About This Article

This article explains on how to develop android UI by applying styles and themes. It allows familiarizing on using style properties such as height, padding, font color, font size, background color, and much more and the theme-ing capabilities like setting background color, image, etc on widgets. Sample style and theme xml are provided to demonstrate the usage of styles and themes.

Scope:

This article is intended for Android developers wishing to create rich UI in android applications using styles and themes. It assumes basic knowledge of android development. This article focuses on creating styles and themes in android and therefore explaining the Java technology is out of the scope in this article.

Introduction

Styles are like Cascading Style Sheets (CSS) in web design. A style in Android is a collection of attribute/value pairs applied to a view. A style is defined in an XML resource that is separate from the XML that specifies the layout. Applying styles for entire Activity or the application are called themes. When a style is applied as a theme, every View in the Activity or application will apply each style property that it supports.

Styles

In layout XML will be in the below format for a button view

```
<Button
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textColor="#FF0000"
    android:gravity="center_vertical|center_horizontal"
    android:focusable="true"
    android:clickable="true"
    android:text="This is style String"
    />
```

To obtain the above results using a style, 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.
Given below is the sample example with <resources> must be root node. 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> </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)

```
<style name="ButtonStyle" parent="@android:style/Widget.Button">
        <item name="android:layout_width">fill_parent</item>
        <item name="android:layout_height">wrap_content</item>
        <item name="android:textColor">#FF0000</item>
        <item name="android:gravity">center_vertical|center_horizontal</item>
        <item name="android:focusable">true</item>
        <item name="android:clickable">true</item>
```

```
    </style>
```
Now redefine the button view in layout XML file as below

```
<Button
      style="@style/ButtonStyle"
      android:text="This is Style String"
      />
```
And the result will be same as shown in figure 1

Figure 1: Button View Style

Inheriting Styles

A style in android has the ability to inherit styles 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.
Initial style for a button is like this

```
<style name="ButtonChild">
     <item name="android:layout_width">fill_parent</item>
     <item name="android:layout_height">wrap_content</item>
     <item name="android:textSize">16dip</item>
     <item name="android:typeface">serif</item>
     <item name="android:textStyle">bold</item>
  </style>
```
The button will appear like as shown in below figure 2

Figure 2: Button View Style without parent

We can make this style inherit from ButtonStyle defined previously by adding the parent attribute to the <style> tag: as below

```
<style name="ButtonChild" parent="ButtonStyle">
```

Now the button appears as shown in figure 3

Figure 2: Button View Style without parent

Applying Android build-in style

An android style has the ability to inherit from the platform built-in styles defined in the android.R.style namespace.

To use this XML In the previous button style example we will set the parent of the style to be @android:style/Widget.Button.Toggle and the button appear as shown in figure 4

