FLUTTER SUCCINCTLY

BY **ED FREITAS**

SUCCINCTLY EBOOK SERIES

Syncfusion®

Flutter Succinctly

By
Ed Freitas

Foreword by Daniel Jebaraj

Technical Reviewer: James McCaffrey
Copy Editor: Courtney Wright
Acquisitions Coordinator: Tres Watkins, content development manager, Syncfusion, Inc.
Proofreader: Jacqueline Bieringer, content producer, Syncfusion, Inc.

Table of Contents

The Story Behind the *Succinctly* Series of Books

Daniel Jebaraj, Vice President
Syncfusion, Inc.

Staying on the cutting edge

As many of you may know, Syncfusion is a provider of software components for the Microsoft platform. This puts us in the exciting but challenging position of always being on the cutting edge.

Whenever platforms or tools are shipping out of Microsoft, which seems to be about every other week these days, we have to educate ourselves, quickly.

Information is plentiful but harder to digest

In reality, this translates into a lot of book orders, blog searches, and Twitter scans.

While more information is becoming available on the Internet and more and more books are being published, even on topics that are relatively new, one aspect that continues to inhibit us is the inability to find concise technology overview books.

We are usually faced with two options: read several 500+ page books or scour the web for relevant blog posts and other articles. Just as everyone else who has a job to do and customers to serve, we find this quite frustrating.

The *Succinctly* series

This frustration translated into a deep desire to produce a series of concise technical books that would be targeted at developers working on the Microsoft platform.

We firmly believe, given the background knowledge such developers have, that most topics can be translated into books that are between 50 and 100 pages.

This is exactly what we resolved to accomplish with the *Succinctly* series. Isn't everything wonderful born out of a deep desire to change things for the better?

The best authors, the best content

Each author was carefully chosen from a pool of talented experts who shared our vision. The book you now hold in your hands, and the others available in this series, are a result of the authors' tireless work. You will find original content that is guaranteed to get you up and running in about the time it takes to drink a few cups of coffee.

Free forever

Syncfusion will be working to produce books on several topics. The books will always be free. Any updates we publish will also be free.

Free? What is the catch?

There is no catch here. Syncfusion has a vested interest in this effort.

As a component vendor, our unique claim has always been that we offer deeper and broader frameworks than anyone else on the market. Developer education greatly helps us market and sell against competing vendors who promise to "enable AJAX support with one click," or "turn the moon to cheese!"

Let us know what you think

If you have any topics of interest, thoughts, or feedback, please feel free to send them to us at succinctly-series@syncfusion.com.

We sincerely hope you enjoy reading this book and that it helps you better understand the topic of study. Thank you for reading.

Please follow us on Twitter and "Like" us on Facebook to help us spread the word about the *Succinctly* series!

About the Author

Ed Freitas is a consultant on software development applied to customer success, mostly related to financial process automation, accounts payable processing, and data extraction.

He really likes technology and enjoys playing soccer, running, traveling, life hacking, learning, and spending time with his family.

You can reach him at: https://edfreitas.me

Acknowledgements

Many thanks to all the people from the amazing Syncfusion team who contributed to this book and helped it become a reality—especially Jacqueline Bieringer, Tres Watkins, Darren West, and Graham High.

The manuscript manager, Jacqueline Bieringer from Syncfusion, and the technical editor, James McCaffrey from Microsoft Research, thoroughly reviewed the book's organization, code quality, and overall accuracy. Thank you both.

This book is dedicated to *Mi Chelin, Lala,* and *Tita,* who inspire me every day and light up my path ahead—God bless you all, always.

Introduction

With the rapid rise of cross-platform mobile frameworks such as Ionic, React Native, and Xamarin, the folks at Google decided to step into the game and develop their own framework with support for both Android and iOS using the same codebase—this is how Flutter came to be.

Flutter is an open-source mobile application development SDK primarily developed and sponsored by Google, used for developing applications for Android and iOS—as well as being the primary method of creating applications for the Google Fuchsia operating system.

Flutter is written in C, C++, and Dart, and uses the Skia Graphics Engine. It offers a rich set of fully customizable widgets for building native interfaces, including the beautiful Material Design library and Cupertino (iOS-flavored) widgets, rich motion APIs, smooth natural scrolling, platform awareness, and hot reload—which helps to quickly build UIs without losing state on emulators, simulators, and any hardware for iOS and Android.

All these great features have helped Flutter take off very quickly, and developers are flocking to the framework. It's also one of the trending projects in GitHub, which has helped it gain even more popularity.

With Flutter gaining momentum, it seems unlikely that it will fade away anytime soon, so I decided to give it a whirl and write an application with it. My personal experience: I was blown away. I felt immediately productive, even though I had never programmed in Dart before.

Throughout this book, I want to go on that same journey with you. We'll do this by creating a fully functional app, which will allow you to get a good grasp of the framework—whether you are coming from another mobile development framework, or have no previous mobile development experience.

I'm quite excited to embark on this journey with you. I hope that by the end of it, you'll have a great impression of the framework and be able to assess whether Flutter is the right choice for your mobile development needs. So, without further ado, let's get going.

Chapter 1 Setup

Project overview

The application that we'll be building throughout this book is one that we can use to keep track of important personal documents that have an expiration date, such as passports, driver's licenses, or credit cards.

This type of application is handy to have so we know when we need to renew these important documents before they expire. If you have read my other book, *Electron Succinctly*, this is the same application concept we explored back then.

Using the Dart programming language with Flutter, we will take our application one step further—we'll also explore how to use a local SQLite database.

Awesome—let's get our engines ready so we can start setting up our development environment straight away.

Installation

The Flutter setup is incredibly easy, with all the installation steps well documented within the official Flutter documentation site.

I'll be using Windows 10, so I'll be describing setup steps and information related to this operating system; however, there are also easy-to-follow setup guidelines for both macOS and Linux.

