# XAMARIN.FORMS

## SUCCINCTLY

BY ALESSANDRO DEL SOLE

SUCCINCTLY EBOOK SERIES

Syncfusion®

# Xamarin.Forms Succinctly

**By**

**Alessandro Del Sole**

---

Foreword by Daniel Jebaraj

# Chapter 1 Getting Started with Xamarin.Forms

Before you start writing mobile apps with Xamarin.Forms, you first need to understand the state of mobile app development today and how Xamarin fits into it. Also, you need to set up your development environment to be able to build, test, debug, and deploy your apps to Android, iOS, and Windows devices. This chapter introduces Xamarin as a set of tools and services, Xamarin.Forms as the platform you will use, and then presents the tools and hardware you need for real-world development.

## Introducing Xamarin and Xamarin.Forms

Xamarin is the name of a company that Microsoft acquired in 2016 and, at the same time, the name of a set of development tools and services that developers can use to build native apps for iOS, Android, and Windows in C#. Xamarin's main goal is to make it easier for .NET developers to build native apps for Android, iOS, and Windows reusing their existing skills. The reason behind this goal is simple: building apps for Android requires you to know Java and Android Studio or Eclipse; building apps for iOS requires you to know Objective-C or Swift and Xcode; building apps for Windows requires you to know C# and Visual Studio. As an existing .NET developer, whether you are experienced or a beginner, getting to know all the possible platforms, languages, and development environments is extremely difficult, and costs are extremely high.

Xamarin allows you to build native apps with C#, based on a cross-platform, open-source porting of the .NET Framework called Mono. From a development point of view, Xamarin offers a number of flavors: Xamarin.iOS and Xamarin.Mac, libraries that wrap native Apple APIs you can use to build apps for iOS and macOS using C# and Visual Studio; Xamarin.Android, a library that wraps native Java and Google APIs you can use to build apps for Android using C# and Visual Studio; and Xamarin.Forms, an open-source library that allows you to share code across platforms and build apps that run on Android, iOS, and Windows from a single C# codebase. The biggest benefit of Xamarin.Forms is that you write code once and it will run on all the supported platforms at no additional cost. As you'll learn throughout this ebook, Xamarin.Forms consists of a layer that wraps objects common to all the supported platforms into C# objects. Accessing native, platform-specific objects and APIs is possible in several ways, all discussed in the next chapters, but it requires some extra work. Additionally, Xamarin integrates with the Visual Studio IDE on Windows and is also part of Visual Studio for Mac, so you can not only create cross-platform solutions, but also write code on different systems.

The Xamarin offering also includes Xamarin University, a paid service that allows you to attend live classes online and watch instructional videos that will help you prepare to get the Xamarin Certified Mobile Developer badge. It also includes the Xamarin Test Cloud service for test automation, a complete cloud solution for the app-management lifecycle from build automation to continuous integration, tests, analytics, and much more (note that Internet Explorer is not supported), which is now part of the Visual Studio Mobile Center. This ebook focuses on Xamarin.Forms and targets Visual Studio 2017 on Windows 10, but all the technical concepts

apply to Visual Studio for Mac as well. However, if you prefer working on a Mac, I recommend that you read my ebook *Xamarin.Forms for macOS Succinctly*. I've also recorded a video series for Syncfusion that provides an overview of what Xamarin offers, and of Xamarin.iOS, Xamarin.Android, and Xamarin.Forms.

### Supported platforms

Out of the box, Xamarin.Forms allows creating apps for Android, iOS, and the Universal Windows Platform from a single C# codebase. Recently, the range of supported platforms has been expanded to include Tizen (an operating system by Samsung for proprietary devices). Additionally, Microsoft is working on previews of support for macOS, WPF, and GTK#. As you can imagine, this opens incredible opportunities in the market of cross-platform development because you can target both mobile and desktop systems. In this ebook, I will target the most popular operating systems (Android, iOS, and Windows 10) because support for other platforms is not yet released (apart from Tizen). You can read the official documentation about targeting macOS, Tizen, WPF, and GTK# in your Xamarin.Forms projects, with the assumption that your shared code will not change.

