Styles and Themes

A **style** is a collection of properties that specify the look and format for a `View` (https://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.

Styles in Android share a similar philosophy to cascading stylesheets in web design—they allow you to separate the design from the content.

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/CodeFont"
    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 `CodeFont`, which is then applied with the `style` attribute. You'll see the definition for this style in the following section.

A **theme** is a style applied to an entire `Activity` (https://developer.android.com/reference/android/app/Activity.html) or application, rather than an individual `View` (https://developer.android.com/reference/android/view/View.html) (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 `CodeFont` style as a theme for an Activity and then all text inside that Activity will have green monospace font.

In this document

See also

Defining Styles

To create a set of styles, save an XML file in the `res/values/` directory of your project. The name of the XML file is arbitrary, but it must use the `.xml` extension and be saved in the `res/values/` folder.

The root node of the XML file must be `<resources>`.

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="CodeFont" 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>
```

Each child of the `<resources>` element is converted into an application resource object at compile-time, which can be referenced by the value in the `<style>` element's `name` attribute. This example style can be referenced from an XML layout as `@style/CodeFont` (as demonstrated in the introduction above).

The `parent` attribute in the `<style>` element is optional and specifies the resource ID of another style from which this style should inherit properties. You can then override the inherited style properties if you want to.

Remember, a style that you want to use as an Activity or application theme is defined in XML exactly the same as a style for a View. A style such as the one defined above can be applied as a style for a single View or as a theme for an entire Activity or application. How to apply a style for a single View or as an application theme is discussed later.

Inheritance

The `parent` attribute in the `<style>` element lets you specify a style from which your style should inherit properties. You can use this to inherit properties from an existing style and then define only the properties that you want to change or add. You can inherit from styles that you've created yourself or from styles that are built into the platform. (See Using Platform Styles and Themes (#PlatformStyles), below, for information about inheriting from styles defined by the Android platform.) For example, you can inherit the Android platform's default text appearance and then modify it:

```
    <style name="GreenText" parent="@android:style/TextAppearance">
        <item name="android:textColor">#00FF00</item>
    </style>
```

If you want to inherit from styles that you've defined yourself, you *do not* have to use the `parent` attribute. Instead, just prefix the name of the style you want to inherit to the name of your new style, separated by a period. For example, to create a new style that inherits the `CodeFont` style defined above, but make the color red, you can author the new style like this:

```
    <style name="CodeFont.Red">
        <item name="android:textColor">#FF0000</item>
    </style>
```

Notice that there is no `parent` attribute in the `<style>` tag, but because the `name` attribute begins with the `CodeFont` style name (which is a style that you have created), this style inherits all style properties from that style. This style then overrides the `android:textColor` property to make the text red. You can reference this new style as `@style/CodeFont.Red`.

You can continue inheriting like this as many times as you'd like, by chaining names with periods. For example, you can extend `CodeFont.Red` to be bigger, with:

```
    <style name="CodeFont.Red.Big">
        <item name="android:textSize">30sp</item>
    </style>
```

This inherits from both `CodeFont` and `CodeFont.Red` styles, then adds the `android:textSize` property.

> **Note:** This technique for inheritance by chaining together names only works for styles defined by your own resources. You can't inherit Android built-in styles this way. To reference a built-in style, such as `TextAppearance` (https://developer.android.com/reference/android/R.style.html#TextAppearance), you must use the `parent` attribute.

Style Properties

Now that you understand how a style is defined, you need to learn what kind of style properties—defined by the `<item>` element—are available. You're probably familiar with some already, such as `layout_width` (https://developer.android.com/reference/android/R.attr.html#layout_width) and `textColor` (https://developer.android.com/reference/android/R.attr.html#textColor). Of course, there are many more style properties you can use.

The best place to find properties that apply to a specific `View` (https://developer.android.com/reference/android/view/View.html) is the corresponding class reference, which lists all of the supported XML attributes. For example, all of the attributes listed in the table of TextView XML attributes (https://developer.android.com/reference/android/widget/TextView.html#lattrs) can be used in a style definition for a `TextView` (https://developer.android.com/reference/android/widget/TextView.html) element (or one of its subclasses). One of the attributes listed in the reference is `android:inputType` (https://developer.android.com/reference/android/widget/TextView.html#attr_android:inputType), so where you might normally place the `android:inputType` (https://developer.android.com/reference/android/widget/TextView.html#attr_android:inputType) attribute in an `<EditText>` element, like this:

```
<EditText
    android:inputType="number"
    ... />
```

You can instead create a style for the `EditText` (https://developer.android.com/reference/android/widget/EditText.html) element that includes this property:

```
<style name="Numbers">
  <item name="android:inputType">number</item>
  ...
</style>
```

So your XML for the layout can now implement this style:

