Xamarin Studio for Android Programming: A C# Cookbook

Multiplatform applications have taken the development world by storm. Xamarin Studio is emerging as the preferred choice among .NET/C# developers. It enables them to design cross-platform applications using their favorite language and IDE.

Starting with introducing Xamarin Studio, its underlying technologies, and the Android ecosystem, the book goes on to cover the graphical aspects of creating Android applications. Moving on, you will learn more about data management with Android services. This is followed by techniques for interacting with the Android OS and the phone’s hardware, before concluding with mobile advertisements and Google Play. By the end of this book, you will have discovered all the techniques related to developing an Android application with Xamarin Studio.

What this book will do for you...

- Build a GUI for your Android applications
- Explore Android activities and understand configuration changes
- Manage multiscreens, icons, and multimedia in your applications
- Start and bind Android services and create notifications
- Create beautiful applications using a camera and animations
- Effectively leverage your phone’s hardware with applications
- Integrate advertisements and select the right advertisement providers for your applications

Inside the Cookbook...

- A straightforward and easy-to-follow format
- A selection of the most important tasks and problems
- Carefully organized instructions to solve problems efficiently
- Clear explanations of what you did
- Solutions that can be applied to solve real-world problems

Prices do not include local sales tax or VAT where applicable
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'Getting Started'
- A synopsis of the book’s content
- More information on Xamarin Studio for Android Programming: A C# Cookbook
Mathieu Nayrolles was born in France and lived in a small village in Côte d'Azur for almost 15 years. He started his studying computer science in France and continued in Montréal, Canada, where he now lives with his wife. He holds a master's degree in software engineering from eXia.Cesi and another in computer science from UQAM. He is currently a PhD student in electrical and computer engineering at Concordia University, Montréal, Canada, under the supervision of Dr. Wahab Hamou-Lhadj.

As well as his academic journey, Mathieu has also worked for worldwide companies such as Eurocopter and Saint-Gobain where he learned how important good technical resources are.

You can discover some of his work through his books *Instant Magento Performances* and *Magento Performance Optimization: How to and Mastering Apache*. You can also check out his blog (https://math.co.de/) or latest realizations: bumper-app.com, mindup.io and toolwatch.io.

Follow him on Twitter at @MathieuNls for more information.
Preface

Xamarin is the leading company in cross-platform application development. This company was created by the same people who brought us Mono, MonoTouch, and Mono for Android, which were the very first cross-platform implementations of the Microsoft CLI (Common Language Structure) and CLS (Common Language Specification). Having a cross-platform CLI and CLS, which is often called .NET, allows us to develop a shared code base in C# to create a Windows Mobile, iOS, and Android application. As of last year, Xamarin has over half a million developers around the world. This success of Xamarin can be explained in many ways. First, they don't have much serious competition in the cross-platform mobile app development, they have a consequent developer base for Mono products and it works like a charm. Indeed, you do not need to have any knowledge about developing mobile applications to give it more than a try. Moreover, they provide a high-end IDE (integrated development environment), in which you can go from displaying a few words in a console to publishing a fully-fledged application in the applications store.

What this book covers

Chapter 1, Getting Started, talks about Xamarin Studio itself and the Android emulator needed to run our first Hello World application.

Chapter 2, Mastering the Life and Death of Android Apps, provides an in-depth and methodical approach to learning about activities’ lifecycles and how to manage the different states of an Android Activity.

Chapter 3, Building a GUI, helps us familiarize ourselves with the Android GUI and prepares us to tackle every situation with no less than 19 different components.

Chapter 4, Using Android Resources, helps us make things look better by covering how to use splash screen, manage multiscreen, play songs or videos, and display our application icon.

Chapter 5, Using On-Phone Data, takes a leap forward in order to persist and access user data between different launches of our applications. To do so, you will learn about user preferences, file writing/reading, SQLite, and LINQ.
Chapter 6, *Populating Your GUI with Data*, completes the loop between the on-phone data of the previous chapter and the GUI of *Chapter 4, Using Android Resources*, by covering how to populate GUI elements with data.

Chapter 7, *Using Android Services*, shows how to create, bind to, and identify running Android services in order to pursue our application processes even when the users are not looking at them.

