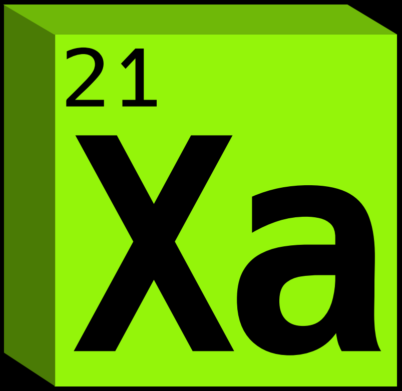
Version 2.0

Exploring Android



Mark L. Murphy

CommonsWare

Exploring Android

by Mark L. Murphy

COMMONSWARE

Exploring Android
by Mark L. Murphy

Printing History:
December 2020: Version 2.0

Table of Contents

Headings formatted in ***bold-italic*** have changed since the last version.

Preface

Thanks!

First, thanks for your interest in Android app development! Android is the world's most popular operating system, but its value comes from apps written by developers like you.

Also, thanks for your interest in this book! Hopefully, it can help "spin you up" on how to create Android applications that meet your needs and those of your users.

And thanks for your interest in CommonsWare! The Warescription program makes this book and others available, to help developers like you craft the apps that your users need.

How the Book Is Structured

Many books — such as *Elements of Android Jetpack*, — present programming topics, showing you how to use different APIs, tools, and so on.

This book is different.

This book has you build an app from the beginning. Whereas traditional programming guides are focused on breadth and depth, this book is focused on being "hands-on", guiding you through the steps to build the app. It provides some details on the underlying concepts, but it relies on other resources — such as *Elements of Android Jetpack* — for the full explanation of those details. Instead, this book provides step-by-step instructions for building the app.

If you are the sort of person who "learns by doing", then this book is for you!

Second-Generation Book

Android app development can be divided into two generations:

- First-generation app development uses Java as the programming language and leverages the Android Support Library and the `android.arch` edition of the Architecture Components
- Second-generation app development more often uses Kotlin as the programming language and leverages AndroidX and the rest of Jetpack (which includes an AndroidX edition of the Architecture Components)

This book is a second-generation book. It will show you step-by-step how to build a Kotlin-based Android app, using AndroidX libraries.

Prerequisites

This book is targeted at developers starting out with Android app development.

You will want another educational resource to go along with this book. The book will cross-reference *Elements of Android Jetpack*, but you can use other programming guides as well. This book shows you each step for building an app, but you will need to turn to other resources for answers to questions like "why do we need to do X?" or "what other options do we have than Y?".

The app that you will build will be written in Kotlin, so you will need to have a bit of familiarity with that language. *Elements of Kotlin* covers this language and will be cross-referenced in a few places in this book.

Also, the app that you will create in this book works on Android 5.0+ devices and emulators. You will either need a suitable device or be in position to use the Android SDK emulator in order to build and run the app.

About the Updates

This book will be updated a few times per year, to reflect new advances with Android, the libraries used by the sample app, and the development tools.

If you obtained this book through the Warescription, you will be able to download updates as they become available, for the duration of your subscription period.

If you obtained this book through other channels… um, well, it's still a really nice book!

Each release has notations to show what is new or changed compared with the immediately preceding release:

- The Table of Contents in the ebook formats (PDF, EPUB, MOBI/Kindle) shows sections with changes in ***bold-italic*** font
- Those sections have changebars on the right to denote specific paragraphs that are new or modified

And, there is the "What's New" section, just below this paragraph.

What's New in Version 2.0?

As with all of the CommonsWare x.0 releases, this is the same as the previous release (1.9), other than some bug fixes.

Copying Code From This Book

You are welcome to copy the code as you see in the book itself, as part of working through the tutorials.

However, copying from the PDF version of the book can be troublesome, depending on your PDF viewer. Some PDF viewers do not handle the syntax highlighting used in this book very well.

Recommended PDF viewers include:

- Adobe Reader (Windows, macOS)
- Foxit Reader (Windows, macOS, Linux)
- Google Chrome (Windows, macOS)
- Google Chromium (Linux)

Also, once we start modifying files, you will find "Final Results" sections towards the end of each chapter. Those will contain the full listings of the source files that were modified in that chapter's tutorial. And, these listings will *not* have syntax highlighting, making them suitable copy sources for a wider range of PDF viewers.

Warescription

If you purchased the Warescription, read on! If you obtained this book from other channels, feel free to jump ahead.

The Warescription entitles you, for the duration of your subscription, to digital editions of this book and its updates, in PDF, EPUB, and Kindle (MOBI/KF8) formats. You also have access to other books that CommonsWare publishes during that subscription period, such as the aforementioned *Elements of Android Jetpack*. You also get access to first-generation Android books, such as the legendary *The Busy Coder's Guide to Android Development*.

Each subscriber gets personalized editions of all editions of each book. That way, your books are never out of date for long, and you can take advantage of new material as it is made available.

However, you can only download the books while you have an active Warescription. There is a grace period after your Warescription ends: you can still download the book until the next book update comes out after your Warescription ends. After that, you can no longer download the book. Hence, **please download your updates as they come out**. You can find out when new releases of this book are available via:

1. The CommonsBlog
2. The CommonsWare Twitter feed
3. Opting into emails announcing each book release — log into the Warescription site and choose Configure from the nav bar
4. Just check back on the Warescription site every month or two

Subscribers also have access to other benefits, including:

- "Office hours" — online chats to help you get answers to your Android application development questions. You will find a calendar for these on your Warescription page.
- A Stack Overflow "bump" service, to get additional attention for a question that you have posted there that does not have an adequate answer.
- A discussion board for asking arbitrary questions about Android app development.

Book Bug Bounty

Find a problem in the book? Let CommonsWare know!

Be the first to report a unique concrete problem in the current digital edition, and CommonsWare will extend your Warescription by six months as a bounty for helping CommonsWare deliver a better product.

By "concrete" problem, we mean things like:

1. Typographical errors
2. Sample applications that do not work as advertised, in the environment described in the book
3. Factual errors that cannot be open to interpretation

By "unique", we mean ones not yet reported. Be sure to check the book's errata page, though, to see if your issue has already been reported. One coupon is given per email containing valid bug reports.

We appreciate hearing about "softer" issues as well, such as:

1. Places where you think we are in error, but where we feel our interpretation is reasonable
2. Places where you think we could add sample applications, or expand upon the existing material
3. Samples that do not work due to "shifting sands" of the underlying environment (e.g., changed APIs with new releases of an SDK)

However, those "softer" issues do not qualify for the formal bounty program.

Questions about the bug bounty, or problems you wish to report for bounty consideration, should be sent to bounty@commonsware.com.

Source Code and Its License

The source code samples shown in this book are available for download from the book's GitLab repository. All of the Android projects are licensed under the Apache 2.0 License, in case you have the desire to reuse any of it.

Copying source code directly from the book, in the PDF editions, works best with

Adobe Reader, though it may also work with other PDF viewers. Some PDF viewers, for reasons that remain unclear, foul up copying the source code to the clipboard when it is selected.

Creative Commons and the Four-to-Free (42F) Guarantee

Each CommonsWare book edition will be available for use under the Creative Commons Attribution-Noncommercial-ShareAlike 4.0 license as of the fourth anniversary of its publication date, or when 4,000 copies of the edition have been sold, whichever comes first. That means that, once four years have elapsed (perhaps sooner!), you can use this prose for non-commercial purposes. That is our Four-to-Free Guarantee to our readers and the broader community. For the purposes of this guarantee, new Warescriptions and renewals will be counted as sales of this edition, starting from the time the edition is published.

This edition of this book will be available under the aforementioned Creative Commons license on *1 December 2024*. Of course, watch the CommonsWare Web site, as this edition might be relicensed sooner based on sales.

For more details on the Creative Commons Attribution-Noncommercial-ShareAlike 4.0 license, visit the Creative Commons Web site

Note that future editions of this book will become free on later dates, each four years from the publication of that edition or based on sales of that specific edition. Releasing one edition under the Creative Commons license does not automatically release *all* editions under that license.

What We Are Building

By following the instructions in this book, you will build an Android app.

But first, let's see what the app is that you are building.

The Purpose

Everybody has stuff to do. Ever since we have had "digital assistants" — such as the venerable Palm line of PDAs — a common use has been for tracking tasks to be done. So-called "to-do lists" are a popular sort of app, whether on the Web, on the desktop, or on mobile devices.

The world has more than enough to-do list apps. Google themselves have published a long list of sample apps that use a to-do list as a way of exploring various GUI architectures.

So, let's build another one!

Ours is not a fork of Google's, but rather a "cleanroom" implementation of a to-do list with similar functionality.

The Core UI

There are three main screens that the user will spend time in: the roster of to-do items, a screen with details of a particular item, and a screen for either adding a new item or editing an existing one.

There is also an "about" screen for displaying information about the app.