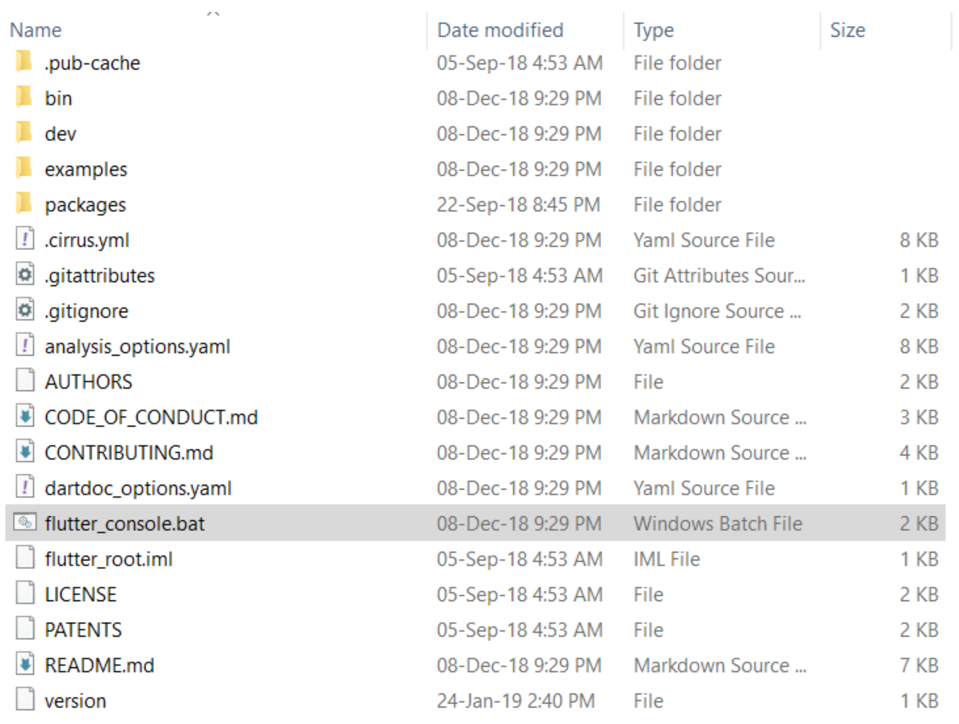
On Windows, there are some essential system requirements that need to be in place, which include having PowerShell 5.0 (or later) and Git for Windows 2.X (or later) installed.

Even though you can write Flutter apps in any editor of your choice—personally, I'm a big fan of Visual Studio Code—Flutter relies on a full installation of Android Studio to supply its Android platform dependencies. You'll also need to set up an Android device emulator. These steps are described in the official documentation.

With the prerequisites in place for Windows, all we need to do is download the installation bundle of the Flutter SDK—at the time of writing, it is Flutter's 1.0.0 stable version for Windows.

Once you've downloaded the zip file, extract it to a desired folder within your drive, such as C:\Flutter. Don't extract the Flutter files to C:\Program Files or C:\Program Files (x86), which require elevated or admin permissions.

Once the files are in the desired folder, locate the file Flutter_console.bat file—this is how it looks on my machine.



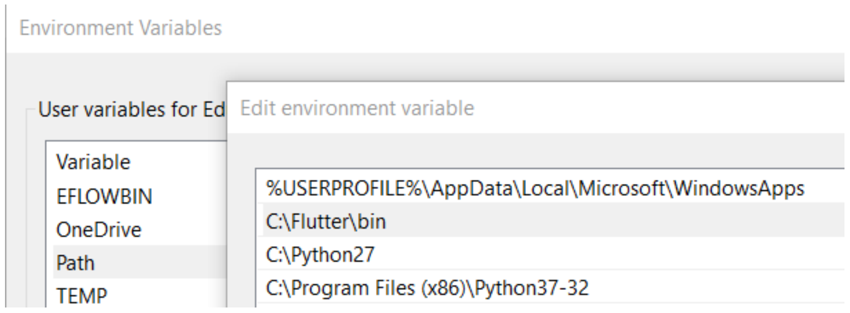
Name	Date modified	Type	Size
.pub-cache	05-Sep-18 4:53 AM	File folder	
bin	08-Dec-18 9:29 PM	File folder	
dev	08-Dec-18 9:29 PM	File folder	
examples	08-Dec-18 9:29 PM	File folder	
packages	22-Sep-18 8:45 PM	File folder	
.cirrus.yml	08-Dec-18 9:29 PM	Yaml Source File	8 KB
.gitattributes	05-Sep-18 4:53 AM	Git Attributes Sour...	1 KB
.gitignore	08-Dec-18 9:29 PM	Git Ignore Source ...	2 KB
analysis_options.yaml	08-Dec-18 9:29 PM	Yaml Source File	8 KB
AUTHORS	08-Dec-18 9:29 PM	File	2 KB
CODE_OF_CONDUCT.md	08-Dec-18 9:29 PM	Markdown Source ...	3 KB
CONTRIBUTING.md	08-Dec-18 9:29 PM	Markdown Source ...	4 KB
dartdoc_options.yaml	08-Dec-18 9:29 PM	Yaml Source File	1 KB
flutter_console.bat	08-Dec-18 9:29 PM	Windows Batch File	2 KB
flutter_root.iml	05-Sep-18 4:53 AM	IML File	1 KB
LICENSE	05-Sep-18 4:53 AM	File	2 KB
PATENTS	05-Sep-18 4:53 AM	File	2 KB
README.md	08-Dec-18 9:29 PM	Markdown Source ...	7 KB
version	24-Jan-19 2:40 PM	File	1 KB

Figure 1-a: The Flutter SDK files

In principle, you are now ready to run the Flutter console by executing the **Flutter_console.bat** file. It's recommended—although not strictly necessary—to add the **Flutter\Bin** folder to the **System Path Environment** variable in Windows.

If you are unsure how to add a folder to the Windows Path variable, please refer to [this nice article]() that explains how to do it, step by step, with screenshots.

In my machine, this looks as follows.