Chapter 8, *Mastering Intents – A Walkthrough*, covers how to switch back and forth between applications with Android Intents to access a contact number or an address on the map, for example.

Chapter 9, *Playing with Advanced Graphics*, will push forward the creation of advanced graphics, such as 2D graphics, in our application and animation. We will also take advantage of the Android API made available by Xamarin to use the camera.

Chapter 10, *Taking Advantage of the Android Platform*, brings the last piece to our GUI building skills by mastering Android fragments that represent the behavior or a portion of the user interface in an Activity.

Chapter 11, *Using Hardware Interactions*, guides us to interact with our phone’s hardware, such as NFC, Bluetooth, Accelerometer, and GPS.

Chapter 12, *Debugging and Testing*, leverages the debugging and testing capacities of Xamarin Studio and the Android emulator.

Chapter 13, *Monetizing and Publishing Your Applications*, prepares us to build our final application and publish it to the Android Play Store.

Appendix, *Mono – The Underlying Technology*, exposes the key concepts about the technology that makes Xamarin possible.
Getting Started

In this first chapter, we will learn how to install Xamarin Studio and the Xamarin plugin for Microsoft Visual Studio and have a quick tour of both. Then we will move towards the creation of our first Hello World application and get it running on the emulator. We will cover the following topics:

- Installing the Xamarin Suite
- Building a HelloWorld App!

Introduction

Xamarin Studio is a cross-platform integrated development environment. It is works across platforms in two ways: the IDE is available on Mac OS and Windows (no Linux support announced), and it allows the development of software for Mac OS, iOS, and Android.

While its design can remind you of Xcode, the Apple IDE is only available to Mac owners, Xamarin Studio offers stunning features. Indeed, this IDE allows users to discover the new API easily through a powerful code completion enhanced by the ability to quickly learn about the method and required types. The following screenshot provides an overview of this code completion and the relevant documentation:
While being trained as a programmer, it seems that I have not developed any artistic capabilities and certainly won't in future. This is quite a problem, and you've certainly already faced an awkward situation like this: You: "Look at these amazing features!" Project Manager/Client: "Meh. It is despicable; the colors don't match with each other. Could you try this in light blue?".

Xamarin Studio, just like any modern IDE, allows the generation of user interfaces in a WYSIWYG (What You See Is What You Get) interface. In this way, it is a tool not only for programmers but also for designers. Using Xamarin Studio, you will build classy and beautiful apps, while Visual Studio and Netbeans barely provide functioning interfaces. The following screenshot shows the user interface menus and an example of such user interfaces:

![User Interface Menus and Example](image)

Every programmer experienced with modern IDEs such as Visual Studio, XCode, and Eclipse generally uses, on a daily basis, a step-by-step debugger. A debugger allows you to set breakpoints to stop code execution and then walk inside the code, instruction by instruction, while inspecting the values of variables. The step-by-step debugger included in Xamarin Studio is really simple to use yet efficient. The best part about it is that it allows debugging in the emulator (a virtual Android phone on your computer; we'll come back to this later) or on an Android device, live. An example implying the introspection of a variable counting the number of clicks on a button is shown in following screenshot:
Among a long list of other really useful and entertaining features, Xamarin Studio also excels in creating a highway to your customers’ mobile devices. Indeed, publishing apps to the Play Store (the Android market) has never been simpler. The packaging, deployment and shipment to the Play Store processes are smoothly integrated inside Xamarin Studio as shown by the following screenshot:

![Android App Publishing](image-url)
A quick tour of Visual Studio

Visual Studio is the Microsoft IDE used to develop in any of the various languages that comprise the Microsoft ecosystem. It can serve as an IDE for pretty much anything in the Microsoft bosom, such as console and graphical interfaces, Windows Presentation Foundation apps, websites, web applications, web services, Windows Mobile, and Windows CE. Honestly, Visual Studio is an excellent IDE that could prove very useful after practicing a little; thus, it comes with a price ranging from free to $13,299 depending on the version. Nevertheless, the Xamarin Company has built a Visual Studio plugin that could definitively fit your needs to develop an Android app if you own Visual Studio and are used to it. Xamarin also costs from $999 to $1899 per platform, but it does have a free version that offers most of the features covered in this book.