The Roster

When initially launched, the app will show a roster of the recorded to-do items, if there are any. Hence, on the first run, it will show just an “empty view”, prompting the user to click the “add” app bar item to add a new item:

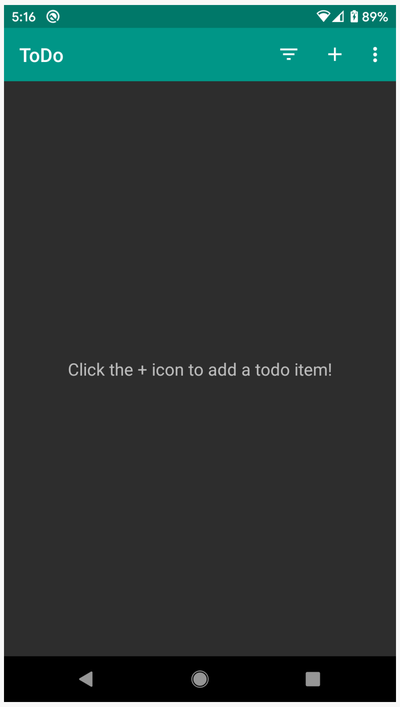
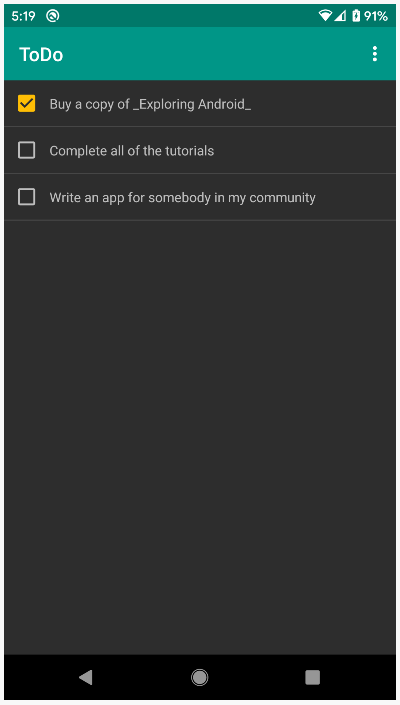


Figure 1: ToDo App, As Initially Launched, with No Items

Once there are some items in the database, the roster will show those items, in alphabetical order by description, with a checkbox indicating whether or not they have been completed:



✓ Buy a copy of _Exploring Android_

☐ Complete all of the tutorials

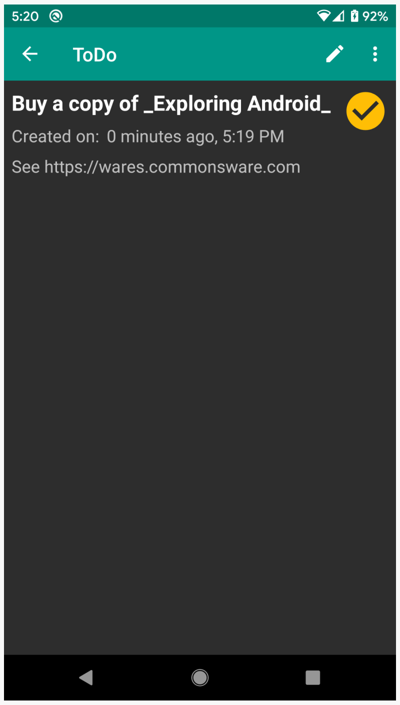
☐ Write an app for somebody in my community

Figure 2: ToDo App, Showing Some Items

From here, the user can tap the checkbox to quickly mark an item as completed (or un-mark it if needed).

The Details

A simple tap on an item in the roster brings up the details screen:



Buy a copy of _Exploring Android_

Created on: 0 minutes ago, 5:19 PM

See https://wares.commonsware.com

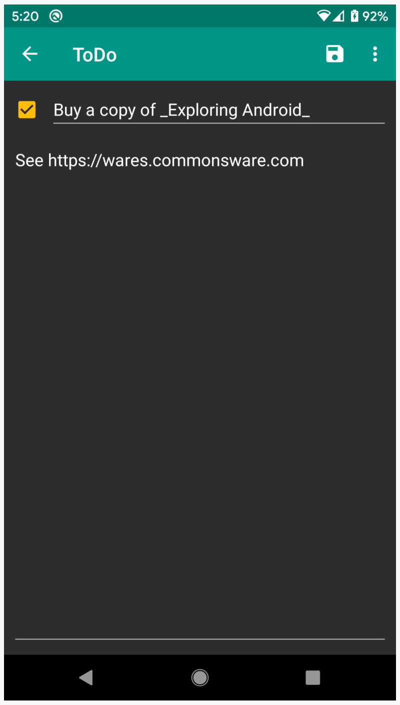
Figure 3: ToDo App, Showing a Completed Item

This just shows additional information about the item, including any notes the user entered to provide more detail than the simple description that gets shown in the roster. The checkmark icon will appear for completed items.

From here, the user can edit this item (via the "pencil" icon).

The Editor

The editor is a simple form, either to define a new to-do item or edit an existing one. If the user taps on the "add" app bar item from the roster, the editor will appear blank, and submitting the form will create a new to-do item. If the user taps on the "edit" (pencil) app bar item from the details screen, the editor will have the existing item's data, which can be altered and saved:



Buy a copy of _Exploring Android_

See https://wares.commonsware.com

Figure 4: ToDo App, Editing a Completed Item

Phase One: Getting a GUI

Installing the Tools

First, let us get you set up with the pieces and parts necessary to build an Android app. Specifically, in this tutorial, we will set up Android Studio.

Step #1: Checking Your Hardware

Compiling and building an Android application, on its own, can be a hardware-intensive process, particularly for larger projects. Beyond that, your IDE and the Android emulator will stress your development machine further. Of the two, the emulator poses the bigger problem.

The more RAM you have, the better. 8GB or higher is a very good idea if you intend to use an IDE and the emulator together. If you can get an SSD for your data storage, instead of a conventional hard drive, that too can dramatically improve the IDE performance.

A faster CPU is also a good idea. The Android SDK emulator supports CPUs with multiple cores. However, other processes on your development machine will be competing with the emulator for CPU time, and so the faster your CPU is, the better off you will be. Ideally, your CPU has 4 or more cores, each 2.5GHz or faster at their base speed.

There are two types of emulator: x86 and ARM. These are the two major types of CPUs used for Android devices. You *really* want to be able to use the x86 emulator, as the ARM emulator is extremely slow. However, to do that, you need a CPU with certain features:

Development OS	CPU Manufacturer	CPU Requirements
mac OS	Intel	any modern Mac should work
Linux/ Windows	Intel	support for Intel VT-x, Intel EM64T (Intel 64), and Execute Disable (XD) Bit functionality
Linux	AMD	support for AMD Virtualization (AMD-V) and Supplemental Streaming SIMD Extensions 3 (SSSE3)
Windows 10 April 2018 or newer	AMD	support for Windows Hypervisor Platform (WHPX) functionality

If your CPU does not meet those requirements, you will want to have one or more Android devices available to you, so that you can test on hardware.

Also, if you are running Windows or Linux, you need to ensure that your computer's BIOS is set up to support the Intel/AMD virtualization extensions. Unfortunately, many PC manufacturers disable this by default. The details of how to get into your BIOS settings will vary by PC, but usually it involves rebooting your computer and pressing some function key on the initial boot screen. In the BIOS settings, you are looking for references to "virtualization" (or perhaps "VT-x" for Intel). Enable them if they are not already enabled. macOS machines come with virtualization extensions pre-enabled, which is really nice of Apple.

Step #2: Install Android Studio

At the time of this writing, the current production version of Android Studio is 4.1 and this book covers that version. Android Studio gets updated often, and so you may be on a newer version — there may be some differences between what you have and what is presented here.

You have two major download options. You can get the latest shipping version of Android Studio from the Android Studio download page. Or, you can download Android Studio 4.1 directly, for:

- Windows

- macOS
- Linux

Windows users can download a self-installing EXE, which will add suitable launch options for you to be able to start the IDE.

Mac users can download a DMG disk image and install it akin to other Mac software, dragging the Android Studio icon into the Applications folder.

Linux users (and power Windows users) can download a ZIP file, then unZIP it to some likely spot on your hard drive. Android Studio can then be run from the `studio` batch file or shell script in your Android Studio installation's `bin/` directory.

Step #3: Run Android Studio

When you first run Android Studio, you may be asked if you want to import settings from some other prior installation of Android Studio:

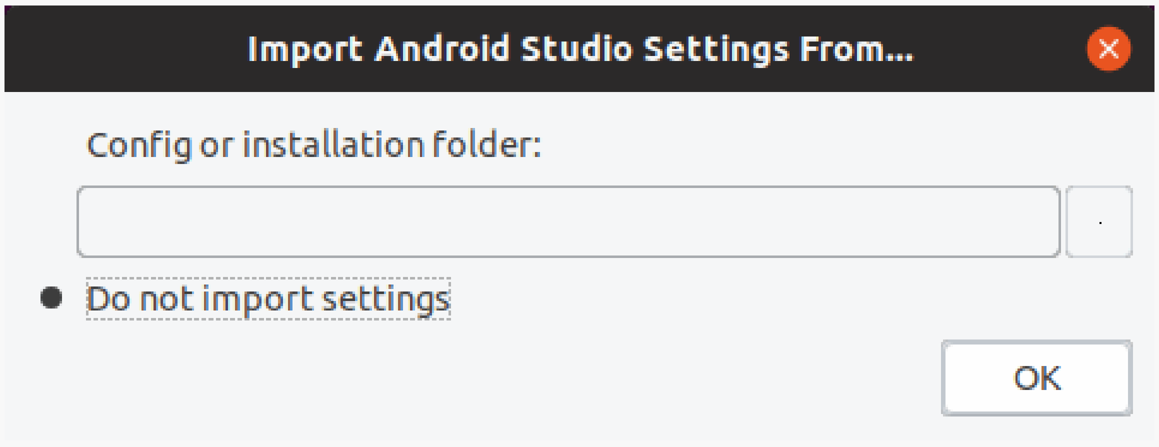
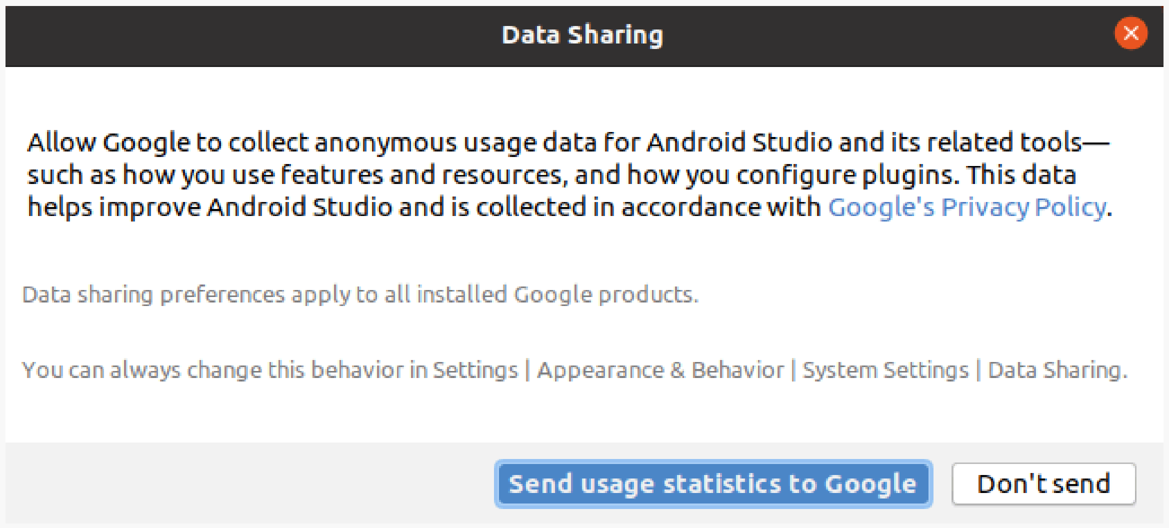

Import Android Studio Settings From...
Config or installation folder:
Do not import settings
OK

Figure 5: Android Studio First-Run Settings Migration Dialog

If you are using Android Studio for the first time, the "Do not import settings" option is the correct choice to make.

