Formation

01
TOUR D’HORIZON

Aujourd'hui, la mobilité est le fait d'avoir accès à l'**information** de manière **continue** et sans rupture des **usages**.

Quelques chiffres

63

63 % des salariés sont en situation de mobilité au moins une fois par semaine

4/10

4 emails sur 10 sont ouverts depuis un Smartphone

52

52 % des utilisateurs utilisent 3 devices pour travailler

90

90 % des entreprises auront 2 ou 3 OS mobiles à gérer en 2017

80

+ 80% des employés utilisent des applications SaaS non-approuvées par l'entreprise

Les enjeux pour l’entreprise

Augmenter la productivité

- Application pensée pour l’utilisateur
- Augmentation de la réactivité et accélération des échanges
- Adéquation aux nouveaux modes de travail : nomadisme, valorisation, …

Aider à la prise de décision

Innover dans les usages

Contrôler l’explosion des usages de manière simple et rationnalisé

S’adapter au nouveau cycle de vie des applications

Les devices actuels

Phone

Phablet

Tablette

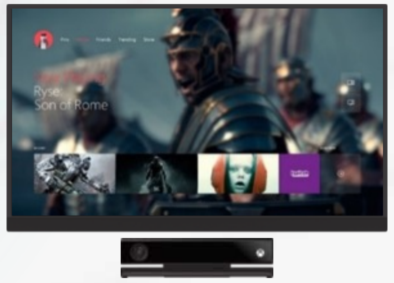
2-en-1s
(Tablette / Portable)

Portable
Classique

PC de
bureau

Mur tactile

TV



VR / AR

IoT

Wearable

Apple

Ecosystème le plus réduit :
- 26 devices
- Fragmentation des versions OS faible

Une version majeure par an de l’OS

Dernières nouveautés à fort impact :
- Stylet
- Multi-fenêtrage
- Téléphone de grande taille
- 3D Touch

Nécessite pour développer :
- Un mac
- Xcode (Une maj mineure par mois)
- ObjectifC ou Swift

Google

Ecosystème extrêmement vaste :

600+ nouveaux modèles de téléphones en 1 an

Fragmentation des version OS forte (4% avec pre-Android 4.1 et seulement 7% sur Android 6.0)

Form factor et qualité du matériel très variable

Parc utilisateurs : 1.5 Milliard d’activations

Microsoft

- Avec Windows 10, Microsoft refond son écosystème pour unifier tous les devices.
- Windows 10 déployé sur 300 millions de devices en un an. Objectif : 1 milliard en 2018
- OS tablette très utilisée en entreprise
- OS le plus utilisé en device hybride
- OS pensé pour l’entreprise et retro-compatible

Le nouveau Microsoft

Microsoft ouvre ces outils : open source et cross-platform.

L'objectif est de faire tomber les barrières techniques pour proposer ses services au plus grand nombre.

Office téléchargé 340 millions de fois sur iOS et Android

25% des machines Azure utilise Linux

Bash est intégré à Windows 10

SQL Server For Linux

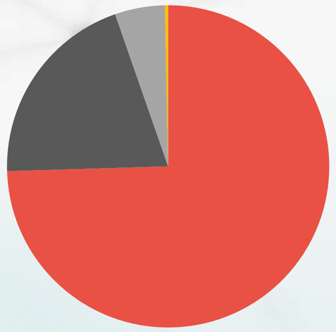
.Net devient Cross-platform

Visual Studio Code : Php, Python, Ruby, C++, Html5, Node.JS, …

Acquisition de Xamarin et mise à disposition du code source

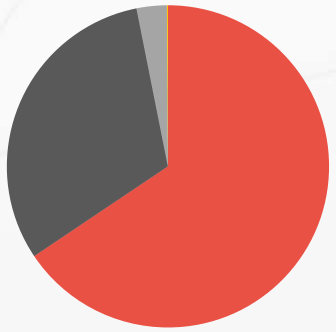
Part de marché (Mars 2016)

En France :



Android	74 %
iOS	20%
Windows	5 %
BlackBerry	0,3 %

Etats-Unis:



Android	65 %
iOS	31%
Windows	3 %
BlackBerry	0,1 %

- L'usage du mobile a dépassé celui du PC en 2014
- Part de marché en entreprise très difficilement mesurable

Les nouveautés techniques

- Toujours connecté
- Biométrie
- BlueTooth LE
- Autonomie
- Résistance
- VR et AR
- Machine Learning
- Bot et AI

02

PRESENTATION DES TECHNOLOGIES DE CROSS-PLATEFORM

‘Write once, Run anywhere’

Les applications ‘Cross-platform’ sont des applications ayant un rendu et des fonctionnalités identiques à celles développées nativement, mais qui on été écrites avec un seul et même code source.

Pourquoi utiliser le cross platform?

Réduire le temps

Développement
Maintenance
Evolution

Fiabiliser l’applicatif

Un seul code métier plutôt que 3

Rationnaliser l’équipe

Réutiliser le savoir faire de ces équipes (C# ou JS)

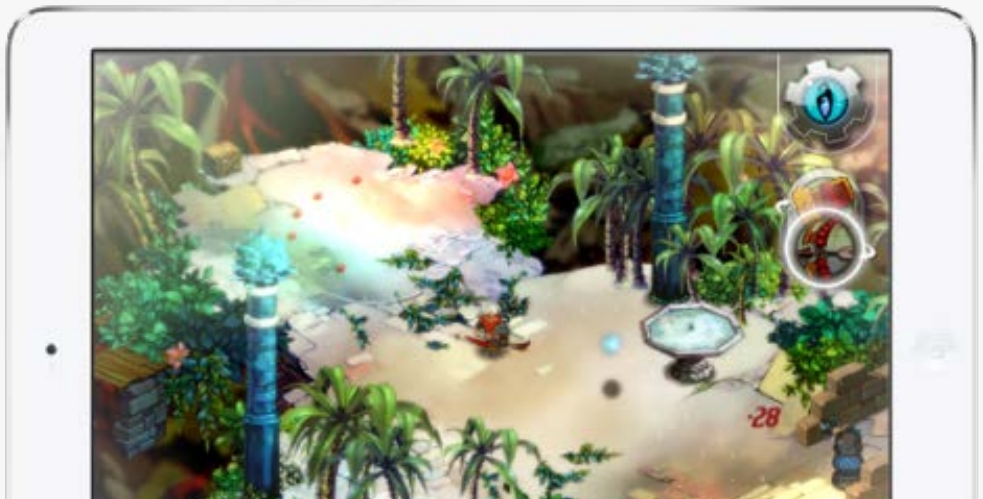
Equipe unique

C# 6.0
IN A NUTSHELL
O'REILLY

Xamarin ?

La possibilité de développer sur toutes les plateformes mobiles majeures :

- l'ergonomie et l'expérience de chaque plateforme
- les performances natives de chaque plateforme
- partager du code entre les plateformes
- C# & .Net Framework



Xamarin, l'historique

2000	2001	2003	2009	2011	2012	2013	2014	2016
Ximian Founded	Mono Started	Ximian Acquired by Novell	First iOS product (now Xamarin.iOS) launches	Xamarin Founded First Release of Xamarin.Android	First release of Xamarin.Mac Launch Partner Program	Xamarin 2.0 Component Store Evolve 2013 Xamarin Test Cloud Microsoft Partnership	Xamarin.Forms Xamarin.Insight Xamarin University	Microsoft Open source

Le concept

Xamarin utilise C#/.Net pour contrôler les objets natifs des plateformes

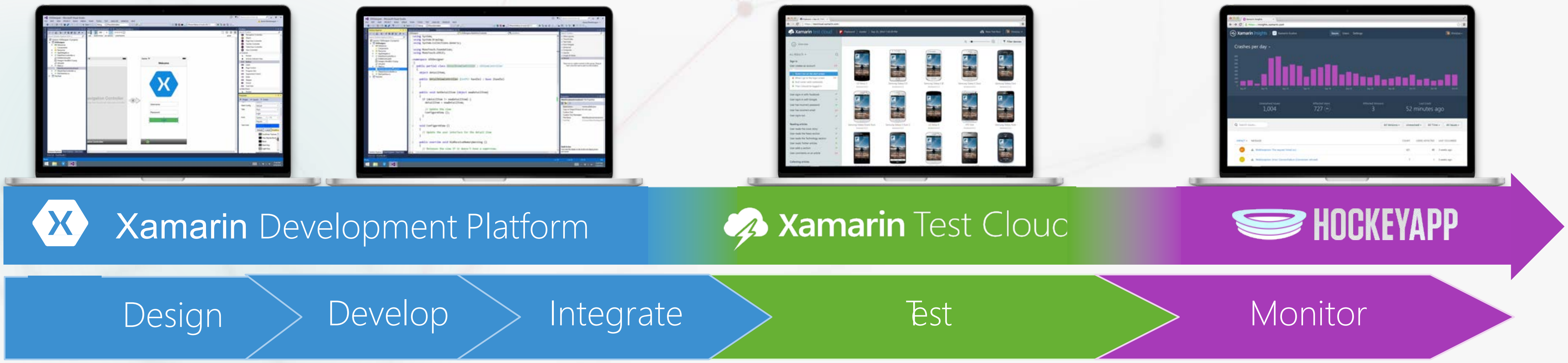
La couverture de l'API iOS, Android et Windows est de 100%

L'IHM rendue est native

Xamarin

Xamarin Platform

- Xamarin platform lets you develop, test, and monitor your released application for iOS, Android, and Windows



Building mobile apps

- There are three common ways to create mobile applications, each with specific strengths and weaknesses

Silo

Black Box

Xamarin

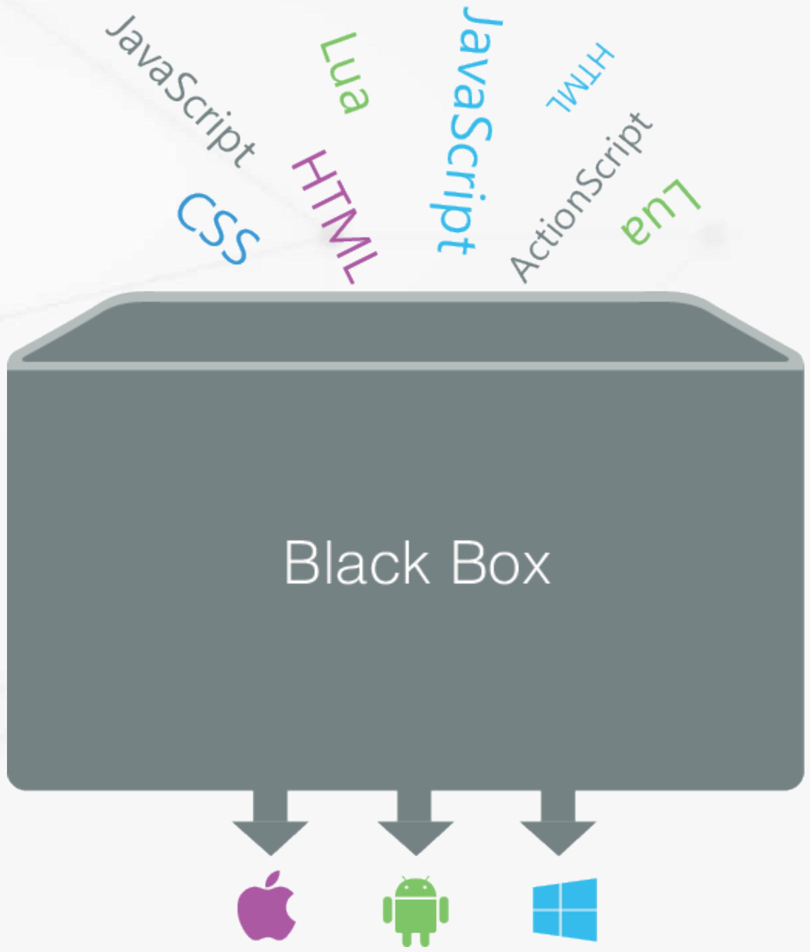
Silo approach

- Can write the same application multiple times using the vendor's tools

iOS App	Android App	Windows App
Obj-C Swift XCode	Java Eclipse A. Studio	C# Visual Studio

