

Mobile Development

using Xamarin Forms

Ahmed El-Araby | Feb-2019

**Introduction of mobile development**

Mobile development has witnessed incredible development in the decade, from introducing new frameworks to enhance an existing one. Thus, it is very important to choose the right technology/framework when starting a new project. in my opinion, all these methods and frameworks can be categorized into four categories, Native development, hybrid, flutter, and cross-platform.

Mobile Development Options

**Hybrid**

(JavaScript and HTML5)

**Flutter**

(Dart)

**Native**

(Swift & Java)

**Cross-Platform**

(Xamarin, React Native, Titanium)

**Native:**

Developing Mobile application using the platform native library, programing language, and compilers. For example, to develop iOS application you’ll need a MacOS with Xcode installed and writing Swift/Objective-C code. However, for Android you need to use Android Studio with Java programming language.

**Hybrid:**

Developing a hybrid application that can be executed on iOS, Android, Windows desktop, Linux … etc. or any other operating system with web viewer.

There are many ways and tools to develop hybrid application but all of them will be converted to JavaScript and HTML at the end. Moreover, most of the tools are using Crodova to convert the compiled code to mobile application, which can be uploaded to App Store or Google Play.

Hybrid apps load in a browser-like component called webview, they are only as good as the webview. Webview is responsible for displaying the UI and for running Javascript code.

Some of this tools and frameworks are Ionic, Angular, Vue and React.

**Flutter:**

It is a framework from Google help you to develop mobile application for iOS and Android in Dart programming language. Which can give the developer the ability to control each and every pixel on the screen.

The compiler will compile your project to machine-code directly, which provide high performance as well.

**Cross-Platform:**

This method help developer to write an application using their preferred programming language and then the compiler will convert it to native library, which can be compiled via native compiler to machine code.

This method will make the application looks and feels much like native applications. And also will have the native limitations.

**How Does Xamarin Work?**

Xamarin offers two commercial products: Xamarin.iOS and Xamarin.Android. They are 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.

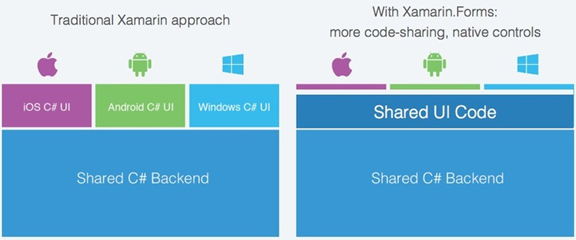
In both cases, Xamarin applications utilize a runtime that automatically handles things such as memory allocation, garbage collection, underlying platform interop, etc.

**Why using Xamarin?**

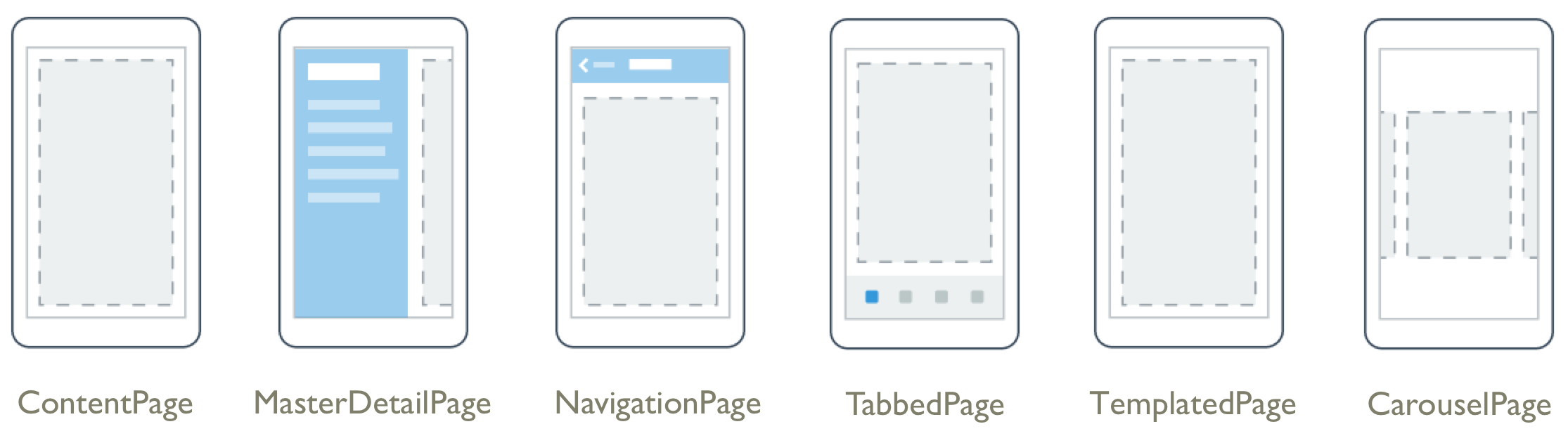
1. Using platform-specific compilers.
2. Direct access to native APIs. These are the C# object libraries that give developers access to iOS SDK and the Android SDK, respectively.
3. Native performance.
4. Develop using the same programming language you have experience with.
5. Developing using preferred IDE.
6. No need to learn iOS or Android platforms and SDK to develop native application for both operating systems. For example, you can use Xamarin Agent on Mac to deploy and test.

