

# Style Definition

# Introduction

- ❖ We can assign a separated style definition for each view object we are working with.
- ❖ The style definition is a collection of properties such as height, padding, font color, font size, background color and others. These properties specify the look and the format of a specific view.

# Style Definition

- ❖ The style definition is defined in a separated XML document we should save within the `res\values` folder. The style definition isn't part of the layout XML document.

# Simple Style Definition

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

```
<LinearLayout  
xmlns:android="http://schemas.android.com/apk/res/android"  
    android:orientation="vertical"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent" >  
<TextView  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:text="@string/hello" />
```

```
<TextView  
android:id="@+id/TextView01"  
android:layout_width="wrap_content"  
android:layout_height="wrap_content"  
android:text="hello earth"  
style="@style/MyImportantFont" />
```

[The Style Definition Resource](#)

```
</LinearLayout>
```

# Simple Style Definition

```
public final class R {  
    public static final class attr {  
    }  
    public static final class drawable {  
        public static final int icon=0x7f020000;  
    }  
    public static final class id {  
        public static final int TextView01=0x7f060000;  
    }  
    public static final class layout {  
        public static final int main=0x7f030000;  
    }  
    public static final class string {  
        public static final int app_name=0x7f050001;  
        public static final int hello=0x7f050000;  
    }  
    public static final class style {  
        public static final int MyImportantFont=0x7f040000;  
    }  
}
```



The Style Definition Resource

# Simple Style Definition

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

```
<resources>
```

```
  <style name="MyImportantFont"
```

The Style Definition Resource

```
    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>
```

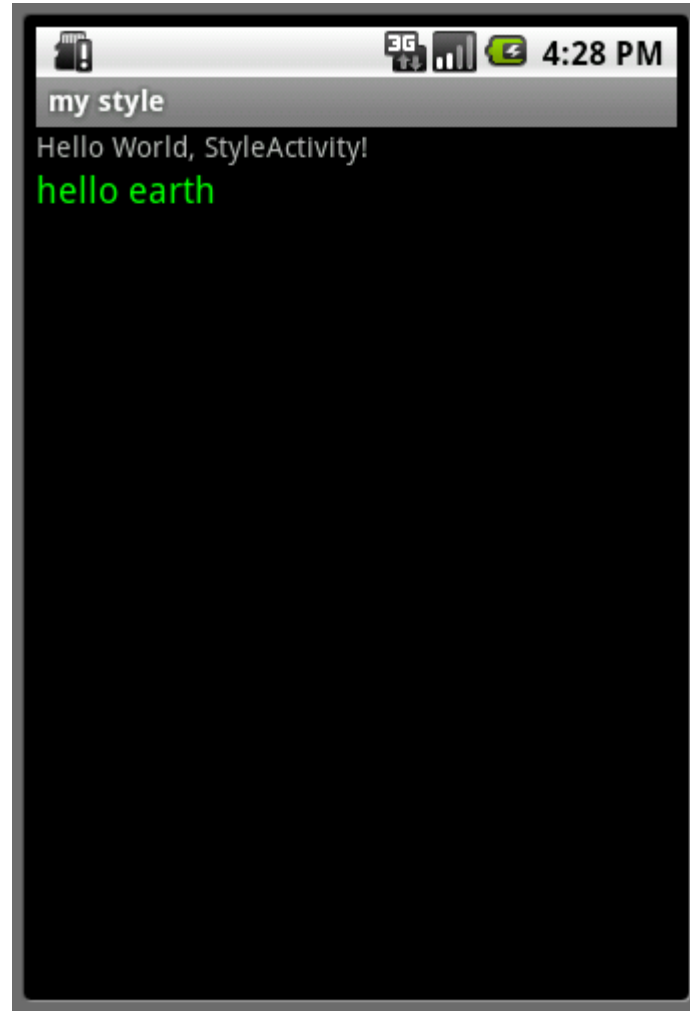
```
      <item name="android:typeface">sans</item>
```

```
    </style>
```

```
</resources>
```

The parent attribute in the <style> element is optional. It specifies the resource ID of another style from which this style should inherit properties. These inherited properties can be overridden

# Simple Style Definition



# Style Definition Inheritance

- ❖ When we define a style we can specify that it should inherit from another already defined style.
- ❖ We can then override the style definitions we want to change.