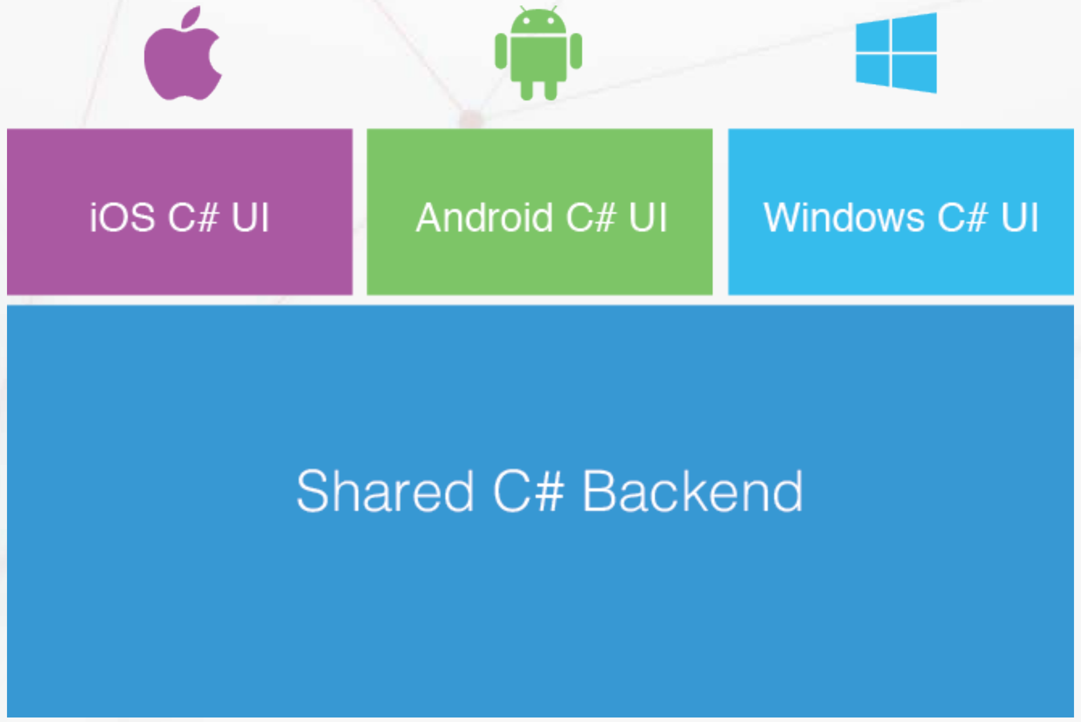
Black Box approach

- Can use high-level tools that convert a single code base to an app for each platform



Xamarin approach

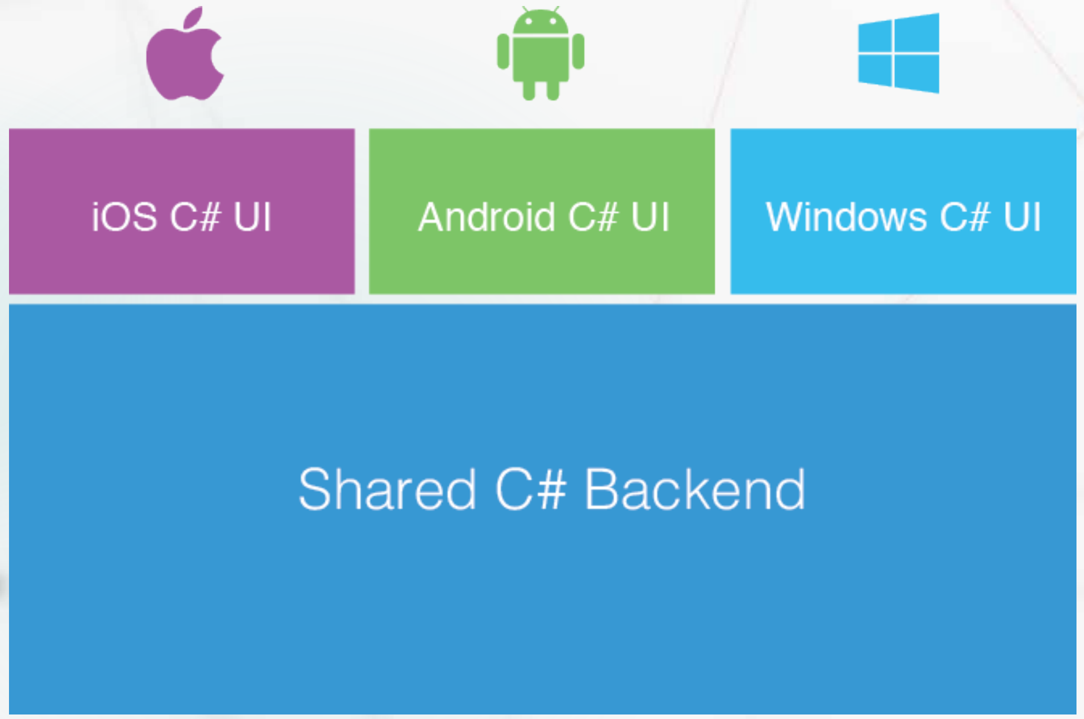
- Can build native apps using C# and .NET, sharing the business logic but leveraging each platform's benefits and paradigms



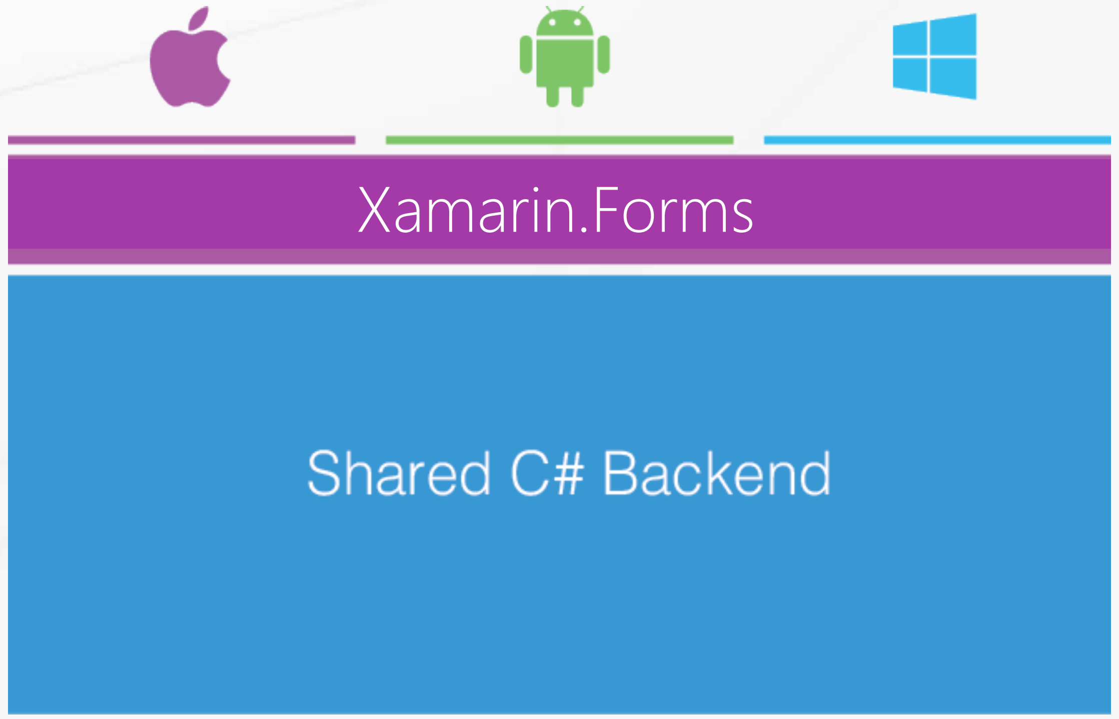
Traditional Xamarin approach

Xamarin approach

- Xamarin.Forms enables even more code-sharing through a shared UI definition when deep platform integration is unnecessary



Traditional Xamarin approach

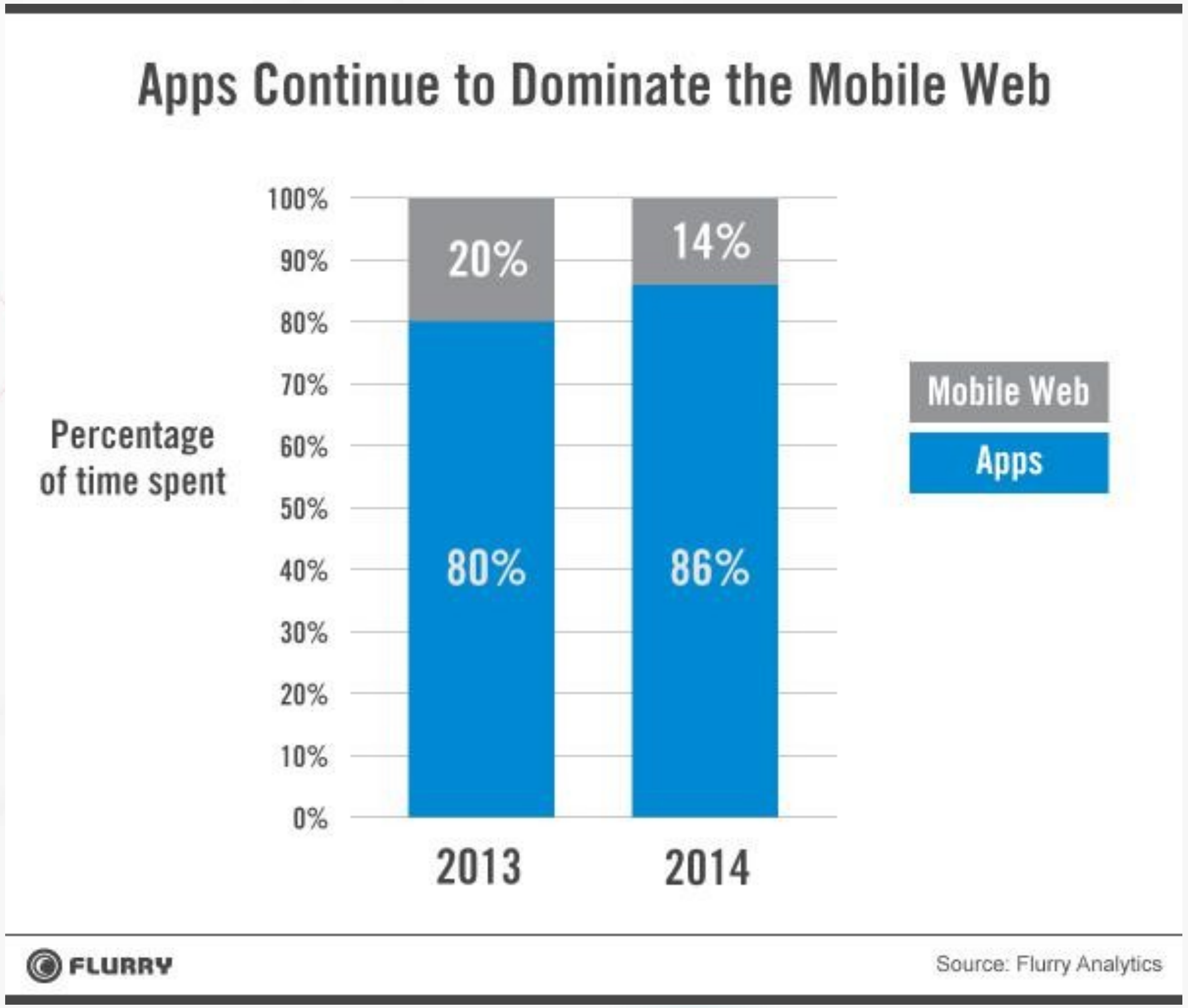


Xamarin.Forms approach

Why Native apps?

- Native apps can integrate better with the platform, are faster, more power efficient, and visually look better

- Plus, users *like* them better!



