Learning Material Design

Google's Material Design language has taken the development and design worlds by storm. Now available on many more platforms than Android, Material Design uses color, light, and movement to not only generate beautiful interfaces, but also to provide intuitive navigation for the user.

Learning Material Design will teach you the fundamental theories of Material Design, using code samples to put these theories into practice.

Focusing primarily on Android Studio, you'll create mobile interfaces using the most widely used and powerful material components. Each section will introduce the relevant Java classes and APIs required to implement these components. We then move into animation and transition, possibly Material Design's most powerful concepts, allowing complex hierarchies to be displayed simply and stylishly.

With all the basic technologies and concepts mastered, the book concludes by showing you how these skills can be applied to other platforms, in particular web apps, using the powerful Polymer library.

Who this book is written for
This book is ideal for developers and designers who are interested in implementing Material Design in their mobile and web apps. No prior knowledge or experience of Material Design is required, but some familiarity with procedural languages, such as Java, and markup languages, such as HTML, will provide an advantage.

What you will learn from this book
- Implement Material Design on both mobile and web platforms that work on older handsets and browsers
- Design stylish layouts with Material Theme
- Create and manage cards, lists, and grids
- Design and implement sliding drawers for seamless navigation
- Coordinate components to work together
- Animate widgets and create transitions and animated program flow
- Use web frameworks to bring Material Design to your web pages

In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'Getting Started with Material Design'
- A synopsis of the book’s content
- More information on Learning Material Design
About the Author

Kyle Mew has been programming since the early eighties and has written for several technology websites. He has also written three radio plays and two other books on Android development.
Welcome to Learning Material Design, a comprehensive guide to the latest and hottest design philosophy for mobile and web applications. More than just a design language, Material Design represents a powerful shift in how modern digital interfaces look and behave. Based largely on traditional design principles, Material Design brings a tactile look and feel to apps and pages, giving screen elements physical properties such as fluid, realistic motion and the ability to depict a third dimension using shadows.

Covering all major design principles and guidelines and including enough of the technologies and code required to implement them, the book is designed so that you can get started with building your own material interfaces from the very beginning.

All the commonly used material components, such as cards and sliding drawers, are covered in terms of both design guidelines and code structures. This element-specific approach is coupled with details on how Material Design can be applied to interfaces in general, and how to use these guidelines to create material transitions and navigation processes.

Although concentrating largely on mobile interface design and using the powerful Android Studio development environment, the latter part of the book focuses on how the principles learned earlier can be just as easily applied to designs of web and desktop interfaces with a number of helpful and simple-to-use CSS frameworks, particularly Materialize and Material Design Lite.

This book is only the beginning of a journey into what may well become one of the most persistent digital design paradigms we have yet seen. But by the end, you will have learned not only the design theory behind materials, but also enough of technical know-how to put what you have learned into practice and be in a position to create or convert Material Design applications on your own.
Preface

What this book covers

Chapter 1, Getting Started with Material Design, introduces some of the basic precepts of Material Design, but concentrates largely on how to set up a development environment and create a simple "Hello World" app. This includes an introduction to the material theme and palette.

Chapter 2, Building a Mobile Layout, is where we concentrate on some fundamental processes in designing an Android interface, such as the content hierarchy and how components are positioned and scaled within it. The second portion of this chapter covers support libraries and how these can help us make Material Design backward compatible.

Chapter 3, Common Components, covers the most frequently used mobile material components, such as app bars, menus, and modal dialogs, along with the creation of action icons for menus.

Chapter 4, Sliding Drawers and Navigation, explains the typical material navigation techniques, in particular, the navigation menu and sliding drawers.

Chapter 5, Lists, Cards, and Data, is where we see how the recycler view can be used to organize data in the form of a list, and how separate fields of mixed media can be applied to the card view widget.

Chapter 6, Animations and Transitions, covers transition from one screen to another, including hide and reveal animations and how components that are shared across screens are animated.

Chapter 7, Material on Other Devices, is where we look at how Material Design is applied to the Android TV and Wear platforms.

Chapter 8, Material Web Frameworks, takes us on a brief tour of one of the most commonly used technologies for applying Material Design to web pages. This is done using ready-made CSS and JavaScript frameworks.

Chapter 9, The Materialize Framework, delves deeper into the Materialize web frameworks, demonstrating how common components, animations, and navigation are achieved.