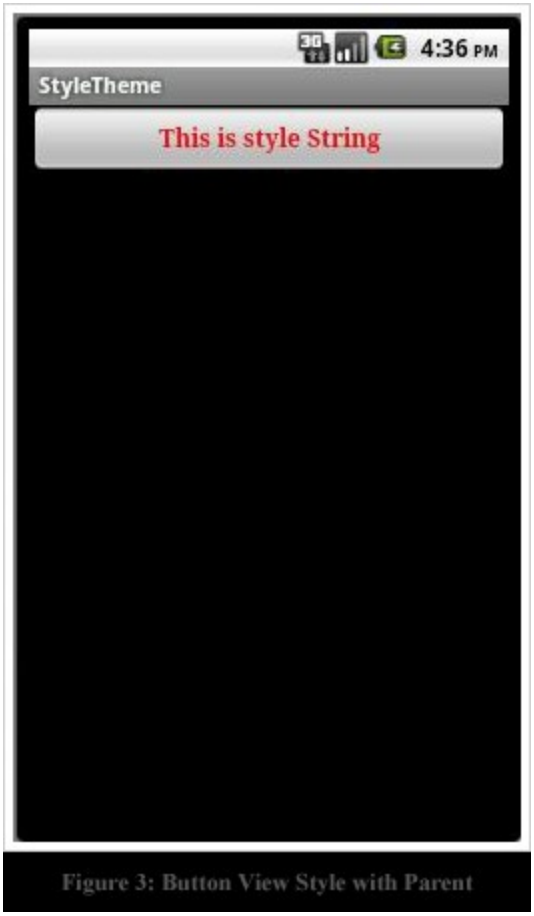

Figure 3: Button View Style with Parent

Themes

Styles can be applied as themes on an activity level or application level. If the theme applied on an activity level then all widgets within that activity will inherit from that theme.

Theme applied to an Activity

To apply Theme to activity modify the AndroidManifest.xml file by adding the android:theme attribute in the tag.

```
<activity android:name=".StyleTheme"
           android:label="@string/app_name"
                    android:theme="@style/ButtonChild">
```

Theme applied to and Application

To apply Theme to application modify the AndroidManifest.xml file by adding the android:theme attribute in the <application> tag.

```
<application android:icon="@drawable/icon"
                 android:label="@string/app_name"
                 android:theme="@style/ButtonChild">
```

The same theme can be set programmatically to an activity by calling setTheme() method in the

Activity class.

```
this.setTheme(R.style.ButtonChild);
```

And on applying theme or application output will be as shown in figure 5

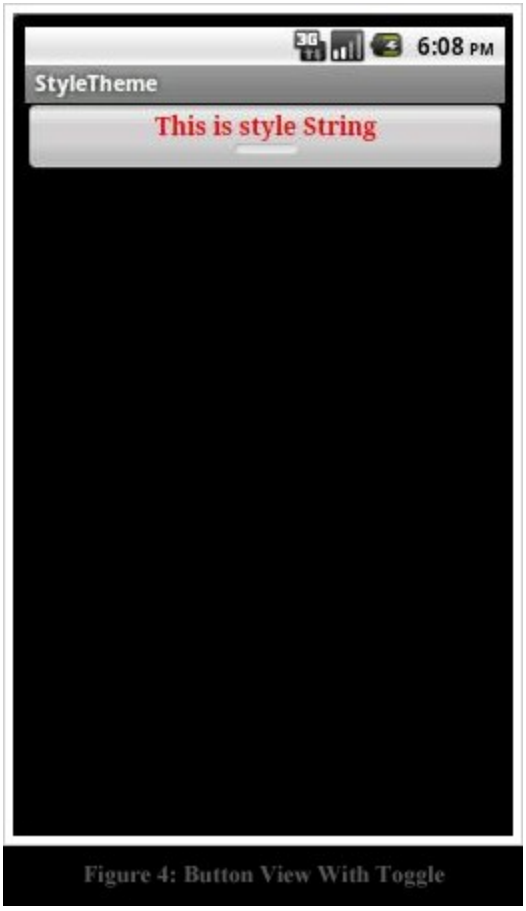

Figure 4: Button View With Toggle

Applying Android build-in Themes

There are several themes that can be added without having to write by our self. Same as android provides other built-in resources.
For example, we can use the Dialog theme to make the activity appear like a dialog box. In the AndroidManifest.xml file by adding the attribute android:theme with value Theme.Dialog

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

Also to modify the style of the build-in android theme, just add the theme as the parent of your custom theme.
For Example, modify the Theme.Dialog theme, to do so, create a style with theme.Dialog as the parent.

```
<style name="CustomDialogTheme" parent="@android:style/Theme.Dialog">>
```

As we have inherited the Dialog theme so its styles can be adjusted as required for your activity. Hence here we redefine and use CustomDialogTheme instead of theme.Dialog in AndroidManifest.xml

ref:http://developer.samsung.com/android/technical-docs/Android-UI-Using-Styles-Themes