Android Developers

Styles and Themes

In this document

- Defining Styles
  - Inheritance
  - Style Attributes
- Applying Styles and Themes to the UI
  - Apply a style to a view
  - Apply a theme to an activity or app
  - Maintaining theme compatibility

See also

- Style and Theme Resources
- `R.style` for Android styles and themes
- `R.attr` for all style attributes

A *style* is a collection of attributes that specify the look and format for a `View` (https://developer.android.com/reference/android/view/View.html) or window. A style can specify attributes 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="match_parent"
    android:layout_height="wrap_content"
    android:textColor="#00FF00"
    android:typeface="monospace"
    android:text="@string/hello" />
```

And turn it into this:

```
<TextView
    android:layout_width="match_parent"
```

The attributes related to style have been removed from the layout XML and put into a style definition called `CodeFont`, which is then applied using the `android:textAppearance` attribute. The definition for this style is covered in the following section.

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

A *theme* is a style applied to an entire `Activity` (https://developer.android.com/reference/android/app/Activity.html) or app, 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 app applies each style attribute that it supports. For example, if you apply the same `CodeFont` style as a theme for an activity, then all text inside that activity appears in a green monospace font.

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 must use the `.xml` extension, and like other resources, it must use lowercase, underscores, 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, complete the following series of steps:

1. Add a `<style>` element to the file, with a `name` that uniquely identifies the style.
2. For each attribute of that style, add an `<item>` element, with a `name` that declares the style attribute. The order of these elements doesn't matter.
3. Add an appropriate value to each `<item>` element.

Depending on the attribute, you can use values with the following resource types in an `<item>` element:

- Fraction
- Float
- Boolean
- Color
- String
- Dimension
- Integer

You can also use values with a number of special types in an `<item>` element. The following list of special types are unique to style attributes:

- Enumerators consisting of a set of integers.
- References which are used to point to another resource.

For example, you can specify the particular value for an `android:textColor` attribute—in this case a hexadecimal color—or you can specify a reference to a color resource so that you can manage it centrally along with other colors.

The following example illustrates using hexadecimal color values in a number of attributes:

```
<resources>
    <style name="AppTheme" parent="Theme.Material">
        <item name="colorPrimary">#673AB7</item>
        <item name="colorPrimaryDark">#512DA8</item>
        <item name="colorAccent">#FF4081</item>
    </style>
</resources>
```

And the following example illustrates specifying values for the same attribute using references:

```
<resources>
    <style name="AppTheme" parent="Theme.Material">
        <item name="colorPrimary">@color/primary</item>
        <item name="colorPrimaryDark">@color/primary_dark</item>
        <item name="colorAccent">@color/accent</item>
    </style>
</resources>
```

You can find information on which resource types can be used with which attributes in the attribute reference, `R.attr` (https://developer.android.com/reference/android/R.attr.html). For more information on centrally managing resources, see Providing Resources (https://developer.android.com/guide/topics/resources/providing-resources.html). For more information on working with color resources, see More Resource Types (https://developer.android.com/guide/topics/resources/more-resources.html#Color).

Here's another 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:textColor">#00FF00</item>
        <item name="android:typeface">monospace</item>
    </style>
</resources>
```

This example style can be referenced from an XML layout as `@style/CodeFont` (as demonstrated in the introduction above).

A style that you want to use as an activity or app theme is defined in XML exactly the same as a style for a view. How to apply a style as an app theme is discussed in Apply a theme to an activity or app (#ApplyATheme).

Inheritance

The `parent` attribute in the `<style>` element lets you specify a style from which your style should inherit attributes. You can use this to inherit attributes from an existing style and define only the attributes 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. For example, you can inherit the Android platform's default text appearance and modify it:

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

For information about inheriting from styles defined by the Android platform, see Using Platform Styles and Themes (#PlatformStyles).