## Setting up the development environment

In order to build native mobile apps with Xamarin.Forms, you need Windows 10 as your operating system and Microsoft Visual Studio 2017 as your development environment. You can download and install the Visual Studio 2017 Community edition for free and get all the necessary tools for Xamarin development. I will discuss the latest stable release of Xamarin.Forms, version 3.1, in this ebook, so make sure you install version 15.8 or later of Visual Studio 2017.

When you start the installation, you will need to select the **Mobile development with .NET** workload in the Visual Studio Installer (see Figure 1).

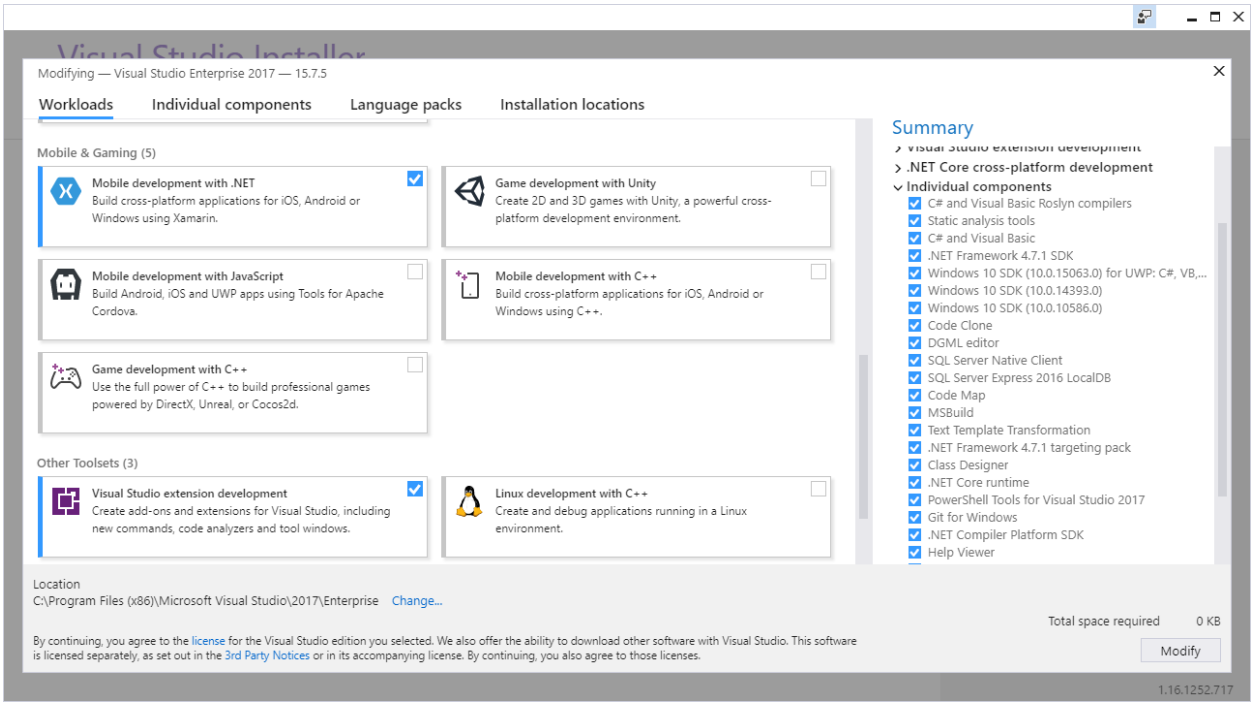


*Figure 1: Installing Xamarin development tools*

When you select this workload, the Visual Studio Installer will download and install all the necessary tools to build apps for Android, iOS, and Windows. iOS requires additional configuration, described in the next section. Also, for Windows 10 development, you need additional tools and SDKs, which you can get by also selecting the **Universal Windows Platform Development** workload. If you select the **Individual components** tab, you will have an option to check if Android and Windows emulators have been selected or to make a choice manually (see Figure 2).

