

Android 3: First Project, Hello World, and Exploring the Development Environment



Introduction

- This unit is a hybrid
- It surveys the Android Studio development environment
- It then covers some of the introductory material in the online tutorials
- It then discusses some of the code aspects of an Android app which you can explore through the studio interface

- The unit is intended as an introduction, not as a comprehensive treatment of these topics
- It is admittedly repetitive because it presents both the tutorial and independent explanations of some topics
- The repetition isn't harmful
- If the basics sink in, more substantive material which comes later will be easier to understand

- The unit is based on creating a new project in Android Studio and accepting the defaults
- What you will find is that Android Studio creates a “Hello World” by default
- Looking at the “Hello World” app components is the beginning of understanding what goes into an Android app
- It is also the starting point for making modifications and making a new app

- Reviewing the “Hello World” app is intended to answer the questions that might spring to the mind of a programmer who is seeing Android Studio and app development for the first time

Outline

- 3.1 The Android Studio Interface, a First Pass
- 3.2 Apps and Manifest Files
- 3.3 The Explorer Pane—Before the Resources
- 3.4 The Explorer Pane—Resources
- 3.5 From the Tutorials—Building Your First App: Creating an Android Project
- 3.6 The Basic Pieces of “Hello World”
- 3.7 From the Tutorials—Building Your First App: Running Your First Application

3.1 The Android Studio Interface, a First Pass

My Application - [C:\Users\kascott\AndroidStudioProjects\MyApplication] - [app] - ...app\src\main\java\com\example\kascott\myapplication\MainActivity.java - Android Studio 2.1.3

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

MyApplication app src main java com example kascott myapplication MainActivity

Android MainActivity.java x AndroidManifest.xml x

```
package com.example.kascott.myapplication;

import ...

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Android Monitor

0a0ebf0d [UNAUTHORIZED] No Debuggable Applications

logcat Monitors → Verbose Regex Show only selected application

TODO Android Monitor Terminal Messages Event Log Gradle Console

Gradle build finished in 14s 206ms (20 minutes ago) 14:1 CRLF+ UTF-8+ Context: <no context>

- Things to notice initially, from the top:
- The menu (File, Build, Run)
- The tool bar (the green arrowhead for Run)
- The third row down gives you explorer capabilities through the folders belonging to the project
- The fourth row down shows a box indicating “Android” and tabs for component files in the editor pane below
- Take a look:

My Application - [C:\Users\kascott\AndroidStudioProjects\MyApplication] - [app] - ...app\src\main\java\com\example\kascott\myapplication\MainActivity.java - Android Studio 2.1.3

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

MyApplication app src main java com example kascott myapplication MainActivity

Android MainActivity.java x AndroidManifest.xml x

```
package com.example.kascott.myapplication;

import ...

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Android Monitor

0a0ebf0d [UNAUTHORIZED] No Debuggable Applications

logcat Monitors → Verbose Regex Show only selected application

TODO Android Monitor Terminal Messages Event Log Gradle Console

Gradle build finished in 14s 206ms (20 minutes ago) 14:1 CRLF+ UTF-8+ Context: <no context>

- Now working from the bottom:
- What extends across the bottom is a pane that will show messages, both success and error, when a project is built and run
- For the time being there is nothing to “do” there except look for any output
- If the build and run were a success, you can typically ignore the output

My Application - [C:\Users\kascott\AndroidStudioProjects\MyApplication] - [app] - ...app\src\main\java\com\example\kascott\myapplication\MainActivity.java - Android Studio 2.1.3

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

MyApplication app src main java com example kascott myapplication MainActivity

Android MainActivity.java x AndroidManifest.xml x

```
package com.example.kascott.myapplication;

import ...

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Android Monitor

0a0ebf0d [UNAUTHORIZED] No Debuggable Applications

logcat Monitors → Verbose Regex Show only selected application

TODO Android Monitor Terminal Messages Event Log Gradle Console

Gradle build finished in 14s 206ms (20 minutes ago) 14:1 CRLF+ UTF-8+ Context: <no context>

- And one more time, considering the middle of the screen, from left to right:
- For the time being, just ignore the items in the left hand margin: Project, Structure, Captures, Build Variants, Favorites
- Next to these are two panes, the one indicated under “Android” and the one indicated under “Favorites”
- Finally, there is the editing pane, which current shows MainActivity.java

My Application - [C:\Users\kascott\AndroidStudioProjects\MyApplication] - [app] - ...app\src\main\java\com\example\kascott\myapplication\MainActivity.java - Android Studio 2.1.3

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

MyApplication app src main java com example kascott myapplication MainActivity

Android MainActivity.java x AndroidManifest.xml x

```
package com.example.kascott.myapplication;

import ...

