Extracted from:

Programming Flutter

Native, Cross-Platform Apps the Easy Way

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Programming Flutter

Native, Cross-Platform Apps the Easy Way

Carmine Zaccagnino
edited by Michael Swaine

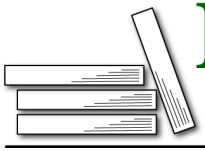
Programming Flutter

Native, Cross-Platform Apps the Easy Way

Carmine Zaccagnino

The Pragmatic Bookshelf

Raleigh, North Carolina



Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

The team that produced this book includes:

Publisher: Andy Hunt
VP of Operations: Janet Furlow
Executive Editor: Dave Rankin
Development Editor: Michael Swaine
Copy Editor: Jasmine Kwityn
Indexing: Potomac Indexing, LLC
Layout: Gilson Graphics

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

ISBN-13: 978-1-68050-695-2
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—February 2020

Preface

This book is about Flutter, Google's open source software development kit (SDK) that can be used to develop applications across a wide range of platforms. We'll begin by taking a brief look at its history, followed by an overview of its features and what we will see during the course of this book.

At the end of this Preface you'll find information about the installation and usage of the Flutter SDK and integrated development environment (IDE) plugins.

A Brief History of Flutter

In 2015 Google unveiled Flutter, a new SDK based on the Dart language, as the next platform for Android development, and in 2017 an alpha version of it (0.0.6) was released to the public for the first time.

At I/O 2017 Google showed off using Flutter and its multi-platform capabilities, and continued promoting it at I/O 2018. Since then, Google has been investing in Flutter and recommending it as the way everyone should be developing mobile apps.

In December 2018 Flutter 1.0 was released and made available so that developers could begin using the SDK to make app creation easier.

At Google I/O 2019, Flutter support for desktop and web platforms was publicly announced. Tools for developing Flutter apps for Windows, macOS, Linux, and the web were released.

In addition to being unstable and untested, desktop development is being held back further by the lack of plugin support, which is very limited mostly because, at the time of writing, plugin tooling is still in the process of being developed, meaning that binaries for the platform-specific code for each platform has to be manually built and linked by editing the Google-provided

Makefiles that can be found in Google's dedicated *flutter-desktop-embedding* GitHub repository.[1]

On the other hand, web support is progressing quickly and shouldn't take much more than a rebuild of a working Flutter mobile project that doesn't have any native plugins or platform-specific code.

Why Flutter Matters and What We'll See in This Book

Flutter's entry into the mobile app development framework space is recent and, because of that, Flutter needs to carry significant improvements over existing frameworks and SDKs to actually be useful—and it does.

For one thing, with Flutter you'll be able to develop apps that work with Android, iOS, and Google Fuchsia,[2] (which might replace Android and/or Chrome OS at some point in the future). Flutter is developed by Google, but it fully supports iOS, and this means you can now also run an iOS emulator and build for iOS in Android Studio. However, you won't be able to build iOS apps on Linux or Windows because iOS emulation and compilation is still done through Xcode.

Flutter makes developing apps incredibly easy by allowing you to define the app's UI declaratively but in the same place and language you define the app logic (no XML UI files required). You can instanly preview the changes you make to your app using stateful hot reload.

Additionally, its cross-platform nature doesn't skimp on having a native look and feel, as the framework supports all of the typical native features of each of the operating systems (different app bar, different list drag to update, Material Design and Apple icons, etc.). The advantages compared to other cross-platform frameworks don't end there: you'll be able to run any native Kotlin/Java and Swift/Objective-C method using platform channels, as we'll see in Integrating Native Code: Making Plugin Packages, on page ?.

Even though it's really new, Flutter is already used by some big and established companies (as well as many smaller ones) to build cross-platform mobile apps, as you can see in Google's Flutter Showcase Page.[3]

We'll be using Flutter packages and plugins (many of them developed by Google) to build ever more useful apps, also introducing more advanced standard Flutter features such as navigation and animations.