After talking about the financial aspect, let's focus on features. It turns out that all Android-focused features are exactly the same and have to be used in the same ways in almost the same menus. Choosing between Visual Studio versus Xamarin Studio will, in the end, be a personal choice that we leave to the reader's discretion. However, Visual Studio consumes a lot more resources than Xamarin Studio. Therefore, if you are not equipped with good hardware, you should definitely go for Xamarin Studio.

```csharp
using System;

namespace AndroidApplication1
{
    [Activity(Label = "AndroidApplication1", MainLauncher = true, Icon = "@drawable/icon")]
    public class Activity1 : Activity
    {
        protected override void OnCreate(Bundle bundle)
        {
            base.OnCreate(bundle);

            // Set our view from the "main" layout resource
            SetContentView(Resource.Layout.Main);

            // Get our button from the layout resource,
            // and attach an event to it
            Button button = FindViewById<Button>(Resource.Id.MyButton);
            button.Click += delegate { button.Text = string.Format("{0} click!", count++); };
        }
    }
}
```
In the rest of this book, screenshots and paths will be based on Xamarin Studio, which we believe is most used by newcomers to Android development (who use Xamarin and C#). We will indicate Visual Studio paths and options when they are different from those in Xamarin Studio.

## Installing the Xamarin suite

In order to develop Android applications using C#, we have to set up our Android development environment, and more specifically, we have to install Xamarin Suite. This first recipe will show you how.

### Getting ready

Xamarin Studio is available for the Mac and Windows operating systems and does not have a set of minimum hardware requirement to match. However, the technologies embedded in Xamarin Studio, such as the Android SDK, do have minimum requirements as follows:

- **Operating system:**
  - Windows XP (32-bit), Vista (32- or 64-bit), 7 (32- or 64-bit), 8 (32- or 64-bit)
  - Mac OS X 10.4.8 or later (x86 only—Intel chips)

- **Hardware requirements:**
  - 600 MB of available space and an additional 100MB per platform (iOS, Android, Windows)
  - This is not official, but I recommend an i3-equivalent processor and 4 GB of RAM to run Xamarin Studio and one emulator without any trouble.
  - Again, this is not official. In case of industrial needs, the debugging available on a real Android phone, while not mandatory, will accelerate your production. The phone cost is definitely worth it in terms of development time.

- **Software requirements:**
  - In case you intend to use the Visual Studio plugin inside Xamarin Studio Suite, you must own a license for Visual Studio 2010 or 2012 in a non-express edition.
How to do it...

In order to install Xamarin Studio, please follow the given steps:

1. Browse to http://xamarin.com/download, enter your information, and choose your operating system.

   ![Tell us about yourself](image)

   - **Tell us about yourself**
     - Mathieu Nayrolles
     - mathieu.nayrolles@gmail.com
     - Mathieu Nayrolles Inc.
     - Academic

   ![Includes the following](image)

   - **Includes the following**
     - Android
       - Develop native Android Apps
     - iOS (MonoTouch)
       - Develop native iOS apps using the entire iOS API
     - Mac
       - Develop apps with Xamarin.Mac

2. Execute the XamarinInstaller.exe or XamarinInstaller.dmg file that you subsequently receive and accept the terms and conditions by clicking on **Next**.

3. Choose what you want to do with your Xamarin tool suite: Android or iOS or both.

4. Accept the directory in which the Android SDK will be downloaded and installed.
5. For now, if you have checked everything, as we did, we should view the requirements of Xamarin Installer. The title is a bit misleading. In reality, this is not a list of requirements but of software that will be downloaded and installed during the Xamarin Suite installation.

![Requirements](image)

6. After a long period of downloading, the various software will install themselves on your computer.
7. The installation is completed. You can now open Xamarin Studio.

How it works...