Then, after a short splash screen, you may be presented with a "Data Sharing" dialog:


Data Sharing

Allow Google to collect anonymous usage data for Android Studio and its related tools—such as how you use features and resources, and how you configure plugins. This data helps improve Android Studio and is collected in accordance with Google's Privacy Policy.

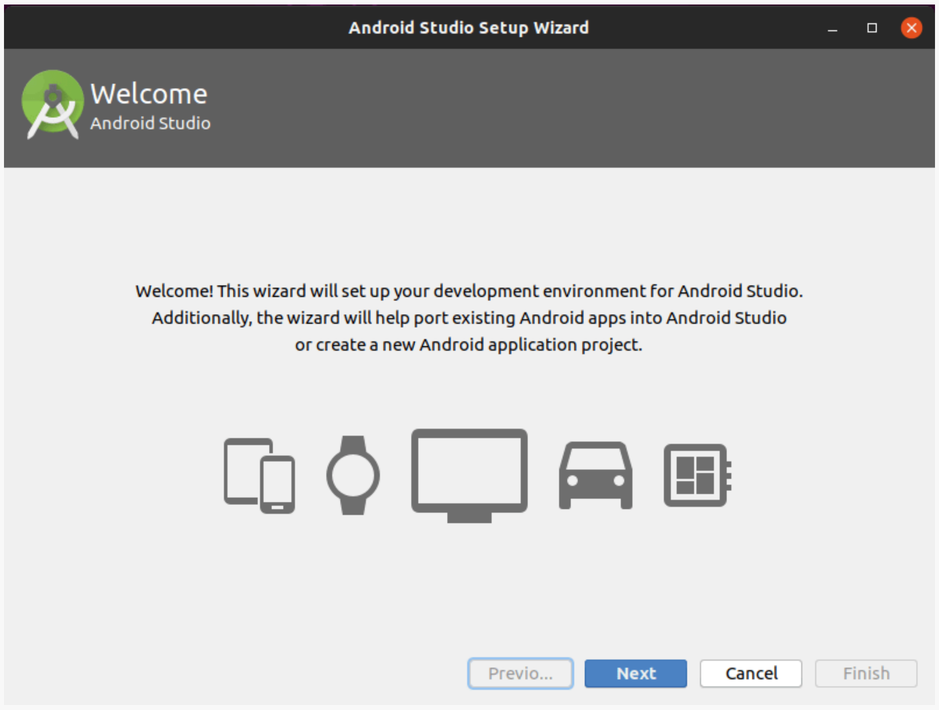
Data sharing preferences apply to all installed Google products.

You can always change this behavior in Settings | Appearance & Behavior | System Settings | Data Sharing.

Figure 6: Android Studio Data Sharing Dialog

Click whichever button you wish.

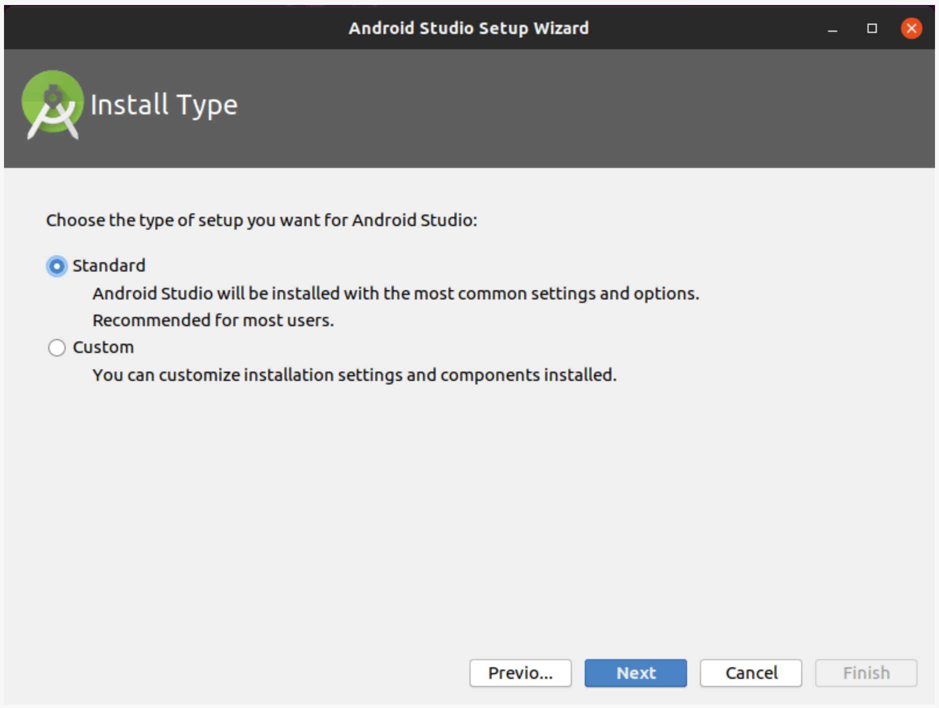
Then, after a potentially long "Finding Available SDK Components" progress dialog, you will be taken to the Android Studio Setup Wizard:



Welcome! This wizard will set up your development environment for Android Studio. Additionally, the wizard will help port existing Android apps into Android Studio or create a new Android application project.

Figure 7: Android Studio Setup Wizard, First Page

Just click "Next" to advance to the second page of the wizard:



Choose the type of setup you want for Android Studio:

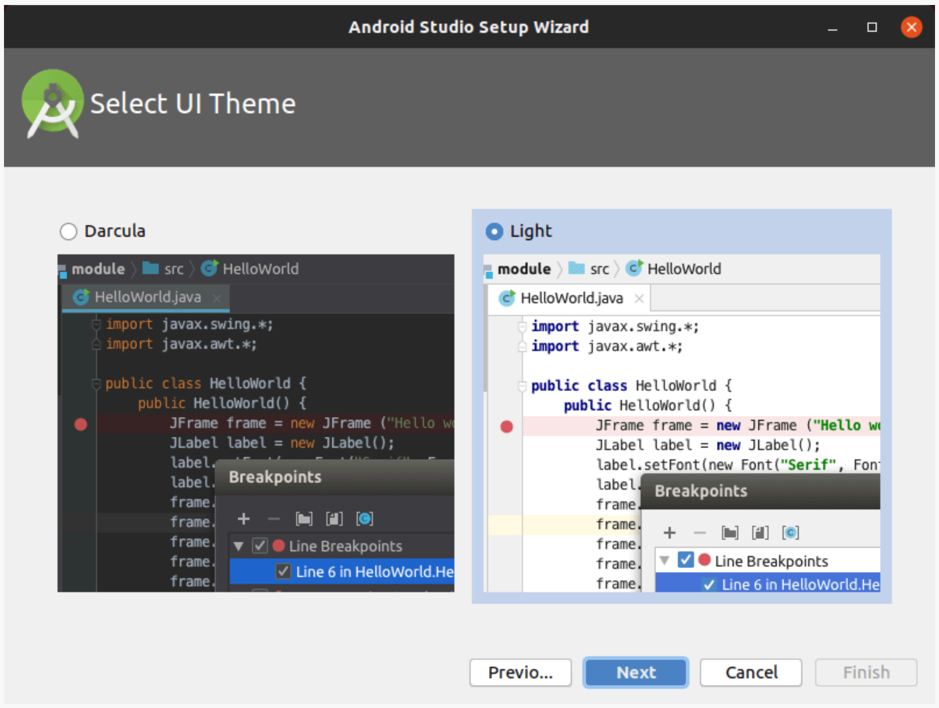
[✓ Standard]
Android Studio will be installed with the most common settings and options.
Recommended for most users.

Custom
You can customize installation settings and components installed.

Figure 8: Android Studio Setup Wizard, Second Page

Here, you have a choice between "Standard" and "Custom" setup modes. Most likely, right now, the "Standard" route will be fine for your environment.

If you go the "Standard" route and click "Next", you should be taken to a wizard page where you can choose your UI theme:

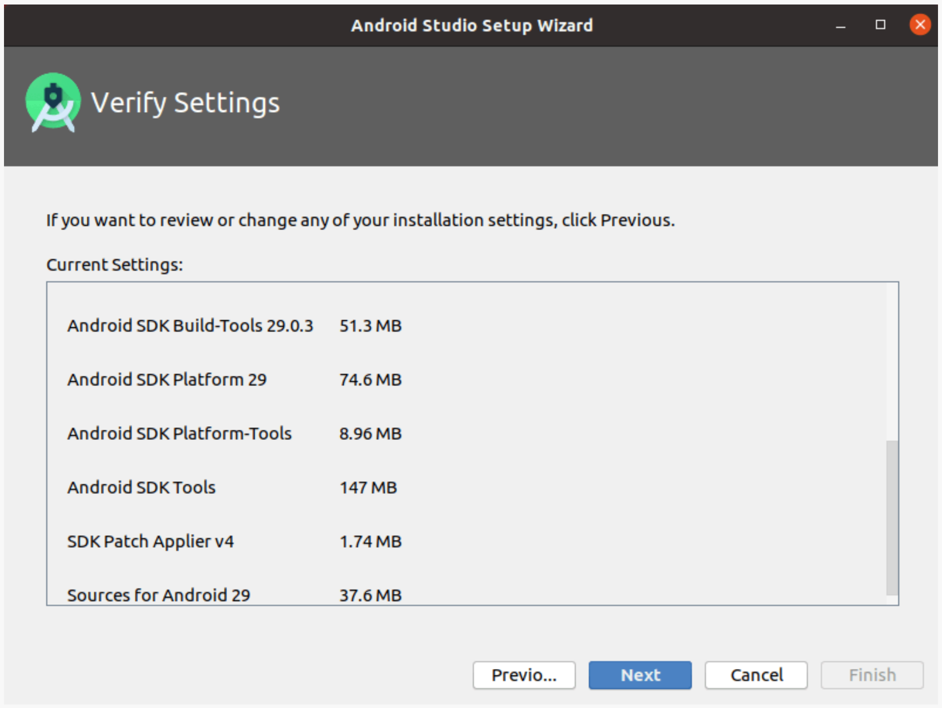


```
import javax.swing.*;
import javax.awt.*;

public class HelloWorld {
    public HelloWorld() {
        JFrame frame = new JFrame ("Hello wo
        JLabel label = new JLabel();
        label.setFont(new Font("Serif", Fon
        label.
        frame.
        frame.
        frame.
        frame.
        frame.
```