1. https://github.com/google/flutter-desktop-embedding
2. https://en.wikipedia.org/wiki/Google_Fuchsia
3. https://flutter.dev/showcase

Don't Know Dart? Don't Worry About It

You might want to read Appendix 1, Introduction to Dart, on page ? if you don't have much programming experience or find even the first chapter difficult to follow because of Dart's syntax; many Dart-specific constructs will be explained during the course of the book, but you might want to consider going through that appendix first if you find yourself struggling to understand the code.

Installing the SDK and the IDE Plugins

To use Flutter, you need to install its SDK, and to be able to get on with your programming in a quick and uncomplicated manner, you'll probably want to install the IDE tools too.

If you prefer to use the command line (maybe because you want to use other, perhaps lighter, IDEs or text editors) there will be guidance on the usage of the `flutter` command throughout the book.

Installing Flutter

The installation process for the Flutter SDK differs slightly for each operating system, so I'll separate the instructions into three sections. Skip ahead to the instructions specific to your platform and, if you want to install them, the part that covers IDE plugin installation.

At the time of writing, the latest stable version is 1.9.1 and requires a 495MB download on Linux (tar.xz archive), a 655MB download on Windows (zip archive), or a 786MB download on macOS (zip archive).

Installing on Linux

On a Linux machine only Android development is supported, so we will install just the Android SDK and the Flutter SDK itself. You'll receive guidance for both CLI and graphical installation methods.

Installing the Android SDK on Linux

If you have never developed Android apps, you need to install the Android SDK, which includes the tools needed to build and debug Android apps.

In order to use Flutter, you need to install Android Studio and the Android SDK tools. Flutter requires Android Studio to be installed, but you don't have to use it for development.

Alternatives to the Official Zip File

On Gentoo and Arch Linux you can use packages to install the Android SDK and/or Android Studio:

- On Gentoo, you can install the Android Studio package[4] by running `emerge --ask dev-util/android-studio`;
- For Arch Linux there is an actively maintained and popular package available on the AUR called *android-studio*[5].

Start by heading over to the downloads page on the Android Developers page:[6] there you'll find links for the .zip download of Android Studio (around 1GB in size).

Regardless of what you choose, you might need to install some 32-bit libraries to make the SDK work on a 64-bit operating system:

- On Debian/Ubuntu, run `sudo apt install libc6:i386 libncurses5:i386 libstdc++6:i386 lib32z1 libbz2-1.0:i386`
- On Fedora, run `sudo dnf install zlib.i686 ncurses-libs.i686 bzip2-libs.i686`.

After that, we're ready to actually install the Android SDK.

To do that using the full Android Studio installer, after extracting the zip file you downloaded from Google's website, run the `studio.sh` script contained in the `bin` subdirectory. This will start a setup wizard. After you complete the setup of the SDK you can launch Android Studio by running that same `studio.sh` script.

Installing the Flutter SDK on Linux

This part will provide guidance for Linux installation aimed at beginners; if you are comfortable with the command line the CLI steps will be more predictable. (For complete Linux newbies it might be easier to follow GUI-oriented guidance since that's usually more familiar.)

Alternatives to the Tarball

There are alternatives to installing the official tarball:

- Arch Linux has an actively maintained package in the AUR to install Flutter.[7]

4. packages.gentoo.org/packages/dev-util/android-studio
5. aur.archlinux.org/packages/android-studio/
6. https://developer.android.com/studio#downloads
7. aur.archlinux.org/packages/flutter/

The Linux download is a source tarball that also contains the script needed to run the flutter command, which means you just need to extract it and add the bin subdirectory of the extracted tarball to the PATH environment variable.

Before we can do that, we need to browse to the SDK archive[8] page on Flutter's official website and download the latest stable version.

Alternatively, in a CLI-only environment, you can download the 1.2.1 tarball using curl https://storage.googleapis.com/flutter_infra/releases/stable/linux/flutter_linux_v1.2.1-stable.tar.xz -o flutter_linux_v1.2.1-stable.tar.xz.