Chapter 10, Material Design Lite, is the final chapter. It covers the most popular material framework—Material Design Lite. As in the previous chapter, we explore the most commonly used components and features.
Google first announced Material Design at their I/O conference in the summer of 2014, and it has since gone on to create quite a storm among developers and designers alike. Originally a formalization and expansion of the Google Now UI, Material Design has grown (and is still growing) into a comprehensive and systematic set of design philosophies.

Material can be thought of as something like smart paper. Like paper, it has surfaces and edges that reflect light and cast shadows, but unlike paper, material has properties that real paper does not, such as its ability to move, change its shape and size, and merge with other material. Despite this seemingly magical behavior, material should be treated like a physical object with a physicality of its own.

Material can be seen as existing in a three-dimensional space, and it is this that gives its interfaces a reassuring sense of depth and structure. Hierarchies become obvious when it is instantly clear whether an object is above or below another.
Based largely on age-old principles taken from color theory, animation, traditional print design, and physics, Material Design provides a virtual space where developers can use surface and light to create meaningful interfaces and movement to design intuitive user interactions.

This book covers both the design principles and the tools required to create the elegant and responsive interfaces that have made Material Design so popular and successful. The journey begins with the Android SDK and Android Studio, which we will use to build working material UIs for a variety of form factors, ranging from tiny wearable devices to large flat screen TVs. This will provide a solid foundation for the design principles behind the subject, and how mobile interfaces are defined using XML and then brought to life with Java.

With the aspects of design covered, the book concludes by exploring the different guidelines and tools required to build material web apps. There are several subtle differences in the design rules, and the possible absence of a touch-screen on desktop computers requires a different approach to user interaction. More significantly, when building web pages, we will need to work in HTML, and to implement material, we will need a tool called Polymer and its associate, Paper Elements.

With the journey outlined, we will begin by taking a quick look at the physical properties of material, and then we'll dive straight in and produce our first interface. To do this, we will need to download and configure Android Studio and SDK tools, as well as create a device emulator to compile and run our apps on. With our development environment in place, we can then take a look at the material themes available as default on recent Android versions.

In this chapter, you will learn how to:

- Understand the physical properties of material
- Install Android Studio
- Configure the Android SDK
- Install Google support libraries
- Create a virtual device
- Set up a real device for development
- Create a hello world app in Android Studio
- Create an Android style with XML
- Customize the material theme
- Apply a material palette
Material properties

As mentioned in the introduction, material can be thought of as being bound by physical laws. There are things it can do and things it cannot. It can split apart and heal again, and change color and shape, but it cannot occupy the same space as another sheet of material or rotate around two of its axes. We will be dealing with these properties throughout the book, but it is a good idea to begin with a quick look at the things material can and can’t do.

The third dimension is fundamental when it comes to material. This is what gives the user the illusion that they are interacting with something more tangible than a rectangle of light. The illusion is generated by the widening and softening of shadows beneath material that is closer to the user. Material exists in virtual space, but a space that, nevertheless, represents the real dimensions of a phone or tablet. The x axis can be thought of as existing between the top and bottom of the screen, the y axis between the right and left edges, and the z axis confined to the space between the back of the handset and the glass of the screen. It is for this reason that material should not rotate around the x or y axes, as this would break the illusion of a space inside the phone.

The basic laws of the physics of material are outlined, as follows, in the form of a list:

- All material is 1 dp thick (along the z axis).
- Material is solid, only one sheet can exist in one place at a time and material cannot pass through other material. For example, if a card needs to move past another, it must move over it.
- Elevation, or position along the z axis, is portrayed by shadow, with higher objects having wider, softer shadows.
- The z axis should be used to prompt interaction. For example, an action button rising up toward the user to demonstrate that it can be used to perform some action.
- Material does not fold or bend.
- Material cannot appear to rise higher than the screen surface.
- Material can grow and shrink along both x and y axes.
- Material can move along any axis.
- Material can be spontaneously created and destroyed, but this must not be without movement. The arrivals and departures of material components must be animated. For example, a card growing from the point that it was summoned from or sliding off the screen when dismissed.
- A sheet of material can split apart anywhere along the x or y axes, and join together again with its original partner or with other material.
Getting Started with Material Design

This covers the basic rules of material behavior but we have said nothing of its content. If material can be thought of as smart paper, then its content can only be described as smart ink. The rules governing how ink behaves are a little simpler:

- Material content can be text, imagery, or any other form of visual digital content
- Content can be of any shape or color and behaves independently from its container material
- It cannot be displayed beyond the edges of its material container
- It adds nothing to the thickness (z axis) of the material it is displayed on

Setting up a development environment

The Android development environment consists mainly of two distinct components: the SDK, which provides the code libraries behind Android and Android Studio, and a powerful code editor that is used for constructing and testing applications for Android phones and tablets, Wear, TV, Auto, Glass, and Cardboard. Both these components can both be downloaded as a single package from http://developer.android.com/sdk/index.html.