Figure 9: Android Studio Setup Wizard, UI Theme Page

Choose whichever you like, then click "Next", to go to a wizard page to verify what will be downloaded and installed:



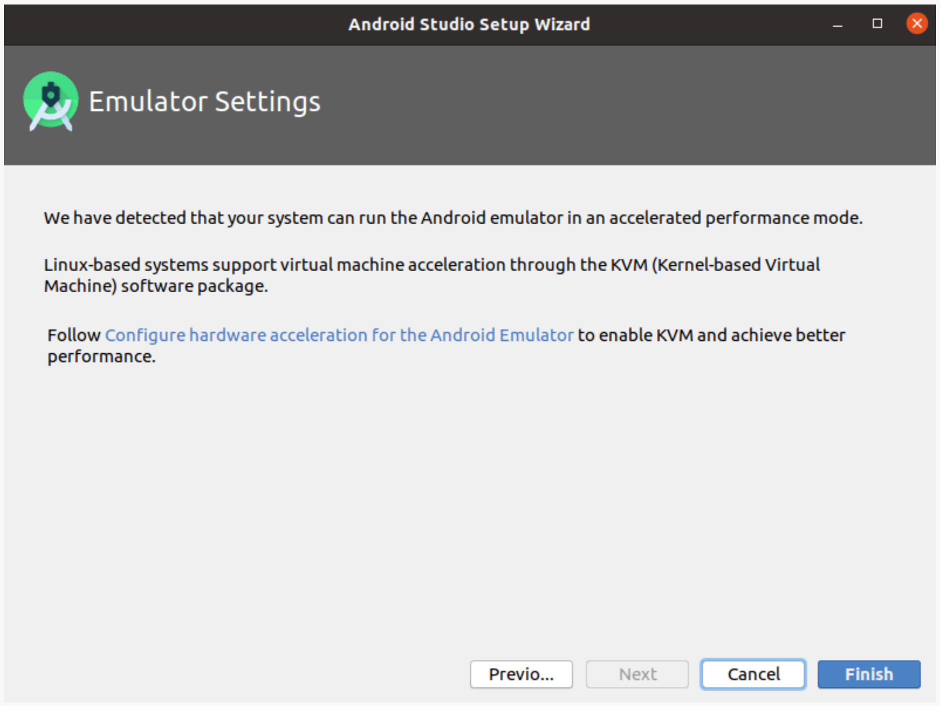
If you want to review or change any of your installation settings, click Previous.

Current Settings:

Android SDK Build-Tools 29.0.3	51.3 MB
Android SDK Platform 29	74.6 MB
Android SDK Platform-Tools	8.96 MB
Android SDK Tools	147 MB
SDK Patch Applier v4	1.74 MB
Sources for Android 29	37.6 MB

Figure 10: Android Studio Setup Wizard, Verify Settings Page

Clicking "Next" may take you to a wizard page explaining some information about the Android emulator:



We have detected that your system can run the Android emulator in an accelerated performance mode.

Linux-based systems support virtual machine acceleration through the KVM (Kernel-based Virtual Machine) software package.

Follow Configure hardware acceleration for the Android Emulator to enable KVM and achieve better performance.

Figure 11: Android Studio Setup Wizard, Emulator Info Page

What is explained on this page may not make much sense to you. That is perfectly normal, and we will get into what this page is trying to say later in the book. Just click "Finish" to begin the setup process. This will include downloading a copy of the Android SDK and installing it into a directory adjacent to where Android Studio itself is installed.

When that is done, Android Studio will busily start downloading stuff to your development machine.

Clicking "Finish" when that is done will then take you to the Android Studio Welcome dialog:

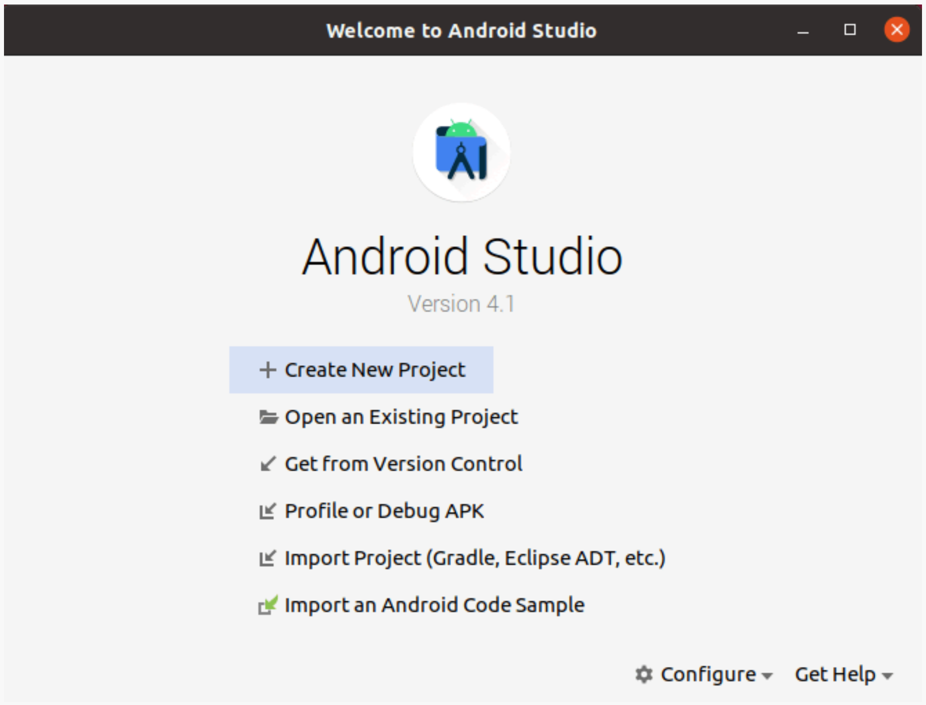


Figure 12: Android Studio 4.0 Welcome Dialog

Creating a Starter Project

Creating an Android application first involves creating an Android "project". As with many other development environments, the project is where your source code and other assets (e.g., icons) reside. And, the project contains the instructions for your tools for how to convert that source code and other assets into an Android APK file for use with an emulator or device, where the APK is Android's executable file format.

Hence, in this tutorial, we kick off development of a sample Android application, to give you the opportunity to put some of what you are learning in this book in practice.

Step #1: Importing the Project

First, we need an Android project to work in.

Sometimes, you will create a new project yourself, using Android Studio's new-project wizard. However, frequently, you will start with an existing project that somebody else created. For example, if you are joining an Android development team, odds are that somebody else will create the project, or the project will already have been created by the time you join. In those cases, you will import an existing project, and that's what we will do here.

Download the starter project from CommonsWare's Web site. Then, UnZIP that project to some place on your development machine. It will unZIP into a `ToDo/` directory.

At that point, look at the contents of `gradle/wrapper/gradle-wrapper.properties`. It should look like this:

```
#Fri Jun 05 08:55:24 EDT 2020
distributionBase=GRADLE_USER_HOME
distributionPath=wrapper/dists
zipStoreBase=GRADLE_USER_HOME
zipStorePath=wrapper/dists
distributionUrl=https\://services.gradle.org/distributions/gradle-6.5-all.zip
```

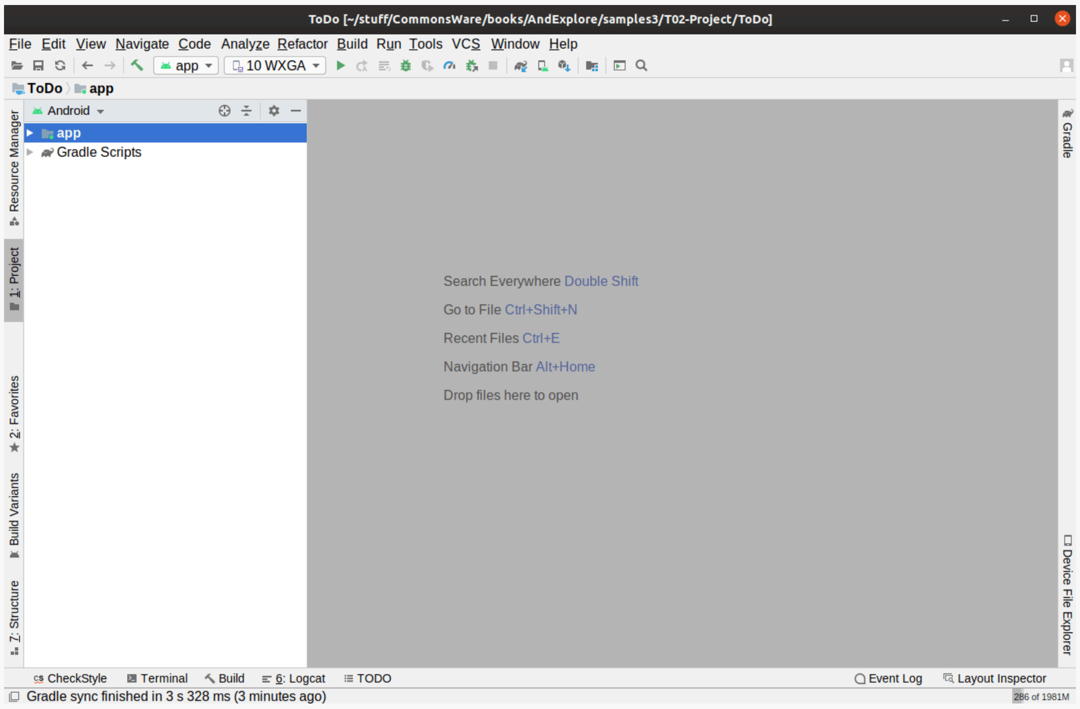
(from T02-Project/ToDo/gradle/wrapper/gradle-wrapper.properties)

In particular, make sure that the `distributionUrl` points to a `services.gradle.org` URL. **Never** import a project into Android Studio without checking the `distributionUrl`, as a malicious person could have `distributionUrl` point to malware that Android Studio would load and execute.