Anything you can do in Objective-C, Swift, or Java can be done in C# (or F#) with Xamarin

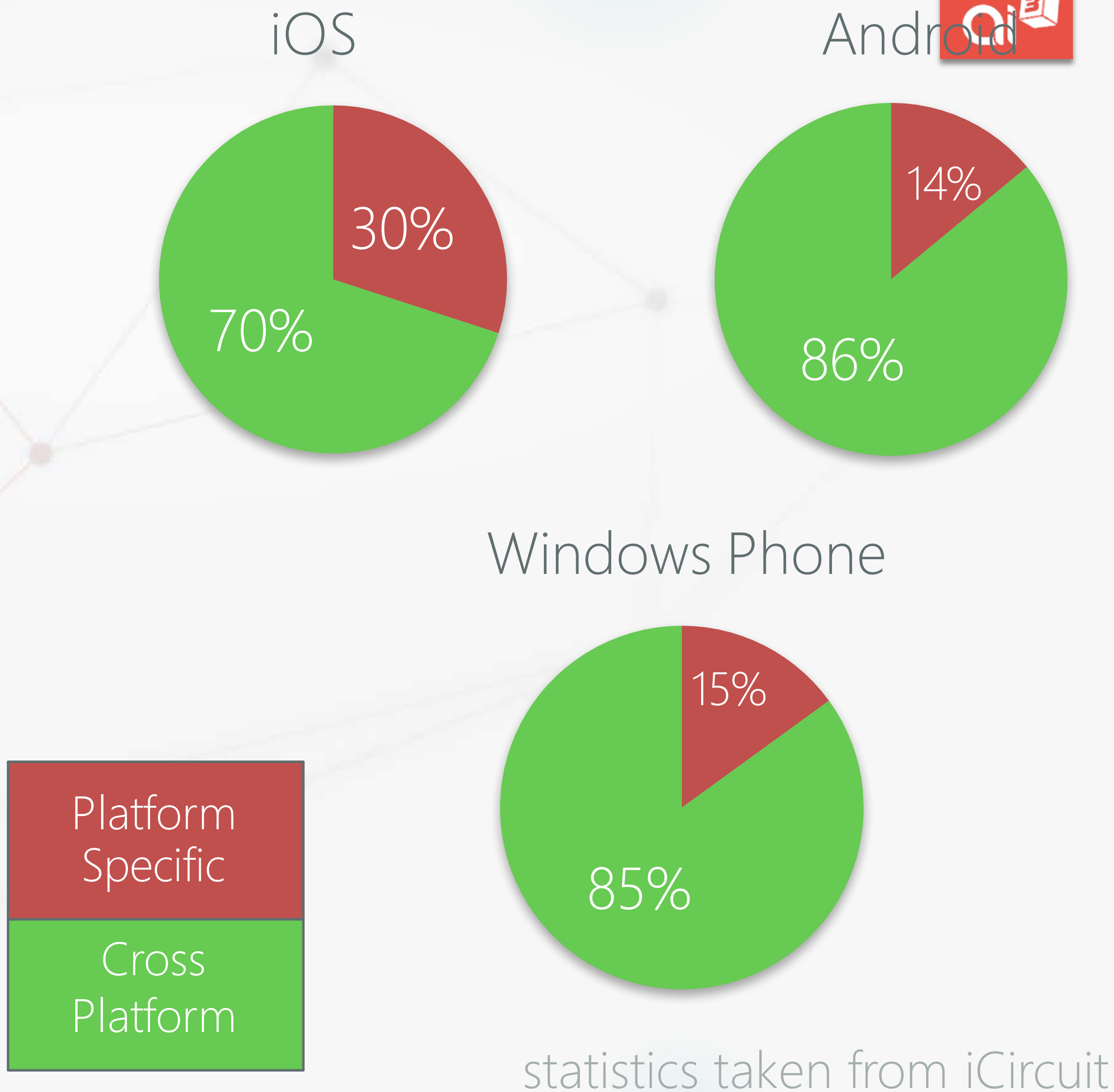
Re-use

Sharing code

v One of the main reasons to use Xamarin is the possibility of sharing a significant portion of your code across all your supported platforms

Sharable Code

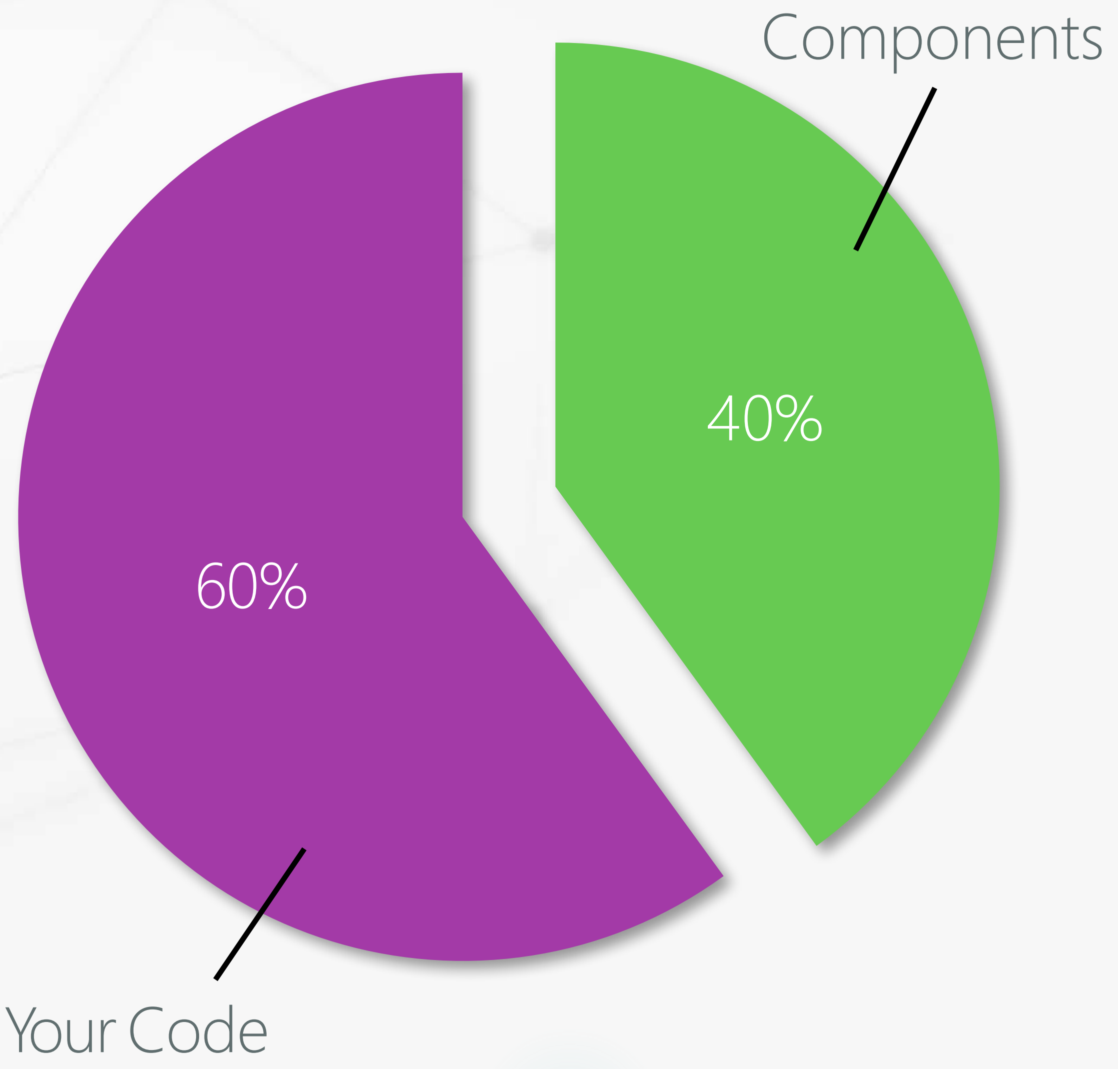
- Xamarin applications are native and therefore will *always* include some platform-specific code



statistics taken from iCircuit

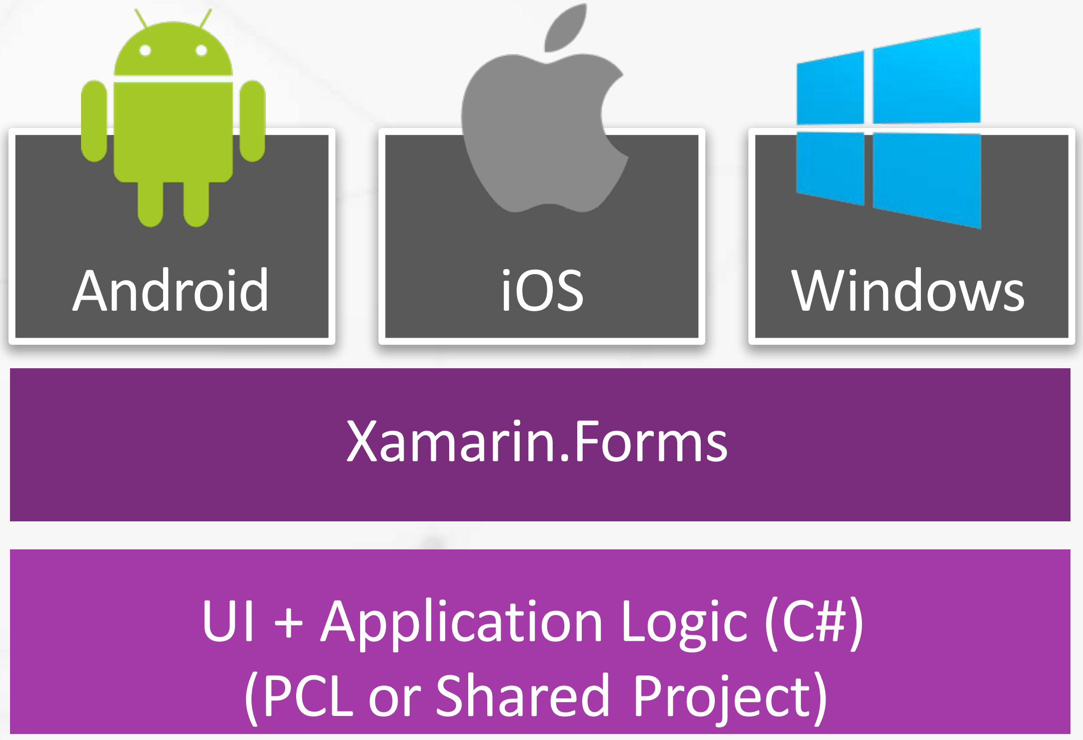
Sharable code

- Sharable code is split between reusable components and platform-independent code



Xamarin.Forms

- Xamarin.Forms provides shared set of UI controls to design the user interface
- Renders native UI on iOS, Android and Windows



Data Access (Database)

- SQLite support available for iOS, Android and Windows
- Can also store in the cloud – Azure Mobile Services, Amazon, Dropbox, etc.

Web Services

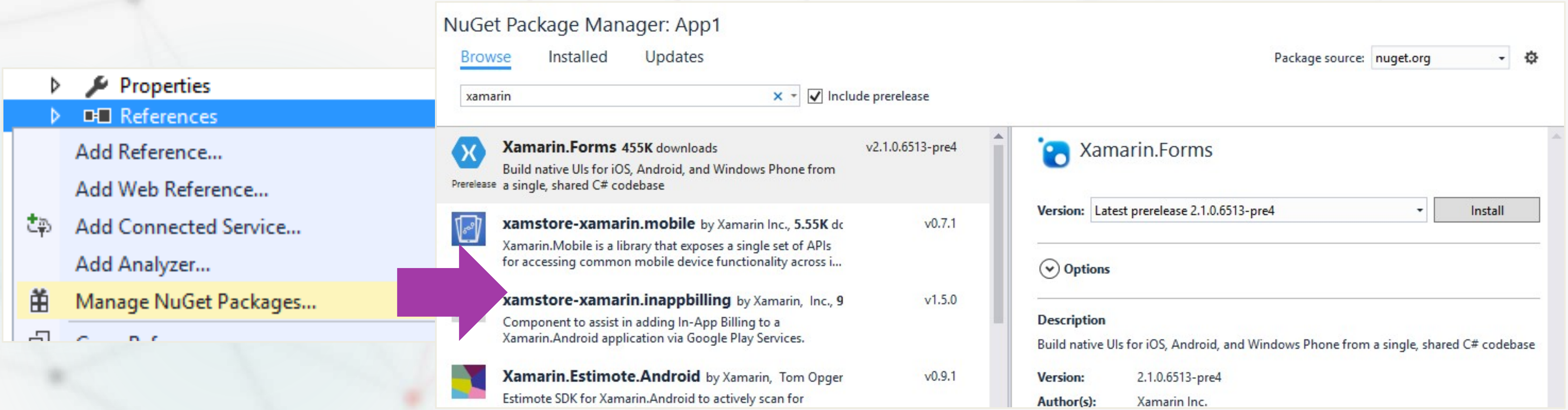
- Use **`HttpClient`** for REST services, can then process with
  - **`System.Xml`** / **`System.Json`**
  - LINQ to XML
  - Json.NET component
- Use WCF or **`.asmx`** for SOAP

Xamarin.* Libraries

- Open-Source, Cross-Platform APIs available from Github.com/Xamarin
  - Xamarin.Social
  - Xamarin.Auth
  - Xamarin.Mobile
- Check out .NET Foundation for more great open source libraries like MailKit and Rx

Using Nuget in Visual Studio

- Can add Nuget components in Visual Studio using References folder



NuGet Package Manager: App1

Browse Installed Updates

Package source: nuget.org

xamarin ✓ Include prerelease

Xamarin.Forms 455K downloads v2.1.0.6513-pre4
Prerelease Build native UIs for iOS, Android, and Windows Phone from a single, shared C# codebase

xamstore-xamarin.mobile by Xamarin Inc., 5.55K dc v0.7.1
Xamarin.Mobile is a library that exposes a single set of APIs for accessing common mobile device functionality across i...

xamstore-xamarin.inappbilling by Xamarin, Inc., 9 v1.5.0
Component to assist in adding In-App Billing to a Xamarin.Android application via Google Play Services.