**Xamarin Forms:**

Xamarin.Forms is a complete array of instruments for the creation of cross-platform software. It works on top of the Xamarin.iOS and Xamarin.Android libraries and allows writing front ends in C# and XAML and may even share this code between implementations. Applications created with Xamarin.Forms are fully endowed with all the characteristics inherent to the best cross-platform solutions. In particular, they allow developers to save time and money (about 80-95% of the code is cross-platform, the remaining 5-20% contain commands to establish connections with native APIs).



**Xamarin Forms Pages:**



**ContentPage:**

ContentPage is the simplest and most common type of page. Set the Content property to a single View object, which is most often a Layout such as StackLayout, Grid, or ScrollView.

**MasterDetailPage:**

A MasterDetailPage manages two panes of information. Set the Master property to a page generally showing a list or menu. Set the Detail property to a page showing a selected item from the master page. The IsPresented property governs whether the master or detail page is visible.

**NavigationPage:**

The NavigationPage manages navigation among other pages using a stack-based architecture. When using page navigation in your application, an instance of the home page should be passed to the constructor of a NavigationPage object.

**TabbedPage**

TabbedPage derives from the abstract MultiPage class and allows navigation among child pages using tabs. Set the Children property to a collection of pages, or set the ItemsSource property to a collection of data objects and the ItemTemplate property to a DataTemplate describing how each object is to be visually represented.

