titanium 9.3.0, apps use the `Theme.MaterialComponents.Bridge` by default. You do not need to create your own theme.

For older Titanium SDK versions, you'll need to create a custom theme. For 9.0.0 and above, it should be extend a `Theme.MaterialComponents.*` based theme. For Titanium 8.x.x and older, it should extend a `Theme.AppCompat.*` based theme.

With a material based theme, you'll be able to set additional color palette attributes.



Color Palette Attribute	Description
`colorPrimaryDark`	Sets the color of the status bar. Only works with Android 5.0 (API 21) and greater.
`colorPrimary`	Sets the color of the action bar.
`colorAccent`	Sets the accent color, which is usually the color of the control when it is activated.
`colorControlNormal`	Sets the color of the control when it is not activated.
`colorControlActivated`	Sets the color when the control is activated. Overwrites the `colorAccent` attribute.
`colorControlHighlight`	Sets the color when the user clicks on a control. Only works with Android 5.0 (API 21) and greater.
`colorSwitchThumbNormal`	Sets the color of a toggle switch's thumb when it is not enabled. Only works with Android 5.0 (API 21) and greater.
`android:colorButtonNormal`	Sets the color of a button when it is not pressed. Only works with Android 5.0 (API 21) and greater.
`android:colorEdgeEffect`	Sets the edge effect color when the user tries to scroll beyond the content's boundaries. Only works with Android 5.0 (API 21) and greater.

`android:navigationBarColor`	Sets the color of the navigation bar (the bar that appears at the bottom of the device that contains the Back, Home and Recent App buttons). Only works with Android 5.0 (API 21) and greater.
`android:textColorPrimary`	Sets the color of text on controls. Prior to Android 5.0, this only sets the color of the action bar title and overflow menu.

Example

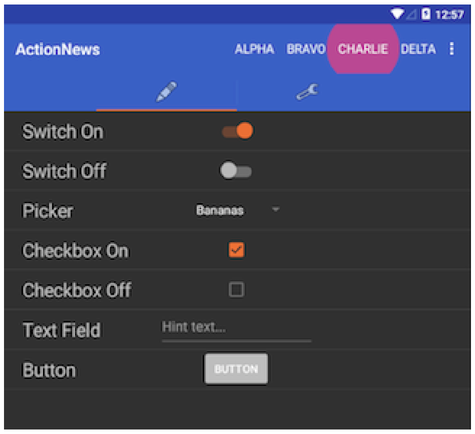
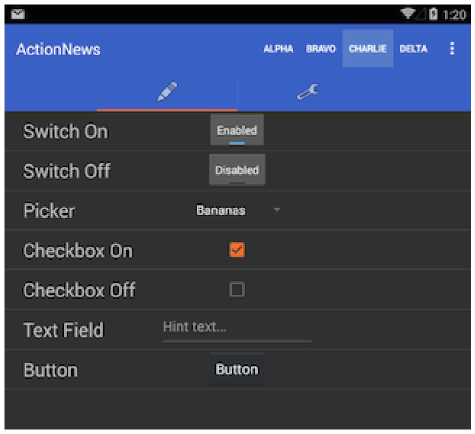
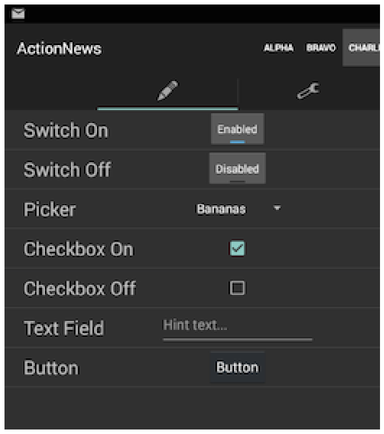
The following XML file below defines a theme that extends a material based theme and applies additional color palette attributes. This theme was applied to the application in the previous screen shot.

