Style, Themes, and Introduction to Material Design

http://developer.android.com/training/material/index.html

Dr. Oren Mishali
What is a style in android?

• A collection of properties that specify the look and format for a view

• Can specify properties such as height, padding, font color, font size, background color, etc.

• Is defined in an XML resource that is separate from the XML that specifies the layout

Styles in Android are like CSS in web design

```xml
<p>
  This is a paragraph
</p>

p {
  color: red;
  text-align: center;
}
```
An example for applying a style

• This is a text view **before** applying a style:

```xml
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textColor="#00FF00"
    android:typeface="monospace"
    android:text="@string/hello" />
```

• And here it is afterwards:

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

All style attributes have been moved to a style definition called **CodeFont**, which we will see next.
What is a theme?

- **A theme** is a style applied to an entire Activity or application rather than an individual View.
- 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 style **CodeFont** as a theme for an Activity and then all text inside that Activity will have green monospace font.

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

```xml
<activity android:theme="@android:style/Theme.Dialog"/>
```
How to create styles?

- Create an XML file in `res/values/`
- Create a `<resources>` root element
- Create a `<style>` element, with `<item>` elements describing the properties of the style
  - Many `<style>` elements may be defined

```xml
<?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 style element is converted into a resource object and can be referenced from an XML layout via `@style`
- Remember that a style may also be used as a theme
Style inheritance

• Use the **parent** attribute to inherit properties from built-in platform styles

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

• To inherit from your own defined style (e.g., from *CodeFont*), you may use this naming style instead:

```xml
<style name="CodeFont.Red">
  <item name="android:textColor">#FF0000</item>
</style>
```
What kind of style properties are available?

- See the Javadoc of the specific view for the list of supported XML attributes
- Note that basically all attributes may be defined in a style, for example:

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

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

Note: do not forget to prefix the attribute name with the android: namespace
Some of the properties are for theme only

- See the `R.attr` reference that lists all possible XML attributes
- Attributes that begin with `window` are effective only when the style is applied as a theme, e.g.,
  - `windowNoTitle`, `windowBackground`

Note: when applying a style to a view, the view will apply only those properties that it supports and will ignore others
How to use newer themes while being backward compatible?

• Define a custom theme that uses resource selection based on platform version

• For example, this holographic theme will be used by applications running android 3.0+

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

• While a default light theme will be used in older versions:

```xml
<style name="LightThemeSelector" parent="android:Theme.Light">
...
</style>
```
Using platform styles and themes

- Android platform provides a large collection of styles and themes that you can use
- All available styles can be found in `R.style` class
- To use those styles, replace each underscore with a period, e.g., `Theme_NoTitleBar` => `@android:style/Theme.NoTitleBar`
- To understand the properties of each provided style, see `styles.xml`, and `themes.xml`

For example, what properties are used to style `Theme.Dialog`?
Material Design

An Introduction

http://developer.android.com/training/material/index.html
Want your app to look that way?

https://www.youtube.com/watch?v=Q8TXgCzxEnw
What is material design?

• “A comprehensive guide for visual, motion, and interaction design across platforms and devices”

• Detailed design guidelines are described in the material design specification

• The guidelines are quite abstract...

(Will talk about that later on)

Note: using material design requires Android 5.0 (API level 21), however many features are supported on earlier versions. See Maintaining Compatibility.
How to apply material design to my app?

• Specify a style that inherits from `android:Theme.Material`
• Your layouts and widgets should conform to the material design guidelines, including
  • Specifying elevation for views
  • Specialized widgets for lists and cards
  • Customized animations
  • Touch feedback

```xml
<!-- res/values/styles.xml -->
<resources>
  <!-- your theme inherits from the material theme -->
  <style name="AppTheme" parent="android:Theme.Material">
    <!-- theme customizations -->
  </style>
</resources>
```
Material theme – easy color customization (1)

• Material design themes allow easy customization of the app’s color palate
• Inherit from a base material theme and define colors that fit your brand (see example below)