Xamarin is based on the Mono Project (refer to the Appendix), which is an Open Source implementation of the CLR (Common Language Runtime), but that does not explain how Xamarin manages to create Android apps. Indeed, the Mono parts explain how we can execute C# based applications on many platforms but not how these apps run on our Android devices.
The Xamarin compiler, which is responsible for the transformation of C#.Net into Android-understandable code, is a very powerful tool. Indeed, it will compile and link all your C#.Net code—using proprietary technologies and processes—directly into an APK file. APK (Android Application Package) is the required format to deploy and install applications onto Android devices. The deployed APK on the targeted Android device will take advantage of the Just In Time (JIT) compilation. This compilation is a hybrid approach between the interpreted language, where an interpreter translates the language to the underlying machine each time we need it, and the compiled approach, where the whole code is compiled into a machine. The interpreted approach loads at the speed of light but comes with poor runtime performance, while the compiled approach loads very slowly but offers better performance. Hence, the JIT strategy consists of translating code continuously, just as an interpreter does, but it saves the translated code with a cache mechanism to avoid performance degradation. In other words, JIT offers the best of both worlds. Last but not least, it runs natively on Android devices.

If you intend to use the Visual Studio plugin, here are the steps to verify that it has been properly installed:

1. Run Visual Studio.
2. In the **File** menu search for **New Project** and select **Visual C#** in the **Other languages** list. Finally, you can select and create a new Android Application Project.

![New Project dialog in Visual Studio](image)

After clicking on the **OK** button, Visual Studio will open the Android Project perspective and a new Android solution.

**There's more...**

Xamarin comes in two different versions: Business and Enterprise. The prices are $999, and $1899 per platform (Android, iOS, and Mac OS) and per developer, respectively. It's quite an investment, especially if you are on your own—understood to be "not sponsored by a company that handles license-related fees"—so it's mandatory to reduce your needs and buy the adapted version.
The starter (free) version will allow you to build very small apps that contain no more than 32 KB of compiled code. In other words, you will have the taste of Xamarin, but your applications will stay very simple—forever. A less constraining limitation related to the free version is the impossibility of invoking native third-party libraries such as P/Invoke. We can definitely build a world-class app without third-party libraries; it just takes longer. The indie version ($299/platform/dev) only takes down the size limitation and will therefore meet the needs of the majority. The two latest and more expensive versions will mainly provide support from the Xamarin team. The Business version offers e-mail support, Visual Studio support, and in-house deployment, while the Enterprise version will add $500 worth of ready-to-use components for Xamarin Studio, a one-day SLA, hotfixes, a technical kick-off session, and code troubleshooting provided by Xamarin engineers.

While many possibilities are offered to you, the Starter Edition should be enough to follow this book. In case you want to determine whether or not a feature is worth its price, go for the trial edition.

**Testing the simulator**

For both systems (Xamarin Studio and Visual Studio), we should determine whether or not the simulator is well configured. If so, there is a good chance that our whole environment is as well.

In Xamarin Studio:

1. Go to the **Emulator** menu via the **Project** menu | **Android Device Target** | **Manage Devices**. The following window appears:
The different versions of MonoForAndroid—which could be referred to as MonoDroid informally—target the different versions of Android. The newer (higher) versions are made for the latest Android releases.

2. Select the higher version (API_21, which targets Android 4.0.3, at the time of writing—November 2015) and click on **Start Simulator**. After a while, especially on the first start, the simulator will show up. It can be used exactly as an Android Phone.
In Visual Studio:

1. Access the **Simulator** menu via the **Tool** menu and then start **Android Emulator Manager**. The following window appears:

   ![Android Virtual Device Manager](image)

2. Select the higher version and click on **Start**. As with Xamarin Studio, the simulator can be used as an Android Phone.

After pressing the **Start** button, the Android emulator will start. This operation may take a few moments depending on your hardware. Nevertheless, after a moment, the emulator will provide a user experience really close to that of a real Android device.
Connecting Xamarin Studio and Visual Studio to a versioning control system

A good habit you should always have is to work with a versioning control system, no matter the size of the project. The integration of git/svn into Xamarin Studio comes off without a hitch. You just have to browse the Version Control menu and then press the Checkout button.

Unfortunately, Visual Studio does not offer any embedded support for git or svn, we recommend giving libgit2 and visualsvn, respectively, a try.
Using another Android SDK

If you already are into Android development you might need, in a particular configuration such as maintaining an application on enterprise devices that are not up-to-date (for security reasons), to use a different Android SDK than the one installed with Xamarin Studio. To do so, browse the Tool menu in Xamarin Studio and then press the Option button. In the new window that comes up, search for Android under SDK Locations in the Projects section. The same menu exists in Visual Studio under Tools | Options | Xamarin | Android Settings.