If you want to inherit from styles that you've defined yourself, you don't have to use the `parent`. Instead, you can use dot notation by prefixing 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 create the new style like this:

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

Notice that there is no `parent` in the `<style>` tag, but because the `name` begins with the `CodeFont` style name, this new style inherits all style attributes from the CodeFont style. The new style then overrides the `android:textColor` attribute to make the text red. You can reference this new style as `@style/CodeFont.Red`.

You can continue inheriting styles 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 style inherits from the `CodeFont.Red` style, that itself inherits from the `CodeFont` style, then adds the `android:textSize` attribute.

> **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, as they reside in a different namespace to that used by your resources. 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.

Now that you understand how a style is defined, you need to learn what kind of style attributes—defined by the `<item>` element—are available.

The attributes that apply to a specific `View` (https://developer.android.com/reference/android/view/View.html) are listed in the corresponding class reference, under 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="flag"
    ... />
```

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"
    ... />
```

For a reference of all style attributes available in the Android Framework, see the `R.attr` (https://developer.android.com/reference/android/R.attr.html) reference. For the list of all available style attributes available in a particular package of the support library, see the corresponding R.attr reference. For example, for the list of style attributes available in the `support-v7` package, see the `R.attr` (https://developer.android.com/reference/android/support/v7/appcompat/R.attr.html) reference for that package. Keep in mind that not all view objects accept all the same style attributes, so you should normally refer to the specific subclass of `View` (https://developer.android.com/reference/android/view/View.html) for a list of the supported style attributes. However, if you apply a style to a view that doesn't support all of the style attributes, the view applies only those attributes that are supported and ignores the others.

As the number of available style attributes is large, you might find it useful to relate the attributes to some broad categories. The following list includes some of the most common categories:

- Text appearance styles, such as `android:textAppearanceSmall`
- Drawables, such as `android:selectableItemBackground`

You can use some style attributes to set the theme applied to a component that is based on the current theme. For example, you can use the `android:datePickerDialogTheme` attribute to set the theme for dialogs spawned from your current theme. To discover more of this kind of style attribute, look at the `R.attr` (https://developer.android.com/reference/android/R.attr.html) reference for attributes that end with `Theme`.

Some style attributes, however, are not supported by any view element and can only be applied as a theme. These style attributes apply to the entire window and not to any type of view. For example, style attributes for a theme can hide the app title, hide the status bar, or change the window's background. These kinds of style attributes don't belong to any view object.To discover these theme-only style attributes, look at the `R.attr` (https://developer.android.com/reference/android/R.attr.html) reference for attributes begin with `window`. For instance, `windowNoTitle` and `windowBackground` are style attributes that are effective only when the style is applied as a theme to an activity or app. 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">`.

You can also create custom style attributes for your app. Custom attributes, however, belong to a different namespace. For more information about creating custom attributes, see Creating a Custom View Class (https://developer.android.com/training/custom-views/create-view.html#customattr).

Applying Styles and Themes to the UI

There are several ways to set a style:

- To an individual view, by adding the `style` attribute to a View element in the XML for your layout.
- To an individual view, by passing the style resource identifier to a `View` (https://developer.android.com/reference/android/view/View.html#View(android.content.Context, android.util.AttributeSet, int, int)) constructor. This is available for apps that target Android 5.0 (API level 21) or higher.
- Or, to an entire activity or app, 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 attributes 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 don't inherit the style attributes; only the

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 app in the Android manifest. When you do so, every `View` (https://developer.android.com/reference/android/view/View.html) within the activity or app applies each attribute that it supports. For example, if you apply the `CodeFont` style from the previous examples to an activity, then the style is applied to all view elements that support the text style attributes that the style defines. Any view that doesn't support the attributes ignores them. If a view supports only some of the attributes, then it applies only those attributes.

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 uses the style named `CodeFont`. (See the sample above, in Defining Styles (#DefiningStyles).)

> **Note:** The `style` attribute doesn't use the `android:` namespace prefix.