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Android Monitor

0a0ebf0d [UNAUTHORIZED] No Debuggable Applications

logcat Monitors → Verbose Regex Show only selected application

TODO Android Monitor Terminal Messages Event Log Gradle Console

Gradle build finished in 14s 206ms (20 minutes ago) 14:1 CRLF+ UTF-8+ Context: <no context>

3.2 Apps and Manifest Files

- If you click on the “AndroidManifest.xml” tab above the editing pane, this is what you see:

My Application - [C:\Users\kascott\AndroidStudioProjects\MyApplication] - [app] - ...app\src\main\AndroidManifest.xml - Android Studio 2.1.3

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

MyApplication app src main AndroidManifest.xml

Android

app

Gradle Scripts

Favorites

My Application

Bookmarks

Breakpoints

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.kascott.myapplication">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="My Application"
        android:supportRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Android Monitor

0a0ebf0d [UNAUTHORIZED]

No Debuggable Applications

logcat

Monitors →

Verbose

Regex

Show only selected application

TODO

Android Monitor

Terminal

Messages

Event Log

Gradle Console

Gradle build finished in 14s 206ms (28 minutes ago)

20:12 CRLF UTF-8 Context: <no context>

- We are happy to rely on Android Studio to create the manifest file by default
- For the time being we don't have to worry about what it means in detail
- However, it's worth noting that this is of fundamental importance

- You know it must be important because Android Studio pops it up as one of the first things you can find through the interface
- This is the basic idea:
- Every executable Java program has a manifest file that goes with it

- Later we'll see that a given app may consist of multiple pieces of executable code
- The manifest file is the place where the multiple pieces of code are indicated so that they are connected together as the constituent parts of a single app

3.3 The Explorer Pane—Before the Resources

- In the following screen shot, other panes have been minimized, leaving the explorer pane open on the left and the editing pane open on the right
- In addition, the items in the explorer have been expanded

My Application - [C:\Users\kascott\AndroidStudioProjects\MyApplication] - [app] - ...app\src\main\java\com\example\kascott\myapplication\MainActivity.java - Android Studio 2.1.3

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

MyApplication app src main java com example kascott myapplication MainActivity

AndroidManifest.xml x MainActivity.java x

Android

- app
 - manifests
 - AndroidManifest.xml
 - java
 - com.example.kascott.myapplication
 - MainActivity
 - com.example.kascott.myapplication
 - ApplicationTest
 - com.example.kascott.myapplication
 - ExampleUnitTest
 - res
 - drawable
 - layout
 - activity_main.xml
 - mipmap
 - ic_launcher.png (5)
 - ic_launcher.png (hdpi)
 - ic_launcher.png (mdpi)
 - ic_launcher.png (xhdpi)
 - ic_launcher.png (xxhdpi)
 - ic_launcher.png (xxxhdpi)
 - values
 - colors.xml
 - dimens.xml (2)
 - dimens.xml
 - dimens.xml (w820dp)
 - strings.xml
 - styles.xml
 - Gradle Scripts
 - build.gradle (Project: MyApplication)
 - build.gradle (Module: app)
 - proguard-rules.pro (ProGuard Rules)
 - gradle.properties (Project Properties)

```
package com.example.kascott.myapplication;

import ...

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Android Monitor Terminal Messages Event Log Gradle Console

Gradle build finished in 14s 206ms (41 minutes ago)

14:1 CRLF+ UTF-8+ Context: <no context>

9:45 AM 8/26/2016

- The explorer shows the contents of a project in tree-like form
- Coming to an initial understanding of what you see there implies a basic understanding of the structure of an Android app

- The first thing to note is that at the bottom of the explorer information is given about Gradle Scripts
- Some of this runs off the screen
- Do not be concerned
- In this initial run-through, we are not interested in the Gradle Script information
- We are interested in what's shown above

- Working from the top down, this is what you find:
- `AndroidManifest.xml`
- This was already briefly explained

- MainActivity (.java)
- This is the Android app Java code which appears in the editor screen
- For a beginning programmer just coming out of a Java programming class, you can think of this as the main() method of the app, saved separately in its own file

- `ApplicationTest`
- `ExampleUnitTest`
- Both of these things we'll be ignoring for the moment
- Throughout the course, they will not be dealt with in detail
- They provide code testing machinery, and would be important in advanced app development

3.4 The Explorer Pane—Resources

- Next comes a folder named “res” with numerous sub-folders
- Not only is it useful to see this in the explorer, but it is critical to how Android apps are structured
- In Android, executable code is stored in one place
- Resources are stored in another

My Application - [C:\Users\kascott\AndroidStudioProjects\MyApplication] - [app] - ...app\src\main\java\com\example\kascott\myapplication\MainActivity.java - Android Studio 2.1.3

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

MyApplication app src main java com example kascott myapplication MainActivity

Android

app

- manifests
 - AndroidManifest.xml
- java
 - com.example.kascott.myapplication
 - MainActivity
 - com.example.kascott.myapplication
 - ApplicationTest
 - com.example.kascott.myapplication
 - ExampleUnitTest
- res
 - drawable
 - layout
 - activity_main.xml
 - mipmap
 - ic_launcher.png (5)
 - ic_launcher.png (hdpi)
 - ic_launcher.png (mdpi)
 - ic_launcher.png (xhdpi)
 - ic_launcher.png (xxhdpi)
 - ic_launcher.png (xxxhdpi)
 - values
 - colors.xml
 - dimens.xml (2)
 - dimens.xml
 - dimens.xml (w820dp)
 - strings.xml
 - styles.xml

- Gradle Scripts
- build.gradle (Project: MyApplication)
- build.gradle (Module: app)
- proguard-rules.pro (ProGuard Rules)
- gradle.properties (Project Properties)

MainActivity.java