Then, import the project. From the Android Studio welcome dialog — where we left off in the previous tutorial — that is handled by the "Import project (Gradle, Eclipse ADT, etc.)" option. From an existing open Android Studio IDE window, you would use File > New > Import Project… from the main menu.

Importing a project brings up a typical directory-picker dialog. Pick the `ToDo/` directory and click "OK" to begin the import process. This may take a while, depending on the speed of your development machine. A "Tip of the Day" dialog may appear, which you can dismiss.

At this point, the IDE window should be open on your starter project:



Search Everywhere Double Shift

Go to File Ctrl+Shift+N

Recent Files Ctrl+E

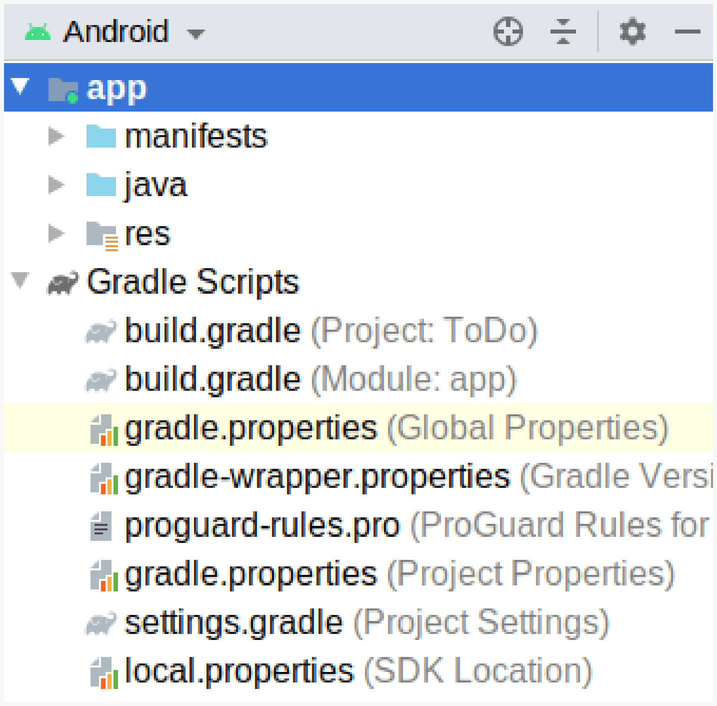
Navigation Bar Alt+Home

Drop files here to open

Gradle sync finished in 3 s 328 ms (3 minutes ago)

Figure 13: ToDo Project, As Initially Imported

The "Project" view — docked by default on the left side, towards the top — brings up a way for you to view what is in the project. Android Studio has several ways of viewing the contents of Android projects. The default one, that you are presented with when creating or importing the project, is known as the "Android view":



```
app
  manifests
  java
  res
Gradle Scripts
  build.gradle (Project: ToDo)
  build.gradle (Module: app)
  gradle.properties (Global Properties)
  gradle-wrapper.properties (Gradle Versi
  proguard-rules.pro (ProGuard Rules for
  gradle.properties (Project Properties)
  settings.gradle (Project Settings)
  local.properties (SDK Location)
```

Figure 14: Android Studio, Project View, Android Content

While you are welcome to navigate your project using it, the tutorial chapters in this book, where they have screenshots of Android Studio, will show the "Project" contents in this view:

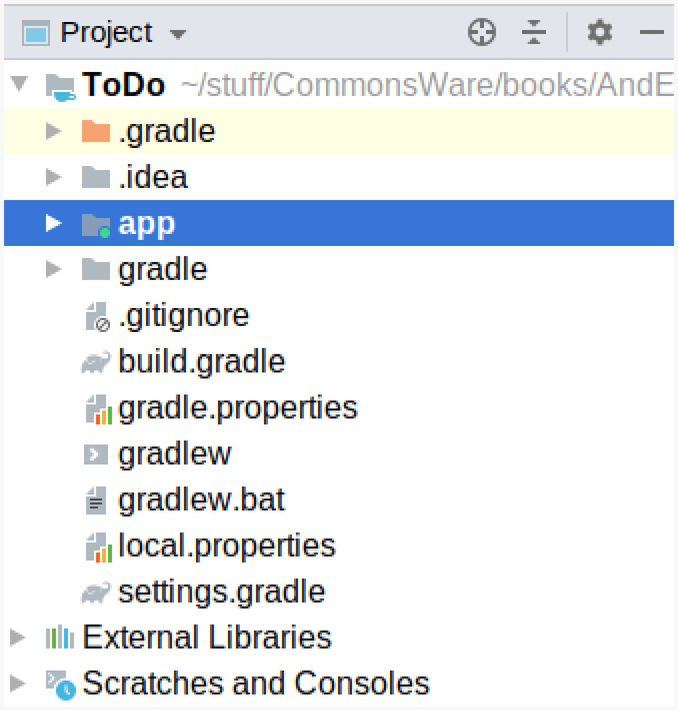


Figure 15: Android Studio, Project View, Project Content

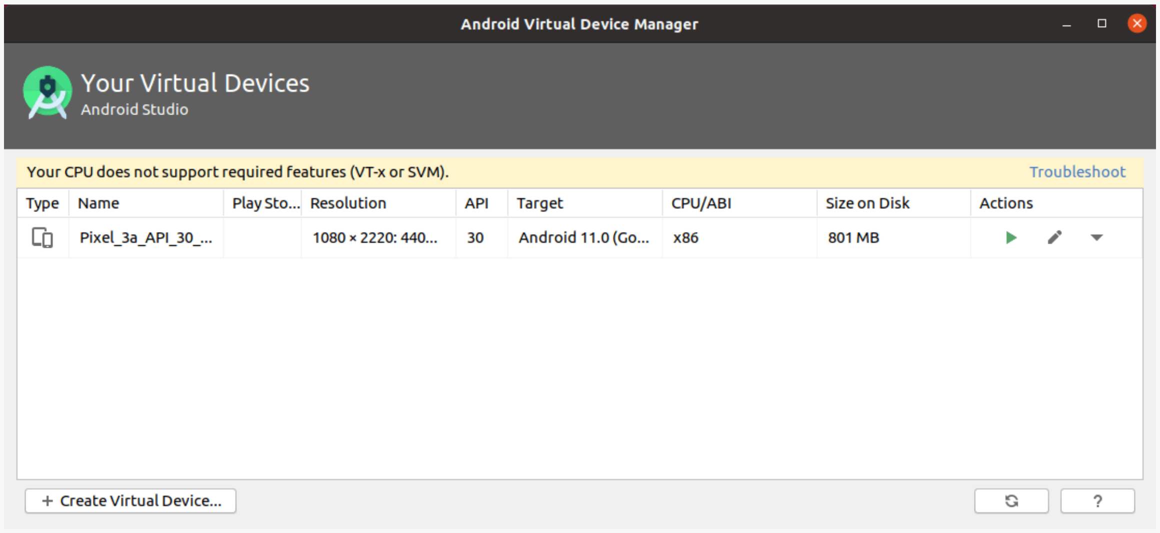
To switch to this view — and therefore match what the tutorials will show you — click the Android drop-down above the tree and choose "Project" from the list.

Step #2: Setting Up the Emulator AVD

Your first decision to make is whether or not you want to bother setting up an emulator image right now. If you have an Android device, you may prefer to start testing your app on it, and come back to set up the emulator at a later point. In that case, skip to Step #3.

The Android emulator can emulate one or several Android devices. Each configuration you want is stored in an "Android virtual device", or AVD. The AVD Manager is where you create these AVDs. To open the AVD Manager in Android Studio, choose Tools > AVD Manager from the main menu.

By default, a fresh installation of Android Studio 4.1 will also set up an initial AVD for you, and the AVD Manager will show it in a list:



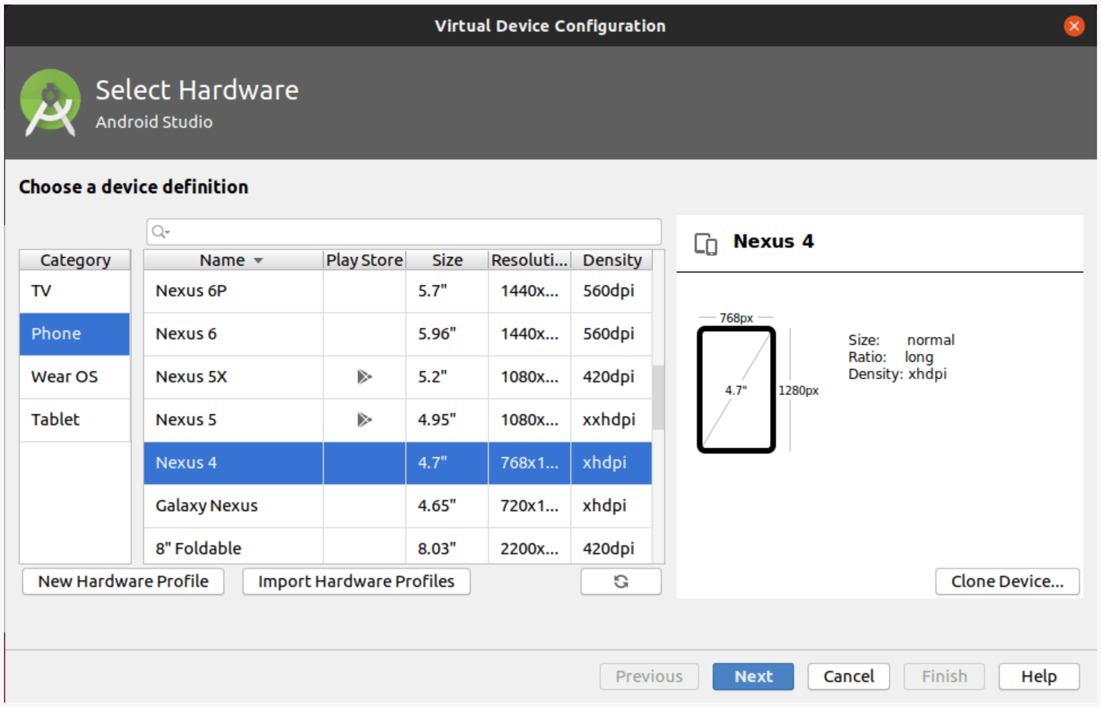
Your CPU does not support required features (VT-x or SVM).