Environment Variables

User variables for Ed

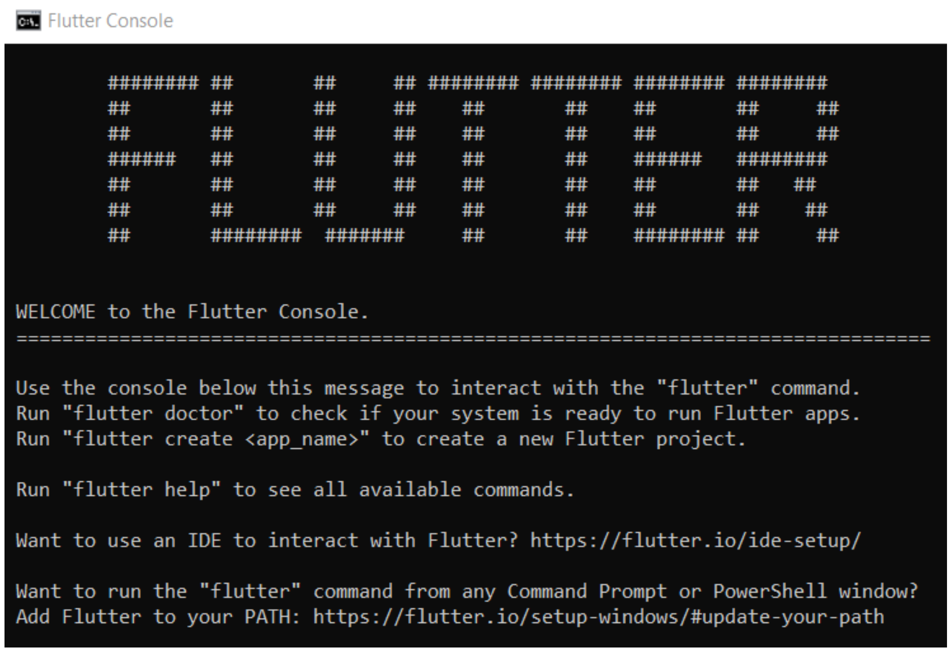
Variable
EFLOWBIN
OneDrive
Path
TEMP

Edit environment variable

```
%USERPROFILE%\AppData\Local\Microsoft\WindowsApps
C:\Flutter\bin
C:\Python27
C:\Program Files (x86)\Python37-32
```

Figure 1-b: Flutter added to the Path variable in Windows

With the SDK file in place, we can now run the **Flutter_console.bat** file—this is what you should see.



```
Flutter Console

        ######## ##       ##     ## ######## ######## ######## ########
        ##       ##       ##     ##    ##       ##    ##       ##     ##
        ##       ##       ##     ##    ##       ##    ##       ##     ##
        ######   ##       ##     ##    ##       ##    ######   ########
        ##       ##       ##     ##    ##       ##    ##       ##   ##
        ##       ##       ##     ##    ##       ##    ##       ##    ##
        ##       ########  #######     ##       ##    ######## ##     ##


WELCOME to the Flutter Console.
================================================================================

Use the console below this message to interact with the "flutter" command.
Run "flutter doctor" to check if your system is ready to run Flutter apps.
Run "flutter create <app_name>" to create a new Flutter project.

Run "flutter help" to see all available commands.

Want to use an IDE to interact with Flutter? https://flutter.io/ide-setup/

Want to run the "flutter" command from any Command Prompt or PowerShell window?
Add Flutter to your PATH: https://flutter.io/setup-windows/#update-your-path
```

Figure 1-c: The Flutter console running

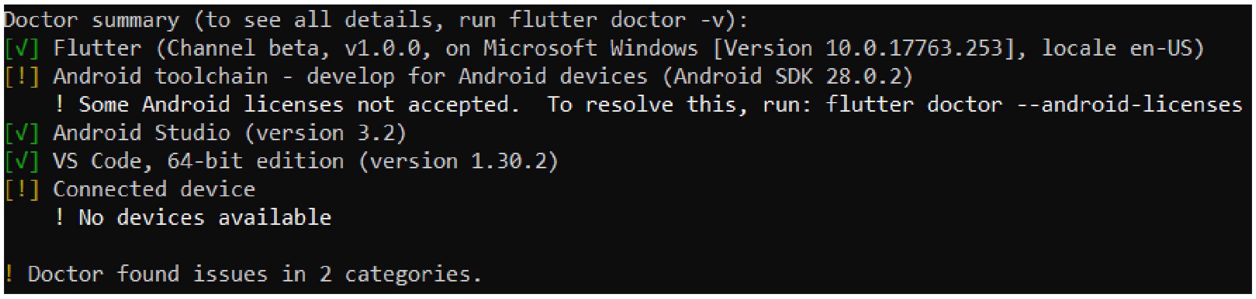
On the prompt, type the following command to check if Flutter is fully operational.

Code Listing 1-a: The "flutter doctor" command

```
flutter doctor
```

After you execute this command, you will get a result with any issues found—in my case, because I had previously installed Android Studio and Visual Studio Code, I get the following information.

When you run the Android Studio installer, please make sure you follow the official documentation so that you end up with a successful Android Studio and SDK setup.



```
Doctor summary (to see all details, run flutter doctor -v):
[√] Flutter (Channel beta, v1.0.0, on Microsoft Windows [Version 10.0.17763.253], locale en-US)
[!] Android toolchain - develop for Android devices (Android SDK 28.0.2)
    ! Some Android licenses not accepted.  To resolve this, run: flutter doctor --android-licenses
[√] Android Studio (version 3.2)
[√] VS Code, 64-bit edition (version 1.30.2)
[!] Connected device
    ! No devices available

! Doctor found issues in 2 categories.
```

Figure 1-d: Results from running the flutter doctor command

In my case, Flutter is telling me that I need to run the **`flutter doctor --android-licenses`** command to resolve an issue with some Android licenses not being accepted.

It also highlighted that I don't have a physical device connected, which is fine for now.

