Xamarin

Xamarin.Forms

A cross-platform mobile UI toolkit

"WRITE ONCE, RUN EVERYWHERE AND STILL BE NATIVE"

Written By- .NET COE Team, PreludeSys Inc.

preludesys

IMPLICIT KNOWLEDGE

Executive Summary

Mobile application development is one of the noticeable and hot trends in software industry in recent years. Every other web-application (specially banking sites, ticket-booking sites, social-networking sites and many popular commercial business sites) either already have a Mobile App or seeking a one.

To cater to the mobile consumer segment and create a reliable mobile app, one need to target the most popular and commonly used mobile platforms like iOS, Android and Windows. Mobile apps development on multiple platforms and devices, has its own challenges like varying size and resolution, different UI components, getting developers/resources with individual platform skill-sets, managing multiple code repositories, handling change management for each platforms and so on which can be a nightmare over a period of time.

As a developer one should spend time to provide a reliable, secured and endurable mobile app, instead of worrying to make it working on different mobile platformsThink about the ease of life, if a developer (specifically a C# .NET developer) has to write a code only once and that too in Microsoft's most popular IDE 'Visual Studio' which can be run on all major platforms. You might be thinking it's just a dream. Well, it was once a dream for most developers till Xamarin introduced Xamarin FormsWith the most awaited integration with Visual Studio, Xamarin has lot more to offer for cross platform mobile development world.

Table of Contents

S.No	Title	Page No.
1	Cross Platform Mobile Development	4
2	Why and When to Prefer Xamarin	6
3	What about Xamarin Mobile Platform	7
4	Introduction to Xamarin Forms	8
5	Xamarin/Xamarin.Forms Architecture	11
6	Deep Dive into Xamarin Forms	13
7	Development Environments	24
9	Conclusion	29

1 Cross Platform Mobile Development

Challenges and Opportunities

In recent years, market for Enterprise-ready cross platform for mobile development has gained significant traction. International Data Corporation (IDC) forecasts the market for such development platforms will see a compound annual growth rate of over 38% reaching $4.8 billion by 2017 and Gartner expects over 20 million enterprise apps to be developed by 2018. Many software companies are trying to deliver mobile apps on as many platforms as possible in order to become competitive and responsive in the market. However, they face tough times when it comes to be on budget and on time, matching with the speed of their competitors.

Since mobile market is highly dictated by iOS, Android and Windows devices, if one wants to develop a particular app, he needs to develop it for these three devices/platforms unless he can afford to ignore one these platforms. The standard approach that companies follow for developing high performance reliable mobile apps, is native development. Speed and cost-efficiency gets compromised , as it requires hiring of native platform developers, creating multiple code bases and designing individual platform UIs.

In these circumstances, it make more sense to follow the new approach of 'Write once and run anywhere' that most companies and developers prefer for cross-platform mobile development. This can reduce the development cost below the threshold of the total sum of native app development costs associated with each platform. It also provides better maintainability with easier app upgradation for each platform. There are still some trade-offs between traditional native development and modern cross platform mobile apps for developing mobile app that require native sensor readings, the wave is towards the cross platform world .

Let us discuss more on currently available cross platform frameworks and tools in the market and see how they tackle the challenge of cross platform mobile development.

Available Cross Platform Tools in the Market

Popular frameworks for cross-platform mobile development that come to one's mind are- Xamarin, PhoneGap and Titanium. There are few more like Appcelerator, iFactr etc. however, they are less popular compared to the other three. All these frameworks fulfill the purpose of developing a single app for multiple platforms. However, there are vast technical, business and philosophical differences among them. None of these are a wrong choice, however, one might be better than the other depending on particular requirements. For example if you are a C# developer you will always prefer Xamarin over PhoneGap and Titanium.

Let's have a closer look at top three cross platform tools (most commonly used by mobile developers) to understand the major differences among them as well as to get an idea about when to prefer which framework.

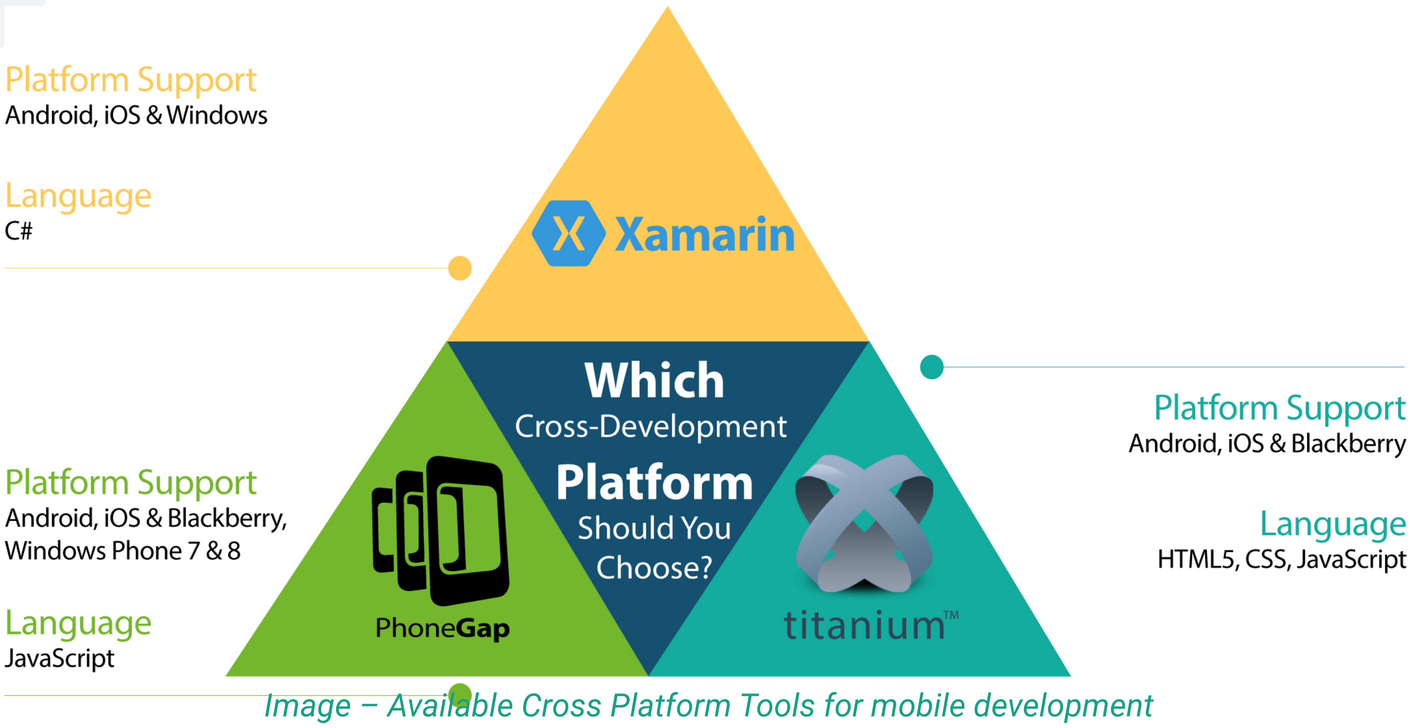


Image – Available Cross Platform Tools for mobile development

Cross Platform Features	PhoneGap	Titanium	Xamarin
Platform Support	iOS , Android , WP7/8, Blackberry	Android , iOS, Blackberry	iOS, Android & Windows
Language	HTML5 , CSS, JavaScript	JavaScript	C#
Open Source	Yes	Yes	No
UI	Web UI	Native	Native
Access to Device API	Limited	Full	Full
Web Standard Support	Yes	No	No
DOM Support	Yes	No	Yes
Native Performance	No	Yes	Yes
Used By	IBM, Sony, Mozilla, Intel	Cisco, VMware, Safeguard Properties, Mitsubishi Electric	GitHub, Microsoft, Foursquare, Expensify, Dow Jones

Table – Difference among top three cross platform mobile development tools

2 Why and When to Prefer Xamarin

→ Xamarin, originally called MonoTouch is a cross-platform framework with its own IDE. It works on C# within .NET framework and allows you to create native apps by utilizing native APIs and UIs of each platform.

→ Xamarin comes with Xamarin Forms library which allows you to write native UIs for once and then share and convert them to platform-specific UIs. Xamarin currently supports iOS, Android and Windows platform. It also allows developing apps for Blackberry by compiling Android apps.

→ Xamarin Forms apps compile natively so the apps run like native apps. Processing click and touch events are fast as well, without noticeable delays.

→ Xamarin Forms focuses on sharing backend logic and UI logic. The Extensible Application Markup Language XAML is converted into native controls so the UI is native. This can feel natural when using the app.

→ Xamarin Forms apps can take advantage of many of the third party libraries written for the .NET stack including retrieving components through Nuget and the Xamarin Component Store.

→ The Xamarin development style feels natural to .NET developers, particularly if they are used to the MVVM pattern.

→ Xamarin can potentially reuse much of the server side business logic if needed for offline capabilities.

→ C# is a strongly typed language allowing bugs to be caught at compile time instead of run time, helping to minimize the number of defects introduced into the application.

→ Xamarin Forms has strong support for advanced development tooling and process control such as unit testing, mocking, dependency injection and inversion of control.

→ Backend code can be written in C# against the .NET framework allowing code to be shared across platforms and from server to client.

→ When required, developers can use native UI controls and renderers if the functionality for a given platform does not feel right given the out of the box functionality.

→ Xamarin do not support sharing of codes outside Xamarin environment for native or HTML5 development can be seen as its limitations.

3 What about Xamarin Mobile Platform

C# Language

Preferred language with sophisticated features like Generics, LINQ and the Parallel Task Library

Mono .NET Framework

A cross-platform implementation of the extensive features in .NET framework.

Compiler

Depending on the platform, produces a native app like iOS or an integrated .NET application and runtime like Android. The compiler also performs many optimizations for mobile deployment such as linking away un-used code.

IDE Tools

The Xamarin Studio IDE and the Xamarin plug-in for Visual Studio allow you to create, build and deploy Xamarin projects.

Deployment

The underlying language is C# with the .NET framework, projects can also be deployed to Windows Phone.

4 Introduction to Xamarin Forms

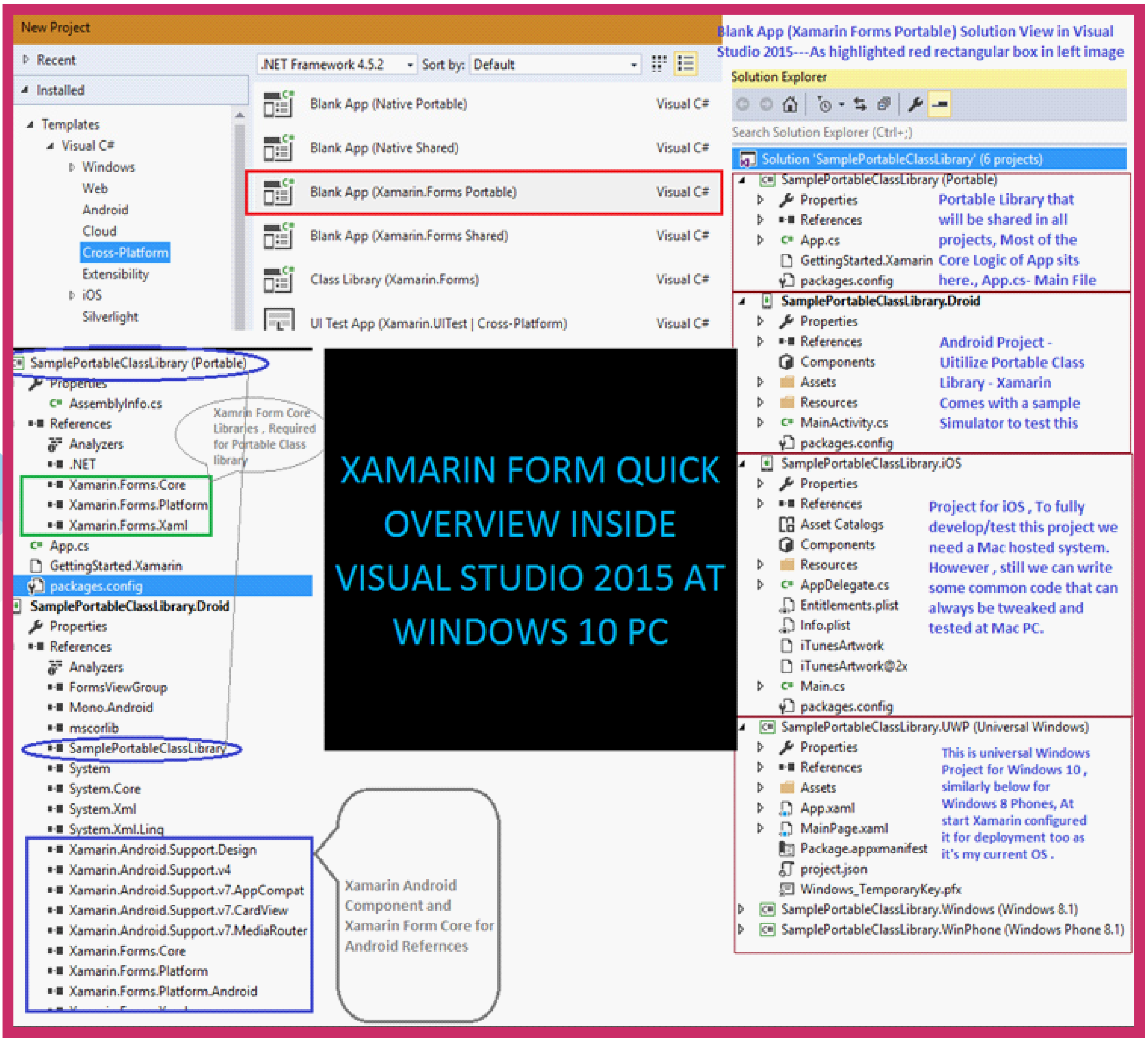
Overview

Xamarin Forms is a cross-platform natively backed User Interface (UI) toolkit which allows developers to easily create user interface that can be shared across various platforms (Android, iOS and Windows Phone). This means applications developed using Xamarin Forms will retain the appropriate look and feel for each platform without sacrificing on the application performance. The user interfaces are rendered using the native controls of the target platform, allowing Xamarin Forms applications to retain the appropriate look and feel for each platform.