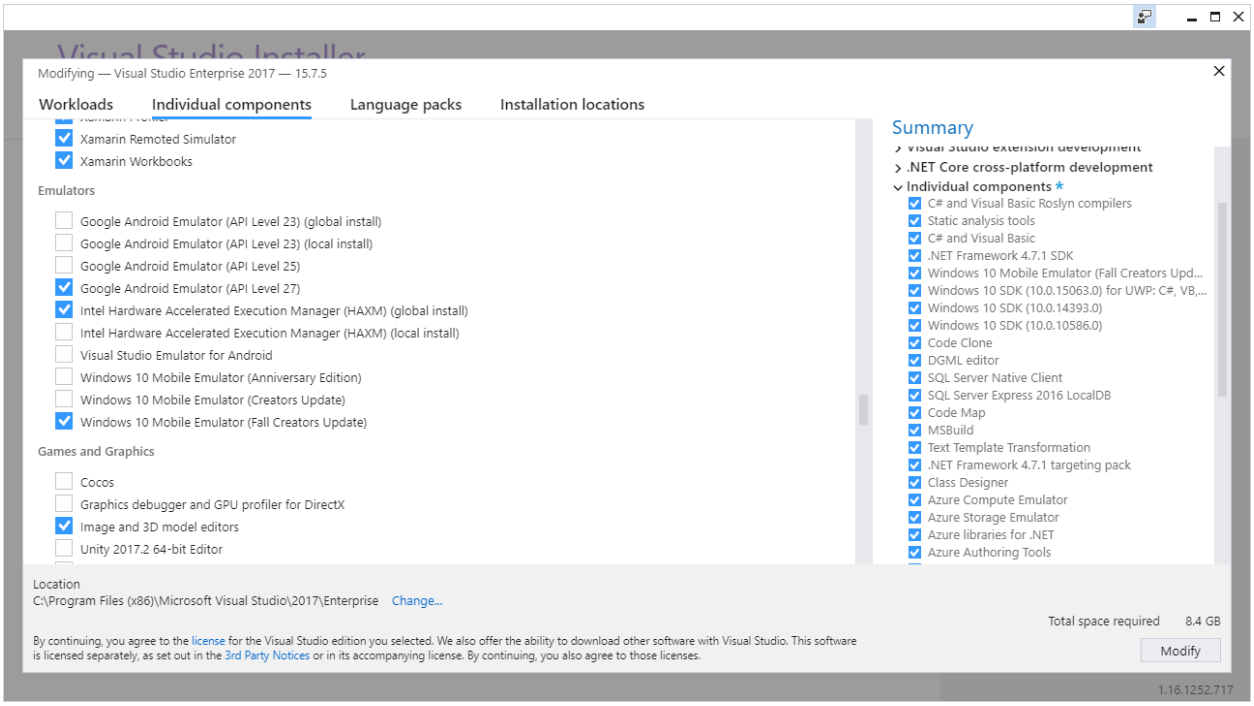


*Figure 2: Selecting emulators*

Whether you will use Visual Studio 2017, Visual Studio for Mac, or both, I suggest you install the Google emulator, which has an identical appearance and behavior on both systems.

> ***Note: With Visual Studio 2017 version 15.8, Microsoft has relesead a Google Android emulator that runs on Hyper-V. This is a very important addition because many developers used to work with the Visual Studio Android Emulator, which is based on Hyper-V, and was recently discontinued. The new emulator requires the Windows 10 April 2018 update with the Windows HyperVisor Platform installed and Visual Studio 2017 15.8. The emulator will not be covered here because of its particular system requirements, and for consistency with development on Mac (which does not have Hyper-V), but you can read the documentation for further details.***

For the Windows 10 emulator, my suggestion is to download the oldest version if you plan to target older versions of Windows 10; otherwise, the most recent is always a good option (Fall Creators Update in Figure 2). Go ahead with the installation and wait for it to complete.