Xamarin.Estimote.Android by Xamarin, Tom Opger v0.9.1
Estimote SDK for Xamarin.Android to actively scan for

Xamarin.Forms

Version: Latest prerelease 2.1.0.6513-pre4 Install

Options

Description
Build native UIs for iOS, Android, and Windows Phone from a single, shared C# codebase

Version: 2.1.0.6513-pre4
Author(s): Xamarin Inc.

Can search, update components and even revert to older revisions

Where can I use shared code?

- Anytime you are writing code which does not depend on a specific platform feature, it is potentially sharable, particularly if it:

Talks to a web service

Parses a data format

Uses a database

Performs processing or logic

Create shared classes + methods and then use them from your platform-specific code to maximize the shareable surface area

When is code *not* sharable?

- If the code you are writing depends on device or platform-specific APIs, or APIs not available in your project, then you will need to isolate it's use or provide some kind of *abstraction* to use it from your shared code

Access system information

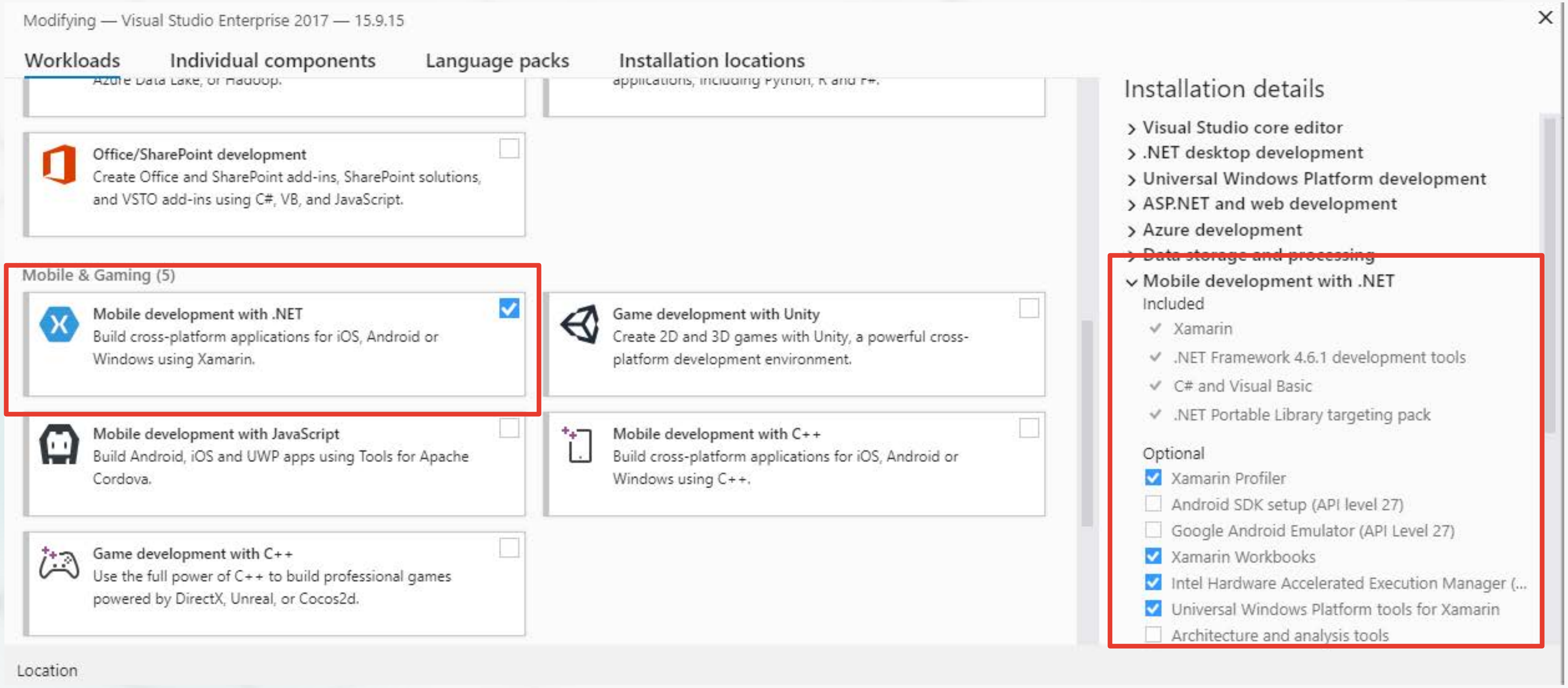
Use files and folders on the device

Access personal information

Use external devices

Préparer votre environnement

Installation



Modifying — Visual Studio Enterprise 2017 — 15.9.15

Office/SharePoint development
Create Office and SharePoint add-ins, SharePoint solutions, and VSTO add-ins using C#, VB, and JavaScript.

Mobile & Gaming (5)

[✓] Mobile development with .NET
Build cross-platform applications for iOS, Android or Windows using Xamarin.

[ ] Game development with Unity
Create 2D and 3D games with Unity, a powerful cross-platform development environment.

[ ] Mobile development with JavaScript
Build Android, iOS and UWP apps using Tools for Apache Cordova.

[ ] Mobile development with C++
Build cross-platform applications for iOS, Android or Windows using C++.

[ ] Game development with C++
Use the full power of C++ to build professional games powered by DirectX, Unreal, or Cocos2d.

Installation details

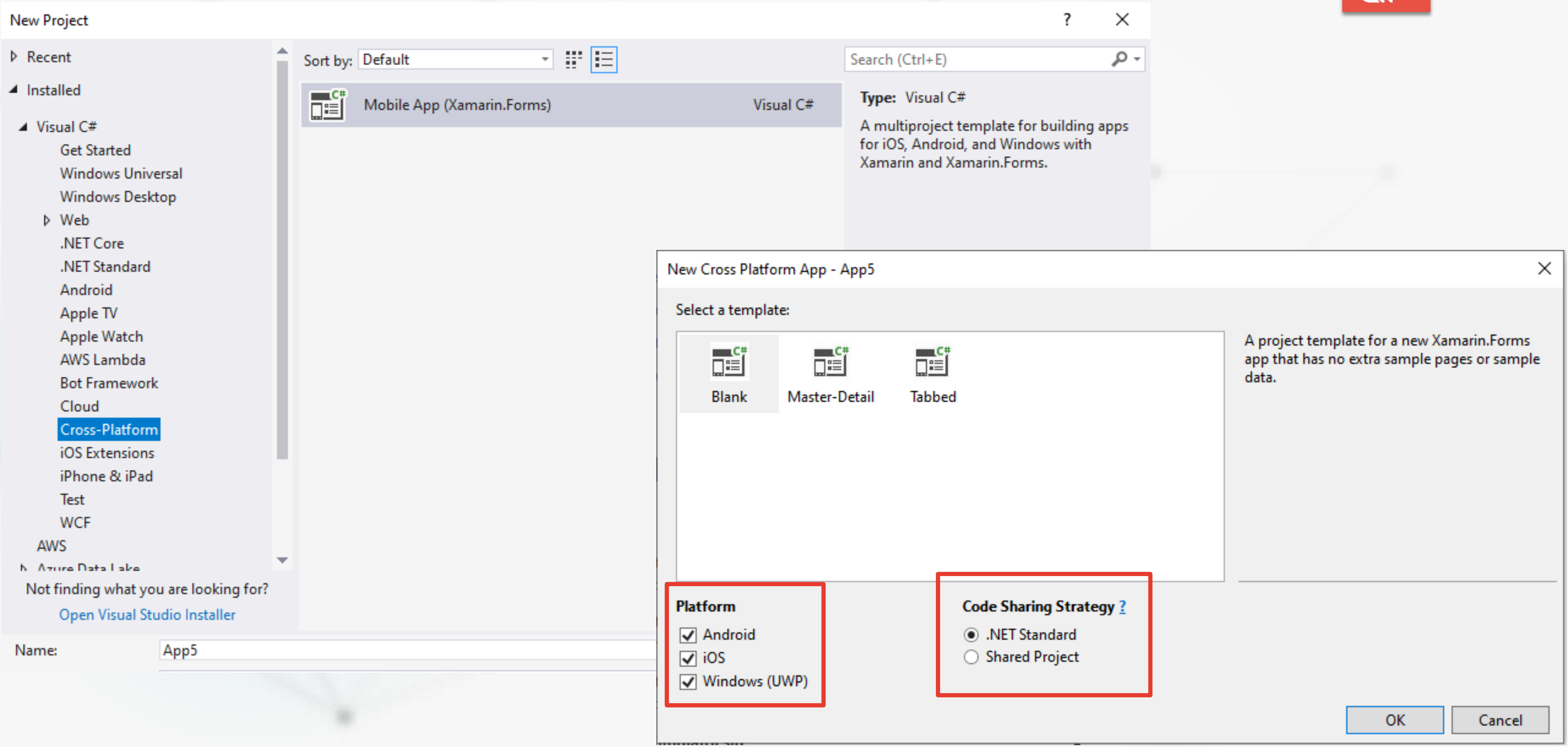
- Visual Studio core editor
- .NET desktop development
- Universal Windows Platform development
- ASP.NET and web development
- Azure development
- Data storage and processing
- Mobile development with .NET

Included
- ✓ Xamarin
- ✓ .NET Framework 4.6.1 development tools
- ✓ C# and Visual Basic
- ✓ .NET Portable Library targeting pack

Optional
- [✓] Xamarin Profiler
- [ ] Android SDK setup (API level 27)
- [ ] Google Android Emulator (API Level 27)
- [✓] Xamarin Workbooks
- [✓] Intel Hardware Accelerated Execution Manager (...
- [✓] Universal Windows Platform tools for Xamarin
- [ ] Architecture and analysis tools

Location

Création d'un nouveau projet



New Project

- Recent
- Installed
  - Visual C#
    - Get Started
    - Windows Universal
    - Windows Desktop
    - Web
    - .NET Core
    - .NET Standard
    - Android
    - Apple TV
    - Apple Watch
    - AWS Lambda
    - Bot Framework
    - Cloud
    - Cross-Platform
    - iOS Extensions
    - iPhone & iPad
    - Test
    - WCF
  - AWS
  - Azure Data Lake

Not finding what you are looking for?
Open Visual Studio Installer

Sort by: Default

Search (Ctrl+E)

Mobile App (Xamarin.Forms) Visual C#

Type: Visual C#

A multiproject template for building apps for iOS, Android, and Windows with Xamarin and Xamarin.Forms.

Name: App5

New Cross Platform App - App5

Select a template:

Blank
Master-Detail
Tabbed

A project template for a new Xamarin.Forms app that has no extra sample pages or sample data.

Platform

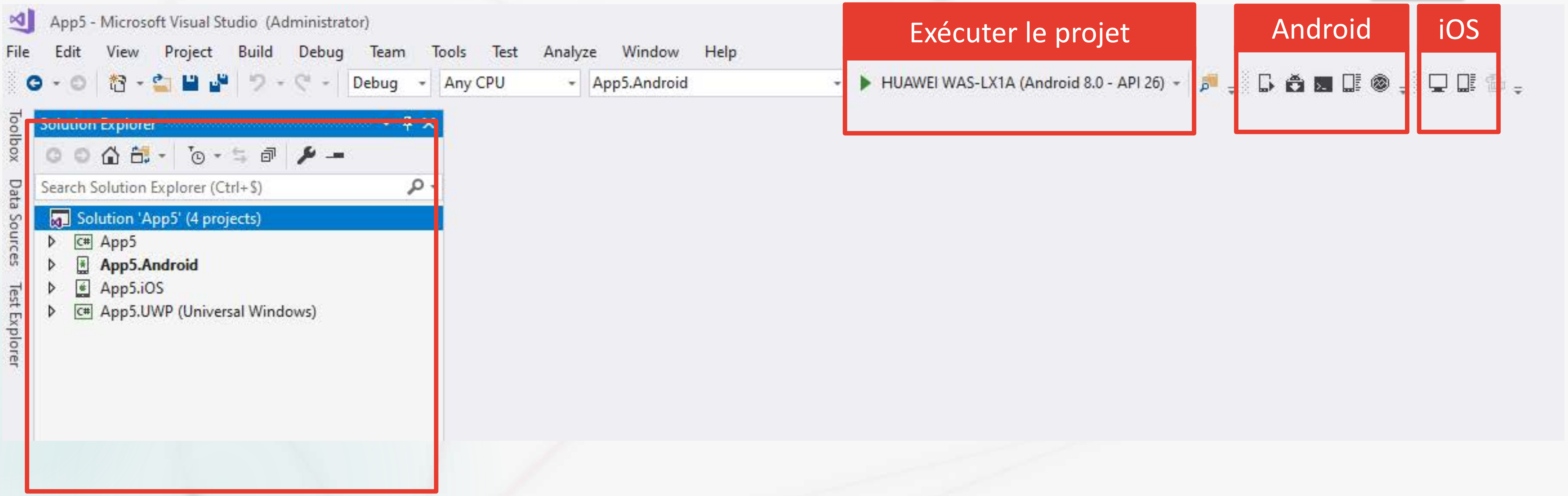
- [✓] Android
- [✓] iOS
- [✓] Windows (UWP)

Code Sharing Strategy ?