“Write once, Run everywhere and still be Native”

Types of Solutions/App

Xamarin Forms provides Portable Class Libraries or Shared Projects to house the shared code, and then create platform specific applications that will consume the shared code.



New Project

Recent
Installed
Templates
Visual C#
Windows
Web
Android
Cloud
Cross-Platform
Extensibility
iOS
Silverlight

.NET Framework 4.5.2 · Sort by: Default

Blank App (Native Portable)	Visual C#
Blank App (Native Shared)	Visual C#
Blank App (Xamarin.Forms Portable)	Visual C#
Blank App (Xamarin.Forms Shared)	Visual C#
Class Library (Xamarin.Forms)	Visual C#
UI Test App (Xamarin.UITest | Cross-Platform)	Visual C#

SamplePortableClassLibrary (Portable)
Properties
AssemblyInfo.cs
References
Analyzers
.NET
Xamarin.Forms.Core
Xamarin.Forms.Platform
Xamarin.Forms.Xaml
App.cs
GettingStarted.Xamarin
packages.config
SamplePortableClassLibrary.Droid
Properties
References
Analyzers
FormsViewGroup
Mono.Android
mscorlib
SamplePortableClassLibrary
System
System.Core
System.Xml
System.Xml.Linq
Xamarin.Android.Support.Design
Xamarin.Android.Support.v4
Xamarin.Android.Support.v7.AppCompat
Xamarin.Android.Support.v7.CardView
Xamarin.Android.Support.v7.MediaRouter
Xamarin.Forms.Core
Xamarin.Forms.Platform
Xamarin.Forms.Platform.Android

Xamrin Form Core Libraries , Required for Portable Class library

Xamarin Android Component and Xamarin Form Core for Android Refernces

XAMARIN FORM QUICK OVERVIEW INSIDE VISUAL STUDIO 2015 AT WINDOWS 10 PC

Blank App (Xamarin Forms Portable) Solution View in Visual Studio 2015---As highlighted red rectangular box in left image

Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'SamplePortableClassLibrary' (6 projects)

SamplePortableClassLibrary (Portable)
Properties
References
App.cs
GettingStarted.Xamarin
packages.config

Portable Library that will be shared in all projects, Most of the Core Logic of App sits here., App.cs- Main File

SamplePortableClassLibrary.Droid
Properties
References
Components
Assets
Resources
MainActivity.cs
packages.config

Android Project - Uitilize Portable Class Library - Xamarin Comes with a sample Simulator to test this

SamplePortableClassLibrary.iOS
Properties
References
Asset Catalogs
Components
Resources
AppDelegate.cs
Entitlements.plist
Info.plist
iTunesArtwork
iTunesArtwork@2x
Main.cs
packages.config

Project for iOS , To fully develop/test this project we need a Mac hosted system. However , still we can write some common code that can always be tweaked and tested at Mac PC.

SamplePortableClassLibrary.UWP (Universal Windows)
Properties
References
Assets
App.xaml
MainPage.xaml
Package.appxmanifest
project.json
Windows_TemporaryKey.pfx

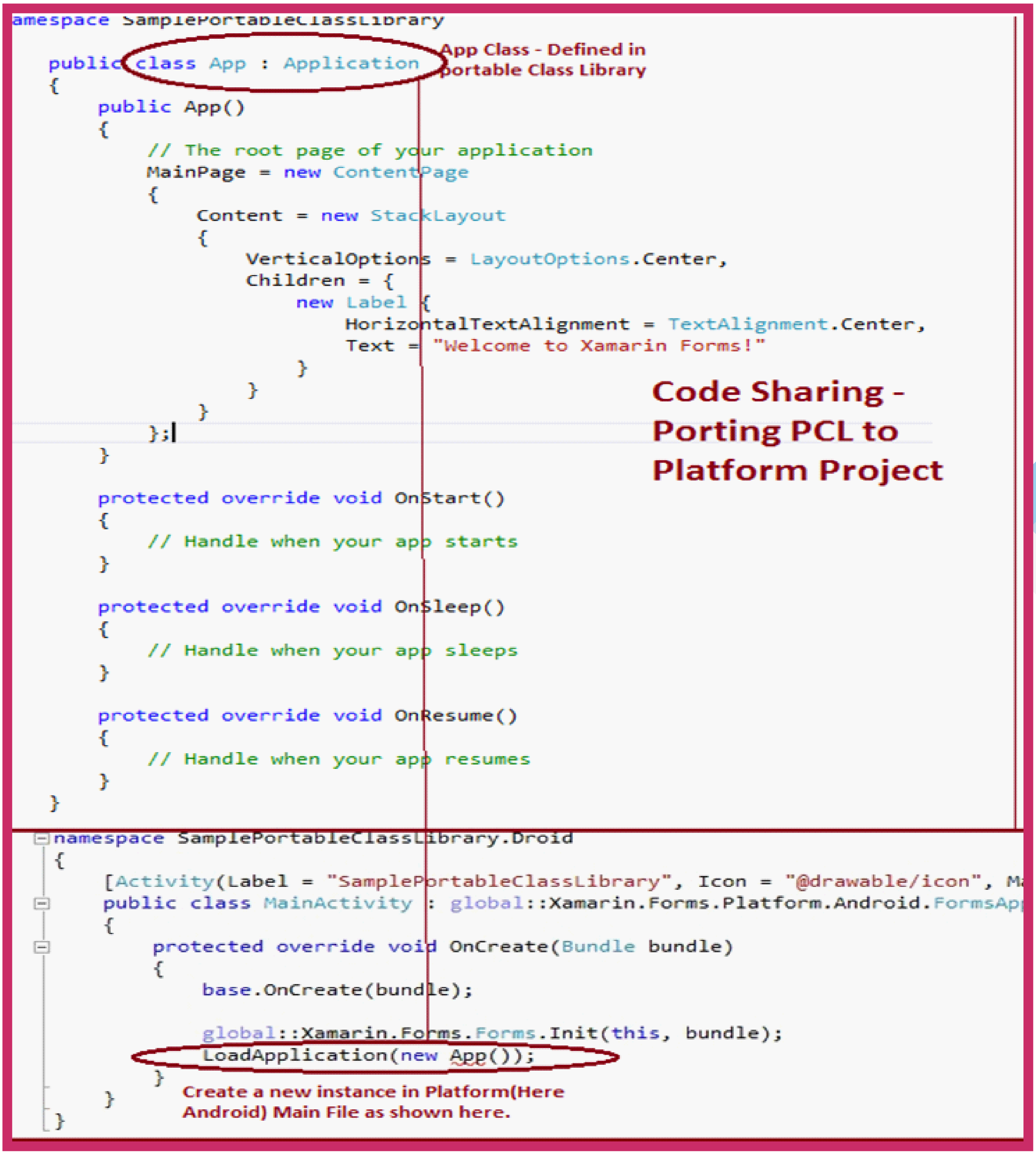
This is universal Windows Project for Windows 10 , similarly below for Windows 8 Phones, At start Xamarin configured it for deployment too as it's my current OS .

SamplePortableClassLibrary.Windows (Windows 8.1)
SamplePortableClassLibrary.WinPhone (Windows Phone 8.1)

Image – Xamarin Forms Cross Platform Application Overview inside Visual Studio 2015

What is so special? Remember Code Sharing- Write Once Approach?

To maximize the reuse of the startup code, Xamarin.Forms applications have a single class named App that is responsible for instantiating the first Page that will be displayed.



```
amespace SamplePortableClassLibrary

    public class App : Application
    {
        public App()
        {
            // The root page of your application
            MainPage = new ContentPage
            {
                Content = new StackLayout
                {
                    VerticalOptions = LayoutOptions.Center,
                    Children = {
                        new Label {
                            HorizontalTextAlignment = TextAlignment.Center,
                            Text = "Welcome to Xamarin Forms!"
                        }
                    }
                }
            };
        }

        protected override void OnStart()
        {
            // Handle when your app starts
        }

        protected override void OnSleep()
        {
            // Handle when your app sleeps
        }

        protected override void OnResume()
        {
            // Handle when your app resumes
        }
    }
```

```
namespace SamplePortableClassLibrary.Droid
{
    [Activity(Label = "SamplePortableClassLibrary", Icon = "@drawable/icon", Ma
    public class MainActivity : global::Xamarin.Forms.Platform.Android.FormsAp
    {
        protected override void OnCreate(Bundle bundle)
        {
            base.OnCreate(bundle);

            global::Xamarin.Forms.Forms.Init(this, bundle);
            LoadApplication(new App());
        }
    }
}
```

App Class - Defined in portable Class Library
Code Sharing - Porting PCL to Platform Project
Create a new instance in Platform(Here Android) Main File as shown here.

Image – How Xamarin Forms shares PCL; 'The App Class'

Where is Cross Platform?

The core building block of building cross-platform apps is to create architecture that assist in maximization of code sharing across platforms.