## Configuring a Mac

Apple's policies establish that a Mac computer is required to build an app. This is because only the Xcode development environment and the Apple SDKs are allowed to run the build process. For local debugging and testing, Xamarin offers the Xamarin Live Player app, which you can download and install on your Android or iOS device and pair with Visual Studio for debugging purposes. More details about this app will be provided shortly. However, it becomes insufficient for serious development. You still need a Mac computer for code signing, setting up profiles, and publishing an app to the App Store. You can use a local Mac in your network, which also allows you to debug and test apps on a physical device, or a remote Mac. In both cases, macOS must be configured with the following software requirements:

- macOS "El Capitan" (10.11) or higher.
- Xcode and Apple SDKs, which you get from the App Store for free.
- Xamarin.iOS engine. The easiest way to get Xamarin properly configured on a Mac is by installing Visual Studio Community for Mac.

Visual Studio will connect to the Mac to launch the Xcode compiler and SDKs; remote connections must be enabled for the latter. The official Xamarin documentation has a specific page that will help you configure a Mac. I recommend you read it carefully, especially because it explains how to configure profiles and certificates, and how to use Xcode to perform preliminary configurations. The documentation is actually about Xamarin.iOS, but the same steps apply to Xamarin.Forms.

***Tip: Starting with Xamarin.Forms 3.0, Visual Studio 2017 introduced integrated tools that simplify the Mac configuration from within the IDE, and without the need to work on the Mac directly. These improvements are discussed in the "Xamarin.iOS project" section later in this chapter.***

# Creating Xamarin.Forms solutions

Assuming that you have installed and configured your development environment, the next step is opening Visual Studio to see how you create Xamarin.Forms solutions and what these solutions consist of. Project templates for Xamarin.Forms are available in the **Visual C#**, **Cross-Platform** node of the **New Project** dialog window (see Figure 3).