```
package com.example.kascott.myapplication;

import ...

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Android Monitor

Terminal

Messages

Event Log

Gradle Console

Gradle build finished in 14s 206ms (41 minutes ago)

14:1 CRLF+ UTF-8+ Context: <no context>

9:45 AM 8/26/2016

- Resources can be detailed things like the display style of the app, dimensions, colors
- But it goes all the way up to the layout of the app
- In other words, in Android the graphical user interface is specified separately from the code in an XML file

- The separation of code and resources leads to challenges in coordinating them
- But it is necessary in an environment where code can run on dozens or hundreds of different hardware platforms
- The layout for each platform can be, and has to be specified in a different resource file

- Not only does this support different display environments
- It also supports internationalization
- You can have different resource files in different languages
- You can have different resource files in different units of measurement (metric, English)
- Etc.

- For the purposes of this first pass, these are the resource files of particular interest:
- activity_main.xml
- strings.xml
- They will be considered in more detail in the next section

3.5 From the Tutorials—Building Your First App: Creating an Android Project

- This lesson shows you how to create a new Android project with [Android Studio](#) and describes some of the files in the project.
- 1. In Android Studio, create a new project:
 - If you don't have a project opened, in the **Welcome** screen, click **New Project**.
 - If you have a project opened, from the **File** menu, select **New Project**.
 - The *Create New Project* screen appears.

- 2. Fill out the fields on the screen.
- For **Application Name** use "My First App".
- For **Company Domain**, use "example.com".
- For the other fields, use the default values and click **Next**
- Here's a brief explanation of each field:
- **Application Name** is the app name that appears to users.
- **Company domain** provides a qualifier that will be appended to the package name;
- Android Studio will remember this qualifier for each new project you create.

- **Package name** is the fully qualified name for the project (following the same rules as those for naming packages in the Java programming language).
- Your package name must be unique across all packages installed on the Android system.
- You can **Edit** this value independently from the application name or the company domain.
- **Project location** is the directory on your system that holds the project files.

- 3. Under **Target Android Devices**, accept the default values and click **Next**.
- The Minimum Required SDK is the earliest version of Android that your app supports, indicated using the [API level](#).
- To support as many devices as possible, you should set this to the lowest version available that allows your app to provide its core feature set.
- If any feature of your app is possible only on newer versions of Android and it's not critical to the app's core feature set, you can enable the feature only when running on the versions that support it (as discussed in [Supporting Different Platform Versions](#)).

- 4. Under **Add an Activity to Mobile**, select **Empty Activity** and click **Next**.
- 5. Under **Customize the Activity**, accept the default values and click **Finish**.
- Your Android project is now a basic "Hello World" app that contains some default files. Take a moment to review the most important of these:

- `app/src/main/java/com.example.myfirstapp/MainActivity.java`
- This file appears in Android Studio after the New Project wizard finishes.
- It contains the class definition for the activity you created earlier.
- When you build and run the app, the [Activity](#) starts and loads the layout file that says "Hello World!"

- `app/src/main/res/layout/activity_main.xml`
- This XML file defines the layout of the activity.
- It contains a `TextView` element with the text "Hello world!".

- `app/src/main/AndroidManifest.xml`
- The [manifest file](#) describes the fundamental characteristics of the app and defines each of its components.
- You'll revisit this file as you follow these lessons and add more components to your app.