Type	Name	Play Sto...	Resolution	API	Target	CPU/ABI	Size on Disk	Actions
	Pixel_3a_API_30_...		1080 × 2220: 440...	30	Android 11.0 (Go...	x86	801 MB	

Figure 16: Android Studio AVD Manager, With Pre-Installed AVD

If you have a fairly powerful development machine, and you want an emulator that emulates an Android 11 device, you are welcome to use that emulator at the outset. For most starting developers, this is not a particularly good emulator choice.

If you would like to create a different AVD — perhaps one for an older Android version or one that will run better on less-powerful development machines — click the "Create Virtual Device" button, which brings up a "Virtual Device Configuration" wizard:



Virtual Device Configuration

Select Hardware
Android Studio

Choose a device definition

Category
TV
Phone
Wear OS
Tablet

Name	Play Store	Size	Resoluti...	Density
Nexus 6P		5.7"	1440x...	560dpi
Nexus 6		5.96"	1440x...	560dpi
Nexus 5X	▶	5.2"	1080x...	420dpi
Nexus 5	▶	4.95"	1080x...	xxhdpi
Nexus 4		4.7"	768x1...	xhdpi
Galaxy Nexus		4.65"	720x1...	xhdpi
8" Foldable		8.03"	2200x...	420dpi

Nexus 4

Size: normal
Ratio: long
Density: xhdpi

Figure 17: Android Studio Virtual Device Configuration Wizard, First Page

The first page of the wizard allows you to choose a device profile to use as a starting point for your AVD. The "New Hardware Profile" button allows you to define new profiles, if there is no existing profile that meets your needs.

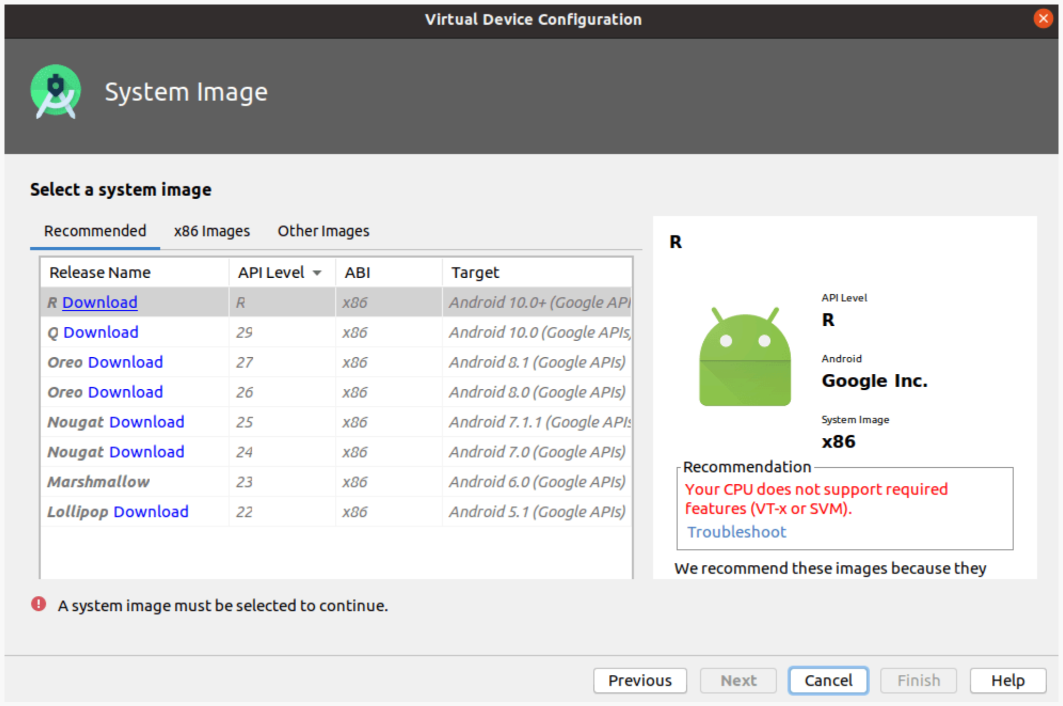
Since emulator speeds are tied somewhat to the resolution of their (virtual) screens, you generally aim for a device profile that is on the low end but is not completely ridiculous. For example, a 1280x768 or 1280x720 phone would be considered by many people to be fairly low-resolution. However, there are plenty of devices out there at that resolution (or lower), and it makes for a reasonable starting emulator.

If you want to create a new device profile based on an existing one — to change a few parameters but otherwise use what the original profile had — click the "Clone Device" button once you have selected your starter profile.

However, in general, at the outset, using an existing profile is perfectly fine. The

Nexus 4 image is a likely choice to start with.

Clicking “Next” allows you to choose an emulator image to use:



Select a system image

Release Name	API Level	ABI	Target
R Download	R	x86	Android 10.0+ (Google API
Q Download	29	x86	Android 10.0 (Google APIs
Oreo Download	27	x86	Android 8.1 (Google APIs)
Oreo Download	26	x86	Android 8.0 (Google APIs)
Nougat Download	25	x86	Android 7.1.1 (Google API
Nougat Download	24	x86	Android 7.0 (Google APIs)
Marshmallow	23	x86	Android 6.0 (Google APIs)
Lollipop Download	22	x86	Android 5.1 (Google APIs)

R

API Level
R

Android
Google Inc.

System Image
x86

Recommendation
Your CPU does not support required features (VT-x or SVM).
Troubleshoot

We recommend these images because they

A system image must be selected to continue.

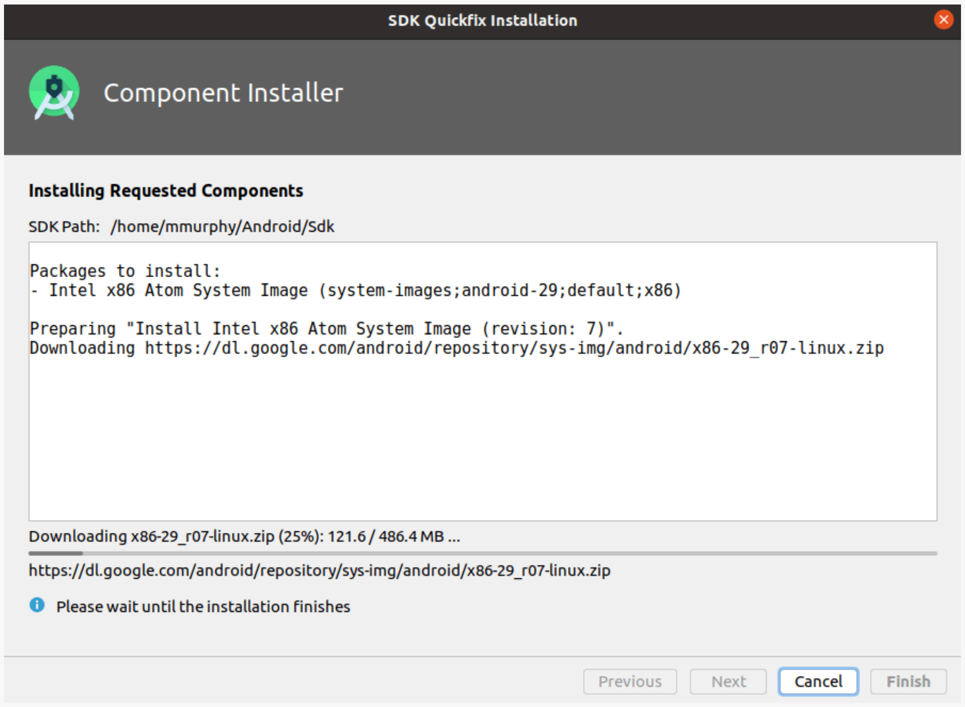
Figure 18: Android Studio Virtual Device Configuration Wizard, Second Page

The emulator images are spread across three tabs:

- “Recommended”
- “x86 Images”
- “Other Images”

For the purposes of the tutorials, you do not need an emulator image with the “Google APIs” — those are for emulators that have Google Play Services in them and related apps like Google Maps. However, in terms of API level, you can choose anything from API Level 21 (Android 5.0) on up.