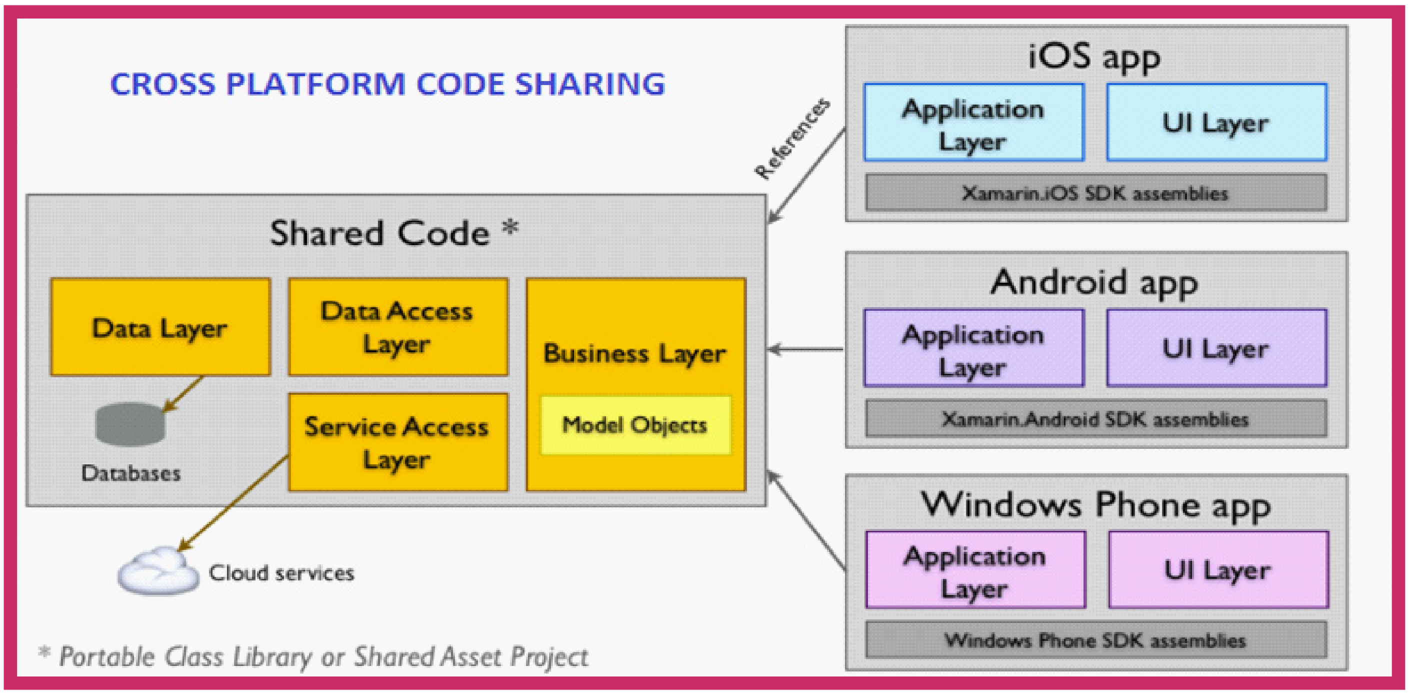


Image – Difference among top three cross platform mobile development tools

5 Xamarin/Xamarin.Forms Architecture

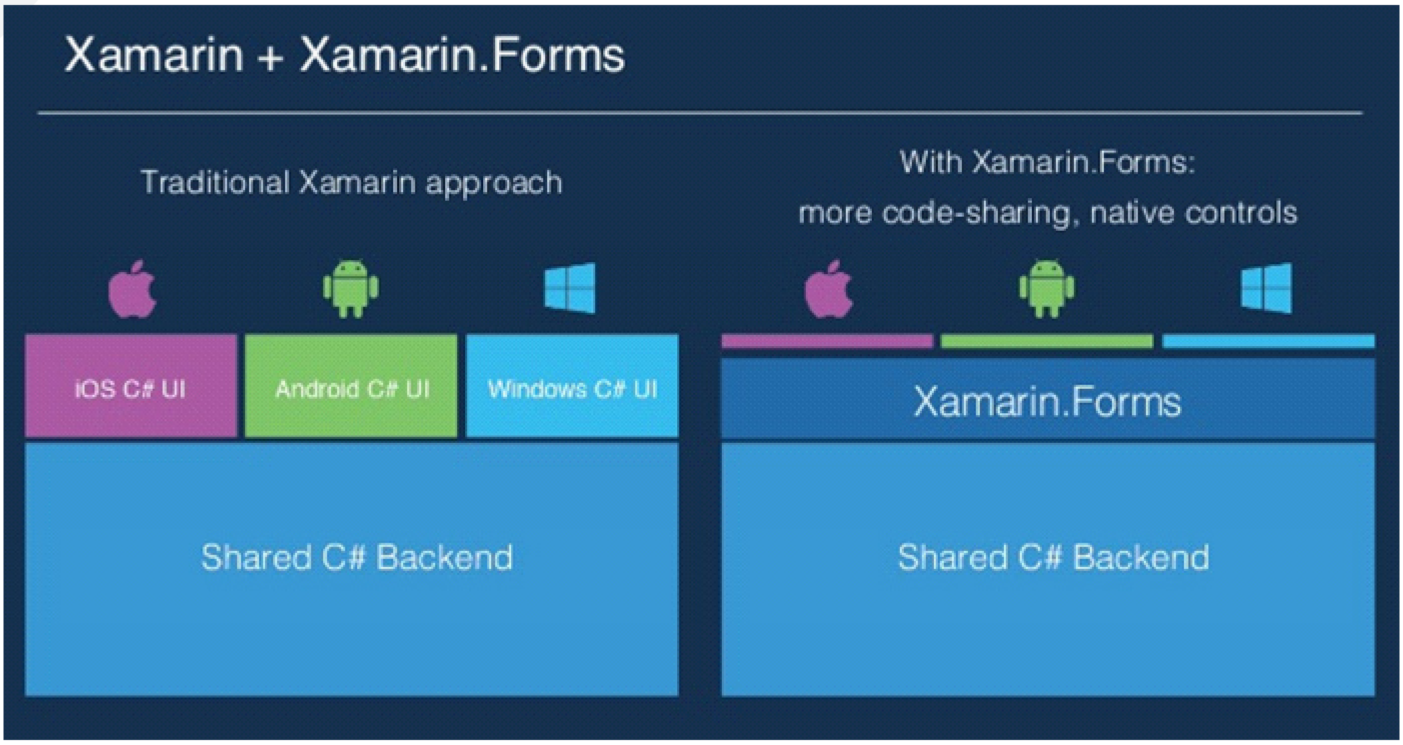


Image – Xamarin Forms Architecture

Cross Platform Architecture

Adhering to the following Object Oriented Programming principles helps in building a well-architected cross platform mobile application.

- **Encapsulation –** Ensuring that classes and even architectural layers only expose a minimal API that performs their required functions, and hides the implementation details. At an architectural level, it means implementing patterns like Façade that encourage a simplified API that orchestrates more complex interactions on behalf of the code in more abstract layers. This means that the UI code should only be responsible for displaying screens and accepting user-input; and never interact with the database directly. Similarly the data-access code should only read and write to the database, but never interact directly with buttons or labels.

- **Separation of Responsibilities –** Ensure that each component has a clear and well-defined purpose. Each component should perform only its defined tasks and expose that functionality via an API that is accessible to the other classes that need to use it.

- **Polymorphism –** Programming to an interface that supports multiple implementations means that core code can be written and shared across platforms, while still interacting with platform-specific features.

Application Layers

Typical architectural layers used in Xamarin Forms Mobile development is quiet similar to traditional cross platform architecture approach.

- **Data Layer –** Non-volatile data persistence

- **Data Access Layer –** Wrapper around the Data Layer that provides Create, Read, Update, Delete (CRUD) access to the data without exposing implementation details to the caller

- **Business Layer –** Business entity definitions (the Model) and business logic.

- **Service Access Layer –** Used to access services in the cloud: from complex web services (REST, JSON, WCF) to simple retrieval of data and images from remote servers

- **Application Layer –** Code that's typically platform specific (not generally shared across platforms) or code that is specific to the application (not generally reusable).

- **User Interface (UI) Layer –** The user-facing layer, contains screens, widgets and the controllers that manage them.

Key Patterns

Few key design patterns that are useful to understand in building/maintaining/understanding mobile applications are- Model, View, Controller (MVC), Business Façade, Singleton, Provider and Async Pattern.

6 Deep Dive into Xamarin Forms MVVM and Bindable Infrastructure

Xamarin Forms provide a MVVM (Model View ViewModel) based UI infrastructure by utilizing the power of XAML markup and C# Designer (Code-behind) files.

Also it is the most recommended pattern when we talk about creating cross platform mobile Apps. It supports 'Separation of Concerns' by decoupling the UI (View) from the Business Logic (Model/View Model). XAML defines the View that links to View Model through XAML-based data bindings. With this approach it also provides a way to utilize UI- designer tools like Microsoft Blend for generating a quick markup and Code-behind.

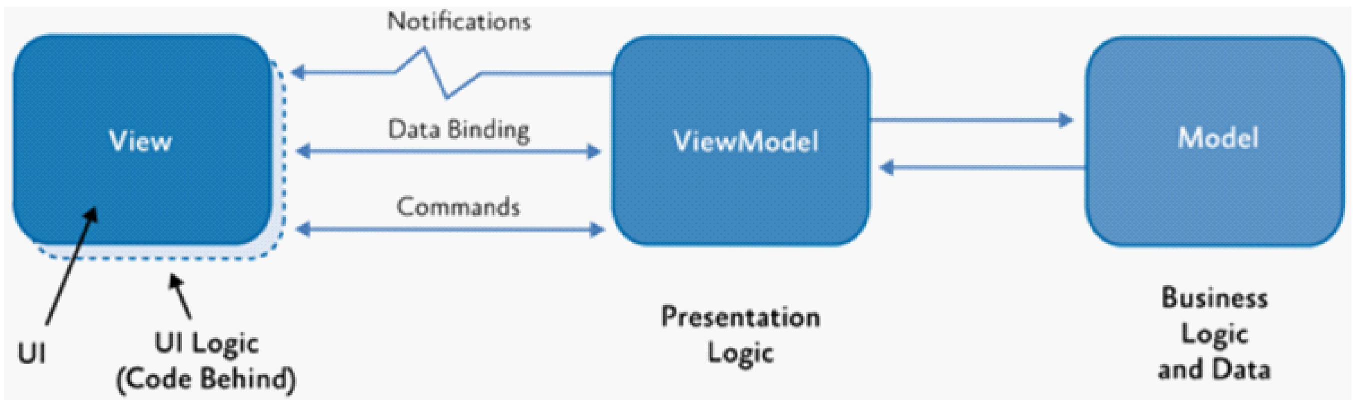
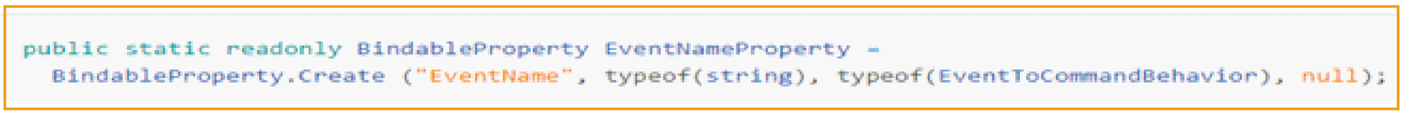


Image – MVVM Infrastructure in Xamarin Forms

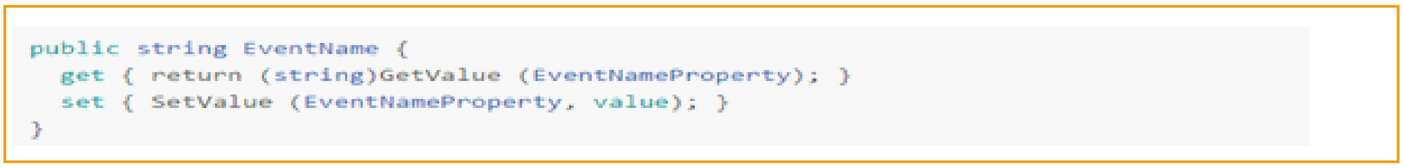
Xamarin Forms extends the commonly used .NET CLR (common language runtime) properties with an additional bindable properties (property, where the property's value is tracked by the Xamarin Forms property system) infrastructure.

Creating a Bindable Property

The bindable property identifier must match the property name specified in the Create method, with "Property" suffix as specified in below code snippet-



```
public static readonly BindableProperty EventNameProperty =
  BindableProperty.Create ("EventName", typeof(string), typeof(EventToCommandBehavior), null);
```

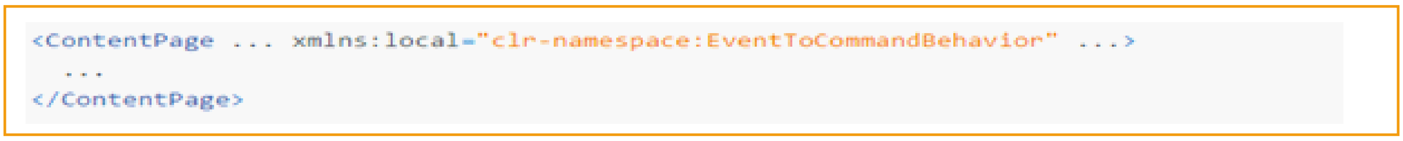


```
public string EventName {
  get { return (string)GetValue (EventNameProperty); }
  set { SetValue (EventNameProperty, value); }
}
```

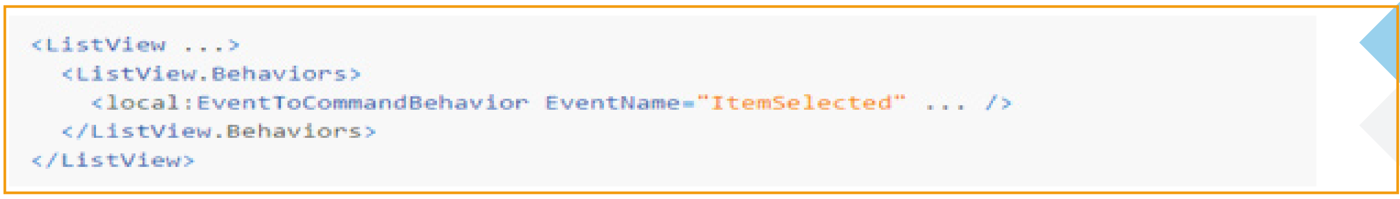
Image – Code Snippet for creating a bindable property

Consuming a Bindable Property

To consume a bindable property, one need to declare a XAML namespace indicating the Common Language Runtime (CLR) namespace name, and optionally, an assembly name as specified in below code snippet-



```
<ContentPage ... xmlns:local="clr-namespace:EventToCommandBehavior" ...>
  ...
</ContentPage>
```

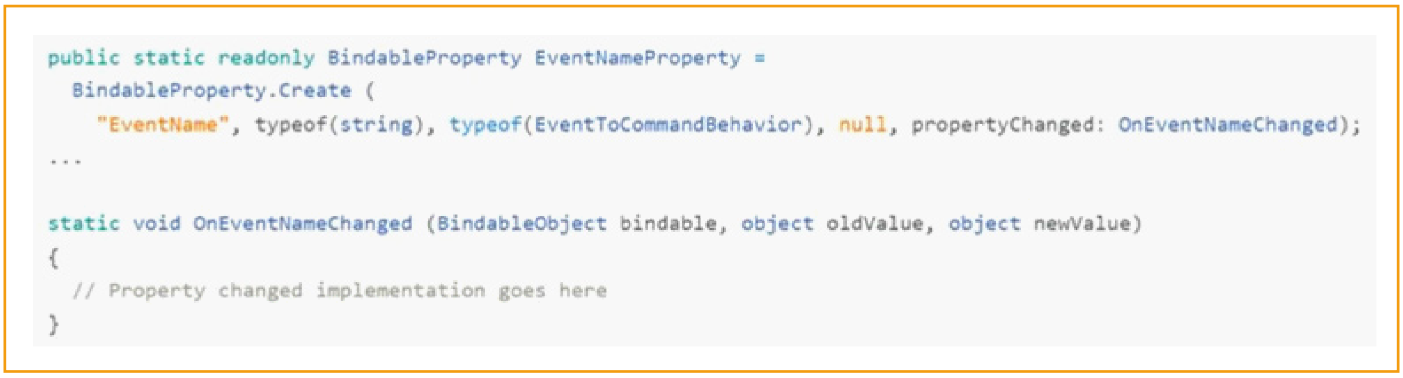


```
<ListView ...>
  <ListView.Behaviors>
    <local:EventToCommandBehavior EventName="ItemSelected" ... />
  </ListView.Behaviors>
</ListView>
```

Image – Code Snippet for consuming the bindable property inside Xaml Page

Advanced Scenarios

Xamarin Forms bindable infrastructure also supports advanced property scenarios like detecting Property Changes, validation, coerce Value parameters etc. All these can be achieved by passing callback methods in Create method that will be triggered on specific scenario as specified in below snippet-



```
public static readonly BindableProperty EventNameProperty =
  BindableProperty.Create (
    "EventName", typeof(string), typeof(EventToCommandBehavior), null, propertyChanged: OnEventNameChanged);
...

static void OnEventNameChanged (BindableObject bindable, object oldValue, object newValue)
{
  // Property changed implementation goes here
}
```

Image – Code Snippet for passing a callback method in the bindable property

Moreover it enables us to utilize Resource Dictionaries for UI element naming and other constant strings. Resource Dictionaries are also useful while sharing the global styles for UI controls.