Extract the tarball you just downloaded with any GUI tool of your liking or by running the following command:

```
$ tar -xf flutter_linux_v1.2.1-stable.tar.xz
```

Now we need to add the executable script to the PATH environment variable.

Before doing this, you need to take note of the directory where you extracted the tarball. It contains a flutter directory, inside which there is a bin directory; we need to know the path to reach that bin directory.

If you are using the GUI, in most distribution it is available by browsing the directory's properties. It will be something along the lines of /home/username/Downloads/flutter_linux_v1.9.1-stable/flutter/bin if you have gone with the default settings for each piece of software used in the steps we described earlier. I suggest moving this to a more permanent path; ideally one at which you'll remember you have installed Flutter.

If you worked in the CLI using the commands just outlined, browse to your Flutter installation directory and then change the working directory to the flutter/bin subdirectory by running:

```
$ cd flutter/bin
```

and get the working directory by running:

```
$ pwd
```

which will return something along the lines of /home/username/flutter/bin (here you'll see the path where you installed Flutter, so your mileage may vary significantly).

8. https://flutter.dev/docs/development/tools/sdk/archive?tab=linux

Know Your Shell

This section supposes that your shell is Bash.[9] This is the case for most Linux distributions (and Unix-like operating systems in general—some exceptions are the BSDs, which have tcsh,[10] ksh,[11] or ash,[12] and Arch Linux's installer, which runs on Zsh[13] but installs Bash by default) and, since you would need to manually install and configure a different one, you probably would know how to add a directory to its PATH.

To add this to the PATH environment variable, we need to edit ~/.bash_profile.

To do that using the GUI, you first need your file manager to display hidden files. If you can't find a file named .bash_profile in your home directory, you need to toggle the option that makes the file manager show hidden files, and this depends on the file manager that you're using:

- In Nautilus (a.k.a. GNOME Files, default in most distributions using the GNOME desktop like Ubuntu, RHEL, Fedora, and default SLED and Debian) you need to press Ctrl+H. This shortcut also works in PCManFM (part of the LXDE, as found in Lubuntu), Caja (part of Mate), and Thunar (part of the XFCE desktop, as found in Xubuntu).
- If you are using the KDE desktop (for example when running Kubuntu or when choosing it when installing distributions like openSUSE or Debian) and its default Dolphin file manager, use Alt++.

Once you have located a file called .bash_profile, open it with any text editor and add the following line to the end of it, in a new line:

```
export PATH=$PATH:/home/username/etc
```

where you'll replace */home/username/etc* with the string you took note of earlier.

If you prefer using the command line or just want a copy-paste experience from this installation guide, you can instead open a terminal window or TTY and run the following command:

```
$ echo "export PATH=$PATH:/home/username/etc" >> ~/.bash_profile
```

9. en.wikipedia.org/wiki/Bash_(Unix_shell)
10. https://en.wikipedia.org/wiki/Tcsh
11. https://en.wikipedia.org/wiki/KornShell
12. https://en.wikipedia.org/wiki/Almquist_shell
13. https://en.wikipedia.org/wiki/Z_shell

replacing /home/username/etc with the path you found earlier for the extracted tarball's flutter/bin directory.

To actually be able to run the flutter command, you need to refresh your terminal's configuration by running:

```
$ source ~/.bash_profile
```

Installing on Windows

Just like on Linux, Windows-only Android development is supported, so we will install just the Android SDK with Android Studio and the Flutter SDK itself.

Installing the Android SDK on Windows

All you need to do to install the Android SDK on Windows is to download and run the Android Studio installation file available on the official download page.[14] This will also install the feature-rich Android Studio IDE. After installing Android Studio, you will be guided through the installation of the Android SDK.

Installing the Flutter SDK on Windows

To install the Flutter SDK on Windows you need to install Git for Windows first. You can find the installation file for it on its official download page.[15] During installation, you need to choose the *Use Git from the Windows Command Prompt* option.