Every framework and Support Library widget has a default style that is applied to it. Many widgets also have alternative styles available that you can apply using the style attribute. For example, by default, an instance of `ProgressBar` (https://developer.android.com/reference/android/widget/ProgressBar.html) is styled using `Widget.ProgressBar` (https://developer.android.com/reference/android/R.style.html#Widget_ProgressBar). The following alternative styles can be applied to ProgressBar:

- `Widget.ProgressBar.Horizontal` (https://developer.android.com/reference/android/R.style.html#Widget_ProgressBar_Horizontal)
- `Widget.ProgressBar.Inverse` (https://developer.android.com/reference/android/R.style.html#Widget_ProgressBar_Inverse)
- `Widget.ProgressBar.Large` (https://developer.android.com/reference/android/R.style.html#Widget_ProgressBar_Large)
- `Widget.ProgressBar.Large.Inverse` (https://developer.android.com/reference/android/R.style.html#Widget_ProgressBar_Large_Inverse)
- `Widget.Progress.Small` (https://developer.android.com/reference/android/R.style.html#Widget_ProgressBar_Smal
- `Widget.ProgressBar.Small.Inverse` (https://developer.android.com/reference/android/R.style.html#Widget_ProgressBar_Small_Inverse)

To apply the `Widget.Progress.Small` (https://developer.android.com/reference/android/R.style.html#Widget_ProgressBar_Small) style to a progress bar, you

```
<ProressBar
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    style="@android:style/Widget.ProgressBar.Small" />
```

To discover all of the alternative widget styles available, look at the `R.style` (https://developer.android.com/reference/android/R.style.html) reference for constants that begin with Widget. To discover all of the alternative widget styles available for a support library package, look at the R.style reference for fields that begin with `Widget`. For example, to view the widget styles available in the `support-v7` package, see the `R.style` (https://developer.android.com/reference/android/support/v7/appcompat/R.style.html) reference for that package. Remember to replace all underscores with periods when copying style names from the reference.

Apply a theme to an activity or app

To set a theme for all the activities of your app, 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 app, 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.Material.Light">
    <item name="android:windowBackground">@color/custom_theme_color</item>
    <item name="android:colorBackground">@color/custom_theme_color</item>
</style>
```

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

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

To discover the themes available in the Android Framework, search the `R.style` (https://developer.android.com/reference/android/R.style.html) reference for constants that begin with `Theme_`.

You can further adjust the look and format of your app by using a theme overlay. Theme overlays allow you to override some of the style attributes applied to a subset of the components styled by a theme. For example, you might want to apply a darker theme to a toolbar in an activity that uses a lighter theme. If you are using `Theme.Material.Light` as the theme for an activity, you can apply `ThemeOverlay.Material.Dark` to the toolbar using the `android:theme` attribute to modify the appearance as follows:

- Change the toolbar colors to a dark theme but preserve other style attributes, such as those relating to size.
- The theme overlay applies to any children inflated under the toolbar.

You can find a list of ThemeOverlays in the Android Framework by searching the `R.style` (https://developer.android.com/reference/android/R.style.html) reference for constants that begin with `ThemeOverlay_`.

Maintaining theme compatibility

To maintain theme compatibility with previous versions of Android, use one of the themes available in the `appcompat-v7` library. You can apply a theme to your app or activity, or you can set it as the parent, when creating your own backwards compatible themes. You can also use backwards compatible theme overlays in the Support Library. To find a list of the available themes and theme overlays, search the `R.style` (https://developer.android.com/reference/android/support/v7/appcompat/R.style.html) reference in the `android.support.v7.appcompat` package for fields that being with `Theme_` and `ThemeOverlay_`respectively.

Newer versions of Android have additional themes available to apps, and in some cases you might want to use these themes 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. 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 material theme when the app is running on Android 5.0 (API Level 21) or higher, you can place an alternative declaration for the theme in an XML file in `res/values-v21`, but make the parent theme the material theme:

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

Now use this theme like you would any other, and your app automatically switches to the material theme if it's running on Android 5.0 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 that are based on the platform version or other device configurations, see the Providing Resources (https://developer.android.com/guide/topics/resources/providing-resources.html) document.