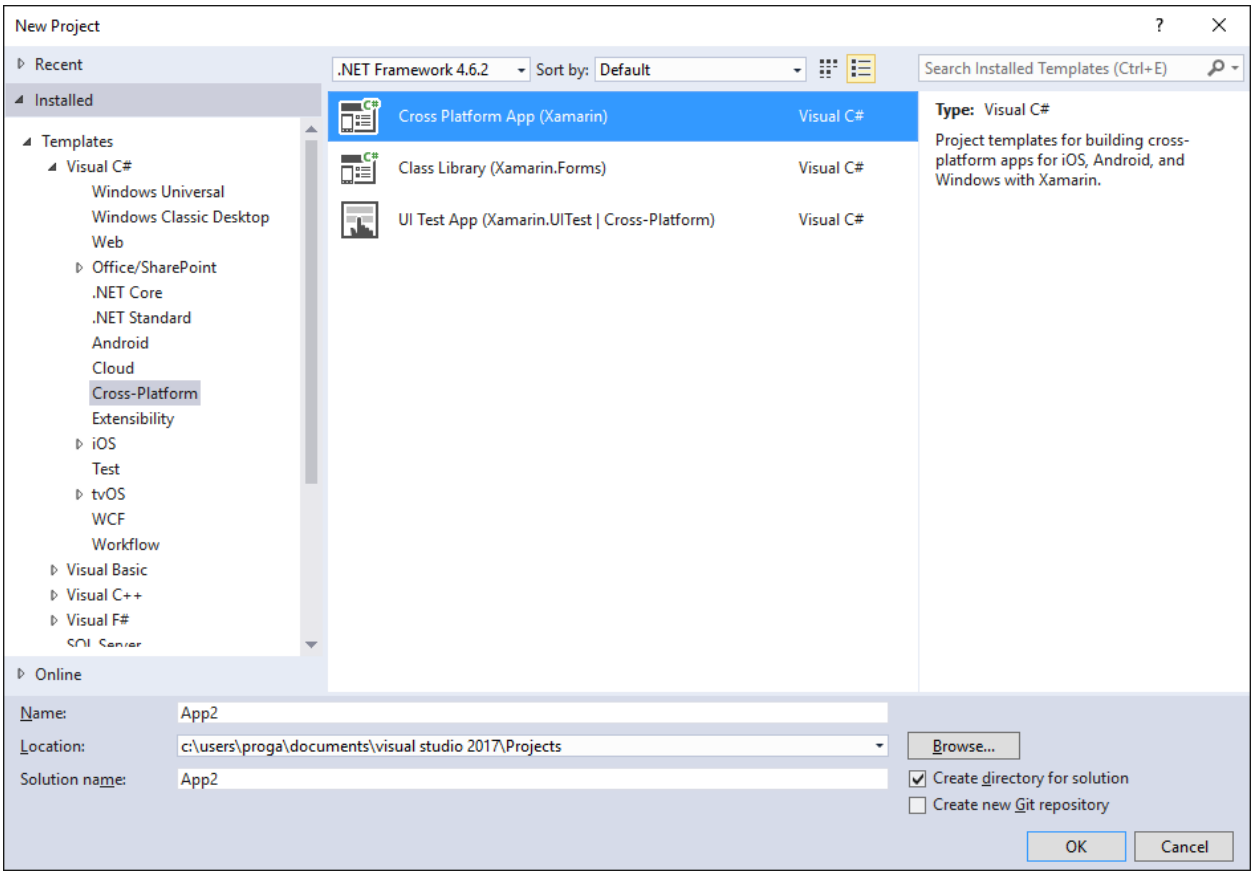


*Figure 3: Project templates for Xamarin.Forms*

The **Cross Platform App (Xamarin)** template is the one you use to build mobile apps. The Class Library (Xamarin.Forms) template allows you to create a reusable class library that can be consumed in Xamarin.Forms projects, and the UI Test App template is used to create automated UI tests, but these two templates will not be discussed in this ebook.

Select the **Cross Platform App** template, and optionally, specify a different project name than the default (this is not relevant right now). Then click **OK**. At this point, Visual Studio will ask you to select between Blank App, Master-Detail, and Tabbed templates (see Figure 4). The Master-Detail template generates a basic user interface based on pages and visual elements that will be discussed later, with some sample data. The Tabbed generates a basic user interface based on tabs to navigate between child pages. Neither Master-Detail nor Tabbed are a good starting point (unless you already have experience with Xamarin), so select the **Blank App** template.

