**STYLES AND THEMES:**

**Style**: A **style** is a collection of properties that specify the look and format for a [View](http://developer.android.com/reference/android/view/View.html) or window. A style can specify properties such as height, padding, font color, font size, background color, and much more. A style is defined in an XML resource that is separate from the XML that specifies the layout.

For Example, by using a style, you can take this layout XML:

<TextView

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:textColor="#00FF00"

android:typeface="monospace"

android:text="@string/hello" />

And turn it into this:

<TextView

style="@style/txt\_font"

android:text="@string/hello" />

All of the attributes related to style have been removed from the layout XML and put into a style definition called as txt\_font, which is then applied with style attribute.

**Defining Styles:** To create a set of styles, save an XML file in the res/values directory of your project.For each style you want to create, add a <style> element to the file with a name that uniquely identifies the style (this attribute is required). Then add an <item> element for each property of that style, with a name that declares the style property and a value to go with it (this attribute is required). The value for the <item> can be a keyword string, a hex color, a reference to another resource type, or other value depending on the style property. Here's an example file with a single style:

<?xml version="1.0" encoding="utf-8"?>

<resources>

<style name="txt\_font " parent="@android:style/TextAppearance.Medium">

<item name="android:layout\_width">fill\_parent</item>

<item name="android:layout\_height">wrap\_content</item>

<item name="android:textColor">#00FF00</item>

<item name="android:typeface">monospace</item>

</style>

</resources>

**Theme**: A **theme** is a style applied to an entire Activity or application, rather than an individual View (as in the example above). When a style is applied as a theme, every View in the Activity or application will apply each style property that it supports. For example, you can apply the same as txt\_font style as a theme for an Activity and then all text inside that Activity.

**Apply a theme to an Activity or application:**

To set a theme for all the activities of your application, open the AndroidManifest.xml file and edit the<application> tag to include the android:theme attribute with the style name. For example:

<application android:theme="@style/CustomTheme">

If you want a theme applied to just one Activity in your application, then add the android:theme attribute to the<activity> tag instead.

Just as Android provides other built-in resources, there are many pre-defined themes that you can use, to avoid writing them yourself. For example, you can use the Dialog theme and make your Activity appear like a dialog box:

<activity android:theme="@android:style/Theme.Dialog">

If you like a theme, but want to change or modify or twist it, just add the theme as the parent of your custom theme. For example, you can modify the traditional light theme to use your own color like this:

### <color name="custom\_theme\_color">#b0b0ff</color>

### <style name="CustomTheme" parent="android:Theme.Light">

### <item name="android:windowBackground">@color/custom\_theme\_color</item>

### <item name="android:colorBackground">@color/custom\_theme\_color</item>

### </style>

### Select a theme based on platform version:

Newer versions of Android have additional themes available to applications, and you might want to use these while running on those platforms while still being compatible with older versions. You can accomplish this through a custom theme that uses resource selection to switch between different parent themes, based on the platform version.

For example, here is the declaration for a custom theme which is simply the standard platforms default light theme. It would go in an XML file under res/values (typically res/values/styles.xml):

<style name="LightThemeSelector" parent="android:Theme.Light">

...

</style>

To have this theme use the newer holographic theme when the application is running on Android 3.0 (API Level 11) or higher, you can place an alternative declaration for the theme in an XML file in res/values-v11, but make the parent theme the holographic theme:

<style name="LightThemeSelector" parent="android:Theme.Holo.Light">

...

</style>

Now use this theme like you would any other, and your application will automatically switch to the holographic theme if running on Android 3.0 or higher.

Choosing Theme to Extend:

Android themes are rich in mediums,and there are many attributes defined that effects the styles of the user interface.Many of those attributes defined for proper operation.Because there are so many attributes,it is best to extend an existing theme.

The parent attribute of the <style> element controls which theme your custom theme will extend.There are four base Android themes that are excellent candidates for extension.

**Theme**:This is the base android theme introduced with the first version of Android. It is a dark theme with light text and works on all versions of Android.

**Theme.Light**: This is a light variation of Theme. It displays dark text on a light background

**Theme.Holo**: This was the new Android theme introduced with Honeycomb. It features more “modern” styling, and it is only available on Android versions 3.0 and above. Theme.Holo cannot be altered by the device manufacturer or carrier. It can still be extended by application themes, however

**Theme.Holo.Light**: This is a light variation of Theme.Holo.

There are many for better reference to the themes source code follow the link:

<https://android.googlesource.com/platform/frameworks/base/+/refs/heads/master/core/res/res/values/themes.xml>