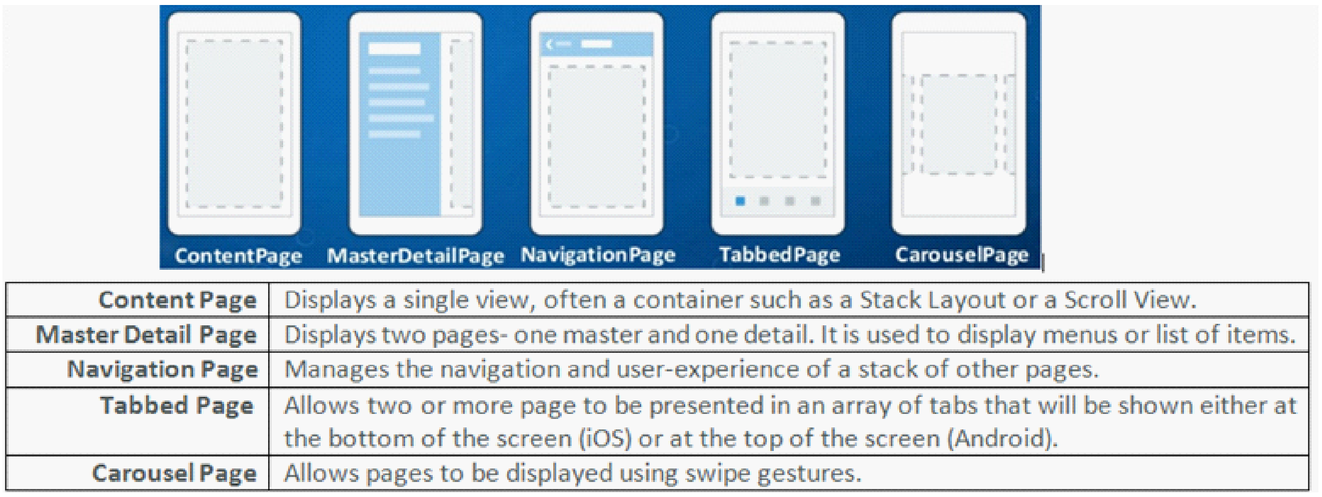
ABC of Xamarin Forms User Interface

Xamarin Forms uses the specific native platform controls by utilizing the PCL (Portable Class Libraries) and Shared Projects in platform specific applications. It provides a flexibility to design UI either in C# or XAML. Each UI Control will generally be mapped to native platform equivalent at the time of rendering.

UI Elements

Xamarin Forms User Interface is further categorized into four control groups **- Pages, Layout, Views & Cells.**

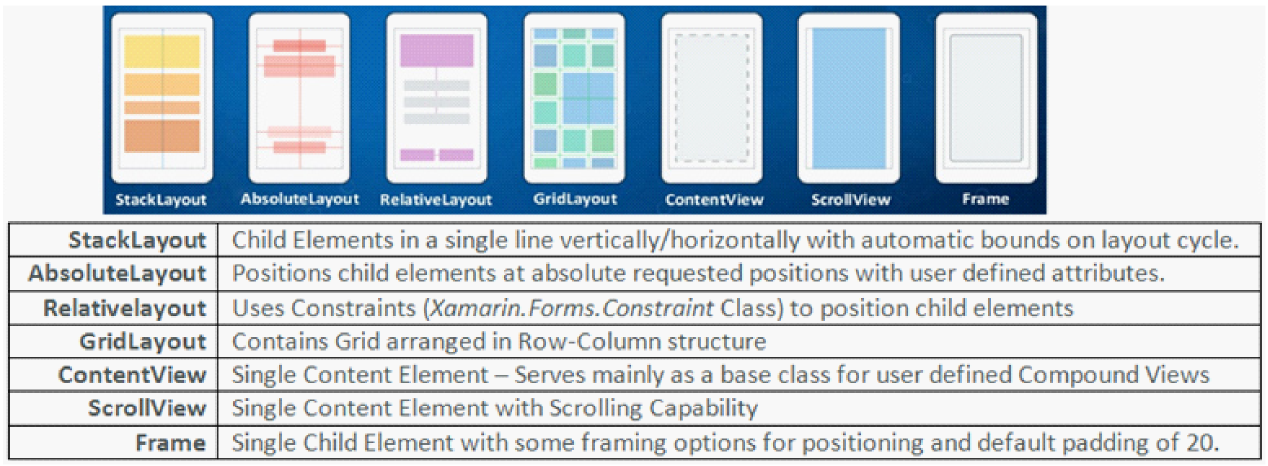
Pages – Cross Platform Mobile App Screens; An abstract class to define content.



Content Page	Displays a single view, often a container such as a Stack Layout or a Scroll View.
Master Detail Page	Displays two pages- one master and one detail. It is used to display menus or list of items.
Navigation Page	Manages the navigation and user-experience of a stack of other pages.
Tabbed Page	Allows two or more page to be presented in an array of tabs that will be shown either at the bottom of the screen (iOS) or at the top of the screen (Android).
Carousel Page	Allows pages to be displayed using swipe gestures.

Image – Xamarin Forms Pages

Layouts – Organize the elements (UI controls) of the Pages into logical structures.

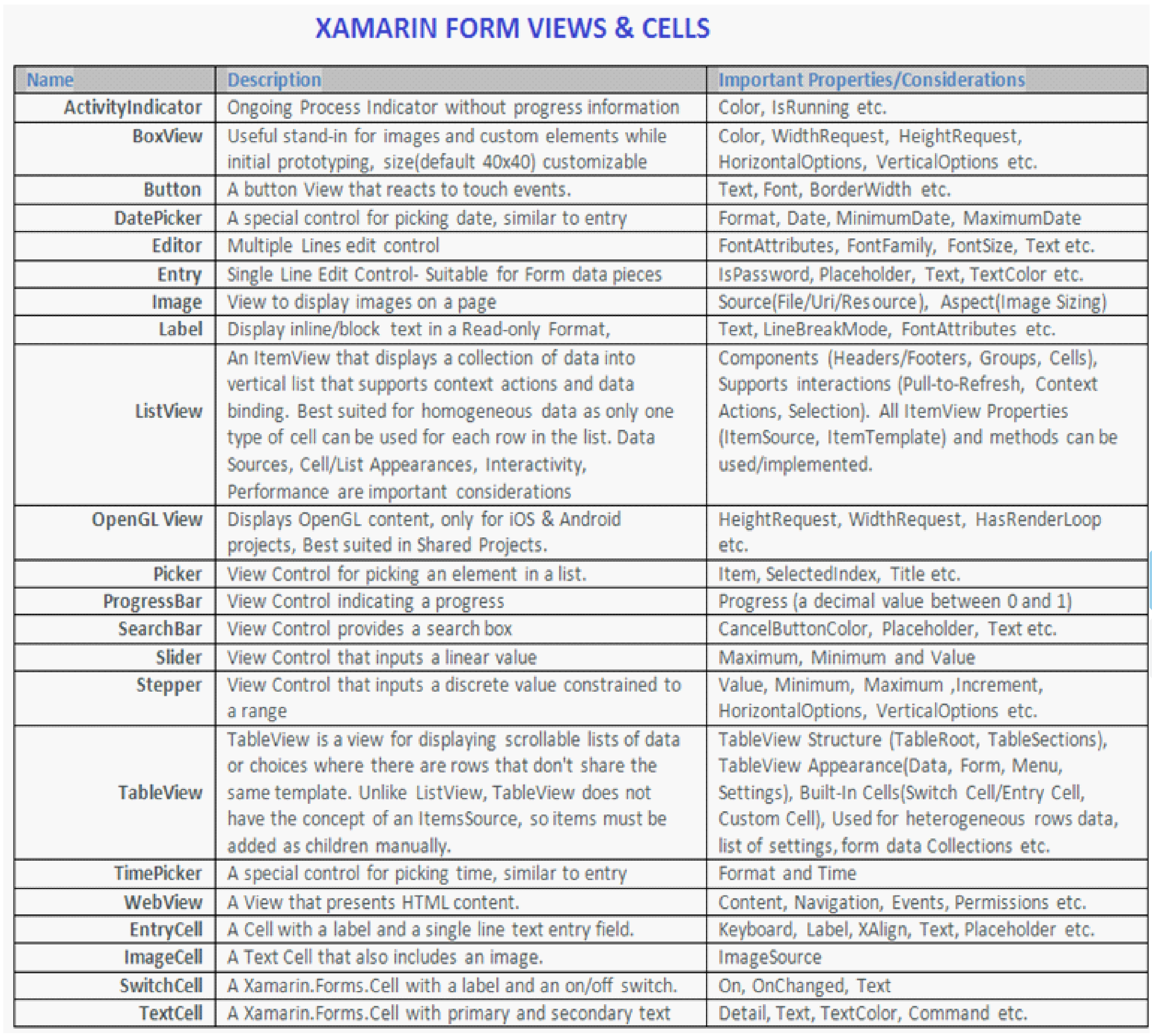


StackLayout	Child Elements in a single line vertically/horizontally with automatic bounds on layout cycle.
AbsoluteLayout	Positions child elements at absolute requested positions with user defined attributes.
Relativelayout	Uses Constraints (*Xamarin.Forms.Constraint* Class) to position child elements
GridLayout	Contains Grid arranged in Row-Column structure
ContentView	Single Content Element – Serves mainly as a base class for user defined Compound Views
ScrollView	Single Content Element with Scrolling Capability
Frame	Single Child Element with some framing options for positioning and default padding of 20.

Image – Xamarin Forms Layouts

Views – Visual Objects, Building blocks of Cross Platform Mobile UI, Controls of Widget
Cells – Visual Element Creation Template; Designed for ListView and TableView controls Items.

Below are few commonly used views and cells that are supported by Xamarin Forms.



XAMARIN FORM VIEWS & CELLS

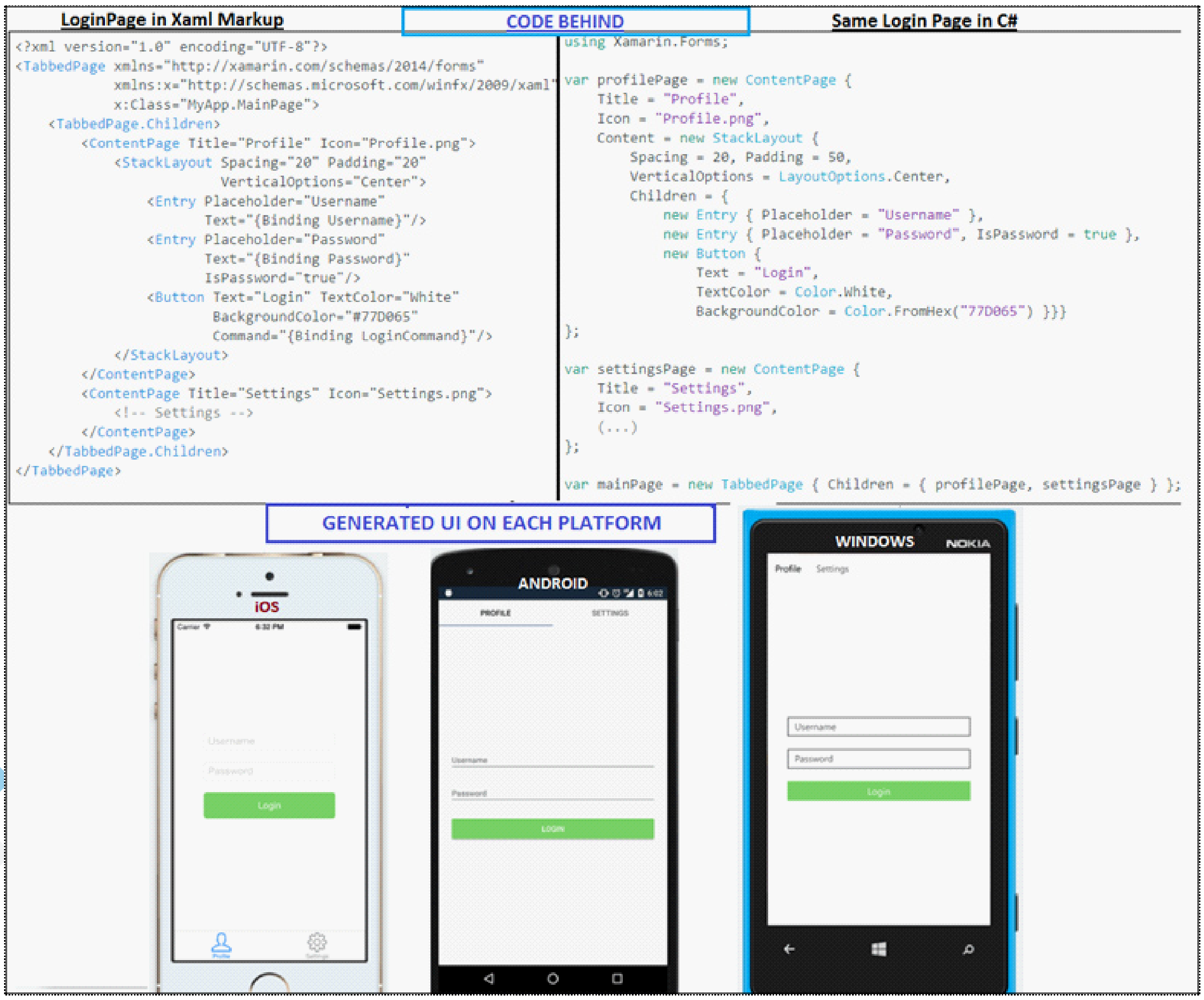
Name	Description	Important Properties/Considerations
ActivityIndicator	Ongoing Process Indicator without progress information	Color, IsRunning etc.
BoxView	Useful stand-in for images and custom elements while initial prototyping, size(default 40x40) customizable	Color, WidthRequest, HeightRequest, HorizontalOptions, VerticalOptions etc.
Button	A button View that reacts to touch events.	Text, Font, BorderWidth etc.
DatePicker	A special control for picking date, similar to entry	Format, Date, MinimumDate, MaximumDate
Editor	Multiple Lines edit control	FontAttributes, FontFamily, FontSize, Text etc.
Entry	Single Line Edit Control- Suitable for Form data pieces	IsPassword, Placeholder, Text, TextColor etc.
Image	View to display images on a page	Source(File/Uri/Resource), Aspect(Image Sizing)
Label	Display inline/block text in a Read-only Format,	Text, LineBreakMode, FontAttributes etc.
ListView	An ItemView that displays a collection of data into vertical list that supports context actions and data binding. Best suited for homogeneous data as only one type of cell can be used for each row in the list. Data Sources, Cell/List Appearances, Interactivity, Performance are important considerations	Components (Headers/Footers, Groups, Cells), Supports interactions (Pull-to-Refresh, Context Actions, Selection). All ItemView Properties (ItemSource, ItemTemplate) and methods can be used/implemented.
OpenGL View	Displays OpenGL content, only for iOS & Android projects, Best suited in Shared Projects.	HeightRequest, WidthRequest, HasRenderLoop etc.
Picker	View Control for picking an element in a list.	Item, SelectedIndex, Title etc.
ProgressBar	View Control indicating a progress	Progress (a decimal value between 0 and 1)
SearchBar	View Control provides a search box	CancelButtonColor, Placeholder, Text etc.
Slider	View Control that inputs a linear value	Maximum, Minimum and Value
Stepper	View Control that inputs a discrete value constrained to a range	Value, Minimum, Maximum ,Increment, HorizontalOptions, VerticalOptions etc.
TableView	TableView is a view for displaying scrollable lists of data or choices where there are rows that don't share the same template. Unlike ListView, TableView does not have the concept of an ItemsSource, so items must be added as children manually.	TableView Structure (TableRoot, TableSections), TableView Appearance(Data, Form, Menu, Settings), Built-In Cells(Switch Cell/Entry Cell, Custom Cell), Used for heterogeneous rows data, list of settings, form data Collections etc.
TimePicker	A special control for picking time, similar to entry	Format and Time
WebView	A View that presents HTML content.	Content, Navigation, Events, Permissions etc.
EntryCell	A Cell with a label and a single line text entry field.	Keyboard, Label, XAlign, Text, Placeholder etc.
ImageCell	A Text Cell that also includes an image.	ImageSource
SwitchCell	A Xamarin.Forms.Cell with a label and an on/off switch.	On, OnChanged, Text
TextCell	A Xamarin.Forms.Cell with primary and secondary text	Detail, Text, TextColor, Command etc.