- [✓ .NET Standard]
- Shared Project

OK
Cancel

Projet Xamarin



App5 - Microsoft Visual Studio (Administrator)

File Edit View Project Build Debug Team Tools Test Analyze Window Help

Debug | Any CPU | App5.Android | HUAWEI WAS-LX1A (Android 8.0 - API 26)

Solution Explorer

Search Solution Explorer (Ctrl+$)

- Solution 'App5' (4 projects)
  - App5
  - **App5.Android**
  - App5.iOS
  - App5.UWP (Universal Windows)

Application – Point d'entrée

```
public partial class App : Application
{
    public App()
    {
        InitializeComponent();
        MainPage = new NavigationPage(page);
    }

    protected override void OnStart()
    {}

    protected override void OnSleep()
    {}

    protected override void OnResume()
    {}
}
```

Atelier

- Créer un nouveau projet Xamarin
- Exécuter votre projet

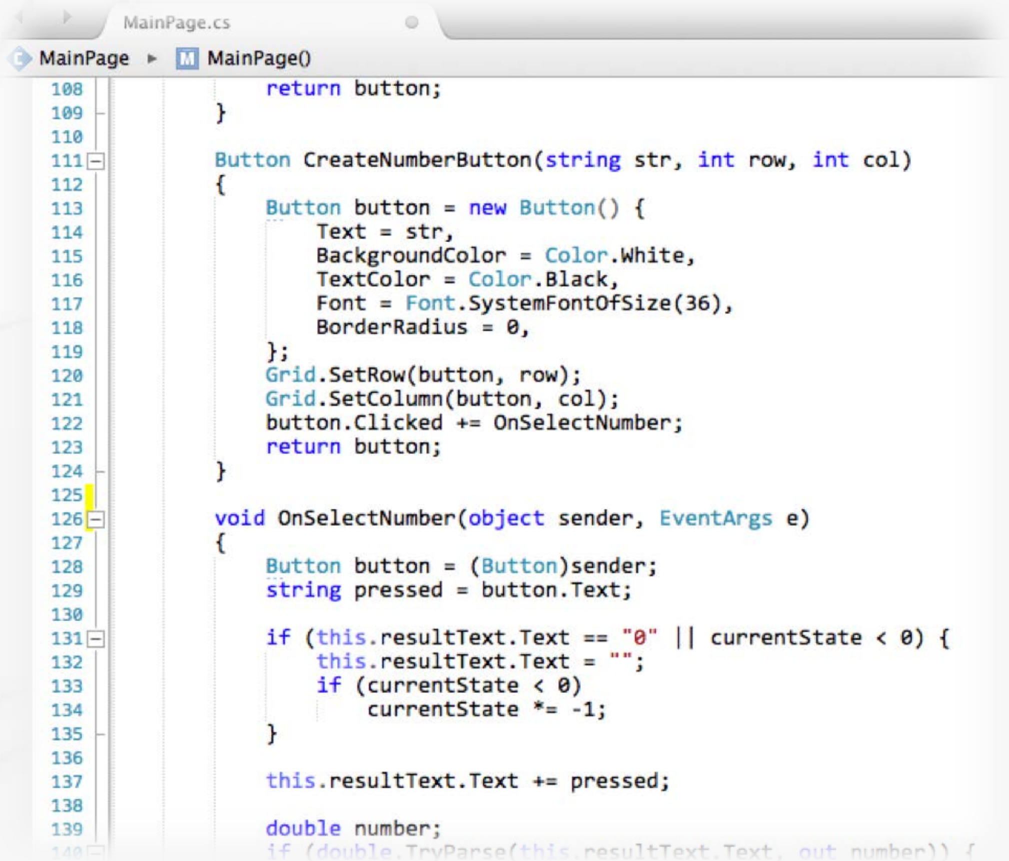
Xamarin XAML

Creating Pages in Code

- Significant portion of code behind tends to be in UI creation: setup and layout

- Mixing of UI and behavior in one file can make both design and behavior harder to understand / evolve

- Prohibits designer role involvement
  – developer is forced to do everything



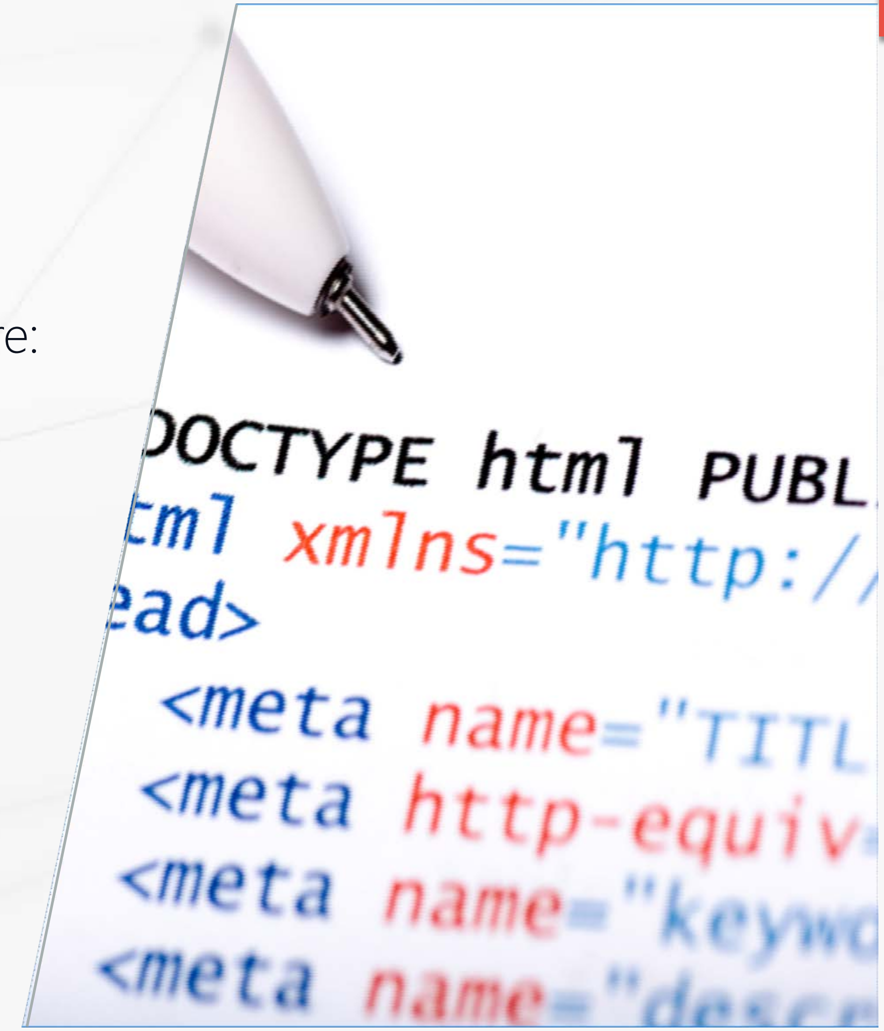
MainPage.cs

MainPage ▸ MainPage()

```
            return button;
        }

        Button CreateNumberButton(string str, int row, int col)
        {
            Button button = new Button() {
                Text = str,
                BackgroundColor = Color.White,
                TextColor = Color.Black,
                Font = Font.SystemFontOfSize(36),
                BorderRadius = 0,
            };
            Grid.SetRow(button, row);
            Grid.SetColumn(button, col);
            button.Clicked += OnSelectNumber;
            return button;
        }

        void OnSelectNumber(object sender, EventArgs e)
        {
            Button button = (Button)sender;
            string pressed = button.Text;

            if (this.resultText.Text == "0" || currentState < 0) {
                this.resultText.Text = "";
                if (currentState < 0)
                    currentState *= -1;
            }

            this.resultText.Text += pressed;

            double number;
            if (double.TryParse(this.resultText.Text, out number)) {
```

Working in Markup

- HTML has taught us that markup languages are a great way to define user interfaces because they are:
  - Toolable
  - Human readable
  - Extensible



Extensible Application Markup Language

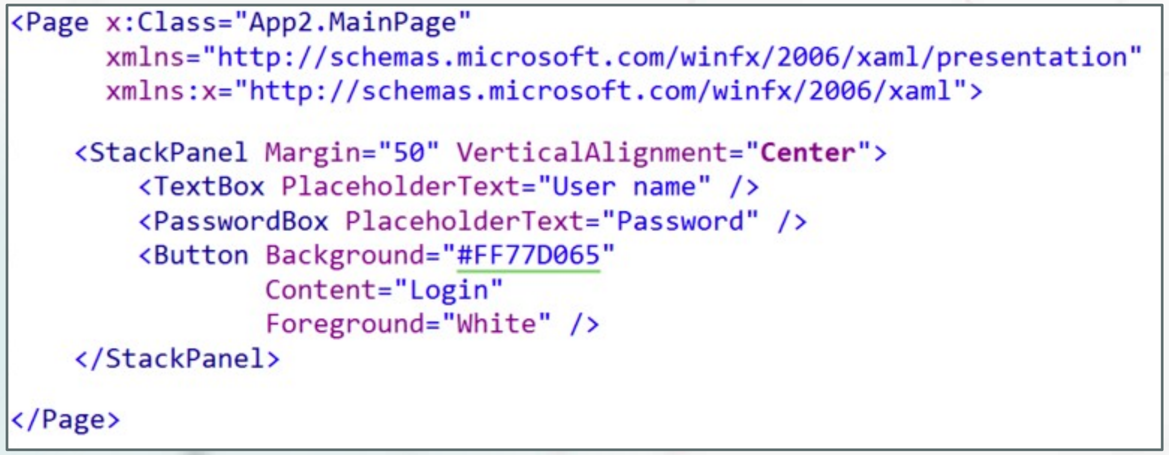
- XAML was created by Microsoft specifically to describe UI

XAML

Xamarin Forms + XAML
= Sweetness!

