# Introduction to Mobile Development

## Understanding Xamarin mobile application projects

# Overview

Building mobile applications can be as easy as opening up your IDE, throwing something together, doing a quick bit of testing, and submitting to an App Store – all done in an afternoon. Or it can be an extremely involved process that involves rigorous up-front design, usability testing, QA testing on thousands of devices, a full beta lifecycle, and then deployment a number of different ways.

In this document, we're going to take a thorough introductory examination of building a Xamarin mobile application, including:

1. **Requirements** – We'll enumerate and examine the requirements for building for iOS and Android applications.
2. **Introduction to Xamarin** – List of features of the Xamarin platform.
3. **How Does Xamarin Work?** – A brief overview of how Xamarin works to bring C# to iOS and Android.
4. **Sample App** – Discusses the Xamarin Store sample application's architecture and how it works. By downloading and working with the code you can receive a free Xamarin C# t-shirt!

This document is intended to introduce the Xamarin platform and show it in action with a working mobile application. To learn more about the process of building mobile applications from design through to testing, refer to the Introduction to the Mobile Software Development Lifecycle document.

# Requirements

If you'd like to develop for iOS, whether you want to code in Xamarin Studio or Visual Studio, you must have an Apple Macintosh computer running at least OS X Mountain Lion on hand. Although Xamarin applications are based on the .NET BCL and are written in C#, Xamarin requires the iOS SDK and Xcode in order to compile. Additionally, the iOS Device Simulator is part of the iOS SDK, and therefore only available on Mac. In order to download the iOS SDK, you must be a member of Apple's Developer Program. This program requires an upgraded membership that costs $99 USD/year.

All the tutorials presented here are based on the latest version. Installation is covered in the next tutorial.

# Introduction to Xamarin

When considering how to build iOS and Android applications, many people think that the indigenous languages, Objective-C and Java, respectively, are the only choice. However, over the past few years, an entire new ecosystem of platforms for building mobile applications has emerged.

Xamarin is unique in this space by offering a single language – C#, class library, and runtime that works across all three mobile platforms of iOS, Android, and Windows Phone (Windows Phone's indigenous language is already C#), while still compiling native (non-interpreted) applications that are performant enough even for demanding games.

Each of these platforms has a different feature set and each varies in its ability to write native applications – that is, applications that compile down to native code and that interop fluently with the underlying Java subsystem. For example, some platforms only allow you to build apps in HTML and JavaScript, whereas some are very low-level and only allow C/C++ code. Some platforms don't even utilize the native control toolkit.

Xamarin is unique in that it combines all of the power of the indigenous platforms and adds a number of powerful features of its own, including:

1. **Complete Binding for the Indigenous SDKs** – Xamarin contains bindings for nearly

the entire underlying platform SDKs in both iOS and Android. Additionally, these bindings are strongly-typed, which means that they're easy to navigate and use, and provide robust compile-time type checking and during development. This leads to fewer runtime errors and higher quality applications.

2. **Objective-C, Java, C, and C++ Interop** – Xamarin provides facilities for directly invoking Objective-C, Java, C, and C++ libraries, giving you the power to use a wide array of 3rd party code that has already been created. This lets you take advantage of existing iOS and Android libraries written in Objective-C, Java or C/C++. Additionally, Xamarin offers binding projects that allow you to easily bind native Objective-C and Java libraries using a declarative syntax.
3. **Modern Language Constructs** – Xamarin applications are written in C#, a modern language that includes significant improvements over Objective-C and Java such as *Dynamic Language Features*, *Functional Constructs* such as *Lambdas*, *LINQ*, *Parallel Programming* features, sophisticated *Generics*, and more.
4. **Amazing Base Class Library (BCL)** – Xamarin applications use the .NET BCL, a massive collection of classes that have comprehensive and streamlined features such as powerful XML, Database, Serialization, IO, String, and Networking support, just to name a few. Additionally, existing C# code can be compiled for use in your applications, which provides access to thousands upon thousands of libraries that will let you do things that aren't already covered in the BCL.
5. **Modern Integrated Development Environment (IDE)** – Xamarin uses Xamarin Studio on Mac OS X, and also Xamarin Studio or Visual Studio 2013 on Windows. These are both modern IDE's that include features such as code auto completion, a sophisticated Project and Solution management system, a comprehensive project template library, integrated source control, and many others.
6. **Mobile Cross Platform Support** – Xamarin offers sophisticated cross-platform support for the three major mobile platforms of iOS, Android, and Windows Phone. Applications can be written to share up to 90% of their code, and our Xamarin.Mobile library offers a unified API to access common resources across all three platforms. This can significantly reduce both development costs and time to market for mobile developers that target the three most popular mobile platforms.