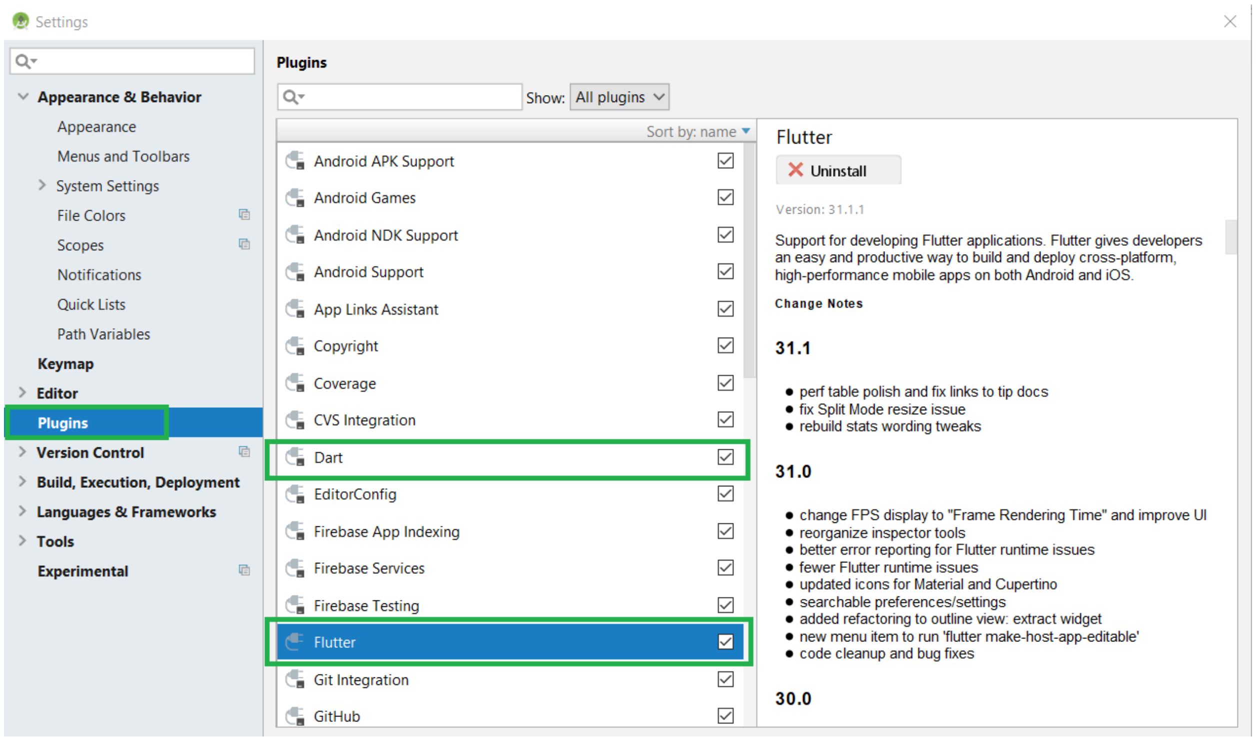
Make sure that you resolve all the conflicts highlighted by the **`flutter doctor`** command before proceeding.

Setting up an editor

Once you have completed all the installation steps, it is necessary to set up Flutter to work with your editor of choice. Although I usually use Visual Studio Code for my projects, this time I decided to use Android Studio to code our Flutter application—to me, it felt more natural, and a better fit for mobile development.

The official Flutter documentation describes how to configure Android Studio (IntelliJ) and Visual Studio Code to work with Flutter—please follow these steps.

If you will also be using Android Studio, once you have followed the steps described, you should see the Dart and Flutter plugins installed. On my machine, this looks as follows.



- Android APK Support ✓
- Android Games ✓
- Android NDK Support ✓
- Android Support ✓
- App Links Assistant ✓
- Copyright ✓
- Coverage ✓
- CVS Integration ✓
- Dart ✓
- EditorConfig ✓
- Firebase App Indexing ✓
- Firebase Services ✓
- Firebase Testing ✓
- Flutter ✓
- Git Integration ✓
- GitHub ✓

Flutter

Version: 31.1.1

Support for developing Flutter applications. Flutter gives developers an easy and productive way to build and deploy cross-platform, high-performance mobile apps on both Android and iOS.

Change Notes

31.1

- perf table polish and fix links to tip docs
- fix Split Mode resize issue
- rebuild stats wording tweaks

31.0

- change FPS display to "Frame Rendering Time" and improve UI
- reorganize inspector tools
- better error reporting for Flutter runtime issues
- fewer Flutter runtime issues
- updated icons for Material and Cupertino
- searchable preferences/settings
- added refactoring to outline view: extract widget
- new menu item to run 'flutter make-host-app-editable'
- code cleanup and bug fixes