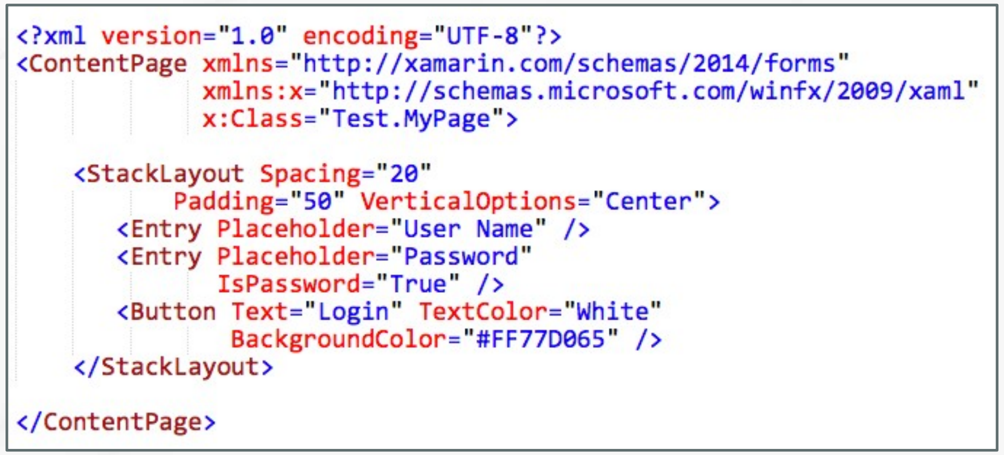
Microsoft XAML vs. Xamarin.Forms

Xamarin.Forms conforms to the XAML 2009 specification;
the differences are really in the controls and layout containers you use



```xml
<Page x:Class="App2.MainPage"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

    <StackPanel Margin="50" VerticalAlignment="Center">
        <TextBox PlaceholderText="User name" />
        <PasswordBox PlaceholderText="Password" />
        <Button Background="#FF77D065"
                Content="Login"
                Foreground="White" />
    </StackPanel>

</Page>
```

Microsoft XAML (WinRT)

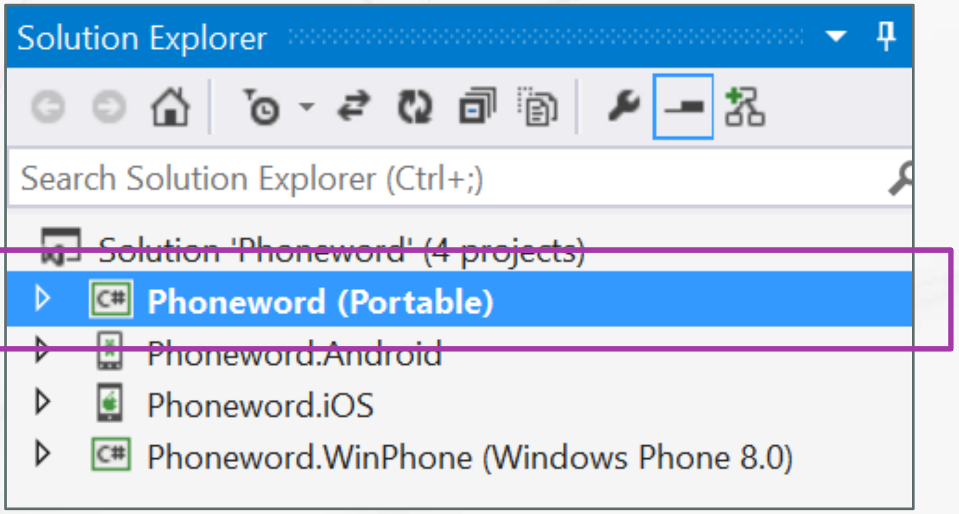


```xml
<?xml version="1.0" encoding="UTF-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Test.MyPage">

    <StackLayout Spacing="20"
          Padding="50" VerticalOptions="Center">
      <Entry Placeholder="User Name" />
      <Entry Placeholder="Password"
             IsPassword="True" />
      <Button Text="Login" TextColor="White"
              BackgroundColor="#FF77D065" />
    </StackLayout>

</ContentPage>
```

Xamarin.Forms

Where do the XAML pages go?

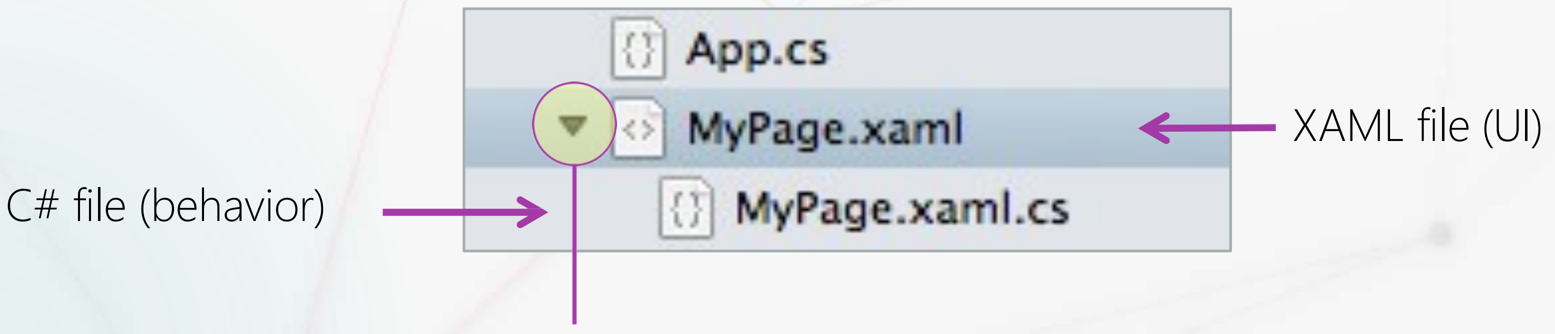
You should always add the XAML content to the *platform-independent* part of your application – this is shared UI and code for all your target platforms



Solution 'Phoneword' (4 projects)
Phoneword (Portable)
Phoneword.Android
Phoneword.iOS
Phoneword.WinPhone (Windows Phone 8.0)

What gets created?

XAML pages have two related files which work together to define the class



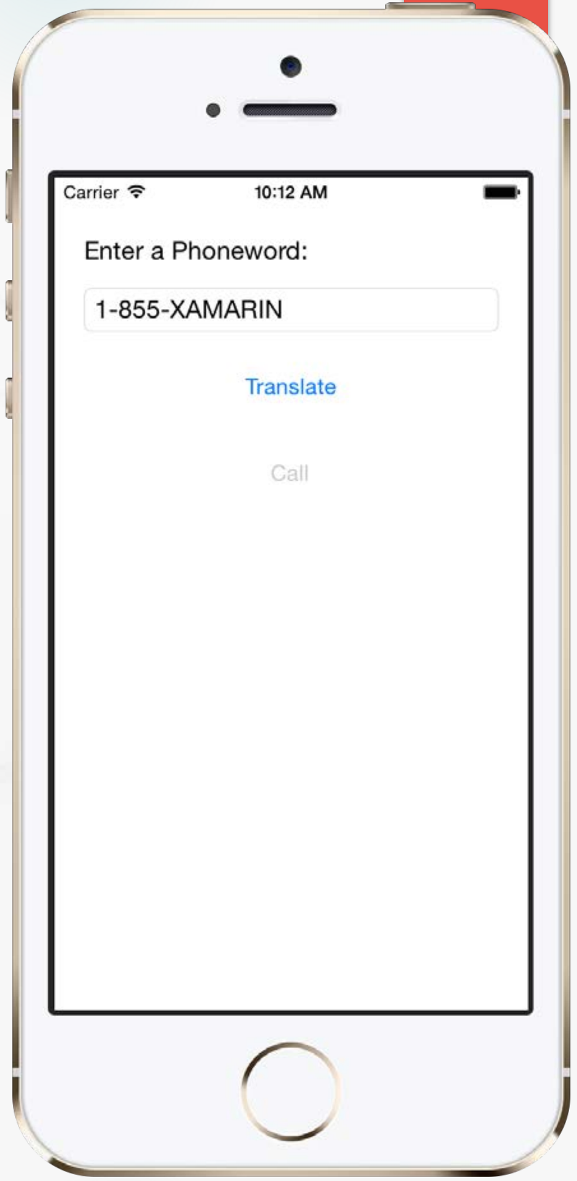
Disclosure arrow *collapses* the C# file and indicates these files go together

Describing a screen in XAML

XAML is used to construct object graphs, in this case a visual **Page**

```
<?xml version="1.0"  encoding="UTF-8" ?>
<ContentPage ...>
   <StackLayout  Padding="20" Spacing="10">
      <Label Text="Enter a Phoneword:"/>
      <Entry Placeholder="Number"  />
      <Button Text="Translate"  />
      <Button Text="Call"  IsEnabled="False"  />
   </StackLayout>
</ContentPage>
```

XML based: case sensitive, open tags must be closed, etc.



Describing a screen in XAML

Element tags create objects

```
<?xml version="1.0" encoding="UTF-8" ?>
<ContentPage ... >
   <StackLayout Padding="20" Spacing="10" >
      <Label Text="Enter a Phoneword:" />
      <Entry Placeholder="Number" />
      <Button Text="Translate" />
      <Button Text="Call" IsEnabled="False" />
   </StackLayout>
</ContentPage>
```

Describing a screen in XAML

```
<?xml version="1.0" encoding="UTF-8" ?>
<ContentPage ...>
   <StackLayout Padding="20" Spacing="10">
      <Label Text="Enter a Phoneword:" />
      <Entry Placeholder="Number" />
      <Button Text="Translate" />
      <Button Text="Call" IsEnabled="False" />
   </StackLayout>
</ContentPage>
```

Attributes set properties or events

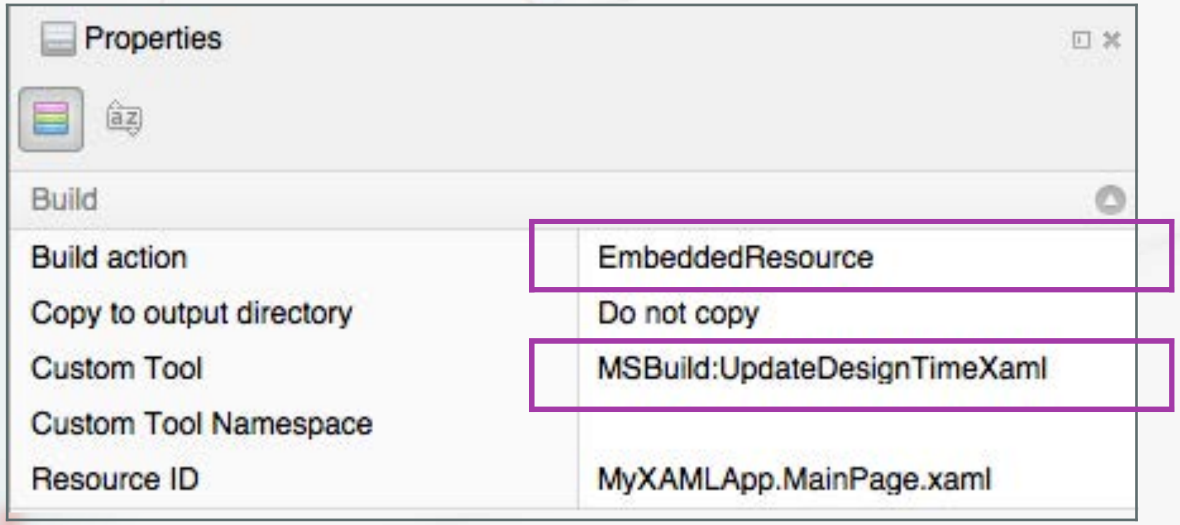
Describing a screen in XAML

```
<?xml version="1.0" encoding="UTF-8" ?>
<ContentPage ... >
   <StackLayout Padding="20" Spacing="10" >
      <Label Text="Enter a Phoneword:" />
      <Entry Placeholder="Number" />
      <Button Text="Translate" />
      <Button Text="Call" IsEnabled="False" />
   </StackLayout>
</ContentPage>
```

Child nodes used to establish relationship

XAML build type

- XAML files are stored as *embedded resources* and have a special build type of `MSBuild:UpdateDesignTimeXaml`



Properties

Build

Build action	EmbeddedResource
Copy to output directory	Do not copy
Custom Tool	MSBuild:UpdateDesignTimeXaml
Custom Tool Namespace	
Resource ID	MyXAMLApp.MainPage.xaml

XAML + Code Behind

- XAML and code behind files are tied together

```
<?xml version="1.0" encoding="UTF-8" ?>
<ContentPage x:Class="Phoneword.MainPage" ...
                                           >
```

x:Class identifies the full name of the class defined in the code behind file

```
namespace Phoneword
{
   public partial class MainPage : ContentPage
   {
      ...
   }
}
```

XAML initialization

- Code behind constructor has call to **`InitializeComponent`** which is responsible for loading the XAML and creating the objects

```
public partial class MainPage : ContentPage
{
    public MainPage ()
    {
        InitializeComponent ();
    }
}
```

implementation of method generated by XAML compiler as a result of the **`x:Class`** tag – added to hidden file (same partial class)

Property Conversions

- XML attributes only allow for string values – works fine for intrinsic types

```
<Label Text="This is a Label" IsVisible="True" Opacity="0.75"
   FontAttributes="Bold,Italic" FontSize="Large"
   Margin="5,20,5,0" TextColor="#fffc0d34" />
```

`Text` is a `string` which is just set directly

Property Conversions

- XML attributes only allow for string values – works fine for intrinsic types

```
<Label Text="This is a Label" IsVisible="True" Opacity="0.75"
   FontAttributes="Bold,Italic" FontSize="Large"
   Margin="5,20,5,0" TextColor="#fffc0d34" />
```

`IsVisible` is a **`bool`** which is converted from the value using `Boolean.TryParse`

Property Conversions

- XML attributes only allow for string values – works fine for intrinsic types

```
<Label Text="This is a Label" IsVisible="True" Opacity="0.75"
   FontAttributes="Bold,Italic" FontSize="Large"
   Margin="5,20,5,0" TextColor="#fffc0d34" />
```

`Opacity` is a `double` which is converted from the value using `Double.TryParse`