```
<EditText
    style="@style/Numbers"
    ... />
```

This simple example may look like more work, but when you add more style properties and factor-in the ability to re-use the style in various places, the pay-off can be huge.

For a reference of all available style properties, see the `R.attr` (https://developer.android.com/reference/android/R.attr.html) reference. Keep in mind that all View objects don't accept all the same style attributes, so you should normally refer to the specific `View` (https://developer.android.com/reference/android/view/View.html) class for supported style properties. However, if you apply a style to a View that does not support all of the style properties, the View will apply only those properties that are supported and simply ignore the others.

Some style properties, however, are not supported by any View element and can only be applied as a theme. These style properties apply to the entire window and not to any type of View. For example, style properties for a theme can hide the application title, hide the status bar, or change the window's background. These kind of style properties do not belong to any View object. To discover these theme-only style properties, look at the `R.attr` (https://developer.android.com/reference/android/R.attr.html) reference for attributes that begin with `window`. For instance, `windowNoTitle` and `windowBackground` are style properties that are effective only when the style is applied as a theme to an Activity or application. See the next section for information about applying a style as a theme.

> **Note:** Don't forget to prefix the property names in each `<item>` element with the `android:` namespace. For example: `<item name="android:inputType">`.

Applying Styles and Themes to the UI

There are two ways to set a style:

- To an individual View, by adding the `style` attribute to a View element in the XML for your layout.
- Or, to an entire Activity or application, by adding the `android:theme` attribute to the `<activity>` or `<application>` element in the Android manifest.

When you apply a style to a single `View (https://developer.android.com/reference/android/view/View.html)` in the layout, the properties defined by the style are applied only to that `View (https://developer.android.com/reference/android/view/View.html)`. If a style is applied to a `ViewGroup (https://developer.android.com/reference/android/view/ViewGroup.html)`, the child `View (https://developer.android.com/reference/android/view/View.html)` elements will **not** inherit the style properties—only the element to which you directly apply the style will apply its properties. However, you *can* apply a style so that it applies to all `View (https://developer.android.com/reference/android/view/View.html)` elements—by applying the style as a theme.

To apply a style definition as a theme, you must apply the style to an `Activity (https://developer.android.com/reference/android/app/Activity.html)` or application in the Android manifest. When you do so, every `View (https://developer.android.com/reference/android/view/View.html)` within the Activity or application will apply each property that it supports. For example, if you apply the `CodeFont` style from the previous examples to an Activity, then all View elements that support the text style properties will apply them. Any View that does not support the properties will ignore them. If a View supports only some of the properties, then it will apply only those properties.

Apply a style to a View

Here's how to set a style for a View in the XML layout:

```
<TextView
    style="@style/CodeFont"
    android:text="@string/hello" />
```

Now this TextView will be styled as defined by the style named `CodeFont`. (See the sample above, in Defining Styles (#DefiningStyles).)

> **Note:** The `style` attribute does *not* use the `android:` namespace prefix.

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">
```

Or if you want the background to be transparent, use the Translucent theme:

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

If you like a theme, but want to tweak 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>
```

(Note that the color needs to supplied as a separate resource here because the `android:windowBackground` attribute only supports a reference to another resource; unlike `android:colorBackground`, it can not be given a color literal.)

Now use `CustomTheme` instead of `Theme.Light` inside the Android Manifest:

```
<activity android:theme="@style/CustomTheme">
```

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.

A list of the standard attributes that you can use in themes can be found at `R.styleable.Theme` (https://developer.android.com/reference/android/R.styleable.html#Theme).

For more information about providing alternative resources, such as themes and layouts, based on the platform version or other device configurations, see the Providing Resources (https://developer.android.com/guide/topics/resources/providing-resources.html) document.

Using Platform Styles and Themes

The Android platform provides a large collection of styles and themes that you can use in your applications. You can find a reference of all available styles in the `R.style` (https://developer.android.com/reference/android/R.style.html) class. To use the styles listed here, replace all underscores in the style name with a period. For example, you can apply the `Theme_NoTitleBar` (https://developer.android.com/reference/android/R.style.html#Theme_NoTitleBar) theme with `"@android:style/Theme.NoTitleBar"`.

The `R.style` (https://developer.android.com/reference/android/R.style.html) reference, however, is not well documented and does not thoroughly describe the styles, so viewing the actual source code for these styles and themes will give you a better understanding of what style properties each one provides. For a better reference to the Android styles and themes, see the following source code:

- Android Styles (styles.xml)
  (https://android.googlesource.com/platform/frameworks/base/+/refs/heads/master/core/res/res/values/styles.xml)
- Android Themes (themes.xml)
  (https://android.googlesource.com/platform/frameworks/base/+/refs/heads/master/core/res/res/values/themes.xml)

These files will help you learn through example. For instance, in the Android themes source code, you'll find a declaration for `<style name="Theme.Dialog">`. In this definition, you'll see all of the properties that are used to style dialogs that are used by the Android framework.

For more information about the syntax for styles and themes in XML, see the Style Resource (https://developer.android.com/guide/topics/resources/style-resource.html) document.

For a reference of available style attributes that you can use to define a style or theme (e.g., "windowBackground" or "textAppearance"), see `R.attr` (https://developer.android.com/reference/android/R.attr.html) or the respective View class for which you are creating a style.