Gradle

- `app/build.gradle`
- Android Studio uses Gradle to compile and build your app.
- There is a `build.gradle` file for each module of your project, as well as a `build.gradle` file for the entire project.
- Usually, you're only interested in the `build.gradle` file for the module, in this case the app or application module.
- This is where your app's build dependencies are set, including the `defaultConfig` settings:

Gradle...

- `compileSdkVersion` is the platform version against which you will compile your app.
- By default, this is set to the latest version of Android available in your SDK.
- By default, this is set to the latest version of Android SDK installed on your development machine.
- You can still build your app to support older versions, but setting this to the latest version allows you to enable new features and optimize your app for a great user experience on the latest devices.

Gradle...

- `applicationId` is the fully qualified package name for your application that you specified in the New Project wizard.
- `minSdkVersion` is the Minimum SDK version you specified during the New Project wizard.
- This is the earliest version of the Android SDK that your app supports.

Gradle...

- `targetSdkVersion` indicates the highest version of Android with which you have tested your application.
- As new versions of Android become available, you should test your app on the new version and update this value to match the latest API level and thereby take advantage of new platform features.
- For more information, read [Supporting Different Platform Versions](#).
- See [Building Your Project with Gradle](#) for more information about Gradle.

- Note also the /res subdirectories that contain the [resources](#) for your application:
- drawable-<*density*>/
- Directories for [drawable resources](#), other than launcher icons, designed for various [densities](#).
- layout/
- Directory for files that define your app's user interface like activity_main.xml, discussed above, which describes a basic layout for the MainActivity class.

- menu/
- Directory for files that define your app's menu items.
- mipmap/
- Launcher icons reside in the mipmap/ folder rather than the drawable/ folders.
- This folder contains the ic_launcher.png image that appears when you run the default app.

- values/
- Directory for other XML files that contain a collection of resources, such as string and color definitions.
- To run the app, continue to the [next lesson](#).

Activities—A Sidebar in the Tutorials in this Section

- An activity is one of the distinguishing features of the Android framework.
- Activities provide the user with access to your app, and there may be many activities.
- An application will usually have a main activity for when the user launches the application, another activity for when she selects some content to view, for example, and other activities for when she performs other tasks within the app.
- See [Activities](#) for more information.

Commentary on Activities

- As you might have gathered, activities are a fundamental concept in Android
- Fragments are another fundamental concept which will be introduced later
- You can't have an app without at least a main activity
- Activities and fragments are the basic building blocks for all Android apps

3.6 The Basic Pieces of “Hello World”

- This is largely repetition of what's gone before
- This is my take on what the tutorials have presented

- At the most basic level, from the beginning programmer's point of view, an Android app consists of two files:
- MainActivity.java
- activity_main.xml
- Some of the most basic resources that may belong to an app are strings, contained in:
- strings.xml