Property Conversions

- XML attributes only allow for string values – works fine for intrinsic types

```
<Label Text="This is a Label" IsVisible="True" Opacity="0.75"
   FontAttributes="Bold,Italic" FontSize="Large"
   Margin="5,20,5,0" TextColor="#fffc0d34" />
```

Enumerations are parsed with **`Enum.TryParse`** and support **`[Flags]`** with comma-separated values

Property Conversions

- XML attributes only allow for string values – works fine for intrinsic types

```
<Label Text="This is a Label" IsVisible="True" Opacity="0.75"
   FontAttributes="Bold,Italic" FontSize="Large"
   Margin="5,20,5,0" TextColor="#fffc0d34" />
```

`Margin` is a `Thickness` object, you can specify as a single number, two numbers, or four numbers (L,T,R,B)

Property Conversions

- XML attributes only allow for string values – works fine for intrinsic types

```
<Label Text="This is a Label" IsVisible="True" Opacity="0.75"
   FontAttributes="Bold,Italic" FontSize="Large"
   Margin="5,20,5,0" TextColor="#fffc0d34" />
```

Colors can be specified as a known value (e.g. **"Red"**, **"Green"**, ...) or as a hex value (RGB or aRGB)

Setting Complex Properties

- When a more complex object needs to be created and assigned, you can use the *Property Element* syntax

- This changes the style to use an element tag (create-an-object) as part of the assignment

```
<BoxView Color="Transparent">
   <BoxView.GestureRecognizers>
      <TapGestureRecognizer
        NumberOfTapsRequired="2"
        ... />
   </BoxView.GestureRecognizers>
</BoxView>
```

Property value is set as a child tag of the **`<Type.PropertyName>`** element

Setting Attached Properties

v Attached Properties provide runtime "attached" data for a visual element

v Used by layout containers to provide container-specific values on each child

```
<Grid>
 <Label Text="Position" />
 <Entry Grid.Column="1" />
</Grid>
```

Set in XAML with **`OwnerType.Property="Value"`** form, can also use property-element syntax for more complexvalues

Content Properties

- Some types have a *default* property which is set when child content is added to the element

- This is the *Content Property* and is identified through a `[ContentAttribute]` applied to the class

```
<ContentPage  ...>
 <Label>
    This is the Text
 </Label>
</ContentPage>
```

These create the same UI

```
<ContentPage     >
 <ContentPage.Content>
   <Label>
     <Label.Text>
       This is the Text
     </Label.Text>
   </Label>
 </ContentPage.Content>
</ContentPage>
```

Identifying Types

- XAML creates objects when it encounters an element tag, XML namespaces are used to correlate .NET types to tags

Default namespace includes most of the Xamarin.Forms types you use

```
<ContentPage ...
    xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml" >

   <StackLayout   ... />

<ContentPage>
```

`x:` namespace includes XAML types and known CLR types (**`Int32`**, **`String`**, etc.)

Custom Types

- XAML can create any public object, including ones with parameterized constructors – you just need to tell it where the type lives

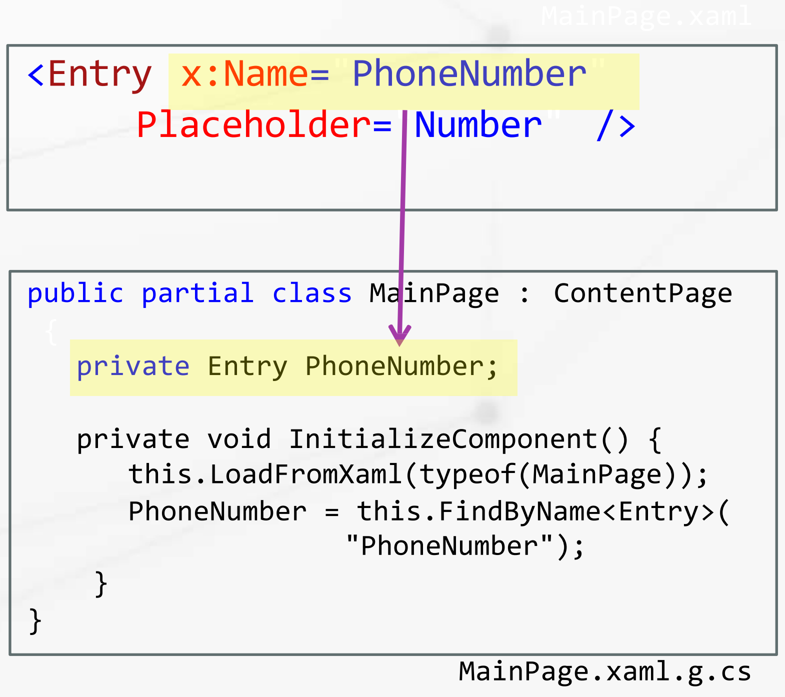
Must supply the namespace, and *possibly* the assembly, the type is defined in

```
<scg:List x:TypeArguments="x:String"
 xmlns:scg="clr-namespace:System.Collections.Generic;assembly=mscorlib">
   <x:String>One</x:String>
   <x:String>Two</x:String>
   <x:String>Three</x:String>
</scg:List>
```

`xmlns` definition can be placed on a single element, or a parent element to use with any children

Naming Elements in XAML

- Use `x:Name` to assign field name
  - allows you to reference element in XAML and code behind
- Adds a private field to the XAML-generated partial class (.g.cs)
- Name must conform to C# naming conventions and be unique in the file



MainPage.xaml

```
<Entry x:Name="PhoneNumber"
       Placeholder="Number"  />
```

```
public partial class MainPage : ContentPage
{
   private Entry PhoneNumber;

   private void InitializeComponent() {
      this.LoadFromXaml(typeof(MainPage));
      PhoneNumber = this.FindByName<Entry>(
                       "PhoneNumber");
   }
}
```

MainPage.xaml.g.cs

Working with named elements

- Can work with named elements as if you defined them in code, but keep in mind the field is not set until *after* **`InitializeComponent`** is called

Can wire up events, set properties, even add new elements to layout

```
public partial class MainPage : ContentPage
{
    public MainPage () {
        InitializeComponent ();
        PhoneNumber.TextChanged += OnTextChanged;
    }

    void OnTextChanged(object sender, TextChangedEventArgs e) {
       ...
    }
}
```

Sharing elements

Generated field is always private, but **Page** owner can wrap in a public property to allow external access

```
public partial class MainPage :  ContentPage
{
    public Entry  PhoneNumberEntry
    {
        get { return this.PhoneNumber;  }
    }
    ...
}
```

should *not* provide a setter – replacing the field's value will not change the actual element on the screen

Handling events in XAML

- Can also wire up events in XAML – event handler *must be defined* in the code behind file and have proper signature or it's a runtime failure

```
<Entry Placeholder="Number" TextChanged="OnTextChanged" />
```

```
public partial class MainPage : ContentPage
{
    ...
    void OnTextChanged(object sender, TextChangedEventArgs e) {
        ...
    }
}
```

Handling events in code behind

- Many developers prefer to wire up all events in code behind by naming the XAML elements and adding event handlers in ode
  - Keeps the UI layer "pure" by pushing all behavior + management into the code behind
  - Names are validated at compile time, but event handlers are not
  - Easier to see how logic is wired up

- Pick the approach that works for your team / preference

Using device-specific values

- XAML is a static (compile-time) definition of the UI; can provide different
  - values for each platform just like we do in code with `Device.OnPlatform`
    - `x:TypeArguments` used for generic instantiation

```
<OnPlatform x:TypeArguments="Thickness"
    iOS="0,20,0,0"  Android="0"  WinPhone="0"  />
```

- can then supply different platform-specific value forproperty

Using Markup Extensions

v Markup Extensions are identified by "`{extension_here}`" curly braces

parser expects to find a class named **`BindingExtension`** that implements **`IMarkupExtension`** when it encounters the curly brace as the first character

```
<StackLayout BindingContext="{Binding Details}">
  <Label Text="{}{Want a Curly Brace Here!}" />
   ...
</StackLayout>
```

literal curly braces need to be escaped properly to avoid a parser error

Reading static properties

- A very useful markup extension is `x:Static` which lets you get the value of public static fields or properties

```
public static class Constants
{
    public static string Title = "Hello, Forms";
    public static Thickness Padding = new Thickness(5, Device.OnPlatform(20, 0, 0), 5, 0);
    public static Font Font = Font.SystemOfSize(24);
    public static Color TextColor = Color.Yellow;
}
```

```
<ContentPage ... Padding="{x:Static me:Constants.Padding}" >
   <Label Text="{x:Static me:Constants.Title}"
          Font="{x:Static me:Constants.Font}"
          TextColor="{x:Static me:Constants.TextColor}" />
</ContentPage>
```

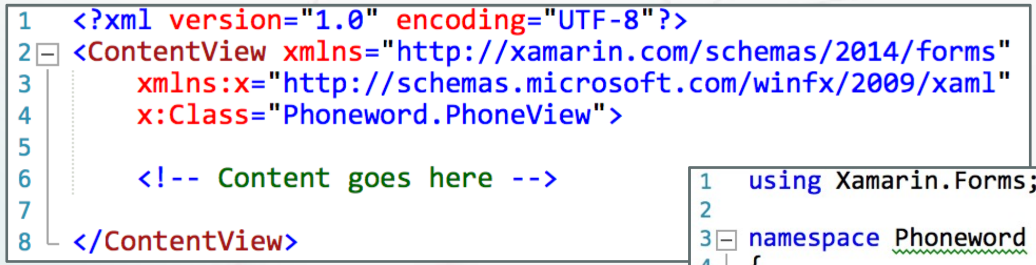
Other built-in Markup Extensions

- Use resource values with **`{StaticResource}`** and **`{DynamicResource}`**
- Supply a **`null`** value with **`{x:Null}`**
- Lookup a **`Type`** with **`{x:Type}`**
- Create an array with **`{x:Array}`**
- Create data bindings with **`{Binding}`**

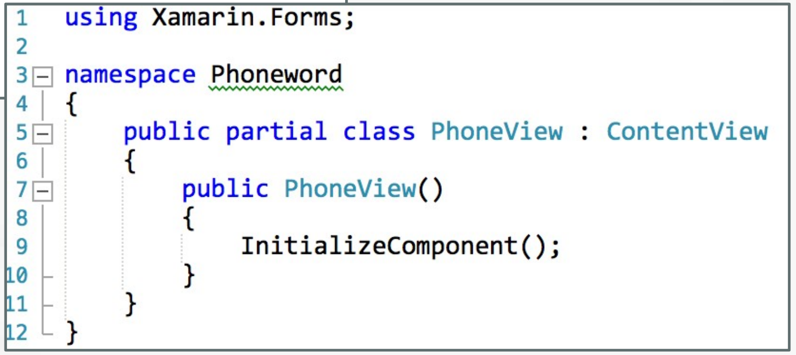
```
<ListView SelectedItem="{x:Null}">
  <ListView.ItemsSource>
    <x:Array Type="{x:Type x:Int32}">
        <x:Int32>10</x:Int32>
        <x:Int32>20</x:Int32>
        <x:Int32>30</x:Int32>
    </x:Array>
  </ListView.ItemsSource>
</ListView>
```

ContentView structure

- ContentView combines a piece of XAML with code behind behavior - just like `ContentPage`, can name elements, wire up events, etc.



```xml
<?xml version="1.0" encoding="UTF-8"?>
<ContentView xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="Phoneword.PhoneView">

    <!-- Content goes here -->

</ContentView>
```

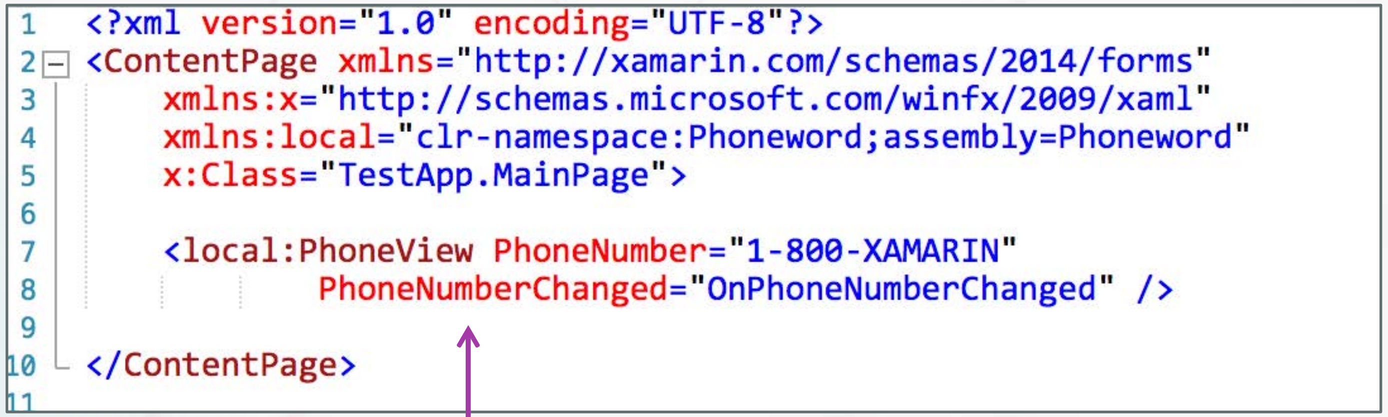


```csharp
using Xamarin.Forms;

namespace Phoneword
{
    public partial class PhoneView : ContentView
    {
        public PhoneView()
        {
            InitializeComponent();
        }
    }
}
```