30.0

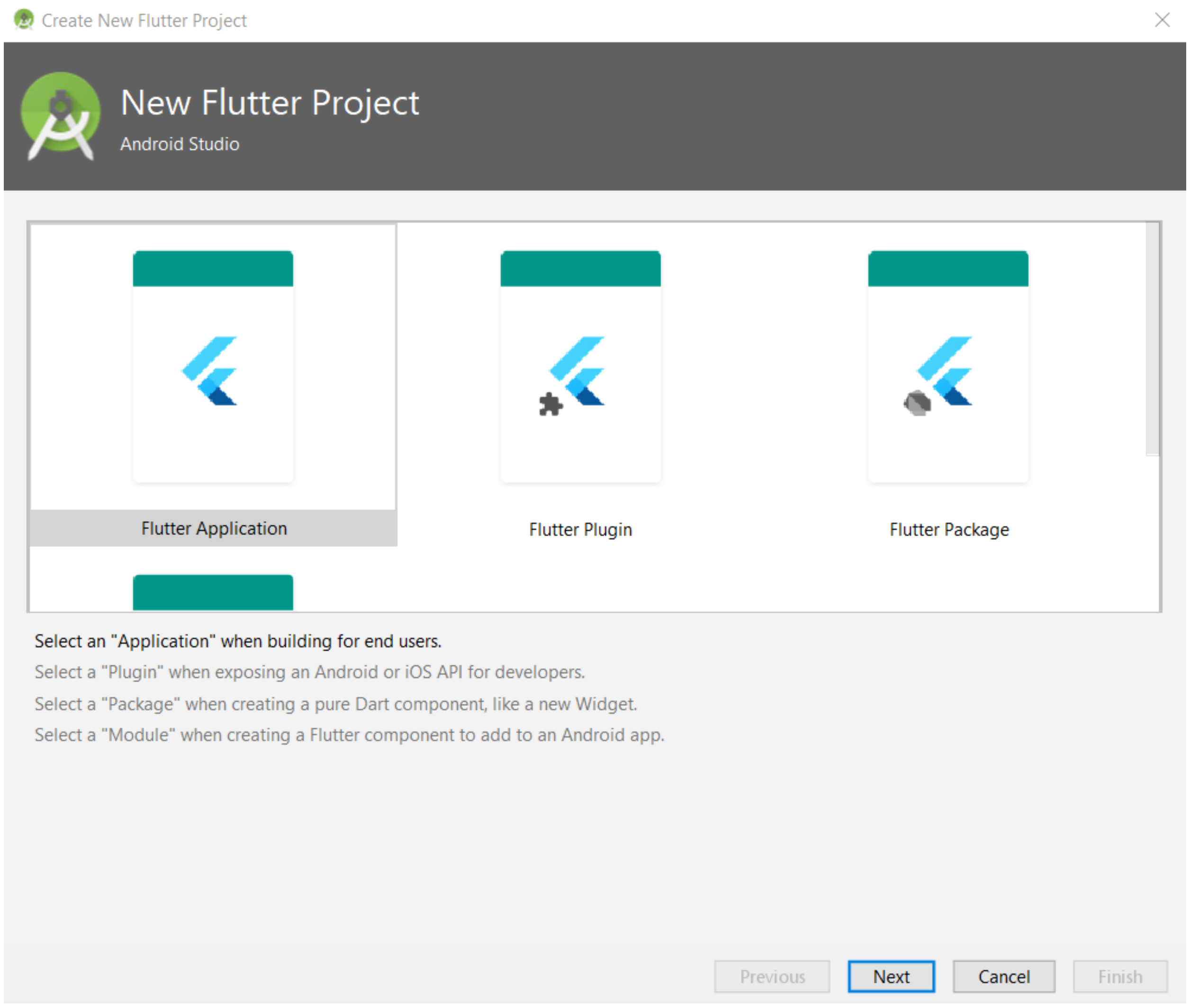
Figure 1-e: The Dart and Flutter plugins installed on Android Studio

Creating the app

Once your editor of choice has been correctly set up following the official documentation guidelines and my previous suggestions, it's time to perform a quick test. We'll create a demo application from one of the predefined templates, and then experience the "hot reload" mechanism after making a change to the app.

The official documentation, which covers the steps that are described and explained in this section, is worth checking.

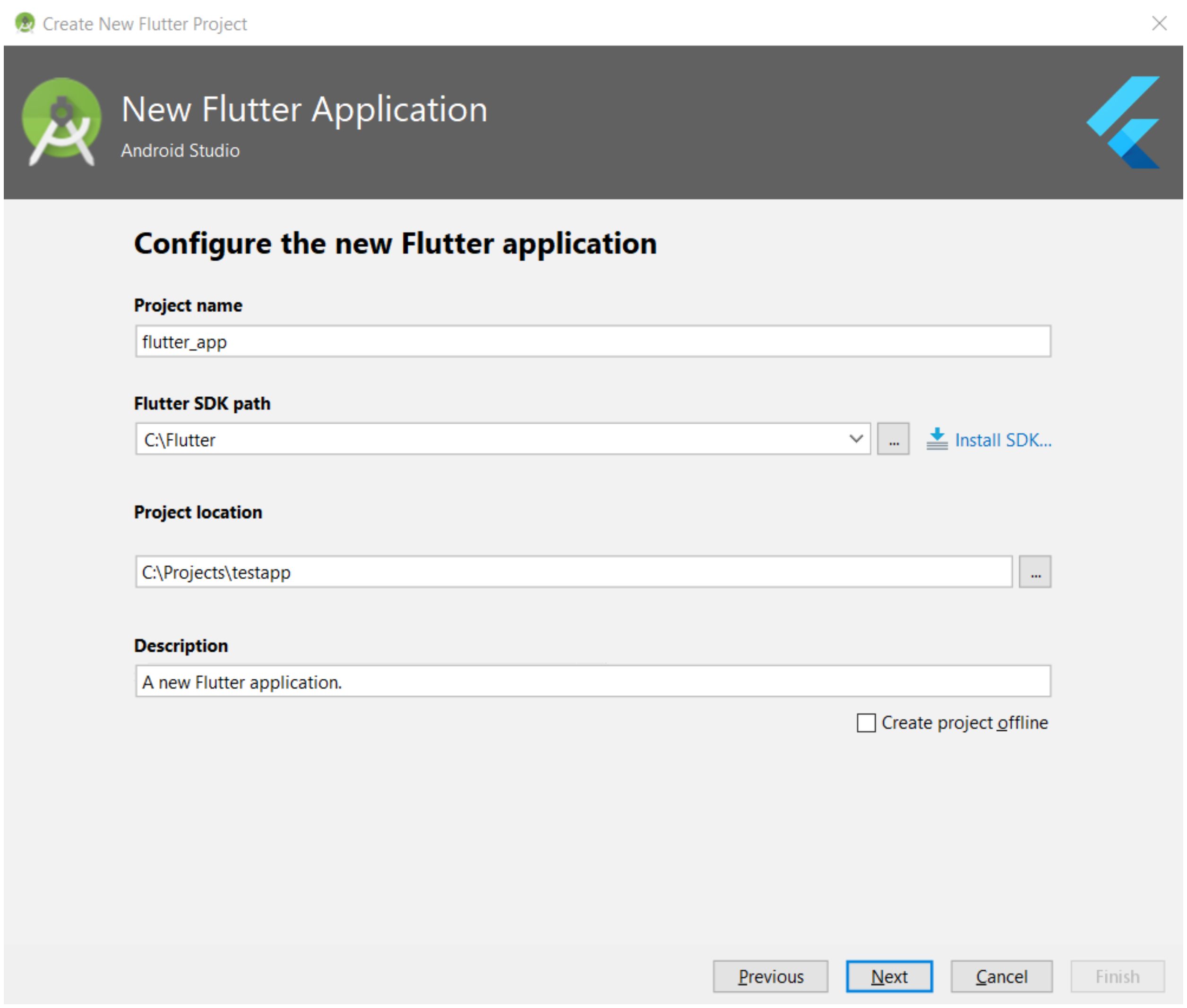
Open Android Studio and navigate to **File** > **New** > **New Flutter Project**. The following screen will be displayed.