Image – Xamarin Forms Views and Cells

Create a Sample Xamarin Form UI

User can build the UI using either Xaml markup or C#; All we need to know is how to use Code intellisense in Visual Studio and of course Coding and UI designing skills. The most important aspect is designing the layout and Page; once the page is setup, using each control is easy with provided properties. If UI glitters on different platform, one can always tweak using platform specific approaches.

Below is a sample code snippet for a user login page which utilizes TabbedPage, Content Page, StackLayout, Entry and Button UI controls.



```
<?xml version="1.0" encoding="UTF-8"?>
<TabbedPage xmlns="http://xamarin.com/schemas/2014/forms"
            xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
            x:Class="MyApp.MainPage">
    <TabbedPage.Children>
        <ContentPage Title="Profile" Icon="Profile.png">
            <StackLayout Spacing="20" Padding="20"
                         VerticalOptions="Center">
                <Entry Placeholder="Username"
                       Text="{Binding Username}"/>
                <Entry Placeholder="Password"
                       Text="{Binding Password}"
                       IsPassword="true"/>
                <Button Text="Login" TextColor="White"
                        BackgroundColor="#77D065"
                        Command="{Binding LoginCommand}"/>
            </StackLayout>
        </ContentPage>
        <ContentPage Title="Settings" Icon="Settings.png">
            <!-- Settings -->
        </ContentPage>
    </TabbedPage.Children>
</TabbedPage>
```

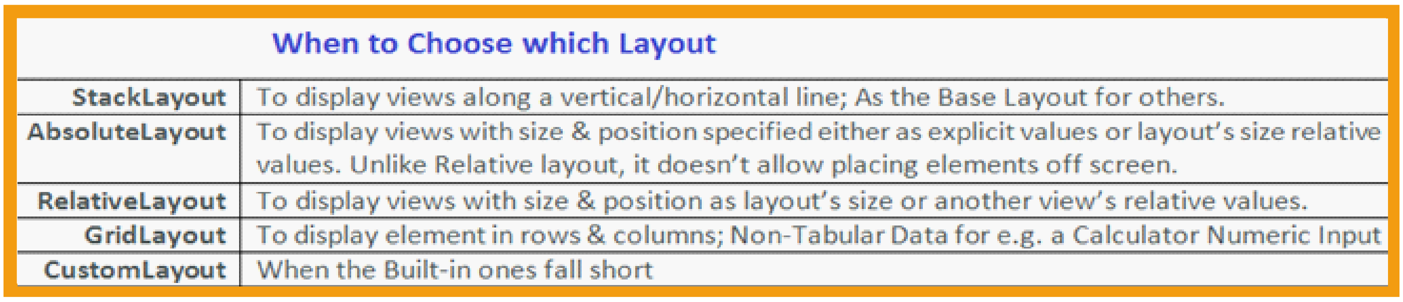
```
using Xamarin.Forms;

var profilePage = new ContentPage {
    Title = "Profile",
    Icon = "Profile.png",
    Content = new StackLayout {
        Spacing = 20, Padding = 50,
        VerticalOptions = LayoutOptions.Center,
        Children = {
            new Entry { Placeholder = "Username" },
            new Entry { Placeholder = "Password", IsPassword = true },
            new Button {
                Text = "Login",
                TextColor = Color.White,
                BackgroundColor = Color.FromHex("77D065") }}}
};

var settingsPage = new ContentPage {
    Title = "Settings",
    Icon = "Settings.png",
    (...)
};

var mainPage = new TabbedPage { Children = { profilePage, settingsPage } };
```

Image – Creating a Sample User Login Page

UI Design Considerations

Choosing a Right Layout

This is the most important question that arises when an app need to be at cross platform. Choosing a right layout not only saves much coding efforts, it improves the app performance as well. When there are multiple valid choices, consider which approach will be the easiest for your situation. Generally, multiple layouts are used together to implement a specific page. Moreover one can easily create more complex UI by nesting different layouts as desired.



When to Choose which Layout	
StackLayout	To display views along a vertical/horizontal line; As the Base Layout for others.
AbsoluteLayout	To display views with size & position specified either as explicit values or layout's size relative values. Unlike Relative layout, it doesn't allow placing elements off screen.
RelativeLayout	To display views with size & position as layout's size or another view's relative values.
GridLayout	To display element in rows & columns; Non-Tabular Data for e.g. a Calculator Numeric Input
CustomLayout	When the Built-in ones fall short

Image – Choosing a Right Layout

Deciding on Page Navigation

Xamarin Forms provides multiple navigation options depending on selected page type.

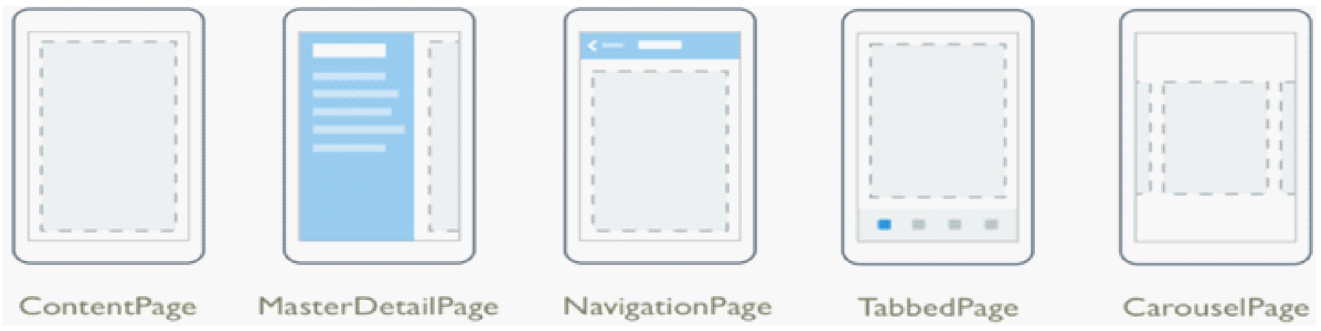
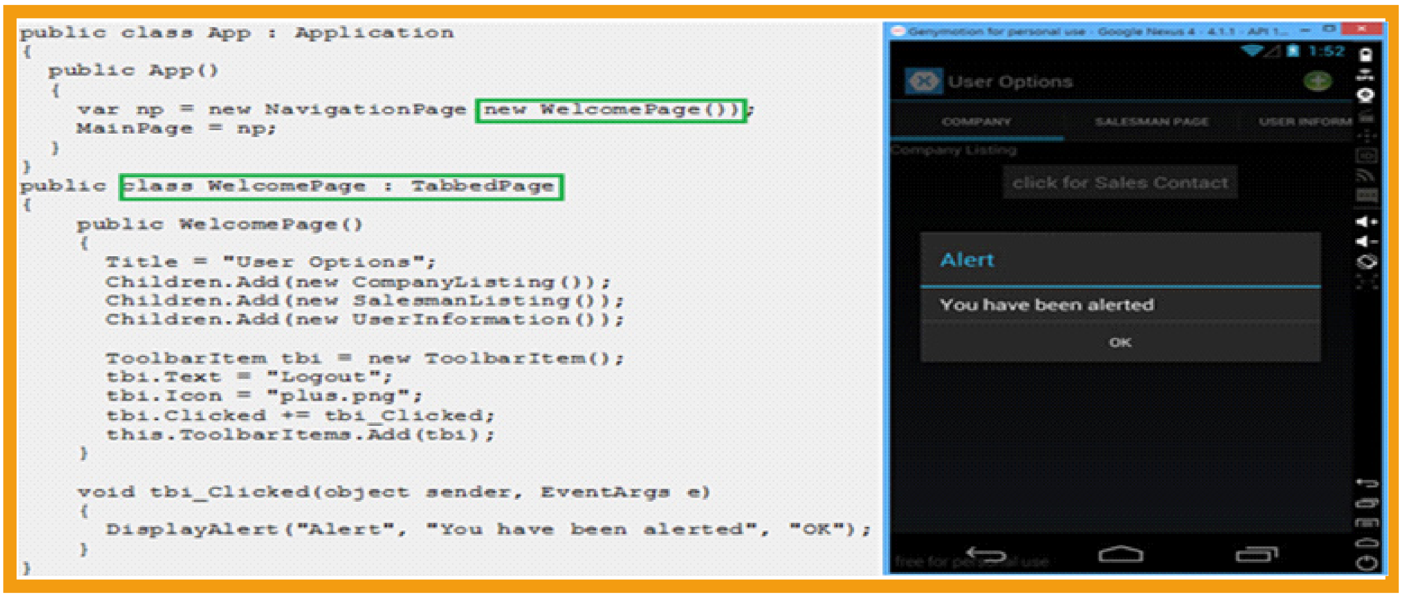


Image – Choosing a Right Layout

Xamarin Forms also provides support for modal pages that encourages users to complete a self-contained task that cannot be navigated away from, until the task is completed or cancelled. Additionally it has two pop-up like user interface elements: an alert and an action sheet. These interface elements can be used to ask users simple questions and to guide users through tasks.

Create a Sample Navigation Page

While creating main root navigation Page, pass the class of navigation page (which can be any of provided base Classes like Tabbed Page, Carousel Page etc.). All the core logic (i.e. callbacks, property settings etc.) will be placed inside navigation page class as per selected base Class.

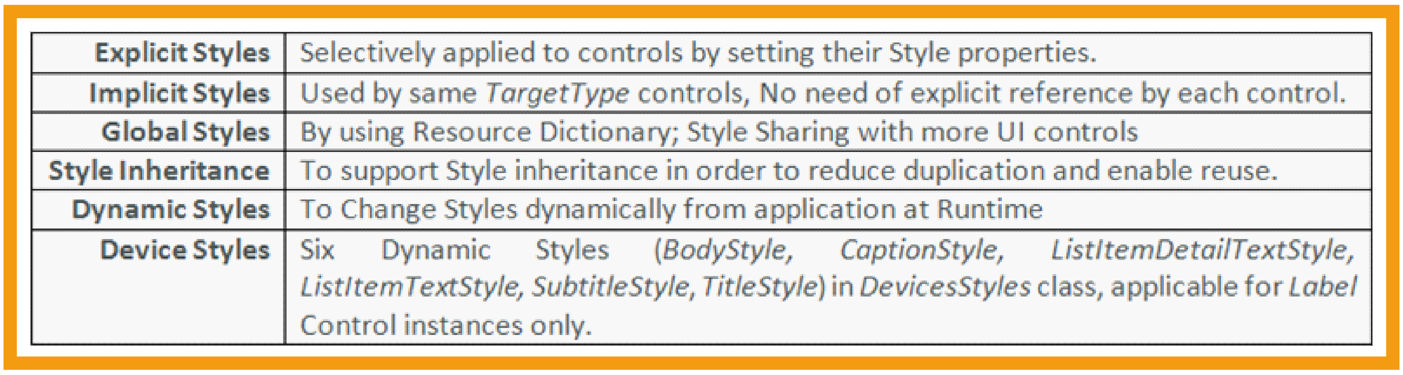


```
public class App : Application
{
  public App()
  {
    var np = new NavigationPage new WelcomePage());
    MainPage = np;
  }
}
public class WelcomePage : TabbedPage
{
    public WelcomePage()
    {
      Title = "User Options";
      Children.Add(new CompanyListing());
      Children.Add(new SalesmanListing());
      Children.Add(new UserInformation());

      ToolbarItem tbi = new ToolbarItem();
      tbi.Text = "Logout";
      tbi.Icon = "plus.png";
      tbi.Clicked += tbi_Clicked;
      this.ToolbarItems.Add(tbi);
    }

    void tbi_Clicked(object sender, EventArgs e)
    {
      DisplayAlert("Alert", "You have been alerted", "OK");
    }
}
```

Image - Sample Page with 3 Tabs Navigation

Customizing Appearances through Styles

As setting the appearance for each individual control is a repetitive and error prone task, Xamarin Forms provide multiple styling options for customizing the control appearances by grouping and settings the properties available on it. Remember CSS Class, oh you got it. Tthis is quite similar!!!!!