```xml
<resources>
    <!-- inherit from the material theme -->
    <style name="AppTheme" parent="android:Theme.Material">
        <!-- Main theme colors -->
        <!-- your app branding color for the app bar -->
        <item name="android:colorPrimary">@color/primary</item>
        <!-- darker variant for the status bar and contextual app bars -->
        <item name="android:colorPrimaryDark">@color/primary_dark</item>
        <!-- theme UI controls like checkboxes and text fields -->
        <item name="android:colorAccent">@color/accent</item>
    </style>
</resources>
```

See more material themes in `R.style`
Material theme – easy color customization (2)

• Note that even the color of a variety of screen elements may be customized, such as the **status bar**
Cards

“A card is a sheet of material that serves as an entry point to more detailed information. A card could contain a photo, text, and a link about a single subject.”

https://www.google.com/design/spec/components/cards.html#

- Cards are implemented using CardView widgets.
- Use card_view:cardElevation attribute for creating a card shadow
- See more details here
Lists

• Use the RecyclerView widget to create complex list in your app

• The RecyclerView widget is a more advanced and flexible version of ListView
  • Flexible layout managers to position elements within the list
  • Animations for adding and removing items

• See more details here
Elevation and shadows

• Material design introduces elevation for UI elements

• The elevation of a view is represented by the Z property and determines the visual appearance of its shadow

• Elevation is also useful to create animations where widgets temporarily rise above the view plane when performing some action

• Key attributes are `android:elevation` and `android:translationZ`
  • `android:elevation` refers to the elevation’s static component
  • `android:translationZ` refers to the dynamic components used for animations
Customizing view shadows

- The bounds of a view's background drawable determine the default shape of its shadow.

```xml
    <solid android:color="#42000000" />
    <corners android:radius="5dp" />
</shape>
```

- The view will have a shadow with rounded corners, since the background drawable defines the view's outline.
- Providing a custom outline overrides the default shape of a view's shadow.
Defining custom animations

• The material theme provides some default animations for buttons and activity transitions

• Android 5.0 (API level 21) and above lets you customize these animations and create new ones such as
  • Touch feedback
  • Activity transitions
  • Reveal animations (show/hide)

• See this guide for more details and code examples
So do we understand how to apply material design?

It feels like there is a gap between design guidelines and implementation.
Chips

- “A chip may contain a photo, short title, and brief information”
- “Chips can be used for various types of entities, including free form text, predefined text, rules, or contacts. Chips may also contain icons”
- “Touching a chip opens a full detailed view (either in a card or full screen) or a menu of options related to that chip”
- “Chips can be deletable or non-deletable. Display a delete icon if a chip is deletable”
- https://www.google.com/design/spec/components/chips.html#

But how to implement a Chip?
Sandra is asking for help

Android material chips

I want to implement an autocomplete edittext with chips in my application and I want to do it in a way that it's done here: material design chips. First I would like to ask if there is some kind of widget (maybe as part of the new support library) or a solution that I can use for easy implementation. (I know that this question has been asked before but I just want to know if something changed in the meantime). Also I found this library, but I don't know how can I use it (and can I use it) for autocompletion of my sets of data... Has anyone worked with this library before and can share their experience?

Any help would be appreciated!

2 unofficial user libraries are suggested

2 Answers

- there is also a new library for material chips!
  1 share improve this answer

- Check out this library! I think it may suit your needs.
  2 share improve this answer
Steppers

• “Steppers convey progress through numbered steps. They may also be used for navigation”

• “Steppers display progress through a sequence by breaking it up into multiple logical and numbered steps”

• https://www.google.com/design/spec/components/steppers.html#
From the spec for steppers

Step height: 72dp
Icon top, bottom, and left padding: 24dp
Icon right padding: 8dp

Step height: 72dp
Icon left and right padding: 8dp
Label right padding: 8dp

But how to implement a Stepper?
Another developer is asking for help

Material design stepper control

Could someone give me an idea how to begin implementing vertical, non-linear stepper control described in the Android Material Design guide here:

http://www.google.com/design/spec/components/stepper

1 Answer

- You can check this library, however, this is still in development.
- Just extend the `mobileStepperSimple` class and implement the methods `init`, `onFinished`.
- You can add the steppers as fragments by extending the `stepperFragment` and implement `onNextButtonHandle` to handle next button click.
- Check the demo for more usage.
- Any contributions and optimization will be helpful.
Final remarks

• Material design apps really have the potential to look great

• However it seems like building such an app requires a considerable effort