Select an "Application" when building for end users.
Select a "Plugin" when exposing an Android or iOS API for developers.
Select a "Package" when creating a pure Dart component, like a new Widget.
Select a "Module" when creating a Flutter component to add to an Android app.

Figure 1-f: Create new Flutter project option (step 1)

Choose the **Flutter Application** option, and then click **Next**. We'll then be presented with a screen where we can enter the **Project name**, **Flutter SDK path**, **Project location**, and a **Description** for the application—we can see this as follows.


Create New Flutter Project

New Flutter Application

Android Studio

Configure the new Flutter application

Project name

flutter_app

Flutter SDK path

C:\Flutter

Project location

C:\Projects\testapp

Description

A new Flutter application.

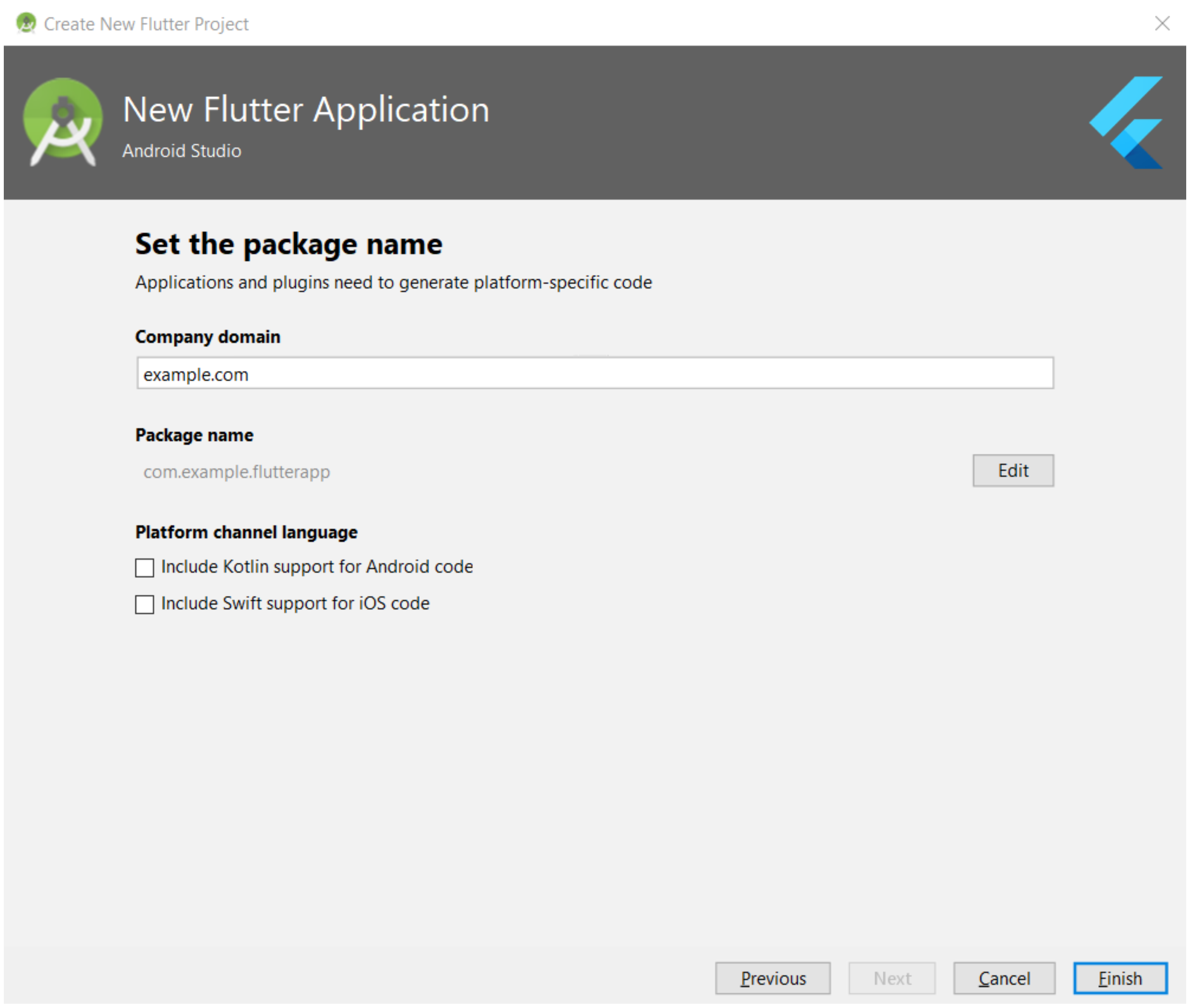
Create project offline

Figure 1-g: Create new Flutter project option (step 2)

Make sure the **Flutter SDK path** text field specifies the correct folder location of the SDK, as previously explained. With those options entered, click **Next**.

In the final step of the app creation process, we are asked to enter the **Company domain** and include (if applicable) **Kotlin support for Android code** and **Swift support for iOS code**—in our case, there's no need to include these options.