Explicit Styles	Selectively applied to controls by setting their Style properties.
Implicit Styles	Used by same *TargetType* controls, No need of explicit reference by each control.
Global Styles	By using Resource Dictionary; Style Sharing with more UI controls
Style Inheritance	To support Style inheritance in order to reduce duplication and enable reuse.
Dynamic Styles	To Change Styles dynamically from application at Runtime
Device Styles	Six Dynamic Styles (*BodyStyle*, *CaptionStyle*, *ListItemDetailTextStyle*, *ListItemTextStyle*, *SubtitleStyle*, *TitleStyle*) in *DevicesStyles* class, applicable for *Label* Control instances only.

Image –Xamarin Forms Styles

Resource Dictionary

Resource Dictionaries can be specified at both Page Level and App Level. When one need to use it across pages then it's better to specify it at app level in App.xaml.

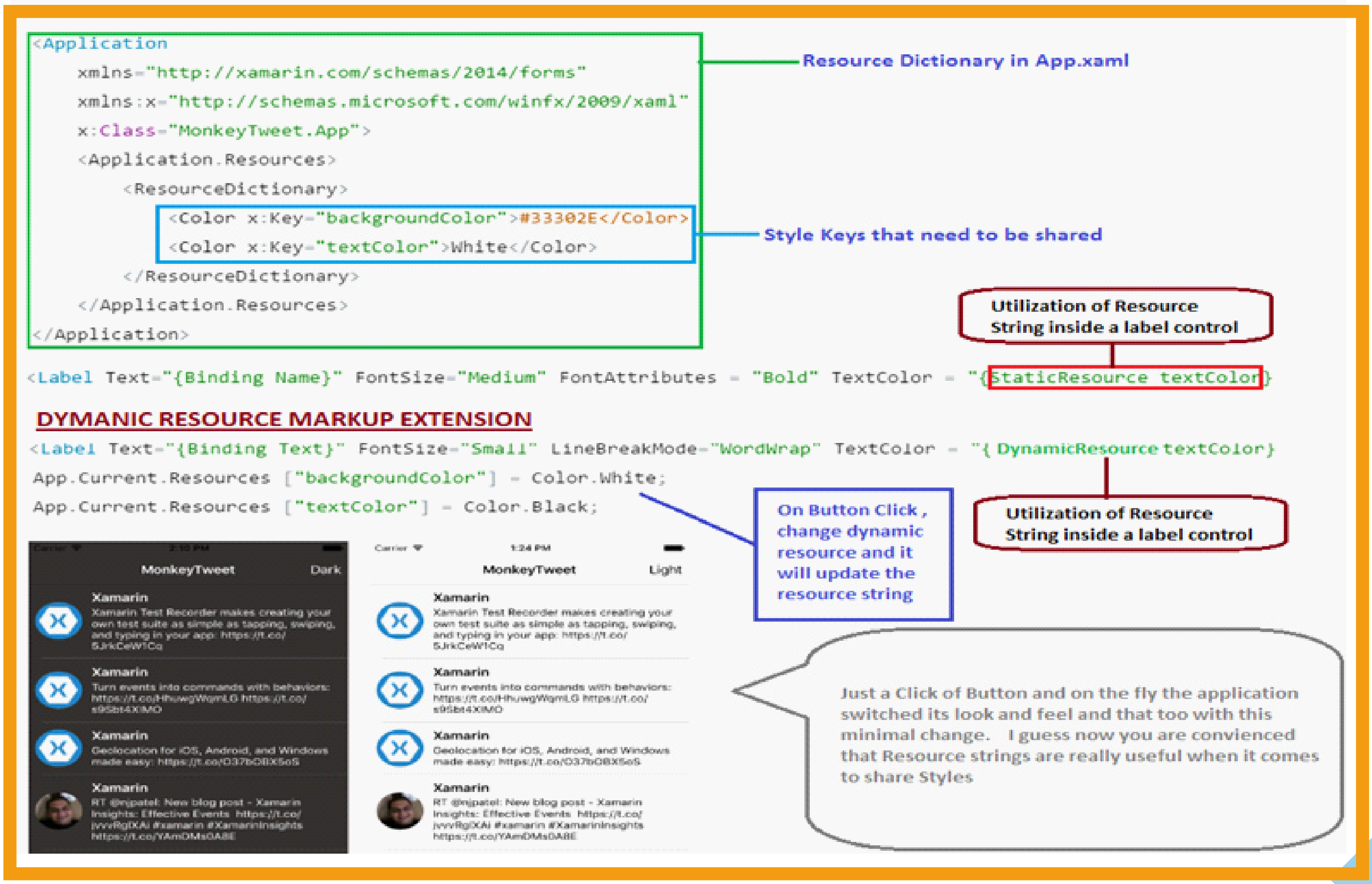
To utilize the resource string key, Xaml provides 2 markup extensions

Static Resource

Used when we want a static value for all controls that cannot be changed Once loaded from Resource Dictionary at start.

Dynamic Resource

To change UI behavior dynamically Let's say on button click



```
<Application
    xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="MonkeyTweet.App">
    <Application.Resources>
        <ResourceDictionary>
            <Color x:Key="backgroundColor">#33302E</Color>
            <Color x:Key="textColor">White</Color>
        </ResourceDictionary>
    </Application.Resources>
</Application>
<Label Text="{Binding Name}" FontSize="Medium" FontAttributes = "Bold" TextColor = "{StaticResource textColor}
```

Resource Dictionary in App.xaml

Style Keys that need to be shared

Utilization of Resource String inside a label control

DYMANIC RESOURCE MARKUP EXTENSION

```
<Label Text="{Binding Text}" FontSize="Small" LineBreakMode="WordWrap" TextColor = "{ DynamicResource textColor}
App.Current.Resources ["backgroundColor"] = Color.White;
App.Current.Resources ["textColor"] = Color.Black;
```

On Button Click , change dynamic resource and it will update the resource string

Utilization of Resource String inside a label control

MonkeyTweet Dark

Xamarin
Xamarin Test Recorder makes creating your own test suite as simple as tapping, swiping, and typing in your app: https://t.co/5JrkCeW1Cq

Xamarin
Turn events into commands with behaviors: https://t.co/HhuwgWqmLG https://t.co/s9Sbt4XIMO

Xamarin
Geolocation for iOS, Android, and Windows made easy: https://t.co/O37bOBX5oS

Xamarin
RT @njpatel: New blog post - Xamarin Insights: Effective Events https://t.co/jvvvRgIXAi #xamarin #XamarinInsights https://t.co/YAmDMsOABE

MonkeyTweet Light

Xamarin
Xamarin Test Recorder makes creating your own test suite as simple as tapping, swiping, and typing in your app: https://t.co/5JrkCeW1Cq

Xamarin
Turn events into commands with behaviors: https://t.co/HhuwgWqmLG https://t.co/s9Sbt4XIMO

Xamarin
Geolocation for iOS, Android, and Windows made easy: https://t.co/O37bOBX5oS

Xamarin
RT @njpatel: New blog post - Xamarin Insights: Effective Events https://t.co/jvvvRgIXAi #xamarin #XamarinInsights https://t.co/YAmDMsOABE

Just a Click of Button and on the fly the application switched its look and feel and that too with this minimal change. I guess now you are convienced that Resource strings are really useful when it comes to share Styles

Image – Sharing Style with Resource String

Advance UI Tricks for Xamarin Form

Map Control - Can I create a re-usable Map UI Component?

Xamarin Forms uses the native map APIs on each platform via Map Control. It can be further enhanced by creating a map custom renderer. Once the Map Control is initialized and configured, it can be used as normal view element in the app.

Gestures - What about my Interaction with the UI?

Xamarin Form has **GestureRecognizer** class that supports tap, pinch and pan gestures on various user interface controls.

Data pages and Theme – Is there any UI readily available?

Data Pages provide an API to bind a data source to pre-built views. List items and detail pages will automatically render the data, and can be customized using themes (Xamarin.Forms.Theme.Base Nuget package).

Data Sources: JsonDataSource, AzureDataSource and AzureEasyTableDataSource.

Pages and Controls: ListDataPage, DirectoryPage, PersonDetailPage, DataView, CardView, HeroImage and ListItem

Data Sources Interaction with Xamarin Form Infrastructure: Using Properties like Data (a read-only collection), IsLoading (Boolean flag indication data load) and [key] (An indexer to retrieve elements). Additionally there are two methods MaskKey and UnmaskKey that can be used to hide/show data item properties.

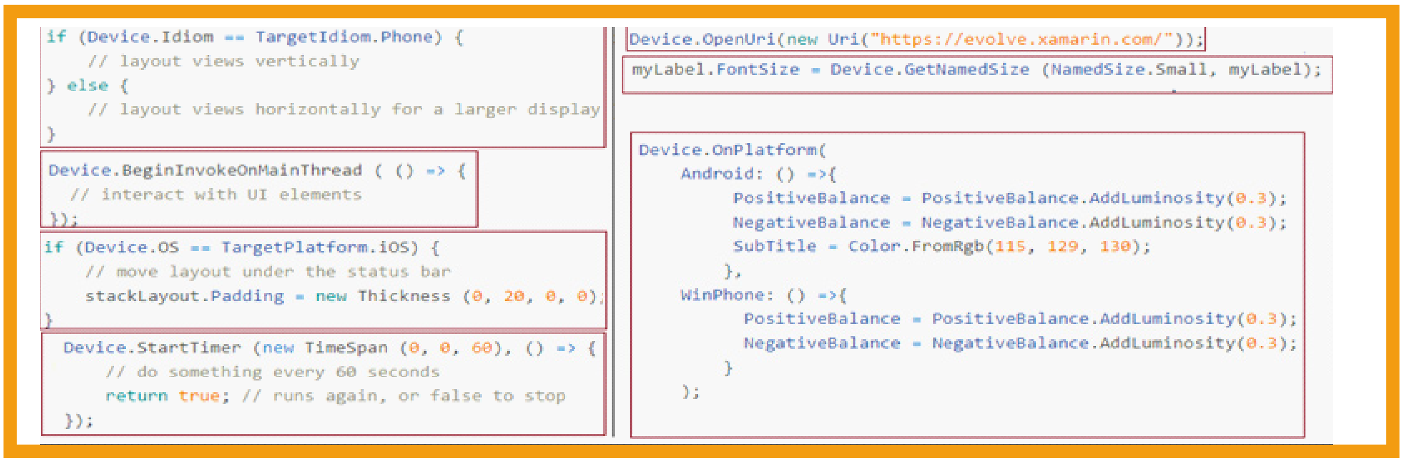
Themes (Light, Dark and Custom): To Define a specific visual appearance for standard controls, generally placed in application's resource dictionary.

Platform Specific Tweaks

Xamarin.Forms provides a number of ways in which developers can make use of the platform specific features. Let's focus on Device Class as it helps not only to customize the shared UI layer specific to device or platform in the common code (either PCL or Shared Projects) but also provides methods and properties to target specific hardware types and sizes.

In a nutshell it can be said that Device class allows fine-grained control over functionality and layouts on a per-platform basis - even in common code (either PCL or Shared Projects).

Device Class Properties/Methods	Use Case Scenarios
Device.OS	Building layouts that take advantage of larger screens, It has 4 different supported values Phone, Tablet, Desktop, Unsupported
Device.OnPlatform	Platform specific tweaks for each OS
Device.Styles	Built-in style definitions that can be applied to some controls. Available Styles are - BodyStyle, CaptionStyle, ListItemDetailTextStyle, ListItemTextStyle, SubtitleStyle, TitleStyle
Device.GetNamedSize	When setting FontSize in C# code:
Device.OpenUri	To trigger operations on the underlying platform, such as open a URL
Device.StartTimer	A simple way to trigger time-dependent tasks that works in Xamarin.Forms common code (including PCLs).
Device.BeginInvokeOnMainThread	Any background code that needs to update the user interface should be wrapped inside BeginInvokeOnMainThread. This is the equivalent of InvokeOnMainThread on iOS, RunOnUiThread on Android, and Dispatcher.BeginInvoke on Windows Phone.



```
if (Device.Idiom == TargetIdiom.Phone) {
    // layout views vertically
} else {
    // layout views horizontally for a larger display
}
```

```
Device.BeginInvokeOnMainThread ( () => {
  // interact with UI elements
});
```

```
if (Device.OS == TargetPlatform.iOS) {
    // move layout under the status bar
    stackLayout.Padding = new Thickness (0, 20, 0, 0);
}
```

```
 Device.StartTimer (new TimeSpan (0, 0, 60), () => {
     // do something every 60 seconds
     return true; // runs again, or false to stop
 });
```

```
Device.OpenUri(new Uri("https://evolve.xamarin.com/"));
```