# Style Definition Inheritance

- ❖ When dealing with inheritance from a style definition defined by the android platform we should use the parent attribute.

```
<style name="RedText" parent="@android:style/TextAppearance">  
    <item name="android:textColor">#FF3300</item>  
</style>
```

# Style Definition Inheritance

- ❖ Instead of using the parent attribute for specifying the parent style (e.g. parent="ParentStyle") from which the new defined style (e.g. "TheNewStyle") extends, we can name the new defined style in the following way:

```
<style name="ParentStyle.TheNewStyle">  
    <item name="android:textColor">#00FF00</item>  
</style>
```

- ❖ We will refer this new style by writing its full qualified name:

```
@style/ParentStyle.TheNewStyle.
```

# Style Definition Inheritance

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="parentstyle">
    <item name="android:textColor">#00FF00</item>
    <item name="android:typeface">sans</item>
    <item name="android:textSize">20px</item>
  </style>
</resources>
```

# Style Definition Inheritance

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="parentstyle.jaja">
    <item name="android:textStyle">italic</item>
    <item name="android:textSize">40px</item>
  </style>
</resources>
```

# Style Definition Inheritance

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/and
roid"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"
        style="@style/parentstyle.jaja"
    />
</LinearLayout>
```

# Style Definition Inheritance

```
package com.absle;

import android.app.Activity;
import android.os.Bundle;

public class GogoActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

# Style Definition Inheritance



# Style Definition Inheritance

- ❖ We can continue the inheritance to as many levels as we want by chaining style names separated with periods.

```
<style name="GrandParentStyle.ParentStyle.ChildStyle">  
    <item name="android:textSize">8sp</item>  
</style>
```

- ❖ We will refer this new style as

```
@style/GrandParentStyle.ParentStyle.ChildStyle
```



# Style Properties

- ❖ You can find the properties that apply for each view browsing the corresponding class reference that lists all attributes supported by an XML document.
- ❖ When trying to apply a specific property the view doesn't support the view will simply ignore it.
- ❖ When applying a style definition to a ViewGroup the child view elements do not inherit the style definition properties.

# Style Properties

The screenshot shows the Android Developers website in a web browser. The page title is "TextView | Android Developers". The URL is "http://developer.android.com/reference/android/widget/TextView.html#attr\_android:typeset". The page content is for the "XML Attributes" section, specifically for the "android:autoLink" attribute. The attribute is described as: "Controls whether links such as urls and email addresses are automatically found and converted to clickable links. The default value is 'none', disabling this feature. Must be one or more (separated by '|') of the following constant values." A table lists the constants: none (0x00), web (0x01), email (0x02), phone (0x04), map (0x08), and all (0x0f). The page also shows related methods like setAutoLinkMask(int).

TextView | Android Developers

http://developer.android.com/reference/android/widget/TextView.html#attr\_android:typeset

Google

Apple Yahoo! Google Maps YouTube Wikipedia News (107) Popular

Gmail - Inbox (13) - haim... Applying Styles and Them... Android: Files TextView | Android Devel... web-safe-colors.gif 474x...

English Android.com

search developer docs Search

Home SDK Dev Guide Reference Resources Videos Blog

Filter by API Level: 7

android.text  
android.text.format  
android.text.method  
android.text.style  
android.text.util  
android.util  
android.view  
android.view.accessib  
android.view.animation  
android.view.inputmet  
android.webkit  
**android.widget**  
dalvik.bytecode  
dalvik.system  
java.awt.font  
java.beans  
java.io  
SimpleCursorAdapter  
SimpleCursorTreeAda  
SimpleExpandableList  
SlidingDrawer

## XML Attributes

**android:autoLink** Since: API Level

Controls whether links such as urls and email addresses are automatically found and converted to clickable links. The default value is "none", disabling this feature.

Must be one or more (separated by '|') of the following constant values.

Constant	Value	Description
none	0x00	Match no patterns (default).
web	0x01	Match Web URLs.
email	0x02	Match email addresses.
phone	0x04	Match phone numbers.
map	0x08	Match map addresses.
all	0x0f	Match all patterns (equivalent to web email phone map).