The following figure shows the final step of the application creation screen.



Set the package name

Applications and plugins need to generate platform-specific code

Company domain: example.com

Package name: com.example.flutterapp

Platform channel language

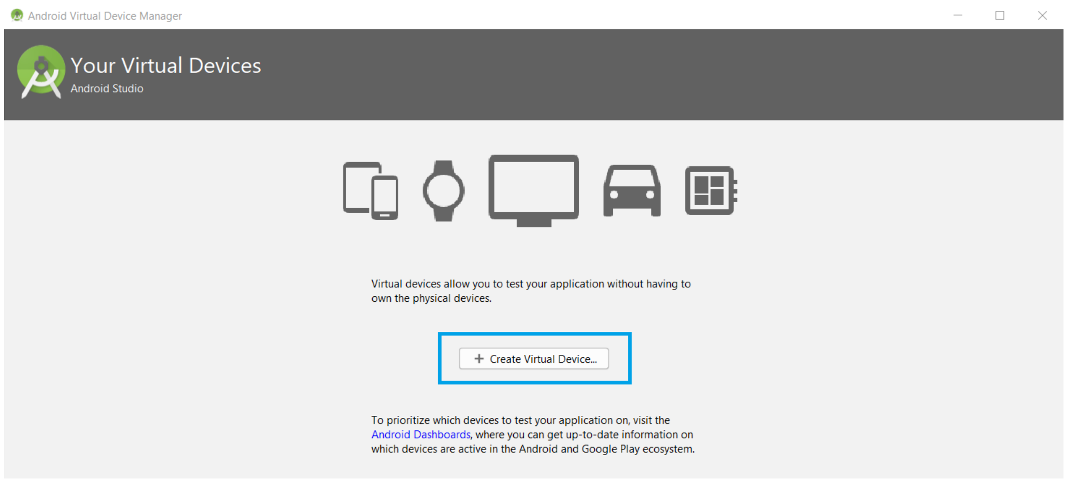
- [ ] Include Kotlin support for Android code
- [ ] Include Swift support for iOS code

Figure 1-h: Create new Flutter project option (step 3)

To finalize the creation of the demo application, click **Finish**. With the demo application created, make sure you have a virtual device created and ready so we can quickly test the app.

Creating a virtual device

Let's quickly go over the steps required to create a virtual device. With Android Studio opened, go to the **Tools** menu and click on the **AVD Manager** option, which will display the following screen.


Android Virtual Device Manager

Your Virtual Devices

Android Studio

Virtual devices allow you to test your application without having to own the physical devices.

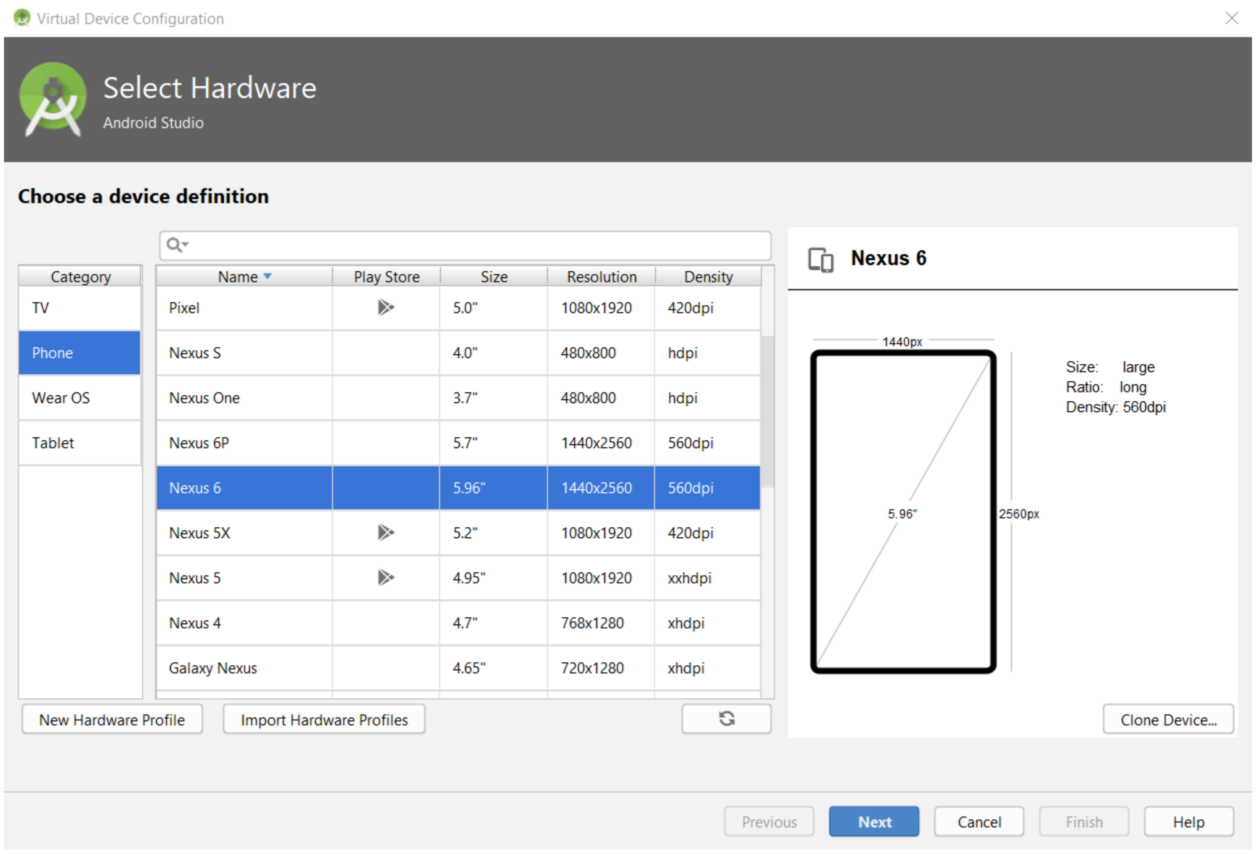
+ Create Virtual Device...