Pay attention while modifying these parameters. Indeed, any misconfiguration will prevent your Xamarin Studio from running any Android applications.
See also

- See also the next recipe for the very first hands-on tutorial on Xamarin Studio.

Building a Hello World App!

It's now time to get our hands dirty and write some simple code by ourselves to output the famous Hello World message. In this recipe, we will create a new Android project in Xamarin Studio. This project will lead to an Android app that outputs Hello World! after pressing on a Say Hello! button. This first application will run in the simulator and on a real Android Device. Moreover, this application will allow us to discover the architecture of an Android project, the same as its specificities.

Getting ready

For this recipe, we will need to have successfully installed the Xamarin Studio tool suite and run the emulator at least once in order to confirm that everything is working well.

How to do it...

1. Run Xamarin Studio, browse the File menu, and then click on New Solution. In the new window that comes up, select Android Application under C# | Android.
2. Make sure that **Create directory for solution** is checked.

3. Name your project **Hello_World** and press **OK**.

   ![Note the project name only accepts the letters A-Z, the digits 0-9 and the hyphen (-), underscore (_), and period (.) characters. This is the reason for the underscore between "Hello" and "World".]

Now that you have a new project named **Hello_World**, it's appropriate to take some time to explain the files and folders comprising the architecture of an Android project. The project you have just created is already filled with a large number of files and folders.

- By order of appearance, the first folder is named **References** and contains four libraries Mono.Android, System, System.core, and System.XML. This folder will contain all libraries that you used in your project in the same way as a pure .Net project.
- The next folder, named **Components**, contains components that you previously developed for Xamarin or downloaded from the rich Xamarin database. Most of the components directly available from the Xamarin Component Store are free and ready-to-use components that will add commonly needed functionality to your project, ranging from interacting with Facebook to including Easter eggs in your applications.
- The two following folders are **Assets** and **Resources**, just like in native Android apps. They are pretty similar however, in the sense that they contain files that are not code, for example, images, songs, XML files, and pretty much anything your application will need. So why do we have two folders if they have the same purposes? In reality, the external files placed in the **Assets** folder will be easily accessible at runtime by using the Asset Manager (we'll come back to this later), while for the ones contained in the **Resources** folder, you will have to declare and maintain a list of resource IDs that you might use at runtime. In general use, we will put all images, sounds, icons, and other external files in the **Resources** folder, and the **Assets** folder will be privileged for dictionaries and XML storage.
- The next folder in line is named **Properties** and contains two files: **AndroidManifest.xml** and **AssemblyInfo.cs**. They are responsible for the Android version and permission your application targets, and your project information (such as the version and build number), respectively.
Finally, the `MainActivity.cs` file is the main class of our application; it contains our very first Android apps lines of code.

4. Open the `Main.axml` file in the `Layout` folder under `Resources`. A new file containing the graphical interface of your Hello World! app should appear.

   The `.axml` file is, in fact, an XML file that allows code completion in Xamarin Studio and Visual Studio for graphical element. It also defines the position and properties of the graphical element.
5. Double-click on the button on the graphical interface and change the text to **Say Hello!**

6. From the **ToolBox** pane, on the right of both Xamarin Studio and Visual Studio, locate **Text (Large)** in the **Widgets** section.

7. Drag and drop the **Text (Large)** widget below the **Say Hello** button.

8. Double-click on the **Text (Large)** widget you just inserted and delete the text.

9. With the **Text (Large)** widget selected, locate the **Properties** pane in the bottom-right corner of the IDE and change the **Id** tag style from `@+id/textView1` to `@+id/myTextView`.