After you have installed Git, download the latest version of Flutter from its official installation page.[16] This, unlike what happens with most of the software available for Windows, will download a zip archive (a.k.a. a compressed folder) that contains a flutter folder: extract it wherever you want (this will be the SDK folder any IDE plugin will ask you to enter).

If you want to run a command on the Flutter Console, run the flutter.bat script contained in the bin subfolder.

Installing on macOS

The advantage macOS has is that it also supports iOS building and debugging. To take advantage of that, we also need to install Xcode and the iOS SDK.

14. https://developer.android.com/studio#downloads
15. https://git-scm.com/download/win
16. https://flutter.dev/docs/development/tools/sdk/archive

Installing Xcode and the iOS SDK

To be able to run Flutter and use it for iOS development you need to download Xcode[17] and the iOS SDK[18] from Apple's official website.

Installing the Android SDK on macOS

Installing the Android SDK on macOS is really simple: head over to the Android Studio downloads page[19] to download the .dmg file that also includes the feature-rich but resource intensive Android Studio IDE.

Installing the Flutter SDK on macOS

Installing Flutter on macOS is very similar to installing Flutter on Linux. The main differences are that macOS uses the zip archive format instead of the more efficient tar.xz format, and there are no OS or GUI differences among distributions to contend with.

Start by downloading the latest stable .zip from Flutter's official website.[20]

Unzip it to any directory, then open the flutter directory you just extracted. Press ⌘+I and take note of the path that appears to the right of *Where:*.

Now, open a Terminal window and run the command:

```
$ nano ~/.bash_profile
```

After doing that, paste the following in the terminal window:

```
export PATH=$PATH:/example/path/to/flutter/bin
```

replacing /example/path/to/flutter/bin with the path you took note of earlier.

All that's left to do is to close the file using Ctrl-X and confirming you want to save the file by pressing Y.

To actually be able to run the flutter command, you need to refresh your terminal's configuration by running:

```
$ source ~/.bash_profile
```

Installing the IDE Plugins

The Flutter IDE plugins for VSCode and Android Studio are installed using the canonical installation tools and techniques for each IDE. They also require the installation of the respective Dart plugin.

17. https://developer.apple.com/xcode/
18. https://developer.apple.com/ios/
19. https://developer.android.com/studio#downloads
20. https://flutter.dev/docs/development/tools/sdk/archive?tab=macos

If you're not familiar with that process, we'll now discuss how to install the Flutter plugin on each IDE.

VSCode

To install the Flutter plugin for Visual Studio Code you need to open Visual Studio Code itself, then use the keyboard shortcut Ctrl-Shift-P to open the command palette, then type "Install extensions" and press Enter.

On the panel that opens up on the left, type Flutter and press Enter.

Click the Install button in the Flutter entry in the list, this will install both the Flutter and the Dart plugins.

Unlike the Flutter plugin for Android Studio, the Flutter plugin for Visual Studio Code plugin will auto-detect the location where the Flutter SDK is installed.

Android Studio

To install the Flutter plugin for Android Studio you need to navigate to File > Settings > Plugins, click *Browse repositories...*, type Flutter and click Install.

To create a Flutter Project in Android Studio you need to restart Android Studio if you haven't since installing the Flutter plugin, navigate to File > New Flutter Project, choose *Flutter Application* and give the app a name and a path, set the package name and you're done.

Using the CLI and the Plugins

There are two ways of interacting with the Flutter SDK: using the `flutter` CLI command directly or through the IDE plugins. The plugins for Android Studio and VSCode are very different to use, as they adapt to the usual conventions of the respective IDE.

Create a Flutter App Project

To create a Flutter app project, run the `flutter create appname` command or follow the procedure corresponding to the IDE you want to use to develop your app:

- In Android Studio, click on *File* > *New* > *New Flutter Project...* and follow the instructions that will appear on screen;
- In Visual Studio Code, hit Ctrl+Shift+P to open the Command Menu and type New Project, *Flutter: New Project* should appear as an option in the menu, click on it and then type a name for your app project.