To prioritize which devices to test your application on, visit the Android Dashboards, where you can get up-to-date information on which devices are active in the Android and Google Play ecosystem.

Figure 1-i: Creating a virtual device (step 1)

Then, click **Create Virtual Device**, which will display the following window with all the available virtual devices that can be created for different categories, such as: phones, TV, tablets, and wearable devices.

I’m going to select the **Nexus 6** model from the **Phone** category, but feel free to choose any other.


Virtual Device Configuration

Select Hardware

Android Studio

Choose a device definition

Category
TV
Phone
Wear OS
Tablet

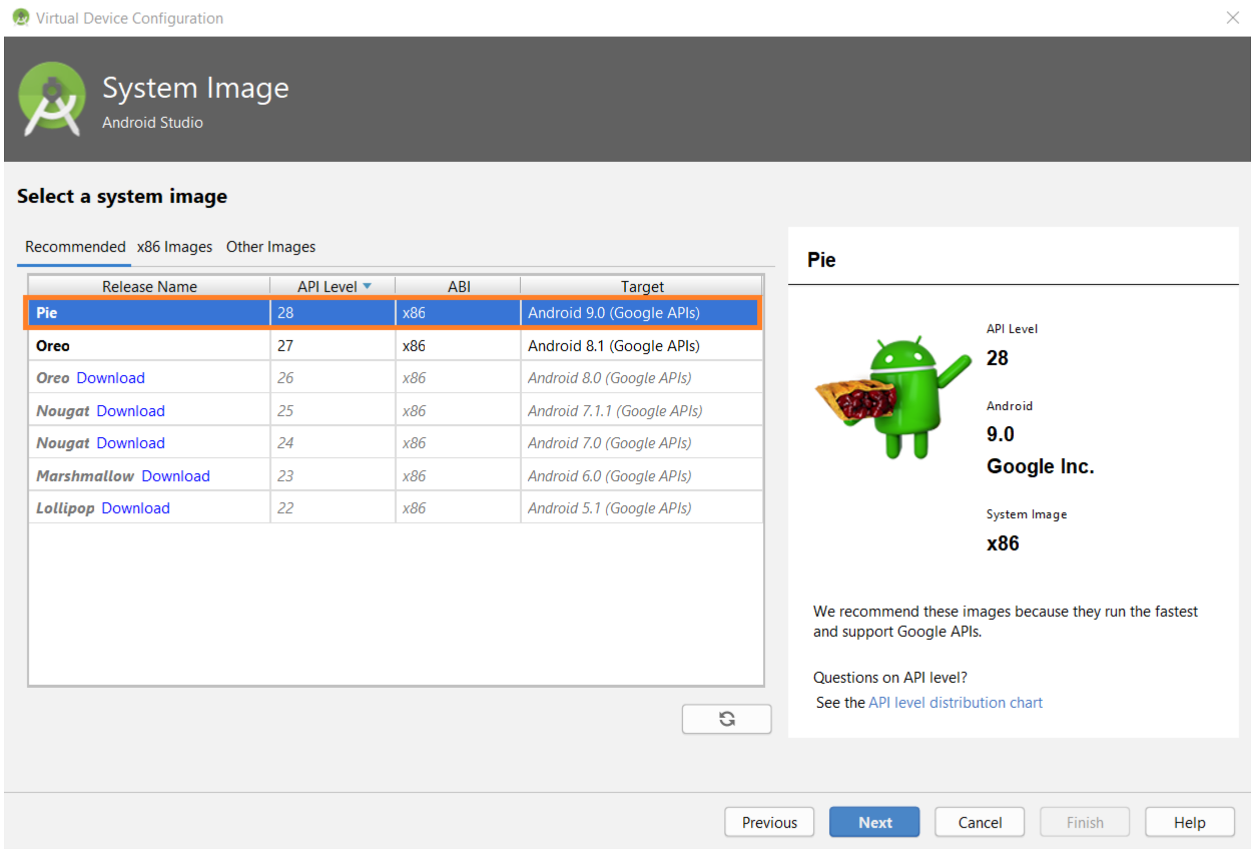
Name	Play Store	Size	Resolution	Density
Pixel	▶	5.0"	1080x1920	420dpi
Nexus S		4.0"	480x800	hdpi
Nexus One		3.7"	480x800	hdpi
Nexus 6P		5.7"	1440x2560	560dpi
Nexus 6		5.96"	1440x2560	560dpi
Nexus 5X	▶	5.2"	1080x1920	420dpi
Nexus 5	▶	4.95"	1080x1920	xxhdpi
Nexus 4		4.7"	768x1280	xhdpi
Galaxy Nexus		4.65"	720x1280	xhdpi

Nexus 6

Size: large
Ratio: long
Density: 560dpi

Figure 1-j: Creating a virtual device (step 2)

Once you have your model selected, click **Next**. You will be prompted to select one of the available device images.



Select a system image

Release Name	API Level	ABI	Target
Pie	28	x86	Android 9.0 (Google APIs)
Oreo	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 APIs)
Nougat Download	24	x86	Android 7.0 (Google APIs)
Marshmallow Download	23	x86	Android 6.0 (Google APIs)
Lollipop Download	22	x86	Android 5.1 (Google APIs)

Pie

API Level
28

Android
9.0
Google Inc.

System Image
x86

We recommend these images because they run the fastest and support Google APIs.

Questions on API level?
See the API level distribution chart

Figure 1-k: Creating a virtual device (step 3)

I'm going with the first recommended option from the available list; however, you can choose any other. It's important to choose an image that plays well with your computer's host operating system. In essence, for emulator performance reasons, it's not recommended to choose an ARM-based image if your computer's host operating system is based on a x86 architecture.

If you've chosen a different image than the one highlighted in Figure 1-k you might have to download the image, using the **Download** link next to the image Release Name field.

Once the image has been selected (and downloaded, if applicable), click **Next** to continue to the last step.