Installing Android Studio

The installation is very straightforward. Run the Android Studio bundle and follow the on-screen instructions, installing HAXM hardware acceleration if prompted, and selecting all SDK components, as shown here:

![Choose Components](image)
Android Studio is dependent on the **Java JDK**. If you have not previously installed it, this will be detected while you are installing Android Studio, and you will be prompted to download and install it. If for some reason it does not, it can be found at [http://www.oracle.com/technetwork/java/javase/downloads/index.html](http://www.oracle.com/technetwork/java/javase/downloads/index.html), from where you should download the latest version.

This is not quite the end of the installation process. There are still some SDK components that we will need to download manually before we can build our first app. As we will see next, this is done using the **Android SDK Manager**.

### Configuring the Android SDK

People often refer to Android versions by name, such as Lollipop, or an identity number, such as 5.1.1. As developers, it makes more sense to use the API level, which in the case of Android 5.1.1 would be API level 22. The SDK provides a platform for every API level since API level 8 (Android 2.2). In this section, we will use the SDK Manager to take a closer look at Android platforms, along with the other tools included in the SDK.

Start a new Android Studio project or open an existing one with the minimum SDK at 21 or higher. You can then open the SDK manager from the menu via **Tools | Android | SDK Manager** or the matching icon on the main toolbar.

The Android SDK Manager can also be started as a stand alone program. It can be found in the `/Android/sdk` directory, as can the **Android Virtual Device (AVD) manager**.
As can be seen in the preceding screenshot, there are really three main sections in the SDK:

- A **Tools** folder
- A collection of platforms
- An **Extras** folder

All these require a closer look. The **Tools** directory contains exactly what it says, that is, tools. There are a handful of these but the ones that will concern us are the SDK manager that we are using now, and the AVD manager that we will be using shortly to create a virtual device.

Open the **Tools** folder. You should find the latest revisions of the SDK tools and the SDK Platform-tools already installed. If not, select these items, along with the latest Build-tools, that is, if they too have not been installed.

> These tools are often revised, and it is well worth it to regularly check the SDK manager for updates.

When it comes to the platforms themselves, it is usually enough to simply install the latest one. This does not mean that these apps will not work on or be available to devices running older versions, as we can set a minimum SDK level when setting up a project, and along with the use of support libraries, we can bring Material Design to almost any Android device out there.

If you open up the folder for the latest platform, you will see that some items have already been installed. Strictly speaking, the only things you need to install are the SDK platform itself and at least one system image. System images are copies of the hard drives of actual Android devices and are used with the AVD to create emulators. Which images you use will depend on your system and the form factors that you are developing for. In this book, we will be building apps for phones and tablets, so make sure you use one of these at least.

> Although they are not required to develop apps, the documentation and samples packages can be extremely useful.

At the bottom of each platform folder are the Google APIs and corresponding system images. Install these if you are going to include Google services, such as Maps and Cloud, in your apps. You will also need to install the Google support libraries from the **Extras** directory, and this is what we will cover next.
The *Extras* folder contains various miscellaneous packages with a range of functions. The ones you are most likely to want to download are listed as follows:

- Android support libraries are invaluable extensions to the SDK that provide APIs that not only facilitate backwards compatibility, but also provide a lot of extra components and functions, and most importantly for us, the design library. As we are developing on Android Studio, we need only install the **Android Support Repository**, as this contains the **Android Support Library** and is designed for use with Android.

- The **Google Play** services and **Google Repository** packages are required, along with the Google APIs mentioned a moment ago, to incorporate Google Services into an application.

- You will most likely need the **Google USB Driver** if you are intending to test your apps on a real device. How to do this will be explained later in this chapter.

- The HAXM installer is invaluable if you have a recent Intel processor. Android emulators can be notoriously slow, and this hardware acceleration can make a noticeable difference.