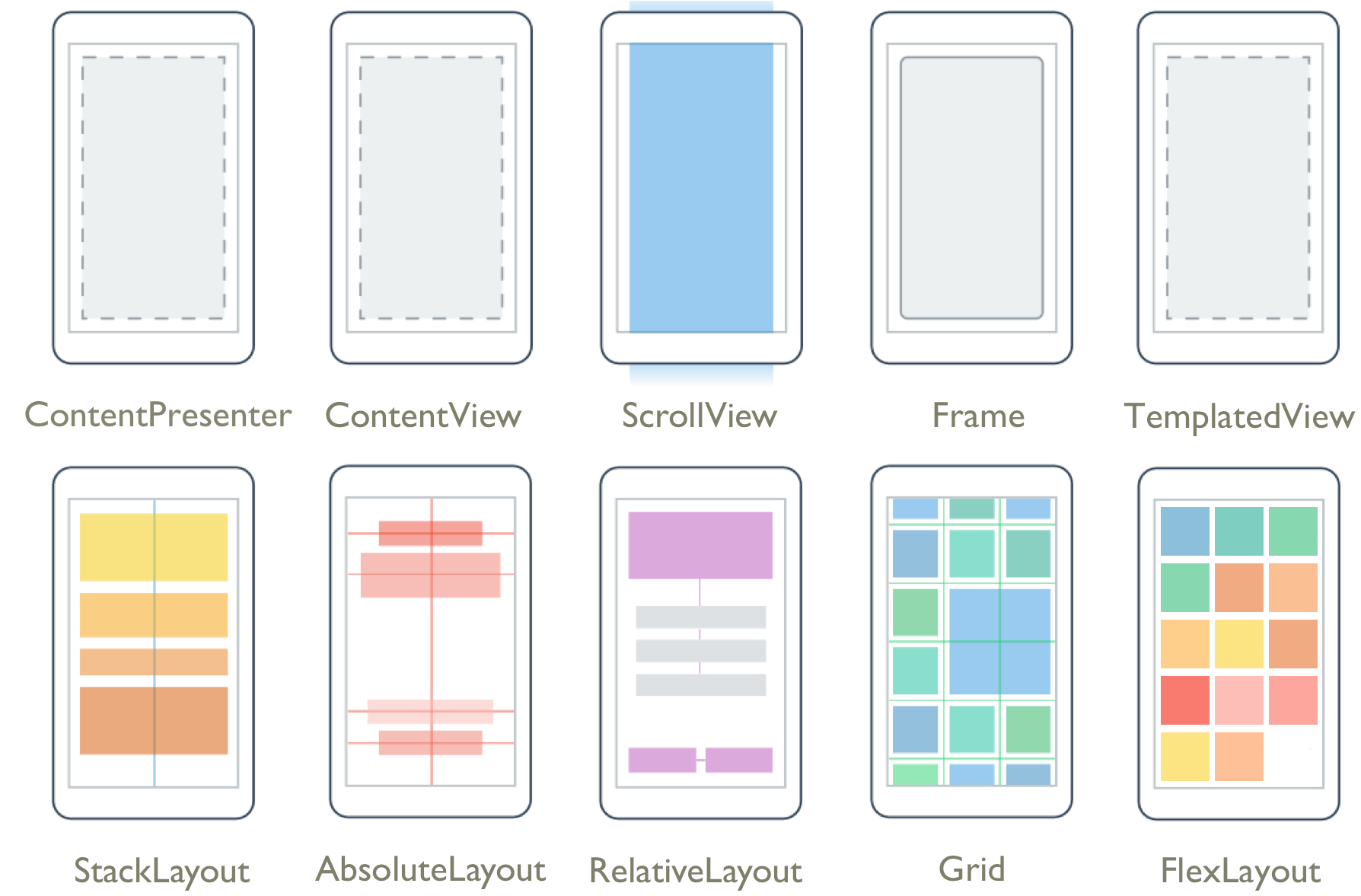
**CarouselPage**

CarouselPage derives from the abstract MultiPage class and allows navigation among child pages through finger swiping. Set the Children property to a collection of ContentPage objects, or set the ItemsSource property to a collection of data objects and the ItemTemplate property to a DataTemplate describing how each object is to be visually represented.

**TemplatedPage**

TemplatedPage displays full-screen content with a control template, using ControlTemplate.

**Xamarin Forms Layouts:**



The classes that derive from Layout can be divided into two categories:

**Layouts with Single Content:**

**ContentView:**

ContentView contains a single child that is set with the Content property. The Content property can be set to any View derivative, including other Layout derivatives. ContentView is mostly used as a structural element and serves as a base class to Frame.

**Frame:**

The Frame class derives from *ContentView* and displays a rectangular frame around its child. Frame has a default Padding value of 20, and also defines *OutlineColor*, *CornerRadius*, and *HasShadow* properties.

**ScrollView:**

*ScrollView* is capable of scrolling its contents. Set the *Content* property to a view or layout too large to fit on the screen. (The content of a *ScrollView* is very often a *StackLayout*.) Set the *Orientation* property to indicate if scrolling should be *vertical*, *horizontal*, or *both*.

**ContentPresenter:**

ContentPresenter is a layout manager for templated views, used within a ControlTemplate to mark where the content that is to be presented appears.

**Layouts with Multiple Children:**

**StackLayout:**

*StackLayout* positions child elements in a stack either horizontally or vertically based on the *Orientation* property. The *Spacing* property governs the spacing between the children, and has a default value of 6.

**Grid:**

Grid positions its child elements in a grid of rows and columns. A child's position is indicated using the attached properties *Row*, *Column*, *RowSpan*, and *ColumnSpan*.

Rows and Columns

Row and column information is stored in Grid's RowDefinitions & ColumnDefinitions properties, which are each collections of [RowDefinition](https://docs.microsoft.com/en-us/dotnet/api/xamarin.forms.rowdefinition) and [ColumnDefinition](https://docs.microsoft.com/en-us/dotnet/api/xamarin.forms.columndefinition) objects, respectively. RowDefinition has a single property, Height, and ColumnDefinition has a single property, Width. The options for height and width are as follows:

* **Auto** – automatically sizes to fit content in the row or column. Specified as [GridUnitType.Auto](https://docs.microsoft.com/en-us/dotnet/api/xamarin.forms.gridunittype) in C# or as Auto in XAML.
* **Proportional(\*)** – sizes columns and rows as a proportion of the remaining space. Specified as a value and GridUnitType.Star in C# and as #\* in XAML, with # being your desired value. Specifying one row/column with \* will cause it to fill the available space.
* **Absolute** – sizes columns and rows with specific, fixed height and width values. Specified as a value and GridUnitType.Absolute in C# and as # in XAML, with # being your desired value.

**AbsoluteLayout:**

AbsoluteLayout positions child elements at specific locations relative to its parent. A child's position is indicated using the attached properties *LayoutBounds* and *LayoutFlags*. An AbsoluteLayout is useful for animating the positions of views.

**RelativeLayout:**

RelativeLayout positions child elements relative to the RelativeLayout itself or to their siblings. A child's position is indicated using the attached properties that are set to objects of type Constraint and BoundsConstraint.

Exasmple:

<BoxView Color="Green" WidthRequest="50" HeightRequest="50"

RelativeLayout.XConstraint =

"{ConstraintExpression Type=RelativeToParent,

Property=Width,

Factor=0.5,

Constant=-100}"

RelativeLayout.YConstraint =

"{ConstraintExpression Type=RelativeToParent,

Property=Height,

Factor=0.5,

Constant=-100}" />

**FlexLayout:**

FlexLayout is based on the CSS Flexible Box Layout Module, commonly known as flex layout or flex-box. FlexLayout defines six bindable properties and five attached bindable properties that allow children to be stacked or wrapped with many alignment and orientation options.

**References:**

<https://applikeysolutions.com/blog/xamarin-forms-vs-xamarin-native-what-fits-you-best>

<https://www.mobiloud.com/blog/native-web-or-hybrid-apps/>

<https://flutter.io/>

<https://docs.microsoft.com/en-us/xamarin/cross-platform/get-started/introduction-to-mobile-development>

<https://www.appcelerator.com/titanium/titanium-sdk/4/>