This corresponds to the global attribute resource symbol [autoLink](#).

### Related Methods

[setAutoLinkMask\(int\)](#)

start Band... Text... 2 W... Java ... C:\W... 5554... untikl... weba... ui.od... EN 7:21 PM

# Theme Definition

- ❖ A theme is a style definition for the entire activity or the application.
- ❖ When applying a style definition for the entire activity (or application) it will apply for each one of the views the activity (or application) includes.
- ❖ We define a theme the same way we define a simple style for a specific view.

# Theme Definition

- ❖ Some of the style properties cannot apply for views. They can apply for the entire window only. Among these properties you can find ones that allow us to hide the application title, hide the status bar, or even change the window's background.
- ❖ You can find detailed information about these properties browsing at the following URL address:  
<http://developer.android.com/reference/android/R.attr.html>.

# Theme Definition

- ❖ We can apply a theme by adding the `android:theme` attribute to the `<activity>` or `<application>` element in our android manifest file.

```
<application android:theme="@style/MyCustomTheme">
```

- ❖ Each and every view within the activity (or application) will apply each property it supports.

# Predefined Themes & Styles

- ❖ There are various predefined themes you can choose to use in your activity (or application). You can find the complete list of predefined themes available browsing at <http://developer.android.com/reference/android/R.style.html>.

# Predefined Themes & Styles

The screenshot displays the 'R.style' page on the Android Developers website. The page is titled 'R.style | Android Developers' and shows the URL 'http://developer.android.com/reference/android/R.style.html'. The left sidebar contains a 'Package Index' with 'android' selected, and 'R.style' is highlighted under the 'R' package. The main content area lists various predefined themes and styles, each with a type (int) and a description.

Type	Theme Name	Description
int	Theme_Black_NoTitleBar_Fullscreen	Variant of the black theme that has no title bar and fills the entire screen
int	Theme_Dialog	Default theme for dialog windows and activities, which is used by the <a href="#">Dialog</a> class.
int	Theme_InputMethod	Default theme for input methods, which is used by the <a href="#">InputMethodService</a> class.
int	Theme_Light	Theme for a light background with dark text on top.
int	Theme_Light_NoTitleBar	Variant of the light theme with no title bar
int	Theme_Light_NoTitleBar_Fullscreen	Variant of the light theme that has no title bar and fills the entire screen
int	Theme_Light_Panel	Default light theme for panel windows.
int	Theme_Light_WallpaperSettings	Theme for a wallpaper's setting activity that is designed to be on top of a light background.
int	Theme_NoDisplay	Default theme for activities that don't actually display a UI; that is, they finish themselves before being resumed.
int	Theme_NoTitleBar	Variant of the default (dark) theme with no title bar
int	Theme_NoTitleBar_Fullscreen	Variant of the default (dark) theme that has no title bar and fills the entire screen

# Predefined Themes & Styles

- ❖ When going over the predefined themes shown in the documentation you should change each '\_' into '.'. For example, in order to refer `Theme_Light_Panel` you should refer `Theme.Light.Panel`.



# Predefined Themes & Styles

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.abelski.samples"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon"
        android:label="@string/app_name"
        android:theme=
            "@android:style/Theme.Light.NoTitleBar.Fullscreen">
        <activity android:name=".MySelectedThemeActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name=
                    "android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-sdk android:minSdkVersion="6" />
</manifest>
```

# Predefined Themes & Styles



# Inherit Predefined Theme

- ❖ It is always possible to inherit from a predefined theme changing one or more of the inherited properties.

# Inherit Predefined Theme

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

<resources>
    <style name="MyCustomTheme"
        parent="@android:style/Theme.Light.NoTitleBar.Fullscreen">
        <item name="android:windowBackground">@drawable/pixa</item>
    </style>
</resources>
```

# Inherit Predefined Theme

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

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.abelski.samples"
    android:versionCode="1"
    android:versionName="1.0">

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

        <activity android:name=".ThemeActivity" android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    </application>

    <uses-sdk android:minSdkVersion="4" />

</manifest>
```

# Inherit Predefined Theme