Building and Running

You can build a Flutter app using the `flutter build` command when the working directory is the project's root directory (where `pubspec.yaml` is located):

- To build an APK for your app (standard Android installation file), use `flutter build apk`.
- To build an iOS application bundle, use `flutter build ios`.
- To build an Android App Bundle (the new generation Android installation file that reduces file size), use `flutter build appbundle`.

If you want to run the app directly on an open emulator or a device connected by USB (with USB Debugging turned on in the device's settings), use `flutter run`.

If you're using an IDE, you can use the Flutter plugin to run apps directly from the IDE.

To run an app on an open emulator or connected USB device:

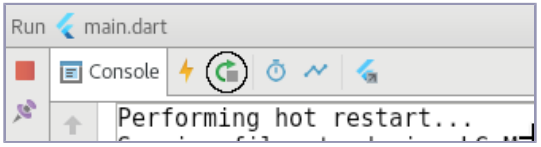
- Using VSCode, press `F5` or click on *Debug* > *Start Debug*.
- Using Android Studio, press `Shift`-`F10` and click the button that looks like the one highlighted in the following image:

Hot Reload

Flutter comes with a feature you might know from other frameworks: hot reload. If you're not familiar with it, the hot reload feature allows you to update the app's current view based on changes you've made to the code without rebuilding the entire app which, as anyone who has ever tried knows, is very painful when you are making many small changes and trying to see how they affect the app.

This feature is accessible in both Android Studio and Visual Studio Code and, actually, also in the command line when you run an app using `flutter run`: pressing `R` in the terminal when that command is running will reload the app.

In Android Studio you need to find an icon like this in the *Run* menu at the bottom:



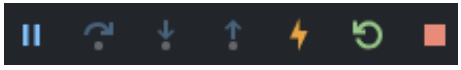
In Visual Studio Code you need to find an icon like this in the menu that appears at the top of the screen when you start debugging:

The app will immediately reload, showing any changes you've made to the app's code immediately on the connected emulator or device.

If you're running a recent version of Android Studio and/or Visual Studio Code, you'll have IDE access to two different but similar features: stateful hot reload and hot restart. The difference between the two is that the stateful hot reload (as the name implies) preserves each of the `State` objects and is faster, but it won't work with bigger changes that require those to be reloaded too, and for that there's hot restart.

When running an app using `flutter run` from the CLI, stateful hot reload is associated to the lowercase `r` character, while hot restart is achieved with the uppercase `R` character.

In both IDEs you'll find, next to the button shown above, a lightning bolt icon like the one you can see in the following VSCode screenshot:



The lightning bolt performs the stateful hot reload, while the typical *restart* icon performs the hot restart.

Updating and Maintaining Your Flutter Installation

You'll know when you need to update your Flutter SDK installation because you'll get a notice whenever you build or run an app.

To update your Flutter installation you need to run the `flutter upgrade` command, which will check and download anything that's needed using Git, so you need to have Git installed.

Flutter includes a tool called Flutter Doctor, available using the `flutter doctor` command, which will list information about the installed Flutter version, the Android/iOS SDK, the IDE plugins, and connected devices.

Other Flutter-Related Commands in the VSCode Plugins

Once the VSCode Flutter plugin is installed, opening the command palette and typing `flutter` will show all available commands.

Among these you'll want to get familiar with:

Flutter: Run Flutter Doctor
: Check if there are dependencies that need to be installed.

Flutter: New Project
: Creates a new Flutter project.

Flutter: Select Device
: Select the device on which you want to debug your app.

Flutter: Get Packages and Flutter: Upgrade Packages
: Commands for interacting with Flutter Packages (we'll discuss packages in Chapter 4).

Where We're Going Next: Let's Start Building Apps

Now that you're set up with all you need to build Flutter apps, we can start building apps.

You can begin by creating a new Flutter project and familiarizing yourself with the files generated by the SDK as well as the IDE's UI for managing Flutter projects. Once you feel comfortable working with it, you can move on to the first chapter, in which we'll learn the structure of a Flutter app and how to implement it.