The code for “Hello World”

- ```
package com.example.kascott.myapplication;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

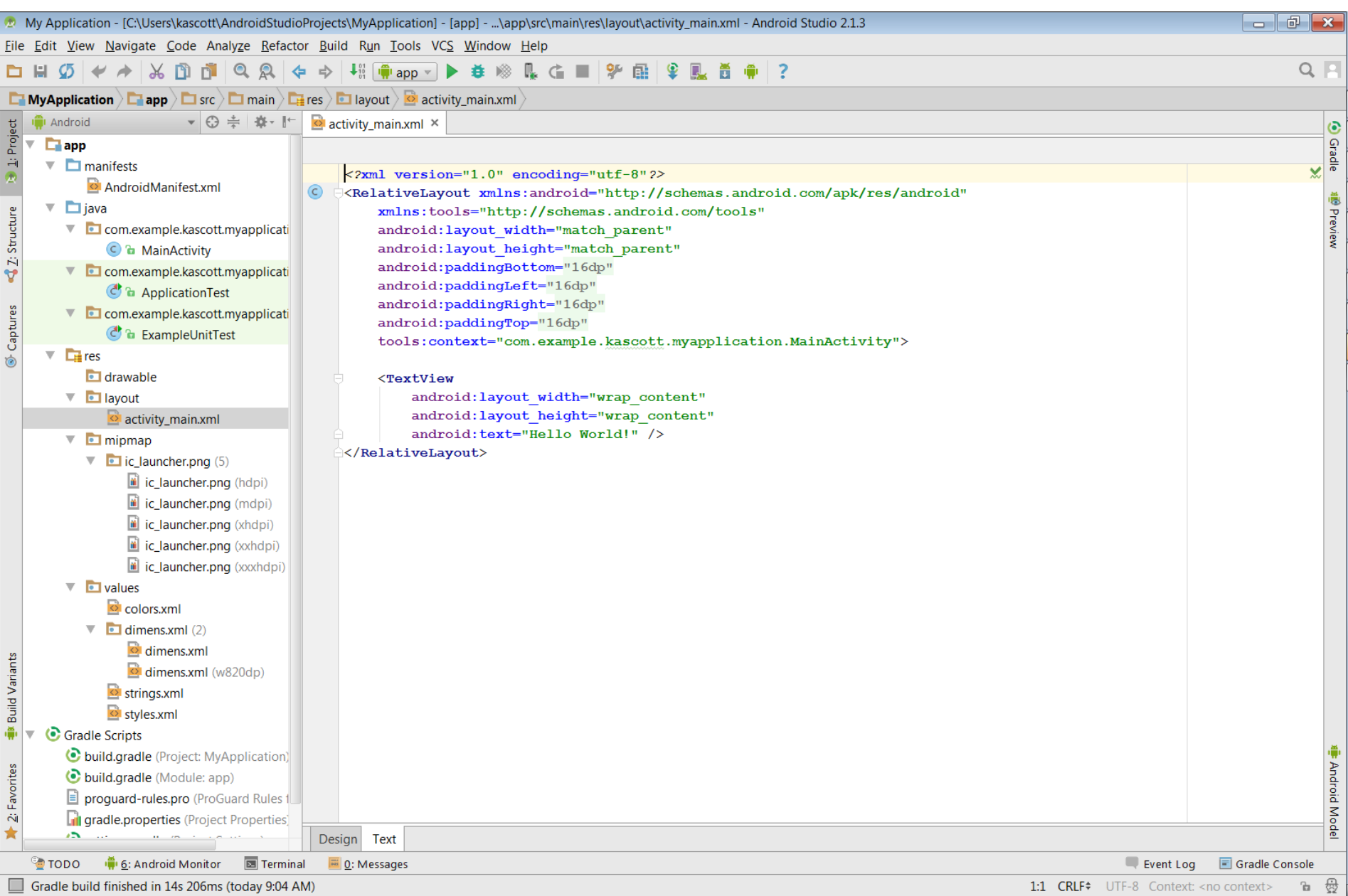
public class MainActivity extends
AppCompatActivity {

 @Override
 protected void onCreate(Bundle
savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);
 }
}
```

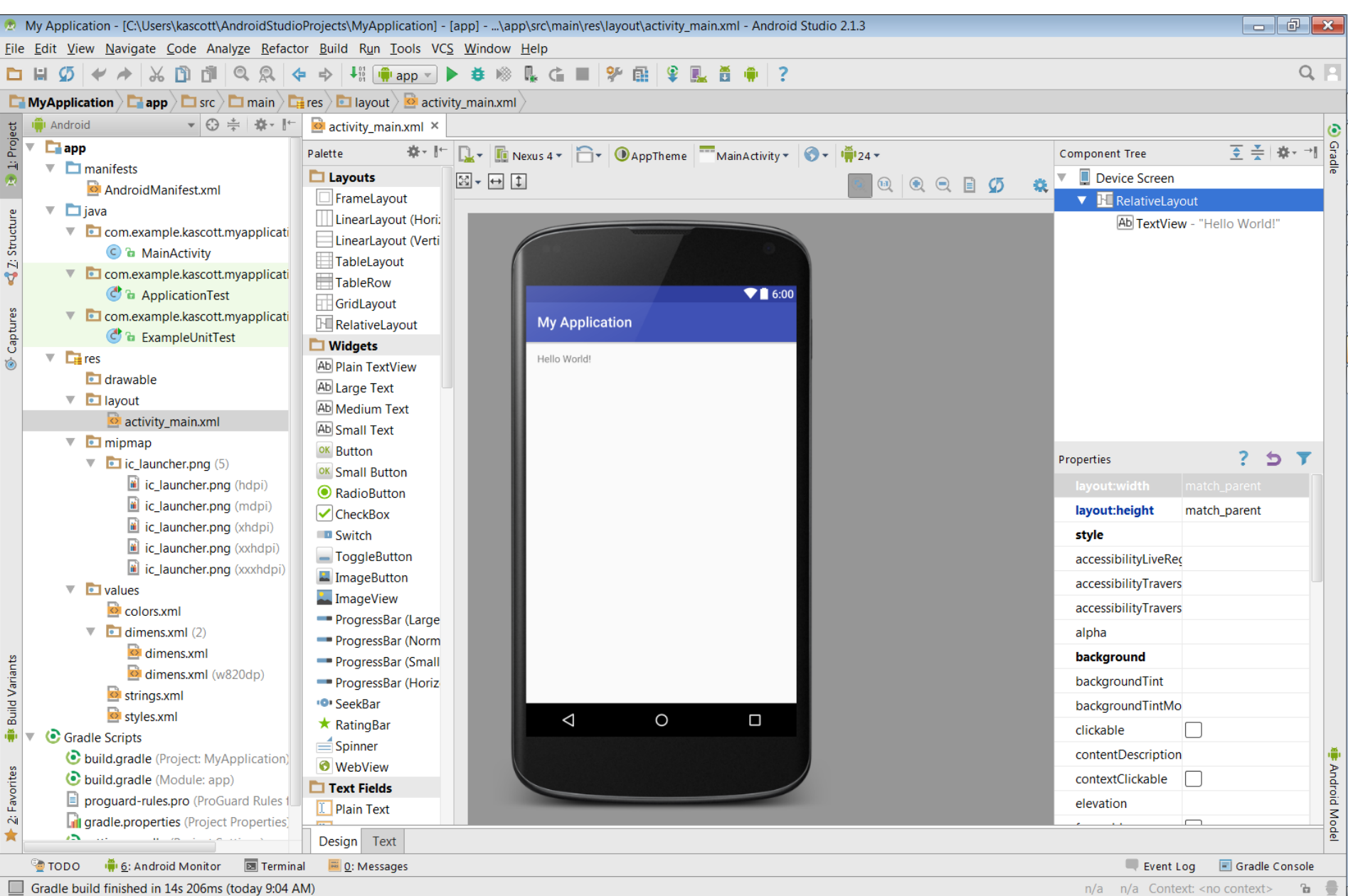
- Remember, this is a “Hello World” program
- It does nothing except present a message on the screen
- There are no output or print commands in this code
- However, the code refers to:
- `R.layout.activity_main`