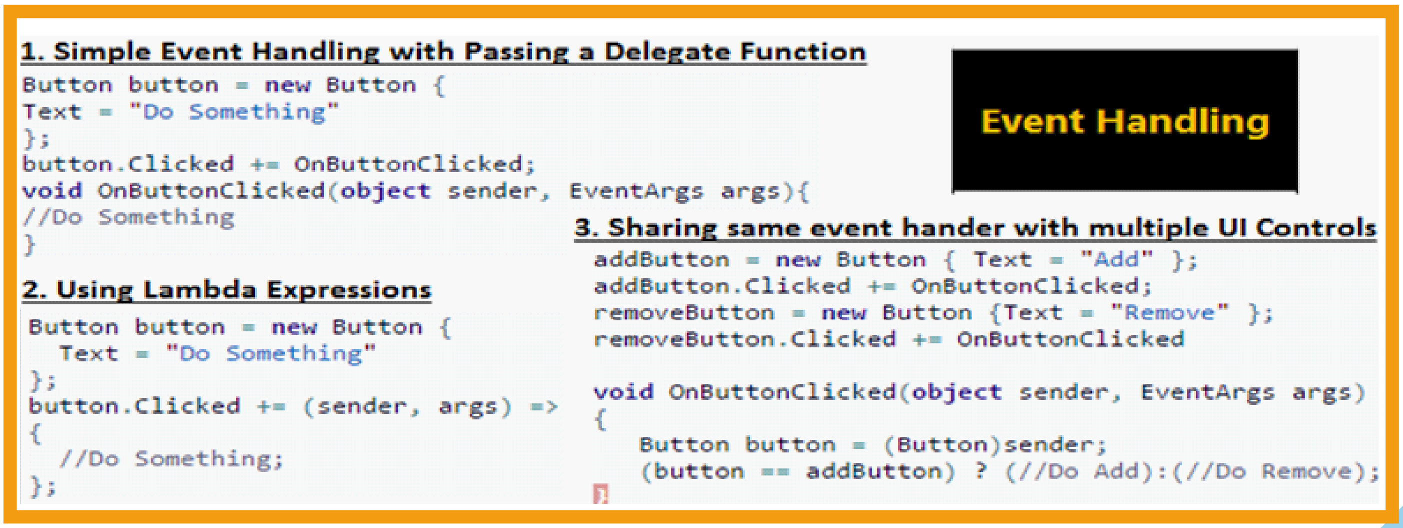
```
myLabel.FontSize = Device.GetNamedSize (NamedSize.Small, myLabel);
```

```
Device.OnPlatform(
    Android: () =>{
        PositiveBalance = PositiveBalance.AddLuminosity(0.3);
        NegativeBalance = NegativeBalance.AddLuminosity(0.3);
        SubTitle = Color.FromRgb(115, 129, 130);
      },
    WinPhone: () =>{
         PositiveBalance = PositiveBalance.AddLuminosity(0.3);
         NegativeBalance = NegativeBalance.AddLuminosity(0.3);
      }
    );
```

Image – Platform Specific tweaks in Xamarin Form using Device Class

Event Handling

Events are the basic means to execute command based on user interaction with the application. The event handling mechanism in Xamarin Form App is similar to event handling used preciously in ASP.NET/Silverlight apps. Let us explore few code snippets for clear picture. Let's take the example of most common user interaction, a button-click event.



1. Simple Event Handling with Passing a Delegate Function

```
Button button = new Button {
Text = "Do Something"
};
button.Clicked += OnButtonClicked;
void OnButtonClicked(object sender, EventArgs args){
//Do Something
}
```

2. Using Lambda Expressions

```
Button button = new Button {
  Text = "Do Something"
};
button.Clicked += (sender, args) =>
{
  //Do Something;
};
```

3. Sharing same event hander with multiple UI Controls

```
addButton = new Button { Text = "Add" };
addButton.Clicked += OnButtonClicked;
removeButton = new Button {Text = "Remove" };
removeButton.Clicked += OnButtonClicked

void OnButtonClicked(object sender, EventArgs args)
{
   Button button = (Button)sender;
   (button == addButton) ? (//Do Add):(//Do Remove);
}
```

Image – Sample Button Click Event Handling in Xamarin Forms

To handle application life-cycle events, the App Class in Xamarin Forms template overrides three virtual protected methods OnStart, OnSleep, and OnResume (defined in base Application class). These are categorized as application level events.

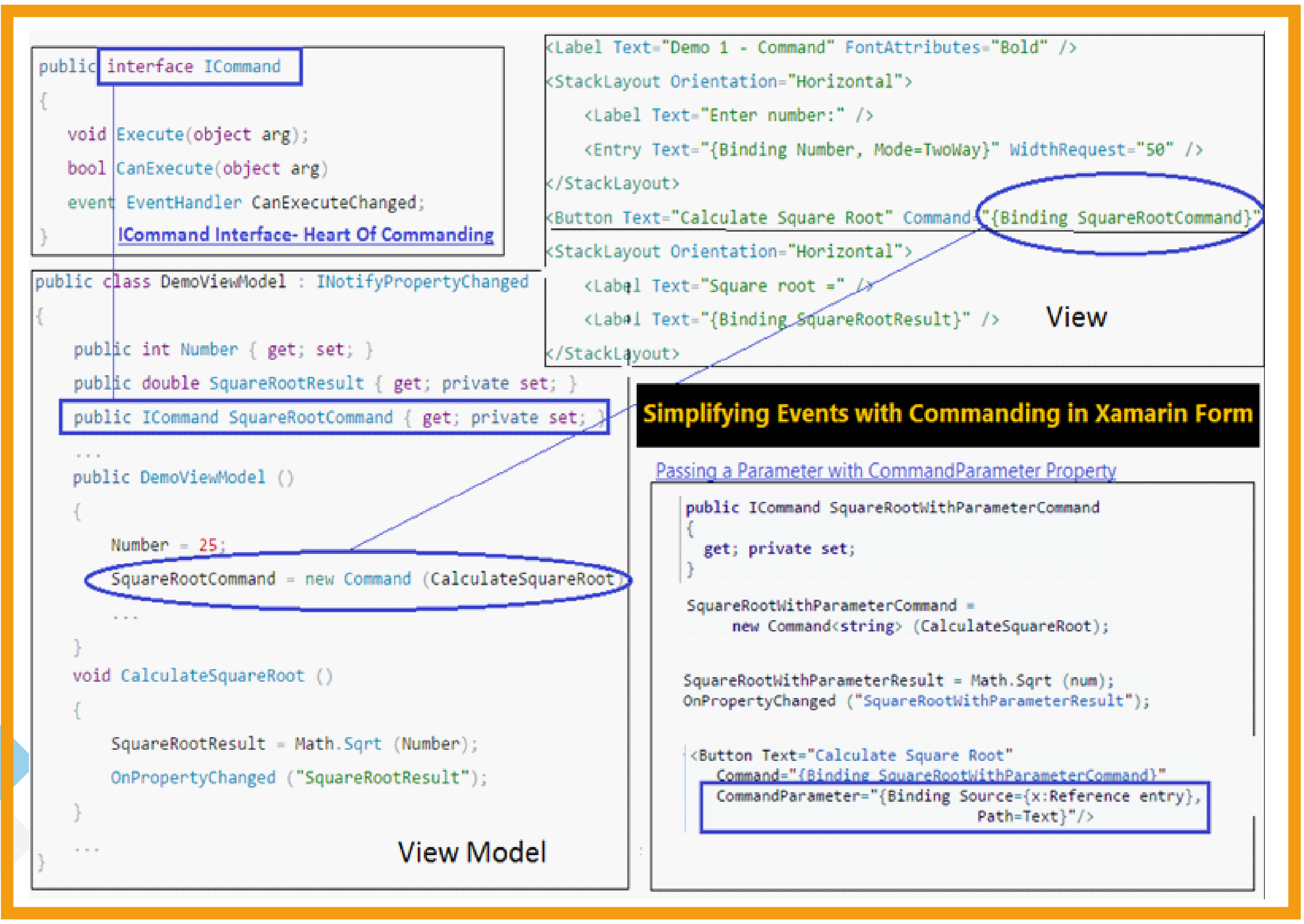
At this point of time you might be thinking, there is nothing new in UI event handling in Xamarin Form, however, it has a new concept to offer called as '**Commanding**'.

Power of Commanding

Special Concept in Xamarin Forms to tackle events in much simpler way.

Commanding makes use of traditional XAML data bindings to make method calls directly to the ViewModel instead of attaching an event handler to the UI Control.

There are eight UI control classes - Button, MenuItem, ToolbarItem, SearchBar, TextCell, ImageCell, ListView and TapGestureRecognizer that can implement commanding using two public properties – Command (of type System.Windows.Input.ICommand) and CommandParameter (of type object).



```
public interface ICommand
{
    void Execute(object arg);
    bool CanExecute(object arg)
    event EventHandler CanExecuteChanged;
}
```

ICommand Interface- Heart Of Commanding

```
public class DemoViewModel : INotifyPropertyChanged
{
    public int Number { get; set; }
    public double SquareRootResult { get; private set; }
    public ICommand SquareRootCommand { get; private set; }
    ...
    public DemoViewModel ()
    {
        Number = 25;
        SquareRootCommand = new Command (CalculateSquareRoot)
        ...
    }
    void CalculateSquareRoot ()
    {
        SquareRootResult = Math.Sqrt (Number);
        OnPropertyChanged ("SquareRootResult");
    }
    ...
}
```

```
<Label Text="Demo 1 - Command" FontAttributes="Bold" />
<StackLayout Orientation="Horizontal">
    <Label Text="Enter number:" />
    <Entry Text="{Binding Number, Mode=TwoWay}" WidthRequest="50" />
</StackLayout>
<Button Text="Calculate Square Root" Command="{Binding SquareRootCommand}"
<StackLayout Orientation="Horizontal">
    <Label Text="Square root =" />
    <Label Text="{Binding SquareRootResult}" />
</StackLayout>
```

Simplifying Events with Commanding in Xamarin Form

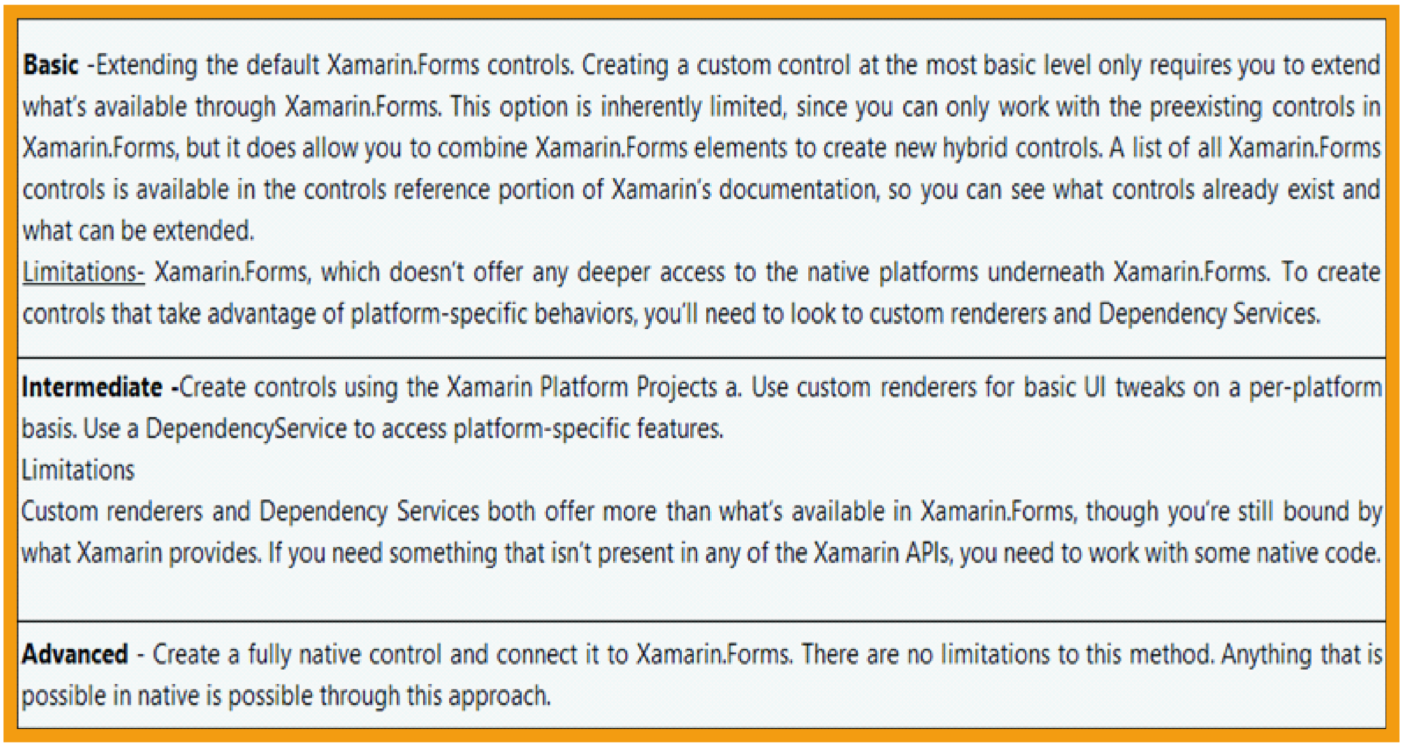
Passing a Parameter with CommandParameter Property

```
public ICommand SquareRootWithParameterCommand
{
    get; private set;
}

SquareRootWithParameterCommand =
        new Command<string> (CalculateSquareRoot);

SquareRootWithParameterResult = Math.Sqrt (num);
OnPropertyChanged ("SquareRootWithParameterResult");

<Button Text="Calculate Square Root"
    Command="{Binding SquareRootWithParameterCommand}"
    CommandParameter="{Binding Source={x:Reference entry},
                               Path=Text}"/>
```

Image – Concept of Commanding in Xamarin Forms

Custom UI Controls

Xamarin.Forms provides many built-in controls for basic UI design, but there are some lacking areas. Since Xamarin.Forms maps to each platform's native controls, they skew toward the more basic and common controls shared among platforms. Complex controls enabling advanced data visualizations and data management aren't represented in this group of controls, either. Fortunately, the ingenious developer can take multiple approaches to add any missing functionality.



Basic -Extending the default Xamarin.Forms controls. Creating a custom control at the most basic level only requires you to extend what's available through Xamarin.Forms. This option is inherently limited, since you can only work with the preexisting controls in Xamarin.Forms, but it does allow you to combine Xamarin.Forms elements to create new hybrid controls. A list of all Xamarin.Forms controls is available in the controls reference portion of Xamarin's documentation, so you can see what controls already exist and what can be extended.
Limitations- Xamarin.Forms, which doesn't offer any deeper access to the native platforms underneath Xamarin.Forms. To create controls that take advantage of platform-specific behaviors, you'll need to look to custom renderers and Dependency Services.