The **Id** tag style is the unique identifier of the graphic element. You will use these IDs in the C# code in order to manipulate the graphical elements. Therefore, you must give them meaningful names.
10. Return to the MainActivity.cs file and locate Button button = 
    FindViewById<Button> (Resource.Id.myButton);. Add these lines of code 
    below it:
    
    // Get our TextView from the layout resource, and attach an 
    // event to it
    TextView view = FindViewById<TextView> 
    (Resource.Id.myTextView);

11. Locate the following lines of code:
    
    button.Click += delegate {
        button.Text = string.Format ("{0} clicks!", count++);
    };

    Modify them as follows:

    button.Click += delegate {
        view.Text = "Hello World !";
    };

12. Delete the code  int  count = 1; in the 14th line.

13. Browse the Project menu, then click on Manage Devices... under the Android 
    Device Target submenu.

14. Locate the MonoForAndroid_API_15 emulator, select it, and click on Start Emulator.

[  You will always need to have an emulator started prior to 
  launching your project as the project requires a device or 
  an emulator to execute on. ]

15. Run your Hello_World project by pressing the button in the top-left corner of 
    Xamarin Studio. After a moment, your application will be pushed on the emulator 
    and executed.
16. Here we go! Our very first Android application is now running on the Android emulator. You can press the **Say Hello!** button and **Hello World!** will appear.

How it works...

The How it works section of our Hello World recipe could appear rudimentary if you already have some graphical user interface development experience with C#. However, mastering the basics is never superfluous.

For this code example, and all the following one throughout this book, we will display the whole piece of code, add some references to it, and then explain the code using the references.

```csharp
using System;
using Android.App;
using Android.Content;
using Android.Runtime;
using Android.Views;  
using Android.Widget;
using Android.OS;
namespace Hello_World {                           
  [Activity (Label = "Hello_World", MainLauncher = true)]
  public class MainActivity : Activity {
    protected override void OnCreate (Bundle bundle) {
      base.OnCreate (bundle);
      // Set our view from the "main" layout resource
      // Get our button from the layout resource
      // and attach an event to it
      Button button = findViewById(R.id.button);  
      button.Click += delegate {
        view.Text = "Hello world!";";
    }
  }
}
```

---

21
Here is an explanation of Hello World MainActivity.cs:

- **[1]**: Most of the C# applications and in general .Net applications files start by using directives. Using directives, enlist all the namespaces that your application will use on a frequent basis, saving you time. For example, the TextView class is inside the namespace Android.Widget, and without the corresponding directive, you will have to type Android.Widget.TextView each time you want to create such objects.

- **[2]**: C# also enables the creation of programmer-defined namespaces. Each class in that namespace will be visible without any need to specify using Hello_World. Moreover, you could refer to your Hello_World namespace in other namespaces and have access to your classes.

- **[3]**: Our layout is defined by the Main.axml file in the Resources/Layout folder. We therefore need to bind this file to the screen. To do so, we call the setContentView() method.

- **[4-5]**: In the fourth and fifth instruction sets, we create a Button and a TextView instance by using the FindViewByIdId constructor. This generic class enables the creation of graphical elements by using their IDs. These IDs are user defined at the creation of these elements.

- **[6]**: delegate is a method signature type, and it's the keystone of event programming in C#. Here we are facing an inline delegate instance, which introduces an anonymous method in reality. This method (view.Text = "Hello World !") will be executed on the button.Click event by means of the += which add this new behavior at the end of the existing ones.
There's more...

Let's have a look at the following sections in order to learn about testing on a physical device and checkout this book's free source code.

**Testing on a physical device**

Testing your applications on physical devices will save you a lot of time, as the simulators are rather slow. You can see the complete procedure to do so, depending on your system, at [http://developer.xamarin.com/guides/android/getting_started/installation/set_up_device_for_development/](http://developer.xamarin.com/guides/android/getting_started/installation/set_up_device_for_development/).

**Source code**

This book comes with a lot of code examples that are freely available from Github. The code for this first chapter can be found here:

[https://github.com/MathieuN1ls/mastering-xamarin-studio/](https://github.com/MathieuN1ls/mastering-xamarin-studio/)

For the following chapters, adapt the following URL:

[https://github.com/MathieuN1ls/mastering-xamarin-studio/tree/master/chap1](https://github.com/MathieuN1ls/mastering-xamarin-studio/tree/master/chap1)

In order to access all the codes given in the book, replace the end of the URL from chap1 to chap2 for Chapter 2 and so on.

**See also**

- The next chapter introduces Android Activities Lifecycle. It will show you how your application lives and dies.
Where to buy this book

You can buy Xamarin Studio for Android Programming: A C# Cookbook from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.