Because of Xamarin's powerful and comprehensive feature set, it fills a void for application

developers that want to use a modern language and platform to develop cross-platform mobile applications.

**Note:**

This Getting Started series focuses on getting started building iOS and Android applications. If you're interested in building for Windows Phone, Microsoft offers tutorials here. If you're interested in learning more about cross-platform development with Xamarin (including Windows Phone), you can find our guide here.

Let's take a look at how this all works.

# How Does Xamarin Work?

Xamarin offers two commercial products: Xamarin.iOS and Xamarin.Android. They're both built on top of *Mono*, an open-source version of the .NET Framework based on the published .NET ECMA standards. Mono has been around almost as long as the .NET framework itself, and runs on nearly every imaginable platform including Linux, Unix, FreeBSD, and Mac OS X.

On iOS, Xamarin's *Ahead-of-Time* (*AOT*) Compiler compiles Xamarin.iOS applications directly to native ARM assembly code. On Android, Xamarin's compiler compiles down to *Intermediate Language* (*IL*), which is then *Just-in-Time* (*JIT*) compiled to native assembly when the application launches.

In both cases, Xamarin applications utilize a runtime that automatically handles things such as memory allocation, garbage collection, underlying platform interop, etc.

## MonoTouch.dll and Mono.Android.dll

Xamarin applications are built against a subset of the .NET BCL known as the Xamarin Mobile Profile. This profile has been created specifically for mobile applications and packaged in the MonoTouch.dll and Mono.Android.dll (for iOS and Android respectively). This is much like the way Silverlight (and Moonlight) applications are built against the

Silverlight/Moonlight .NET Profile. In fact, the Xamarin Mobile profile is equivalent to the Silverlight 4.0 profile with a bunch of BCL classes added back in.

For a full list of available assemblies and classes, see the Xamarin.iOS Assembly List and the Xamarin.Android Assembly List

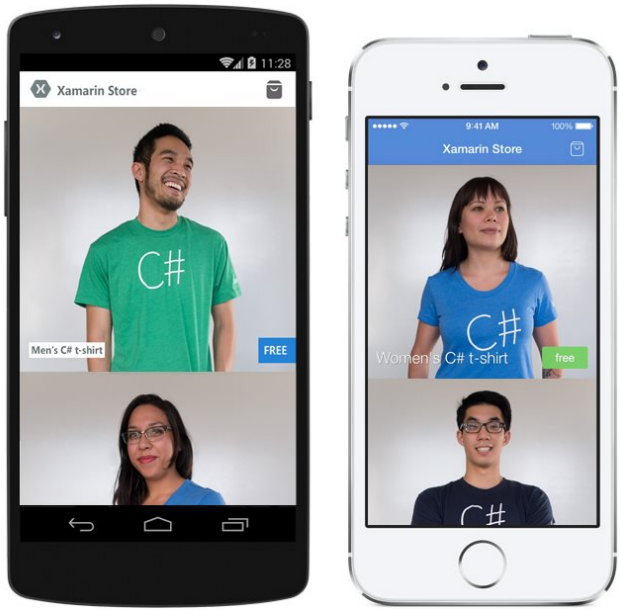
In addition to the BCL, these .dlls include wrappers for nearly the entire iOS SDK and Android SDK that allow you to invoke the underlying SDK APIs directly from C#.

## Application Output

When Xamarin applications are compiled, the result is an Application Package, either an .app file in iOS, or .apk file in Android. These files are indistinguishable from indigenous application packages and are deployable in the exact same way.