# The Layout File in Android Studio

- Notice in the following screen shot that the tab at the bottom that's highlighted is “Text”
- This is the editor view of the contents of the layout file
- And notice that along with the specific layout information, at the bottom of the file the output string “Hello World” is defined



- If you click on the “Design” tab at the bottom of the editing pane, you get to see how the layout would appear in use
- Keep in mind that this is just a static presentation of the XML
- Unlike the emulator, this is quick, easy, and foolproof
- And unlike the emulator, this is not a runnable representation of the app

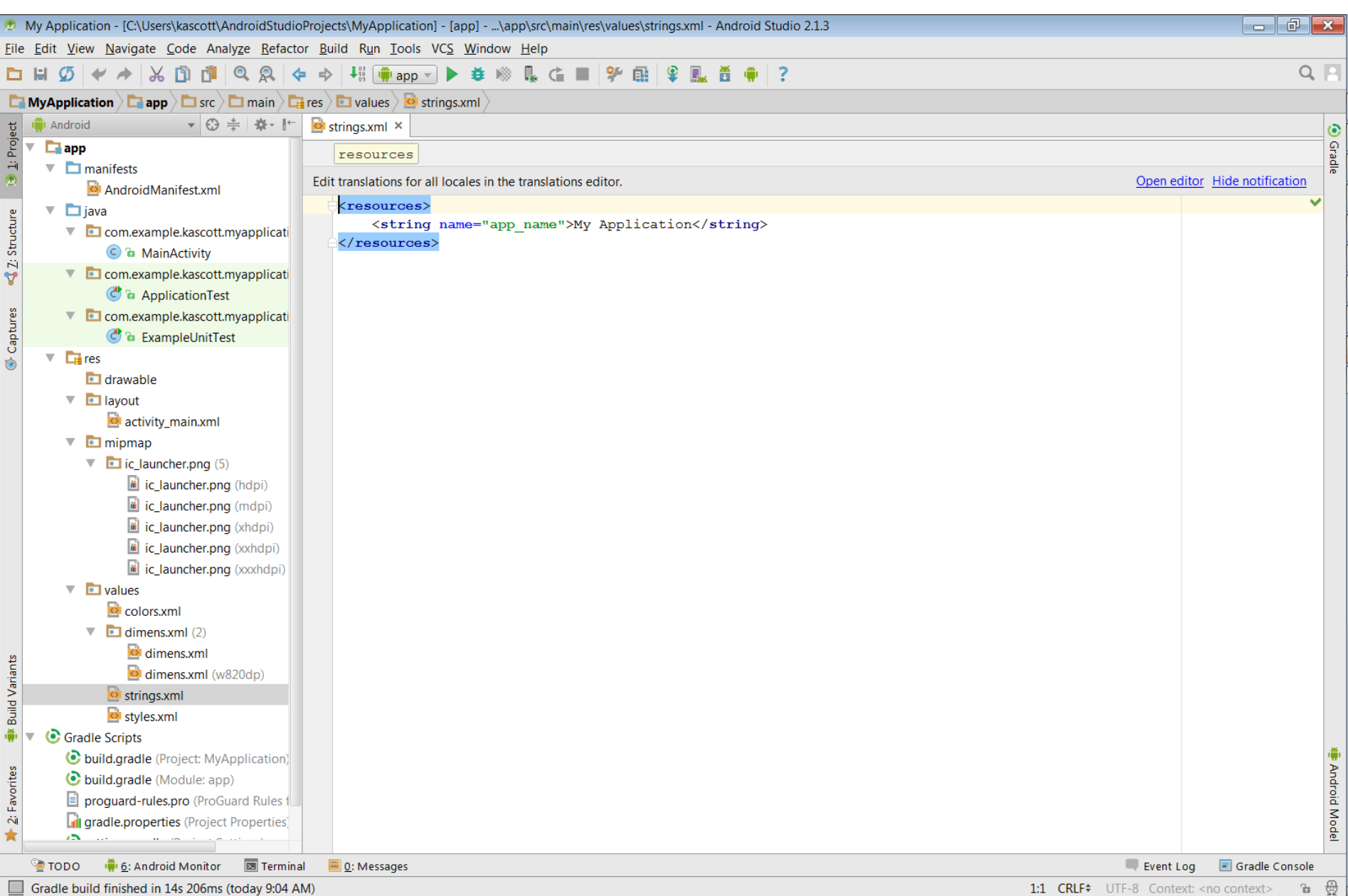


- You will notice on the left, above, and on the right of the device representation in the Design view there are graphical tools for building a graphical user interface
- If you alternate between the Text View and the Design View, you can see the XML for the things which you add using the graphical development environment



# The strings.xml File

- The strings.xml file at this point is quite simple
- The only string that's defined by default when creating the initial “Hello World” app is that name of the app



- You might foresee already how to make the app more flexible:
- Define the string “Hello World” in the strings.xml file
- Then have the layout file refer to the string in that location
- Understanding and coordinating resources in this way will be covered in future units

## 3.7 From the Tutorials—Building Your First App: Running Your Application

- This section is a combination of the happy generalities of the tutorials plus my commentary
- If you run into trouble, you may discover that my commentary is relevant or helpful
- On the other hand, it may be that you can follow along with the tutorials without encountering any problems

- In the [previous lesson](#), you created an Android project.
- The project contains a default app that displays "Hello World".
- In this lesson, you will run the app on a device or emulator.

# Run on a Real Device

- Set up your device as follows:
- 1. Connect your device to your development machine with a USB cable.
- If you're developing on Windows, you might need to install the appropriate USB driver for your device.
- For help installing drivers, see the [OEM USB Drivers](#) document.

# Commentary on Drivers

- Drivers appear to be working better now under Android Studio than they did under earlier development environments
- However, they are critical to running apps on an attached device, and they could be a source of trouble
- One kind of problem that has happened before is that the O/S would override the specific drivers you installed for your Android device with generic drivers which didn't support app functionality



- To be specific, some installations of Windows would only recognize an attached Android tablet as a simple media player
- You would install the drivers again, and Windows would again replace them with its out of date media player drivers
- With luck, it will work like magic
- If not, you will have to do some research

- 2. Enable **USB debugging** on your device by going to **Settings > Developer options**.
- **Note:** On Android 4.2 and newer, **Developer options** is hidden by default.
- To make it available, go to **Settings > About phone** and tap **Build number** seven times.
- Return to the previous screen to find **Developer options**.

- Run the app from Android Studio as follows:
- In Android Studio, select your project and click **Run** from the toolbar.
- In the **Select Deployment Target** window, select your device, and click **OK**.
- *Android Studio installs the app on your connected device and starts it.*

# Run on the Emulator

- Before you run your app on an emulator, you need to create an [Android Virtual Device](#) (AVD) definition.
- An AVD definition defines the characteristics of an Android phone, tablet, Android Wear, or Android TV device that you want to simulate in the Android Emulator.