Once you have downloaded your selected SDK components, depending on your system and/or project plans, you should have a list of installed packages similar to the one shown next:

<table>
<thead>
<tr>
<th>Name</th>
<th>API</th>
<th>Rev.</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tools</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Android SDK Tools</td>
<td>24.3.3</td>
<td></td>
<td>Installed</td>
</tr>
<tr>
<td>Android SDK Platform-tools</td>
<td>22</td>
<td></td>
<td>Installed</td>
</tr>
<tr>
<td>Android SDK Build-tools</td>
<td>22.0.1</td>
<td></td>
<td>Installed</td>
</tr>
<tr>
<td>Android 5.1.1 (API 22)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Documentation for Android SDK</td>
<td>22</td>
<td>1</td>
<td>Installed</td>
</tr>
<tr>
<td>SDK Platform</td>
<td>22</td>
<td>2</td>
<td>Installed</td>
</tr>
<tr>
<td>Samples for SDK</td>
<td>22</td>
<td>6</td>
<td>Installed</td>
</tr>
<tr>
<td>Intel x86 Atom_64 System Image</td>
<td>22</td>
<td>1</td>
<td>Installed</td>
</tr>
<tr>
<td>Google APIs</td>
<td>22</td>
<td>1</td>
<td>Installed</td>
</tr>
<tr>
<td>Google APIs Intel x86 Atom System Image</td>
<td>22</td>
<td>1</td>
<td>Installed</td>
</tr>
<tr>
<td>Extras</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Android Support Repository</td>
<td>15</td>
<td></td>
<td>Installed</td>
</tr>
<tr>
<td>Google Play services</td>
<td>25</td>
<td></td>
<td>Installed</td>
</tr>
<tr>
<td>Google Repository</td>
<td>19</td>
<td></td>
<td>Installed</td>
</tr>
<tr>
<td>Google USB Driver</td>
<td>11</td>
<td></td>
<td>Installed</td>
</tr>
<tr>
<td>Intel x86 Emulator Accelerator (HAXM installer)</td>
<td>5.3</td>
<td></td>
<td>Installed</td>
</tr>
</tbody>
</table>
Getting Started with Material Design

The SDK is finally ready, and we can start developing material interfaces. All that is required now is a device to test it on. This can, of course, be done on an actual device, but generally speaking, we will need to test our apps on as many devices as possible. Being able to emulate Android devices allows us to do this.

Emulating Android devices

The AVD allows us to test our designs across the entire range of form factors. There are an enormous number of screen sizes, shapes, and densities around. It is vital that we get to test our apps on as many device configurations as possible. This is actually more important for design than it is for functionality. An app might operate perfectly well on an exceptionally small or narrow screen, but not look as good as we had wanted, making the AVD one of the most useful tools available to us. This section covers how to create a virtual device using the AVD Manager.

The AVD Manager can be opened from within Android Studio by navigating to Tools | Android | AVD Manager from the menu or the corresponding icon on the toolbar. Here, you should click on the Create Virtual Device... button.

The easiest way to create an emulator is to simply pick a device definition from the list of hardware images and keep clicking on Next until you reach Finish. However, it is much more fun and instructive to either clone and edit an existing profile, or create one from scratch.

Click on the New Hardware Profile button. This takes you to the Configure Hardware Profile window where you will be able to create a virtual device from scratch, configuring everything from cameras and sensors, to storage and screen resolution. When you are done, click on Finish and you will be returned to the hardware selection screen where your new device will have been added:
As you will have seen from the **Import Hardware Profiles** button, it is possible to download system images for many devices not included with the SDK. Check the developer sections of device vendor's websites to see which models are available.

So far, we have only configured the hardware for our virtual device. We must now select all the software it will use. To do this, select the hardware profile you just created and press **Next**. In the following window, select one of the system images you installed earlier and press **Next** again. This takes us to the **Verify Configuration** screen where the emulator can be fine-tuned. Most of these configurations can be safely left as they are, but you will certainly need to play with the scale when developing for high density devices. It can also be very useful to be able to use a real SD card. Once you click on **Finish**, the emulator will be ready to run.

An emulator can be rotated through 90 degrees with left **Ctrl** + **F12**. The menu can be called with **F2**, and the back button with **ESC**. Keyboard commands to emulate most physical buttons, such as call, power, and volume, and a complete list can be found at [http://developer.android.com/tools/help/emulator.html](http://developer.android.com/tools/help/emulator.html).