# Sample App: Xamarin Store

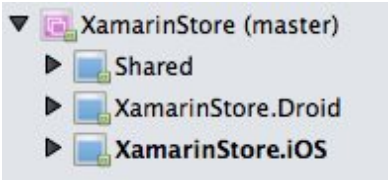
Download the code from xamarin.com/sharp-shirt or github.com/xamarin/xamarin-store-app.




## Application Structure

A Xamarin cross-platform solution typically includes:

- Shared code project
- iOS application project
- Android application project




The shared project is referenced by the application projects, allowing all non-platform-specific code to be re-used. If you are using Xamarin with Visual Studio you can also include Windows Phone and Windows Store applications in your solution, and share code with them as well.

## Shared Code

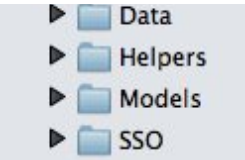
Xamarin apps are written entirely in C# and have full access to the .NET framework on each platform. This means you can write code that uses any of the features of .NET, including file operations, web access, database storage, XML manipulation and more, then share that code across iOS, Android and Windows Phone.

Xamarin supports two mechanisms for sharing code:

- Portable Class Libraries
- Shared Application Projects

The pros and cons of each method are discussed in detail in the Sharing Code Options article.

This example uses a **Shared Application Project** which is compatible with the latest version of Xamarin Studio and Visual Studio 2013 Update 2. This screenshot of the Shared project shows some the folders of code included in this project and re-used in both iOS and Android applications:




Functionality that our sample application shares includes:

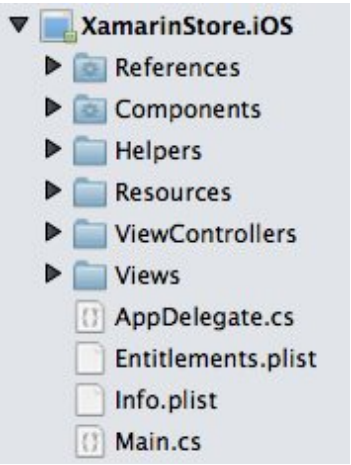
- Web services access for login and ordering
- Gravatar image download and caching
- Product information display
- Order validation logic

These classes consist of non-user-interface functionality that takes advantage of the .NET framework that Xamarin makes available on iOS and Android, as well as being able to run on Windows and Windows Phone.

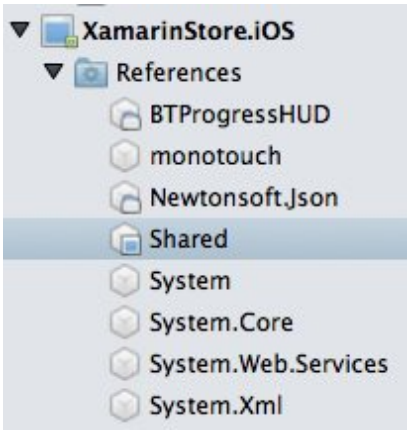
The goal of the platform-specific projects is to build a native user interface that takes advantage of the shared code to provide the application functionality in a fast and familiar way.

# iOS Application

The iOS application project is shown below. This project contains all the iOS-specific code to produce the native user interface for the app. You'll find the screens for the app defined in the **ViewControllers** folder and a number of custom controls in the **Views** folder - these are the bulk of the application code.

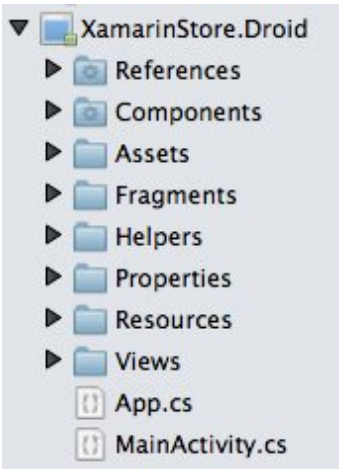


The references list shows a number of BCL assemblies, the monotouch assembly that provides the bindings to the native iOS SDK, a couple of components, and a reference to the Shared project.