- Create an AVD Definition as follows:
- 1. Launch the Android Virtual Device Manager by selecting **Tools > Android > AVD Manager**, or by clicking the AVD Manager icon in the toolbar.
- 2. On the AVD Manager main screen, click **Create Virtual Device**.
- 3. In the Select Hardware page, select a phone device, such as Nexus 6, then click **Next**.
- 4. In the Select Image page, choose the desired system image for the AVD and click **Next**.
- 5. Verify the configuration settings (for your first AVD, leave all the settings as they are), and then click **Finish**.
- For more information about using AVDs, see [Create and Manage Virtual Devices](#).

- Run the app from Android Studio as follows:
- 1. In **Android Studio**, select your project and click **Run** from the toolbar.
- 2. In the **Select Deployment Target** window, select your emulator and click **OK**.
- It can take a few minutes for the emulator to start. You may have to unlock the screen.
- When you do, *My First App* appears on the emulator screen.
- That's how you build and run your Android app on the emulator!
- To start developing, continue to the [next lesson](#).

# Commentary on Using the Emulator

- As pointed out in an earlier set of overheads, although a desirable addition to the development environment, the emulator can be problematic
- Presumably it has gotten better as the environment has progressed, but it is still prone to causing difficulties

- If you can download the drivers successfully and run on an attached device, that is the most trouble-free way to develop
- On the other hand, if you can't download the drivers, you can run on the emulator
- And if you're preparing a project presentation on your desktop, it can be handy to take screen shots of the emulator



- The online tutorials are a bit vague on some of the things you may have to keep in mind when using the emulator
- What kind of virtual device should you create?
- Ideally, you would want a virtual device to emulate the hardware device that you have
- On the other hand, it is also valid to create a simpler smaller virtual device if you can't get the emulator working with a virtual device that matches your actual physical device

- Another thing that crops has to do with the memory size of the virtual device that you will be trying to emulate
- There is not one central location where you can answer questions that might arise, but the general idea is this:
- The emulation of your virtual device may or may not work correctly depending on the amount of memory that is allocated in the virtual device


# Practical Realities





- The tutorial says that when the virtual device is created, you can emulate on it
- Experience suggests these further things to keep in mind:
- If you want to run on the virtual device, remember to disconnect the real device

- This is the most critical thing I've run into in practice:
- Don't just run the project, relying on the emulator to start the virtual device
- Instead, start the virtual device emulation first
- Then run the project


- The following screen shot shows what you get by following Tools/Android/AVD Manager in the menu
- At the right hand side there is a green arrowhead
- If you click this, it will start the AVD in the emulator
- Patience, patience, patience, plus vigilance...

Android Virtual Device Manager

Your Virtual Devices  
Android Studio

| Type                                                                             | Name           | Resolution         | API | Target                    | CPU/ABI | Size on Disk | Actions                                                                                                                                                                                                                                                     |
|----------------------------------------------------------------------------------|----------------|--------------------|-----|---------------------------|---------|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | Nexus 7 API 23 | 1200 × 1920: xhdpi | 23  | Android 6.0 (Google APIs) | x86     | 4 GB         |    |

+ Create Virtual Device...



- It will take a while for the emulator to start, plus it is quite likely that it won't show up on top of the Android Studio window
- Minimize everything else that is on your desktop and watch for the emulator to appear, as shown on the next overhead

5554:Nexus\_7\_2012\_API\_23





- You're not out of the woods yet, because the emulator (although slow) accurately represents the fact that the device starts up
- When the screen of the emulation is still acting like a device starting up, don't try to run your app
- Wait for the emulation of the virtual device to stabilize, like a successfully started device

- Then re-maximize Android Studio
- Click the green arrowhead to run the app
- And return to the emulator to see the app run

- Doing as directed above is my recommendation
- In theory everything will start and run correctly if you just click on the green arrowhead to run the app without starting the emulated virtual device first

- The problem is that you will run out of patience
- Nothing will seem to be happening
- You won't know if the emulation is starting successfully or not
- You will do something like try to restart the app, give up and close things, etc., etc.

- This will be especially useful when you are actually doing projects of your own where the app code may be flaky
- If you know the emulation is fully and successfully started first, then if the app doesn't run, or doesn't run as expected, you know that you will need to debug your app
- But at least you won't be in doubt, wondering whether it was the starting of the emulation that messed things up

- There is one final thing to keep in mind if you're in a position where you want or need to use the emulator
- During a programming session, once you start the emulation of your virtual device, don't close it
- It will be there any time you want to test your app and you won't have to wait for it to restart

# Summary and Mission

- That's a brief survey of the Android Studio working environment and the default, "Hello World", project it will make for you
- You have a single mission, which is not graded homework:
  1. Create and run the "Hello World" app on your installation of Android Studio
- For completeness' sake, you should probably try running it in the emulator and also running it on an attached device

# The End