Android emulators are notoriously slow, during both loading and operating, even on quite powerful machines. The Intel hardware accelerator we encountered earlier can make a significant difference. Between the two choices offered, the one that you use should depend on how often you need to open and close a particular emulator. More often than not, taking advantage of your GPU is the more helpful of the two. Apart from this built-in assistance, there are a few other things you can do to improve performance, such as setting lower pixel densities, increasing the device's memory, and building the website for lower API levels. If you are comfortable doing so, set up exclusions in your anti-virus software for the Android Studio and SDK directories.

There are several third-party emulators, such as Genymotion, that are not only faster, but also behave more like real devices.

The slowness of Android emulators is not necessarily a big problem, as most early development needs only one device, and real devices suffer none of the performance issues found on emulators. As we shall see next, real devices can be connected to our development environment with very little effort.
Connecting a real device

Using an actual physical device to run and test applications does not have the flexibility that emulators provide, but it does have one or two advantages of its own. Real devices are faster than any emulator, and you can test features unavailable to a virtual device, such as accessing sensors, and making and receiving calls.

There are two steps involved in setting up a real phone or tablet. We need to set developer options on the handset and configure the USB connection with our development computer:

1. To enable developer options on your handset, navigate to **Settings | About phone**. Tap on **Build number** 7 times to enable **Developer options**, which will now be available from the previous screen.
2. Open this to enable **USB debugging** and **Allow mock locations**.
3. Connect the device to your computer and check that it is connected as a **Media device (MTP)**.

Your handset can now be used as a test device. Depending on your we need only install the Google USB. Connect the device to your computer with a USB cable, start Android Studio, and open a project. Depending on your setup, it is quite possible that you are already connected. If not, you can install the Google USB driver by following these steps:

1. From the Windows start menu, open the device manager.
2. Your handset can be found under **Other Devices** or **Portable Devices**. Open its **Properties** window and select the **Driver** tab.
3. Update the driver with the Google version, which can be found in the *sdk\extras\google\usb_driver* directory.

An application can be compiled and run from Android Studio by selecting **Run 'app'** from the **Run** menu, pressing **Shift + F10**, or clicking on the green play icon on the toolbar. Once the project has finished building, you will be asked to confirm your choice of device before the app loads and then opens on your handset.
With a fully set up development environment and devices to test on, we can now start taking a look at Material Design, beginning with the material theme that is included as the default in all SDKs with APIs higher than 21.

**The material theme**

Since API level 21 (Android 5.0), the material theme has been the built-in user interface. It can be utilized and customized, simplifying the building of material interfaces. However, it is more than just a new look; the material theme also provides the automatic touch feedback and transition animations that we associate with Material Design.

To better understand Android themes and how to apply them, we need to understand how Android styles work, and a little about how screen components, such as buttons and text boxes, are defined.

Most individual screen components are referred to as widgets or views. Views that contain other views are called view groups, and they generally take the form of a layout, such as the relative layout we will use in a moment.
An **Android style** is a set of graphical properties defining the appearance of a particular screen component. Styles allow us to define everything from font size and background color, to padding elevation, and much more. An Android theme is simply a style applied across a whole screen or application. The best way to understand how this works is to put it into action and apply a style to a working project. This will also provide a great opportunity to become more familiar with Android Studio.

### Applying styles

Styles are defined as XML files and are stored in the resources (res) directory of Android Studio projects. So that we can apply different styles to a variety of platforms and devices, they are kept separate from the layout code. To see how this is done, start a new project, selecting a minimum SDK of 21 or higher, and using the blank activity template. To the left of the editor is the project explorer pane. This is your access point to every branch of your project.

Take a look at the `activity_main.xml` file, which would have been opened in the editor pane when the project was created. At the bottom of the pane, you will see a **Text** tab and a **Design** tab. It should be quite clear, from examining these, how the XML code defines a text box (**TextView**) nested inside a window (**RelativeLayout**). Layouts can be created in two ways: textually and graphically. Usually, they are built using a combination of both techniques. In the design view, widgets can be dragged and dropped to form layout designs. Any changes made using the graphical interface are immediately reflected in the code, and experimenting with this is a fantastic way to learn how various widgets and layouts are put together. We will return to both these subjects in detail later on in the book, but for now, we will continue with styles and themes by defining a custom style for the text view in our Hello world app.
Open the res node in the project explorer; you can then right-click on the values node and select the New | Values resource file from the menu. Call this file my_style and fill it out as follows:

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="myStyle">
    <item name="android:layout_width">match_parent</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:elevation">4dp</item>
    <item name="android:gravity">center_horizontal</item>
    <item name="android:padding">8dp</item>
    <item name="android:background">#e6e6e6</item>
    <item name="android:textSize">32sp</item>
    <item name="android:textColor">#727272</item>
  </style>
</resources>
```

This style defines several graphical properties, most of which should be self-explanatory with the possible exception of gravity, which here refers to how content is justified within the view. We will cover measurements and units later in the book, but for now, it is useful to understand dp and sp:

- **Density-independent pixel (dp):** Android runs on an enormous number of devices, with screen densities ranging from 120 dpi to 480 dpi and more. To simplify the process of developing for such a wide variety, Android uses a virtual pixel unit based on a 160 dpi screen. This allows us to develop for a particular screen size without having to worry about screen density.

- **Scale-independent pixel (sp):** This unit is designed to be applied to text. The reason it is scale-independent is because the actual text size on a user’s device will depend on their font size settings.

To apply the style we just defined, open the activity_main.xml file (from res/layouts, if you have closed it) and edit the TextView node so that it matches this:

```xml
<TextView
  style="@style/myStyle"
  android:text="@string/hello_world" />
```
The effects of applying this style can be seen immediately from the design tab or preview pane, and having seen how styles are applied, we can now go ahead and create a style to customize the material theme palette.

**Customizing the material theme**

One of the most useful features of the material theme is the way it can take a small palette made of only a handful of colors and incorporate these colors into every aspect of a UI. Text and cursor colors, the way things are highlighted, and even system features such as the status and navigation bars can be customized to give our apps brand colors and an easily recognizable look.

The use of color in Material Design is a topic in itself, and there are strict guidelines regarding color, shade, and text, and these will be covered in detail later in the book. For now, we will just look at how we can use a style to apply our own colors to a material theme.

So as to keep our resources separate, and therefore easier to manage, we will define our palette in its own XML file. As we did earlier with the `my_style.xml` file, create a new values resource file in the `values` directory and call it `colors`. Complete the code as shown next:

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <color name="primary">#FFC107</color>
  <color name="primary_dark">#FFA000</color>
  <color name="primary_light">#FFECB3</color>
  <color name="accent">#03A9F4</color>
  <color name="text_primary">#212121</color>
  <color name="text_secondary">#727272</color>
  <color name="icons">#212121</color>
  <color name="divider">#B6B6B6</color>
</resources>
```

In the gutter to the left of the code, you will see small, colored squares. Clicking on these will take you to a dialog with a color wheel and other color selection tools for quick color editing.
We are going to apply our style to the entire app, so rather than creating a separate file, we will include our style in the theme that was set up by the project template wizard when we started the project. This theme is called AppTheme, as can be seen by opening the res/values/styles/styles.xml (v21) file. Edit the code in this file so that it looks like the following:

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="AppTheme" parent="android:Theme.Material.Light">
    <item name="android:colorPrimary">@color/primary</item>
    <item name="android:colorPrimaryDark">@color/primary_dark</item>
    <item name="android:colorAccent">@color/accent</item>
    <item name="android:textColorPrimary">@color/text_primary</item>
    <item name="android:textColor">@color/text_secondary</item>
  </style>
</resources>
```

Being able to set key colors, such as colorPrimary and colorAccent, allows us to incorporate our brand colors throughout the app, although the project template only shows us how we have changed the color of the status bar and app bar. Try adding radio buttons or text edit boxes to see how the accent color is applied. In the following figure, a timepicker replaces the original text view:

The XML for this looks like the following lines:

```xml
<TimePicker
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_centerHorizontal="true" />
```
For now, it is not necessary to know all the color guidelines. Until we get to them, there is an online material color palette generator at http://www.materialpalette.com/ that lets you try out different palette combinations and download color XML files that can just be cut and pasted into the editor.

With a complete and up-to-date development environment constructed, and a way to customize and adapt the material theme, we are now ready to look into how material specific widgets, such as card views, are implemented.

**Summary**

The Android SDK, Android Studio, and AVD comprise a sophisticated development toolkit, and even setting them up is no simple task. But, with our tools in place, we were able to take a first look at one of Material Design's major components: the material theme. We have seen how themes and styles relate, and how to create and edit styles in XML. Finally, we have touched on material palettes, and how to customize a theme to utilize our own brand colors across an app.

With these basics covered, we can move on to explore Material Design further, and in the next chapter, we will look at layouts and material components in greater detail.
Where to buy this book

You can buy Learning Material Design from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.