platform/android/res/values/custom_theme.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- For Titanium 8.x.x and older, use "Theme.AppCompat" instead. -->
 <style name="MyMaterialTheme" parent="@style/Theme.MaterialComponents.Bridge">
     <item name="colorPrimary">#1565C0</item>
     <item name="colorPrimaryDark">#0D47A1</item>
     <item name="colorAccent">#FF80AB</item>
     <item name="colorControlNormal">#757575</item>
     <item name="colorControlActivated">#FF6E40</item>
     <item name="colorControlHighlight">#FF4081</item>
     <item name="colorSwitchThumbNormal">#BDBDBD</item>
     <item name="android:colorButtonNormal">#BDBDBD</item>
     <item name="android:colorEdgeEffect">#FF4081</item>
 </style>
</resources>
```

Theme comparison

The following table compares the same application when using different Android versions and themes. Note that the Android 4.4. screenshots use the old style toggle button due to Android bug #78262.

Example Material Theme / Android 5.0	Example Material Theme / Android 4.4	Default AppCompat / Android 4.4
 12:57 ActionNews ALPHA BRAVO CHARLIE DELTA Switch On Switch Off Picker Bananas Checkbox On Checkbox Off Text Field Hint text... Button BUTTON	 1:20 ActionNews ALPHA BRAVO CHARLIE DELTA Switch On Enabled Switch Off Disabled Picker Bananas Checkbox On Checkbox Off Text Field Hint text... Button Button	 ActionNews ALPHA BRAVO CHARLI Switch On Enabled Switch Off Disabled Picker Bananas Checkbox On Checkbox Off Text Field Hint text... Button Button

Material theme further reading

- Android Developer: Using Material Theme
- Android Developers Blog: AppCompat v21 — Material Design for Pre-Lollipop Devices!
- Google Design Guidelines: Style - Color

Override a theme

If you have a global theme set, the application can override the theme for a window or activity, or if you want to specify a custom theme for a specific window or activity.

Override a window theme

Use the `theme` property to override the global theme for an individual window. Set the property to the name of the theme you want to apply to the window. The property can only be set when creating the Window object and cannot be changed after it is set. For Titanium 9.3.0 and higher, you should use a `Theme.Titanium.*` based theme since they are based on the theme assigned to `<application/>`, making it look consistent with other windows. For Titanium versions older than 9.3.0, you should use a `Theme.AppCompat.*` based theme.

```
var win = Ti.UI.createWindow({
    theme: "Theme.Titanium.Fullscreen"
});
```

Override an activity theme

As you can see in the previous examples, an application-wide theme can be specified in the `<application>` element of your `tiapp.xml` file, but it can also be overridden on a per-activity basis. These activities are defined in the `AndroidManifest.xml` file, generated by the build process. You can find the generated `AndroidManifest.xml` file in the `build/android` folder under your project folder. Inside the `AndroidManifest.xml`, you'll find code like this:

```
<activity android:name=".ThemetestActivity" android:theme="@style/Theme.Titanium">
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
</activity>

<activity android:name="org.appcelerator.titanium.TiActivity"/>
<activity android:name="org.appcelerator.titanium.TiTranslucentActivity"
android:theme="@android:style/Theme.Titanium.Translucent"/>
```

The `build` folder is hidden by default in Studio's **App Explorer** and **Project Explorer** views, but visible in the **Navigator** view.

To override the theme for one of these activities, copy the activity definition into the `tiapp.xml` file, and place it inside the `<android><manifest>` element. The `<activity>` element must be nested inside an `<application>` element, as in the original `AndroidManifest.xml` file. The end result should look something like this:

```
<android xmlns:android="http://schemas.android.com/apk/res/android">
    <manifest>
        <application>
            <!-- Override the root splash screen activity's theme. -->
            <activity android:name=".ThemetestActivity"
android:theme="@style/Theme.MyTheme">
                <intent-filter>
                    <action android:name="android.intent.action.MAIN"/>
                    <category android:name="android.intent.category.LAUNCHER"/>
                </intent-filter>
            </activity>
        </application>
    </manifest>
</android>
```

If you change the name of the application, you will need to copy the newly generated `android:name` value from the `AndroidManifest.xml` file and overwrite the old value in the `tiapp.xml` file.

Custom theme generator

Android Action Bar Style Generator

The Android Action Bar Style Generator is a website tool that builds and generates an Android Action Bar style based on the customizations you select. This tool generates the XML files and graphic assets you need and packages them in to a ZIP file, which you can unpack in your project.

To create a custom Action Bar style:

1. Go to Android Action Bar Style Generator.
2. Enter a name for your style. This name will be used to reference your style in the Android manifest. If you want to use a theme generated by the Android Holo Colors Generator, do not use the same name.
3. For **Style Compatibility**, use **AppCompat.**
4. Choose the style customizations you want to use.
5. Click **Download .ZIP** button near the button of the web page to download your custom theme.

Once you have your custom Action Bar style, unpack the ZIP file and copy the `res` folder to your project's `platform/android` folder. Then, modify your `tiapp.xml` file to use the custom style, for example:

```
<android xmlns:android="http://schemas.android.com/apk/res/android">
    <manifest>
        <!-- Replace StyleName with the name of your style -->
        <application android:theme="@style/Theme.StyleName"/>
    </manifest>
</android>
```

Further reading

- Android Developers: Styles and Themes