It is best to use one of the x86 images for the best emulator performance. On the “x86 Images” tab, you should see some entries with a “Download” link, and you might see others without it. The emulator images with “Download” next to them will trigger a one-time download of the files necessary to create AVDs for that particular API level and CPU architecture combination, after another license dialog and progress dialog:



Installing Requested Components

SDK Path: /home/mmurphy/Android/Sdk

```
Packages to install:
- Intel x86 Atom System Image (system-images;android-29;default;x86)

Preparing "Install Intel x86 Atom System Image (revision: 7)".
Downloading https://dl.google.com/android/repository/sys-img/android/x86-29_r07-linux.zip
```

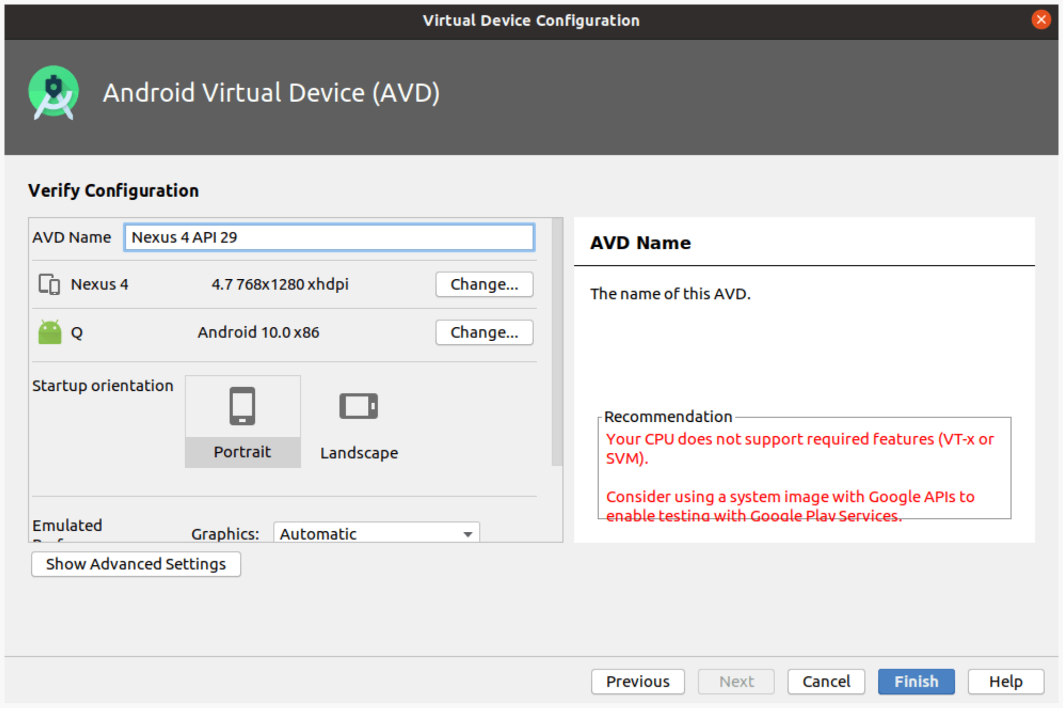
Downloading x86-29_r07-linux.zip (25%): 121.6 / 486.4 MB ...

https://dl.google.com/android/repository/sys-img/android/x86-29_r07-linux.zip

Please wait until the installation finishes

Figure 19: Android Studio Component Installer Dialog, Downloading API 29 Image

Once you have identified the image that you want — and have downloaded it if needed — click on one of them in the wizard. Clicking “Next” allows you to finalize the configuration of your AVD:



AVD Name: Nexus 4 API 29

Nexus 4 — 4.7 768x1280 xhdpi

Q — Android 10.0 x86

Emulated Performance — Graphics: Automatic

AVD Name

The name of this AVD.

Recommendation

Your CPU does not support required features (VT-x or SVM).

Consider using a system image with Google APIs to enable testing with Google Play Services.

Figure 20: Android Studio Virtual Device Configuration Wizard, Third Page

If you get the “Recommendation” box with the red “Your CPU does not support required features…” message, your development machine is not set up to support this type of emulator image. For example, you may need to enable virtualization extensions in your PC’s BIOS, as was noted in the previous tutorial.

A default name for the AVD is suggested, though you are welcome to replace this with your own value. However, that name must be something valid: only letters, numbers, spaces, and select punctuation (e.g., ., _, -, (,)) are supported.

The rest of the default values should be fine for now.

Clicking “Finish” will return you to the main AVD Manager, showing your new AVD. You can then close the AVD Manager window.

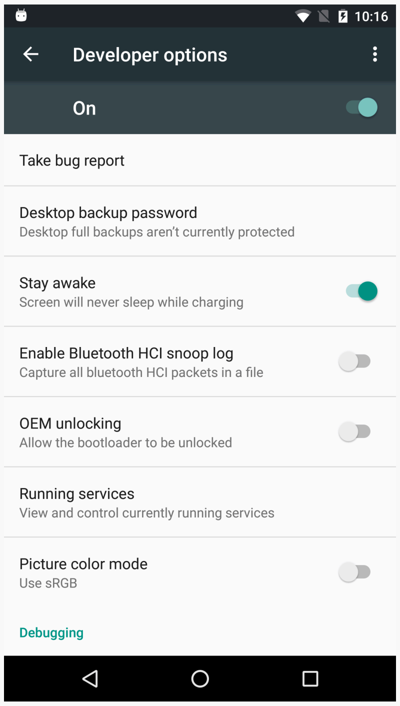
If you also have a physical device that you want to use for testing, continue with Step #3. Otherwise, feel free to skip to Step #4.

Step #3: Setting Up the Device

You do not need an Android device to get started in Android application development. Having one is a good idea before you try to ship an application (e.g., upload it to the Play Store). And, perhaps you already have a device – maybe that is what is spurring your interest in developing for Android.

If you do not have an Android device that you wish to set up for development, skip this step.

The first thing to do to make your device ready for use with development is to go into the Settings application on the device. On Android 8.0+, go into System > About phone. On older devices, About is usually a top-level entry. In the About screen, tap on the build number seven times, then press BACK, and go into "Developer options" (which was formerly hidden)



On

Take bug report

Desktop backup password
Desktop full backups aren't currently protected

Stay awake
Screen will never sleep while charging

Enable Bluetooth HCI snoop log
Capture all bluetooth HCI packets in a file

OEM unlocking
Allow the bootloader to be unlocked

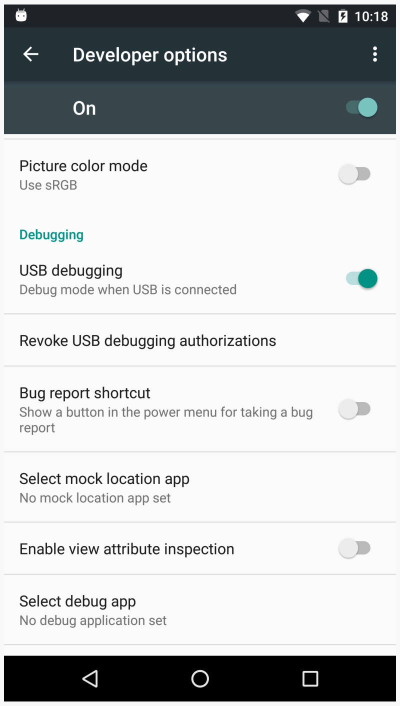
Running services
View and control currently running services

Picture color mode
Use sRGB

Figure 21: Developer Options, in Android 9.0 Settings App

You may need to slide a switch in the upper-right corner of the screen to the "ON" position to modify the values on this screen.

Generally, you will want to scroll down and enable USB debugging, so you can use your device with the Android build tools:


10:18

Developer options

On

Picture color mode
Use sRGB

Debugging

USB debugging
Debug mode when USB is connected

Revoke USB debugging authorizations

Bug report shortcut
Show a button in the power menu for taking a bug report

Select mock location app
No mock location app set

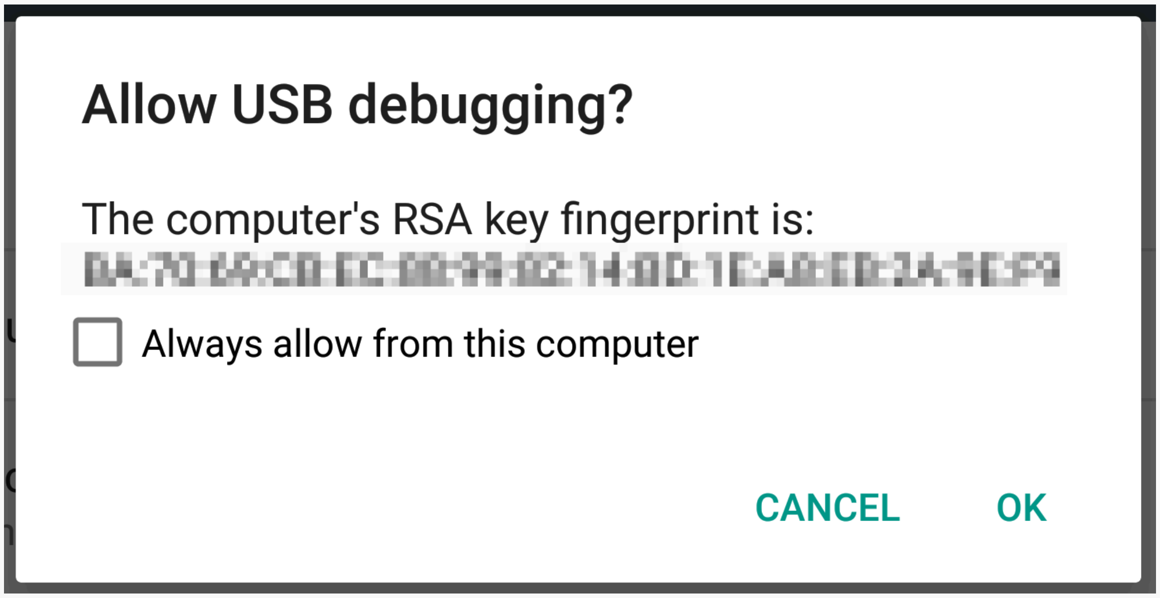
Enable view attribute inspection

Select debug app
No debug application set

Figure 22: Debugging Options, in Android 9.0 Settings App

You can leave the other settings alone for now if you wish, though you may find the "Stay awake" option to be handy, as it saves you from having to unlock your phone all of the time while it is plugged into USB.

Note that on Android 4.2.2 and higher devices, before you can actually use the setting you just toggled, you will be prompted to allow USB debugging with your *specific* development machine via a dialog box:



Allow USB debugging?

The computer's RSA key fingerprint is:
[redacted]

Always allow from this computer

Figure 23: Allow USB Debugging Dialog

This occurs when you plug in the device via the USB cable and have the driver appropriately set up. That process varies by the operating system of your development machine, as is covered in the following sections.

Windows

When you first plug in your Android device, Windows will attempt to find a driver for it. It is possible that, by virtue of other software you have installed, that the driver is ready for use. If it finds a driver, you are probably ready to go.

If the driver is not found, here are some options for getting one.

Windows Update

Some versions of Windows (e.g., Vista) will prompt you to search Windows Update for drivers. This is certainly worth a shot, though not every device will have supplied its driver to Microsoft.

Standard Android Driver

In your Android SDK installation, if you chose to install the "Google USB Driver" package from the SDK Manager, you will find an `extras/google/usb_driver/` directory, containing a generic Windows driver for Android devices. You can try pointing the driver wizard at this directory to see if it thinks this driver is suitable for your device. This will often work for Nexus devices.