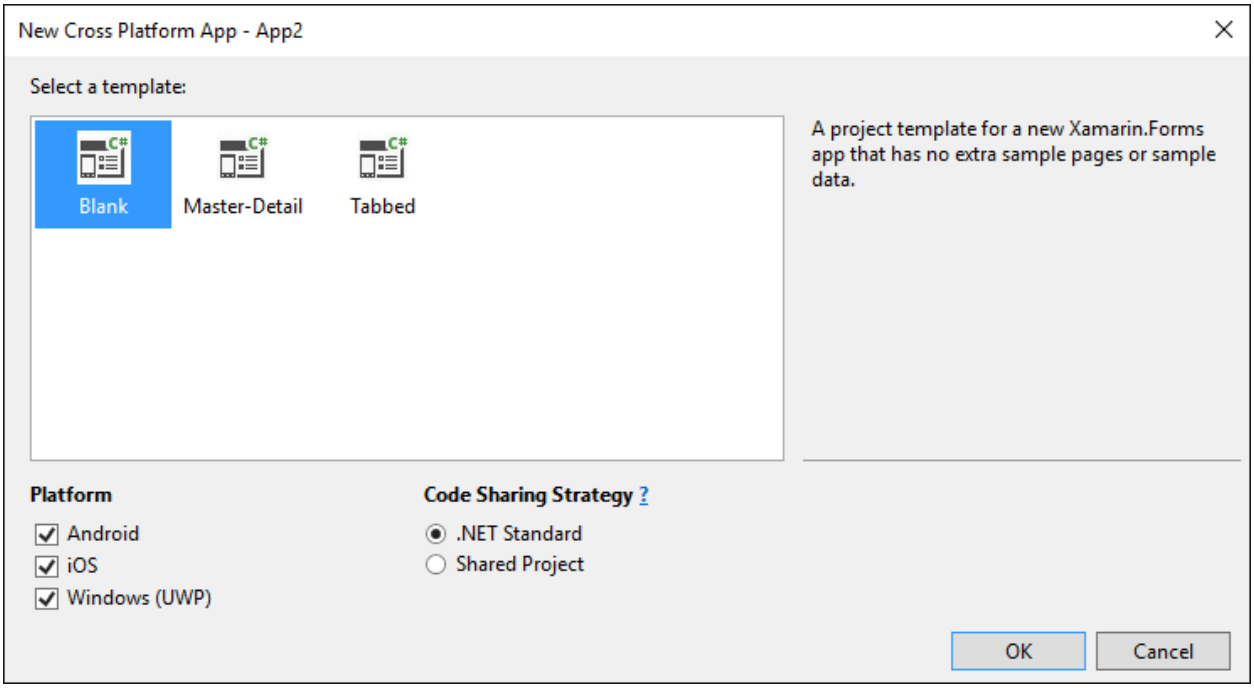


*Figure 4: Selecting the template, UI technology, and code sharing strategy for a new project*

In the **Platform** group, select all the platforms you want to target. The default is Android, iOS, and UWP. In the **Code Sharing Strategy** group, you can choose between Shared Project and .NET Standard. The code-sharing strategy is a very important topic in Xamarin.Forms, and Chapter 2 will provide a detailed explanation. For now, select the **.NET Standard** option and click **OK**.

After a few seconds, Visual Studio will ask you to specify the target version of Windows 10 for your new app. Leave the default selection unchanged and continue. It will also show a welcome dialog window where, in the second screen, you will have an option to specify the location of a Mac computer. Skip this step for now, as it will be discussed in the next section. In Solution Explorer, you will see that the solution is made up of four projects, as demonstrated in Figure 5.

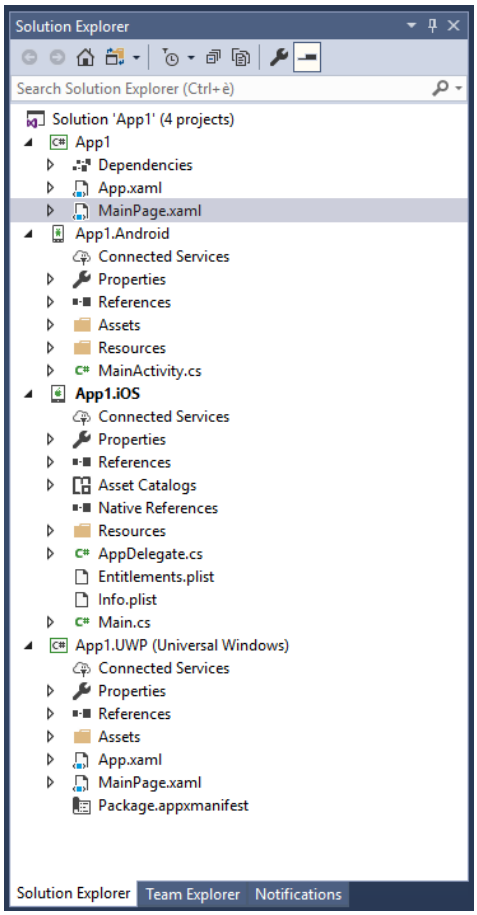


*Figure 5: The structure of an Xamarin.Forms solution*

The first project is either a .NET Standard library or a shared project, depending on your selection at project creation. This project contains all the code that can be shared across platforms, and its implementation will be discussed in the next chapter. For now, what you need to know is that this project is the place where you will write all the user interface of your app, and all the code that does not require interacting with native APIs.

The second project, whose suffix is **Android**, is a Xamarin.Android native project. It has a reference to the shared code and to Xamarin.Forms, and implements the required infrastructure for your app to run on Android devices.

The third project, whose suffix is **iOS**, is a Xamarin.iOS native project. This one also has a reference to the shared code and to Xamarin.Forms, and implements the required infrastructure for your app to run on the iPhone and iPad.

The fourth and last project is a native Universal Windows project (UWP) that has a reference to shared code and implements the infrastructure for your app to run on Windows 10 devices, for both the desktop and mobile devices. I will now provide more details on each platform project, so that you have a basic knowledge of their properties. This is very important, because you will need to fine-tune project properties every time you create a new Xamarin.Forms solution.