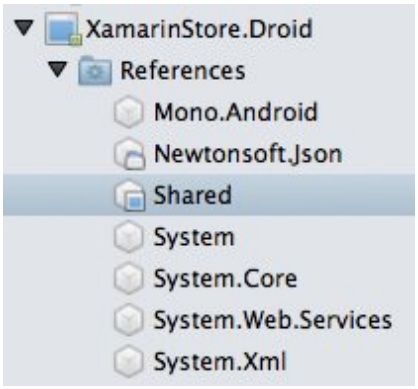

## Android Application

The Android application project is shown below. This project contains all the Android-specific code to produce the native user interface for the app. You'll find the screens for the app defined in the **Resources/layout** folder and a number of custom controls in the **Views** and **Fragments** folders - these are the bulk of the application code.




The references list shows a number of BCL assemblies, the Mono.Android assembly that provides the bindings to the native Android SDK, a couple of components, and a reference

to the Shared project.




# Build and Run

You can run the sample on either the iOS or Android simulator, or an iOS or Android device.

## Xamarin Studio

To start the app on iOS, select the desired simulator or device from the list and press the Play button.

To use Android, press the Play button and choose an emulator from the dialog (or select a plugged in device).

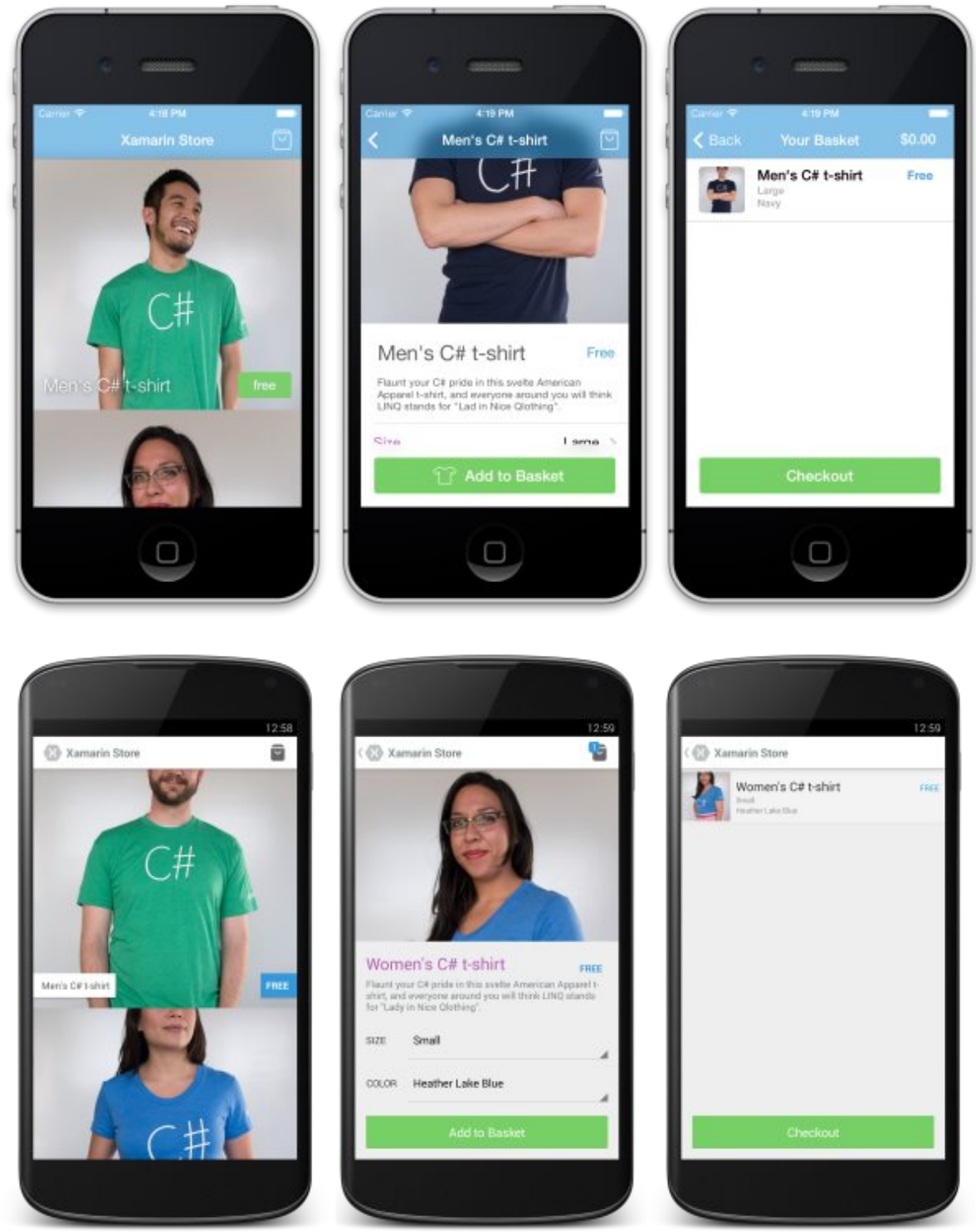
## Visual Studio

To start the app on iOS, select the desired simulator from the list and press the |> play button.

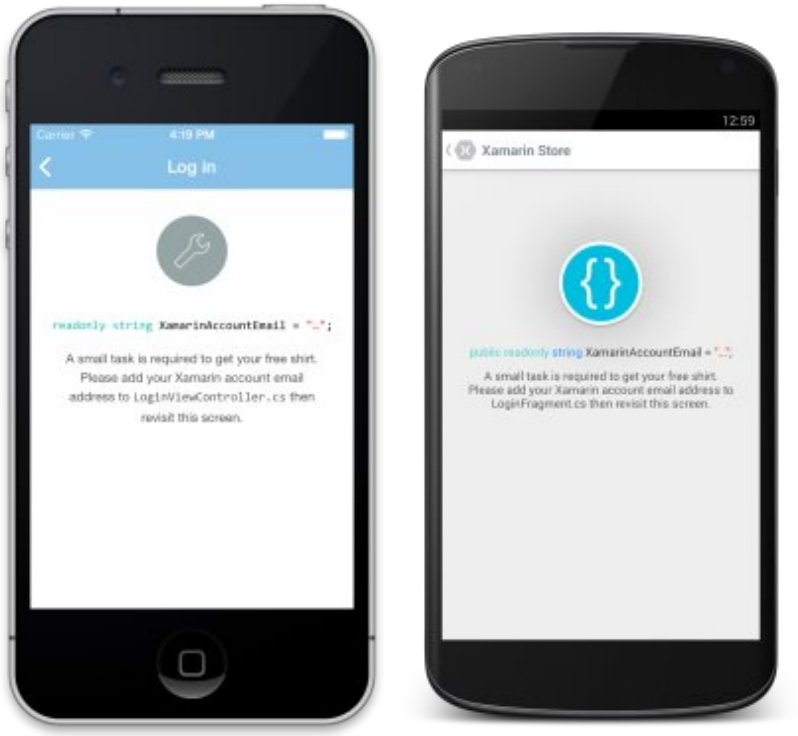
To use Android, press the **Start** button and choose an emulator from the dialog (or select a plugged in device).

# Writing your first mobile C# code

When you first run the app you'll be able to follow the ordering process for your free t-shirt (shown on both iOS and Android):




However when you attempt to Checkout, the follow message will appear:




This is where you need to do some work!

First stop running the app so you can edit the code. Then follow the instructions to edit the `LoginViewController` (on iOS) or `LoginFragment` (Android) – enter the email address you used to download Xamarin.

```
// TODO: Enter your Xamarin account email address here
// If you do not have a Xamarin Account please sign up here:
https://store.xamarin.com/account/register
readonly string XamarinAccountEmail = "";
```

Only basic C# knowledge is required to find and make this change. Once you've edited the file (and perhaps scanned through the application code for any other minor changes you might like to try), re-start the app. This time you'll be able to complete the check-out process, enter your delivery information and order a free Xamarin C# shirt!

# Summary

This document demonstrates how to quickly and easily build and run your first cross-platform mobile application with Xamarin. By downloading our t-shirt sample app you can learn how Xamarin solutions are structured, edit and run C# code on a mobile simulator or device – and get a free t-shirt in the process.

It serves as a primer and gave context for the next steps of the Getting Started Series. From here, the next step is read either the Hello, iOS, or Hello, Android guides, depending on which platform you'd like to begin developing for first.