Manufacturer-Supplied Driver

If you still do not have a driver, the OEM USB Drivers in the developer documentation may help you find one for download from your device manufacturer. Note that you may need the model number for your device, instead of the model name used for marketing purposes (e.g., GT-P3113 instead of "Samsung Galaxy Tab 2 7.0").

macOS and Linux

It is likely that simply plugging in your device will "just work".

If you are running Ubuntu (or perhaps other Linux variants), and when you later try running your app it appears that Android Studio does not "see" your device, you may need to add some `udev` rules. This GitHub repository contains some instructions and a large file showing the rules for devices from a variety of manufacturers, and this blog post provides more details of how to work with `udev` rules for Android devices.

Step #4: Running the Project

Now, we can confirm that our project is set up properly by running it on a device or emulator.

Android Studio has a toolbar just below the main menu. In that toolbar, you will find two drop-down lists, followed by the Run toolbar button (usually depicted as a green rightward-pointing triangle):

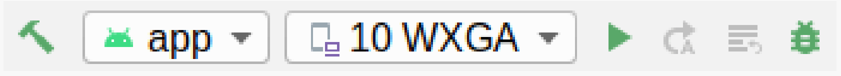


Figure 24: Android Studio Toolbar Segment

The first drop-down says "this is what I want to run". Right now, your only viable option is "app", referring to the app that this project builds.

The second drop-down says "this is where I want to run it". Here, you will find a list of devices and emulators that are available to you.

To run the app, choose your desired device or emulator in the second drop-down, then click the Run toolbar button. If you choose an emulator, and the emulator is not already running, Android Studio will start it up. Then, after a short wait, your app should appear on it:

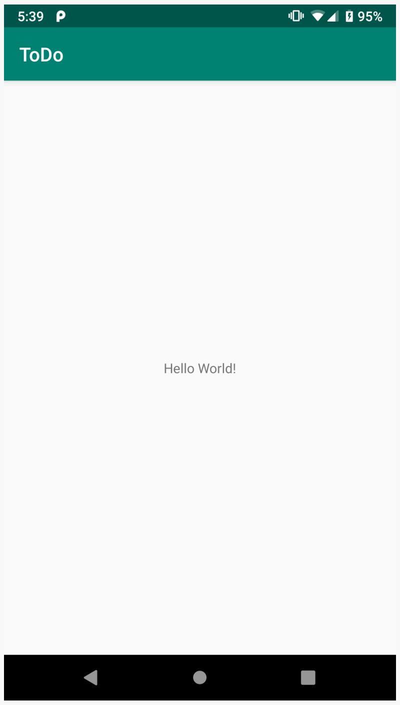


Figure 25: Android 8.1 Device, Showing ToDo App

Note that you may have to unlock your device or emulator to actually see the app running.

The first time you launch the emulator for a particular AVD, you may see this message:

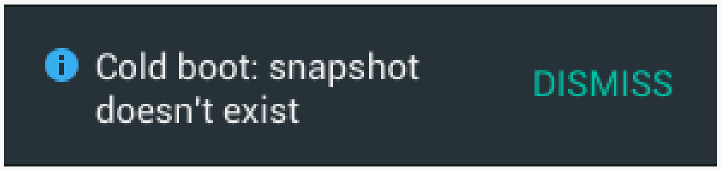

Cold boot: snapshot doesn't exist

Figure 26: Android Emulator Cold-Boot Warning

The emulator behaves a bit more like an Android device. Closing the emulator window is like tapping the POWER button to turn off the screen. The next time you start that particular AVD, it will wake up to the state in which you left it, rather than booting from scratch ("cold boot"). This speeds up starting the emulator. Occasionally, though, you will have the need to start the emulator as if the device were powering on. To do that, in the AVD Manager, in the drop-down menu in the Actions column, choose "Cold Boot Now".

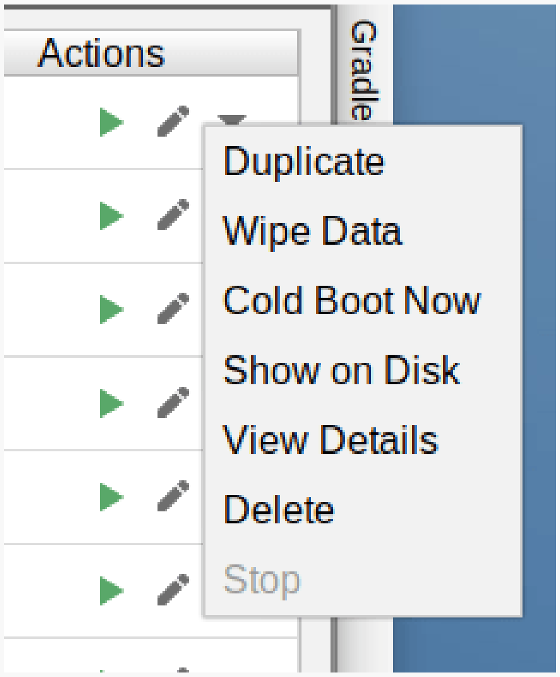


Figure 27: AVD Manager, Showing Actions Drop-Down Menu

Modifying the Manifest

Now that we have our starter project, we need to start making changes, as we have a *lot* of work to do.

In this tutorial, we will start with the Android manifest, one of the core files in an app. Here, we will make a few changes, just to help get you familiar with editing this file. We will be returning to this file — and other core files, like Gradle build files — many times over the course of the rest of the book.

This is a continuation of the work we did in the previous tutorial. The book's GitLab repository contains the results of the previous tutorial as well as the results of completing the work in this tutorial.

You can learn more about the contents of the manifest in the "Inspecting Your Manifest" chapter of *Elements of Android Jetpack*!

Some Notes About Relative Paths

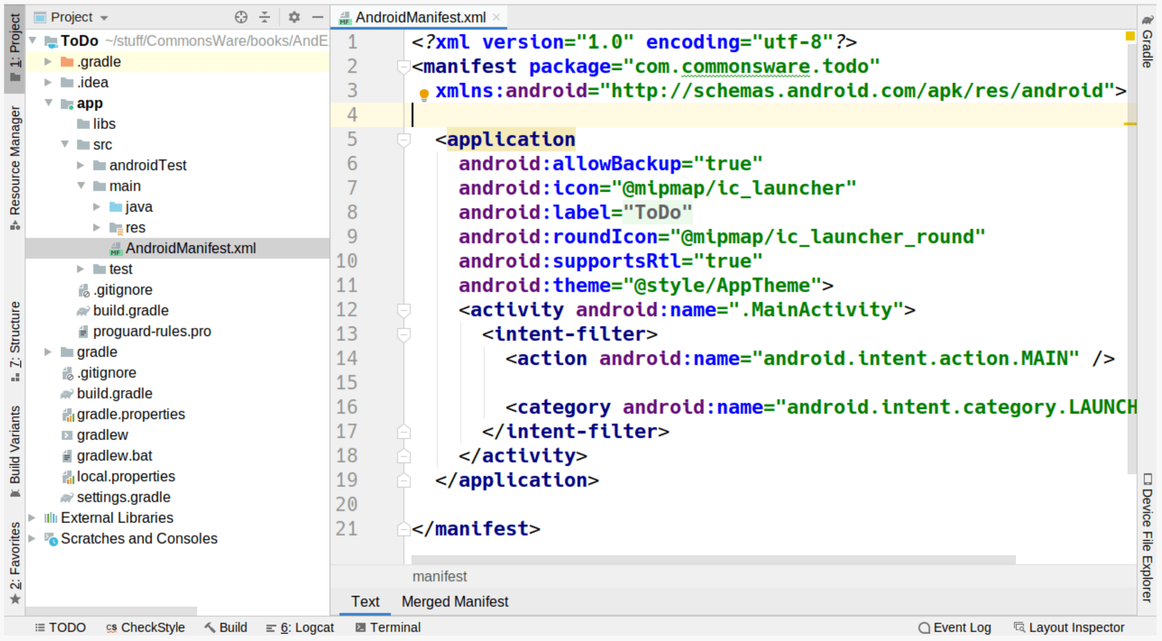
In these tutorials, you will see references to relative paths, like `AndroidManifest.xml`, `res/layout/`, and so on.

You should interpret these paths as being relative to the `app/src/main/` directory within the project, except as otherwise noted. So, for example, Step #1 below will ask you to open `AndroidManifest.xml` — that file can be found in `app/src/main/AndroidManifest.xml` from the project root.

Step #1: Supporting Screens

Android devices come in a wide range of shapes and sizes. Our app can support them all. However, we should advise Android that we are indeed willing to support any screen size. To do this, we need to add a `<supports-screens>` element to the manifest.

To do this, double-click on `AndroidManifest.xml` in the project explorer:



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

  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="ToDo"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

Figure 28: Android Studio, Showing Manifest Editor

As a child of the root `<manifest>` element, add a `<supports-screens>` element as follows:

```
<supports-screens
    android:largeScreens="true"
    android:normalScreens="true"
    android:smallScreens="true"
    android:xlargeScreens="true"/>
```

At this point, the manifest should resemble:

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

  <supports-screens
    android:largeScreens="true"
    android:normalScreens="true"
    android:smallScreens="true"
    android:xlargeScreens="true"/>

  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="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>
```

Step #2: Blocking Backups

If you look at the `<application>` element, you will see that it has a few attributes, including `android:allowBackup="true"`. This attribute indicates that `ToDo` should participate in Android's automatic backup system.

That is not a good idea, until you understand the technical and legal ramifications of that choice.

In the short term, change `android:allowBackup` to be `false`.

Final Results

At this point, your manifest should look like:

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

```
<manifest package="com.commonsware.todo"
  xmlns:android="http://schemas.android.com/apk/res/android">

  <supports-screens
    android:largeScreens="true"
    android:normalScreens="true"
    android:smallScreens="true"
    android:xlargeScreens="true"/>

  <application
    android:allowBackup="false"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="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>
```

(from T03-Manifest/ToDo/app/src/main/AndroidManifest.xml)

What We Changed

The book's GitLab repository contains the entire result of having completed this tutorial. In particular, it contains the changed files:

- `app/src/main/AndroidManifest.xml`