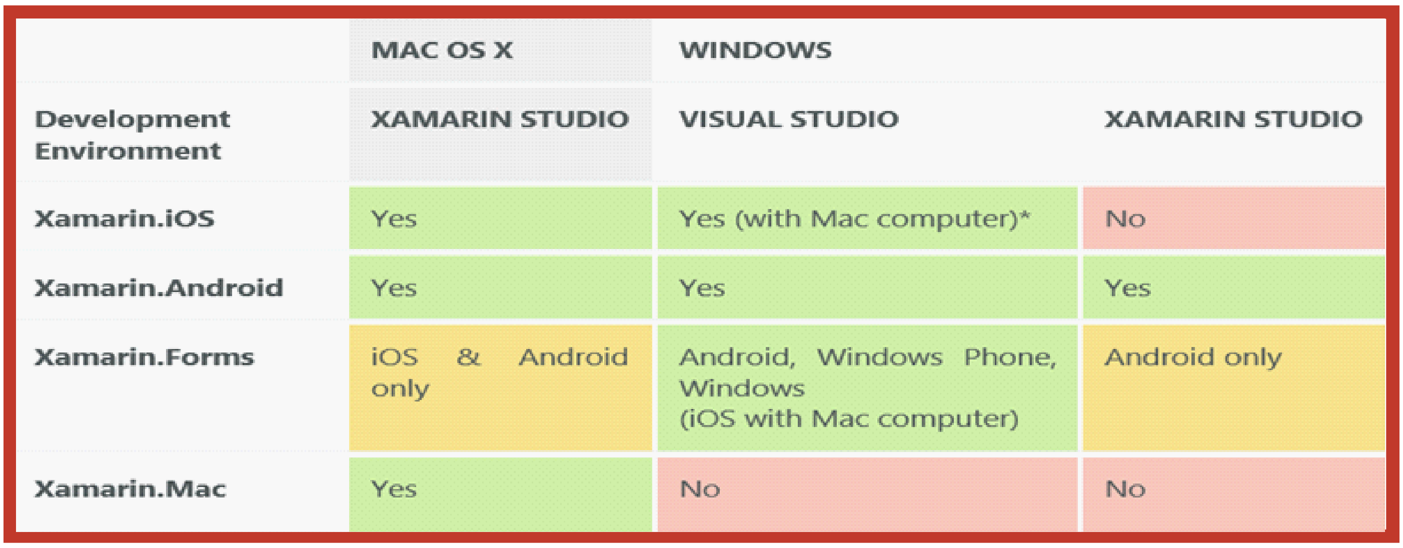
Intermediate -Create controls using the Xamarin Platform Projects a. Use custom renderers for basic UI tweaks on a per-platform basis. Use a DependencyService to access platform-specific features.
Limitations
Custom renderers and Dependency Services both offer more than what's available in Xamarin.Forms, though you're still bound by what Xamarin provides. If you need something that isn't present in any of the Xamarin APIs, you need to work with some native code.

Advanced - Create a fully native control and connect it to Xamarin.Forms. There are no limitations to this method. Anything that is possible in native is possible through this approach.

Image – Various Approaches to create custom UI control in Xamarin Forms

7 Development Environments

Below table shows major platforms that can be built with different development tools and operating system combinations for Xamarin.



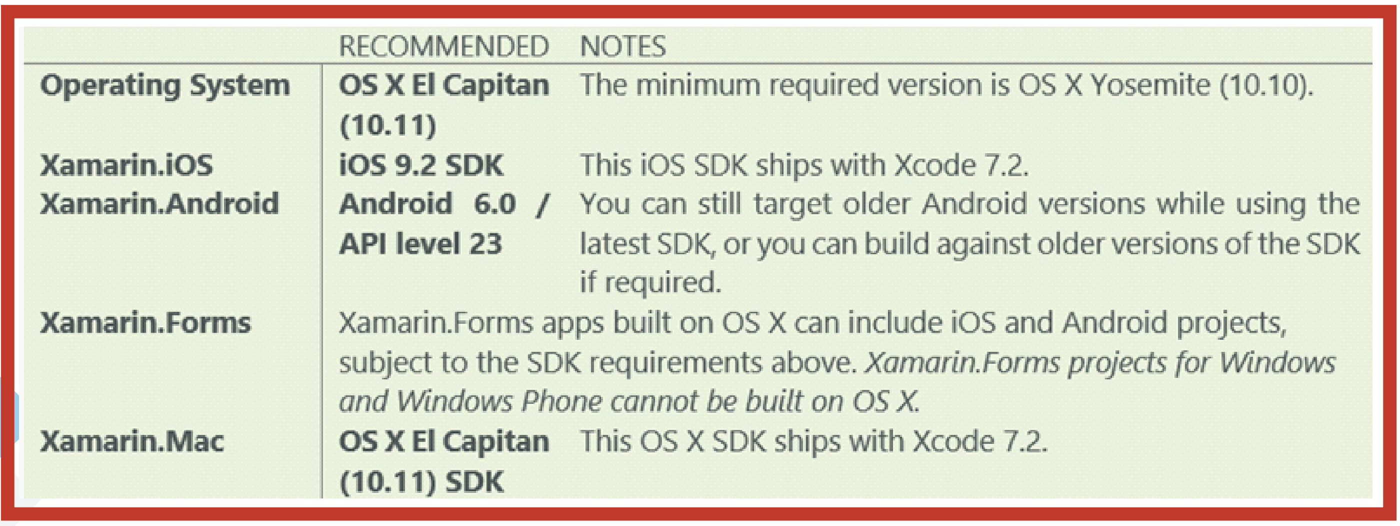
	MAC OS X	WINDOWS	
Development Environment	**XAMARIN STUDIO**	**VISUAL STUDIO**	**XAMARIN STUDIO**
Xamarin.iOS	Yes	Yes (with Mac computer)*	No
Xamarin.Android	Yes	Yes	Yes
Xamarin.Forms	iOS & Android only	Android, Windows Phone, Windows (iOS with Mac computer)	Android only
Xamarin.Mac	Yes	No	No

Image – Xamarin Forms Cross Platform App Development Environment

** To develop for iOS on Windows computers there must be a Mac computer accessible on the network, for remote compilation and debugging. This also works if you have Visual Studio running inside a Windows VM on a Mac computer.*

Mac OS X Requirements

Using a Mac computer for Xamarin development requires the following software/SDK versions. Check your operating system version and follow the instructions for the Xamarin installer.



	RECOMMENDED	NOTES
Operating System	**OS X El Capitan (10.11)**	The minimum required version is OS X Yosemite (10.10).
Xamarin.iOS	**iOS 9.2 SDK**	This iOS SDK ships with Xcode 7.2.
Xamarin.Android	**Android 6.0 / API level 23**	You can still target older Android versions while using the latest SDK, or you can build against older versions of the SDK if required.
Xamarin.Forms	Xamarin.Forms apps built on OS X can include iOS and Android projects, subject to the SDK requirements above. *Xamarin.Forms projects for Windows and Windows Phone cannot be built on OS X.*	
Xamarin.Mac	**OS X El Capitan (10.11) SDK**	This OS X SDK ships with Xcode 7.2.

Image – Xamarin Forms Cross Platform App Development Environment on Mac OS X

** Xcode can be installed (and updated) via the Mac App Store.*

Testing and Debugging on OS X

Xamarin mobile applications can be deployed to physical devices via USB for testing and debugging (Xamarin.Mac apps can be tested directly on the development computer; Apple Watch apps are deployed first to the paired iPhone).

TESTING NOTES	
Xamarin.iOS	The easiest way to get started is using the iPhone, iPad, Apple Watch, and Apple TV simulators that are included with Xcode. To use a device for testing, follow these instructions http://developer.xamarin.com/guides/ios/getting_started/installation/device_provisioning/
Xamarin.Android	Follow these instructions to configure your device, or use an emulator: https://developer.xamarin.com/guides/android/getting_started/installation/set_up_device_for_development/ The Xamarin installer includes the Google Emulator Manager which lets you configure Google Android emulators for testing. https://developer.xamarin.com/guides/android/getting_started/installation/configure_emulator/ The Xamarin Android Player (Preview) can be downloaded and installed. https://developer.xamarin.com/guides/android/getting_started/installation/android-player/ Genymotion also provides a free-for-personal-use emulator for download.
Xamarin.Forms	Xamarin.Forms apps for iOS and Android can be deployed to the relevant platforms as described above.
Xamarin.Mac	Xamarin.Mac apps can be tested directly on the development computer.

Windows OS Requirements

Using a Windows computer for Xamarin development requires the following software/SDK versions. Check your operating system version (and for Visual Studio users, confirm that you are not using an Express version). If using Visual Studio, it should be installed first - the Visual Studio 2015 installer includes an option to install Xamarin automatically; otherwise follow the instructions for the Xamarin installer.

	RECOMMENDED	NOTES
Operating System	Windows 10	The minimum operating system version is Windows 7. Xamarin.Forms Windows support requires Windows 8.1, and Xamarin.Forms UWP support requires Windows 10.
Xamarin.iOS	iOS 9.2 SDK installed on a Mac	**To build iOS projects on Windows requires:** Visual Studio 2012 or newer, and a Mac computer, network-accessible from the Windows computer, that conforms to the minimum requirements for running Xamarin on OS X above.
Xamarin.Android	Android 6.0 / API level 23	You can still target older Android versions while using the latest SDK, or you can build against older versions of the SDK if required.
Xamarin.Forms	Xamarin.Forms apps for iOS and Android can be deployed to the relevant platforms as described above. Using Visual Studio also means you can test apps for Windows Phone and the Universal Windows Platform (on Windows 10) using Microsoft's emulators. Windows apps can be tested directly on the development computer.	
Xamarin.Mac	Xamarin.Mac development (OS X desktop apps) is NOT supported on Windows.	

** Xamarin for Visual Studio supports any Visual Studio 2015 (Community, Professional, and Enterprise) and non-Express editions of Visual Studio 2013 Update 2 and Visual Studio 2012. To develop Xamarin.Forms apps for the Universal Windows Platform (UWP) requires Windows 10 with Visual Studio 2015.*

Testing and Debugging on Windows

Xamarin mobile applications can be deployed to physical devices via USB for testing and debugging (iOS devices must be connected to the Mac computer, not the computer running Visual Studio).

TESTING NOTES	
Xamarin.iOS	The easiest way to get started is using the iPhone, iPad, Apple Watch, and Apple TV simulators that are included with Xcode. The simulators can be accessed on the connected Mac while debugging with Visual Studio. To use a device for testing, follow these instructions (performing most steps on the connected Mac computer). https://developer.xamarin.com/guides/ios/getting_started/installation/device_provisioning/
Xamarin.Android	Follow these instructions to configure your device, or use an emulator: https://developer.xamarin.com/guides/android/getting_started/installation/set_up_device_for_development/ The Xamarin installer includes the Google Emulator Manager which lets you configure Google Android emulators for testing. https://developer.xamarin.com/guides/android/getting_started/installation/configure_emulator/ The Xamarin Android Player (Preview) can be downloaded and installed. https://developer.xamarin.com/guides/android/getting_started/installation/android-player/ Other emulators are available from Genymotion and Microsoft (with Visual Studio 2015).

Xamarin.Forms	Xamarin.Forms apps can be deployed to the relevant devices and emulators as described above. The iOS app can only be tested via the connected Mac hardware; and the Windows tablet/ desktop apps for Windows 8.1 or UWP can be tested directly on the development computer.

** The Windows Phone 10 emulator is included with the Visual Studio 2015 UWP SDK. Windows Phone 8.1 emulator and Android emulator can be downloaded*

8 Conclusion

On a final note mobile technology enables new kinds of customer engagement, makes distributed workforces vastly more efficient, and further lubricates the flow of data that serves as the lifeblood of our information economy.

The importance of mobile technology will continue to increase as hardware improvements and declining costs propel us into a world where connectivity is present in everything around us. Companies that have adopted a Xamarin-based mobile strategy are winning, rapidly delivering quality, platform-specific experiences to customers, employees and partners—apps that take full advantage of the latest capabilities across a heterogeneous device landscape. Organizations that are still formulating their mobile strategy can rely on Xamarin to accelerate their ability to move from strategy to this level of mobile excellence.

With collaboration of Microsoft, Xamarin seems to standardize the mobile app development in C#. There are some highly appreciated features like – On average 75 percent source code sharing across multiple platforms, leveraging the existing skills of .NET C# developers and Stack of multiple tools and codebase to rapidly deliver great apps with broad reach.

Moreover, Xamarin is **used by over 450,000 developers** from **more than 100 Fortune 500 companies** and **over 20,000 paying customers** including Dow Jones, Bosch, McKesson, Halliburton, Cognizant, GitHub, Rdio and WebMD, to accelerate the creation of mission-critical consumer and enterprise apps.

That's why Xamarin.Forms is quoted as 'Write Once, Run everywhere and don't suck even with native features on Cross Platforms'.

References

https://developer.xamarin.com/guides

https://blog.xamarin.com/easy-app-theming-with-xamarin-forms-styles/

https://blog.xamarin.com/simplifying-events-with-commanding/

https://visualstudiomagazine.com/articles/2015/06/01/navigation-with-xamarin-forms.aspx

http://appindex.com/blog/ten-best-cross-platform-development-mobile-enterprises/

Authors

→ Hemanth Kumar Gaddale

→ Perumal Rajapandi

→ Rajkumar Ganesan

→ Santoskumar Surendar

→ Sunder Raman Kasi

→ Vishal Kumar Pandey