Can be placed into a separate class library if desired

Using a ContentView

- `ContentView` is not displayed on it's own - must be added to a `Page`



```
<?xml version="1.0" encoding="UTF-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:Phoneword;assembly=Phoneword"
    x:Class="TestApp.MainPage">

    <local:PhoneView PhoneNumber="1-800-XAMARIN"
            PhoneNumberChanged="OnPhoneNumberChanged" />

</ContentPage>
```

`ContentView` can expose it's own properties and events to provide customization or "hooks" into the logic

XAML resources

- By default, your XAML files are included as a plain-text resource in the generated assembly which is parsed at runtime to generate the page

```
private void InitializeComponent()
{
    this.LoadFromXaml(typeof(MainPage));
}
```

This **Page** method looks up the embedded resource by name, parses it, and creates each object found; it returns the root created object

Compiling XAML

- XAML can be optionally compiled to intermediate language (IL)
  - Provides compile-time validation of your XAML files
  - Reduces the load time for pages
  - Reduces the assembly size by removing text-based .xaml files

Enabling XAMLC

- XAMLC (the XAML compiler) is disabled by default to ensure backwards compatibility; can be enabled through a .NET attribute

```
using Xamarin.Forms.Xaml;

[assembly: XamlCompilationAttribute( XamlCompilationOptions.Compile)]
```

↑

Can enable the compiler for all XAML files in the assembly

Enabling XAMLC

- XAMLC (the XAML compiler) is disabled by default to ensure backwards compatibility; can be enabled through .NET attribute

```
using Xamarin.Forms.Xaml;

[XamlCompilationAttribute(XamlCompilationOptions.Compile)]
public partial class MainPage : ContentPage   {
```

… or on a specific XAML-based class

What does it do?

- Attribute presence causes MSBuild command to be run which parses the XAML and generates **`InitializeComponent`** to create the page in code

```
private void InitializeComponent()
{
   Label label = new Label();
   StackLayout stackLayout = new StackLayout();
   stackLayout.SetValue(VisualElement.BackgroundColorProperty,
        new ColorTypeConverter().ConvertFrom("Red"));
   stackLayout.SetValue(Layout.PaddingProperty,
        new ThicknessTypeConverter().ConvertFrom("10"));
   stackLayout.SetValue(StackLayout.SpacingProperty, 5);
   label.SetValue(Label.TextProperty, "Hello, Forms");
   stackLayout.Children.Add(label);
   ...
   this.Content = stackLayout;
}
```

Disabling XAMLC

- Attribute also lets you disable XAMLC for a specific class

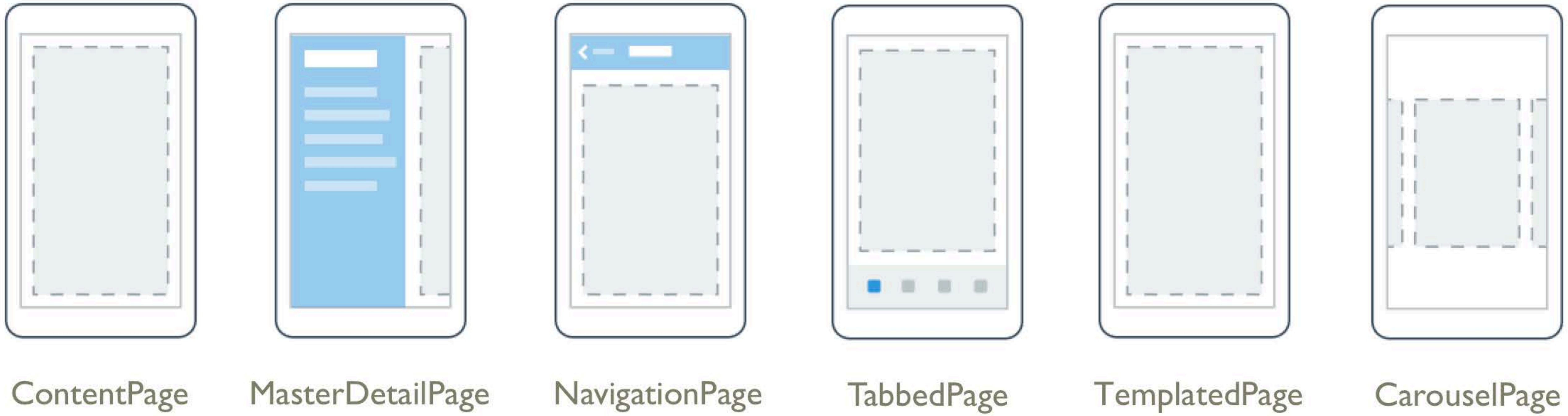
```
using Xamarin.Forms.Xaml;

[XamlCompilationAttribute(XamlCompilationOptions.Skip)]
public partial class DetailsPage  : ContentPage  {
```

Specify Skip to turn off compiler for this specific page; goes back to using `LoadFromXaml`

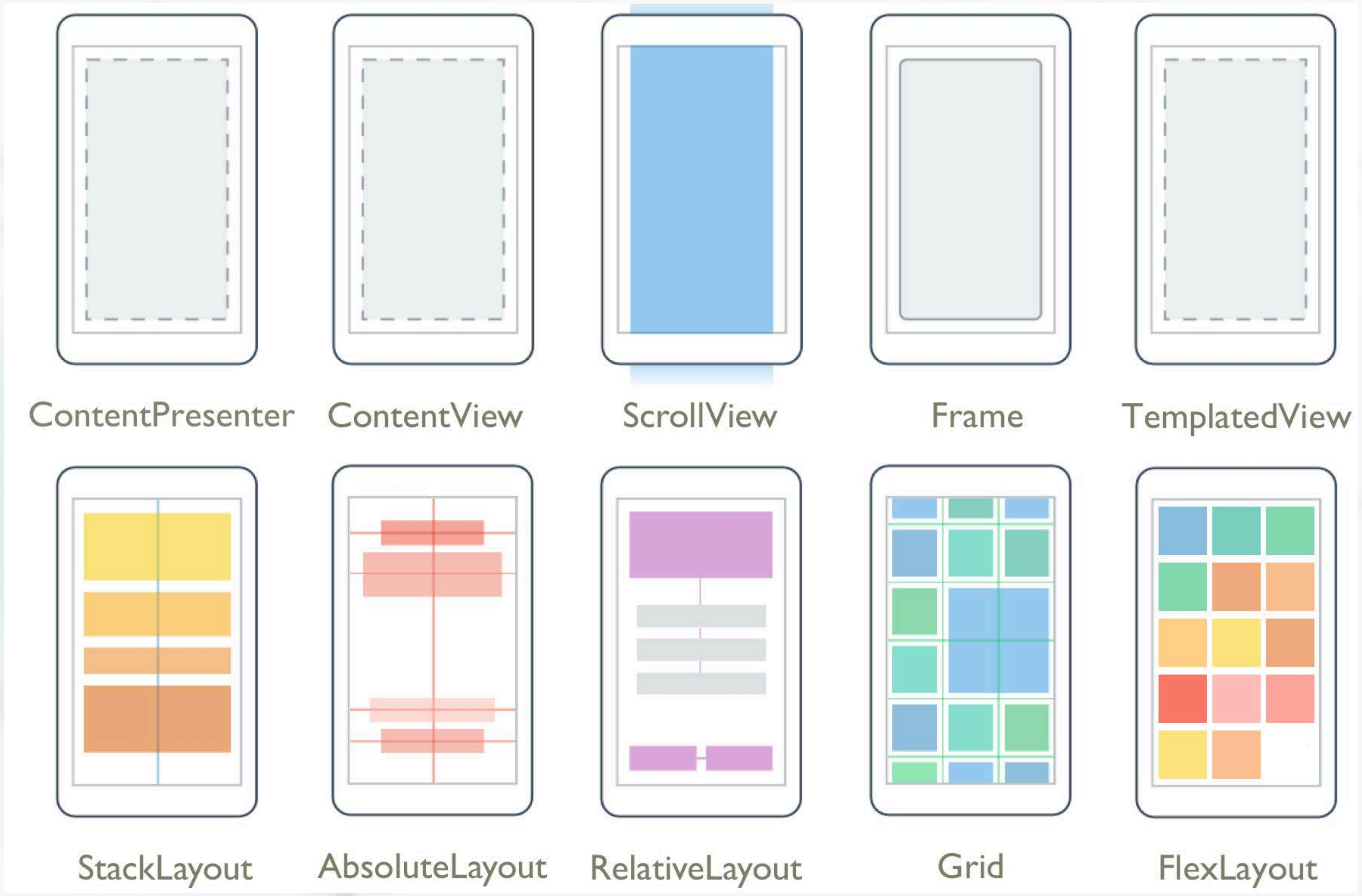
Layout

PAGE, LAYOUT, POSITIONNEMENT

Pages

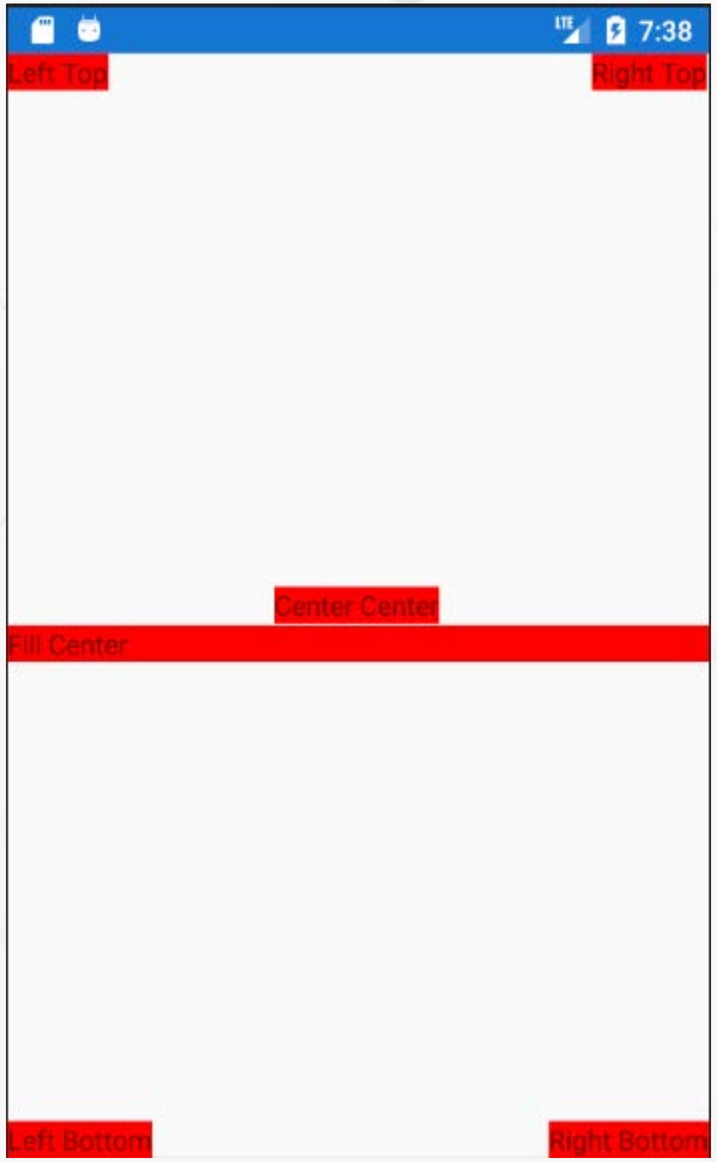

ContentPage
MasterDetailPage
NavigationPage
TabbedPage
TemplatedPage
CarouselPage

Layouts


ContentPresenter
ContentView
ScrollView
Frame
TemplatedView
StackLayout
AbsoluteLayout
RelativeLayout
Grid
FlexLayout

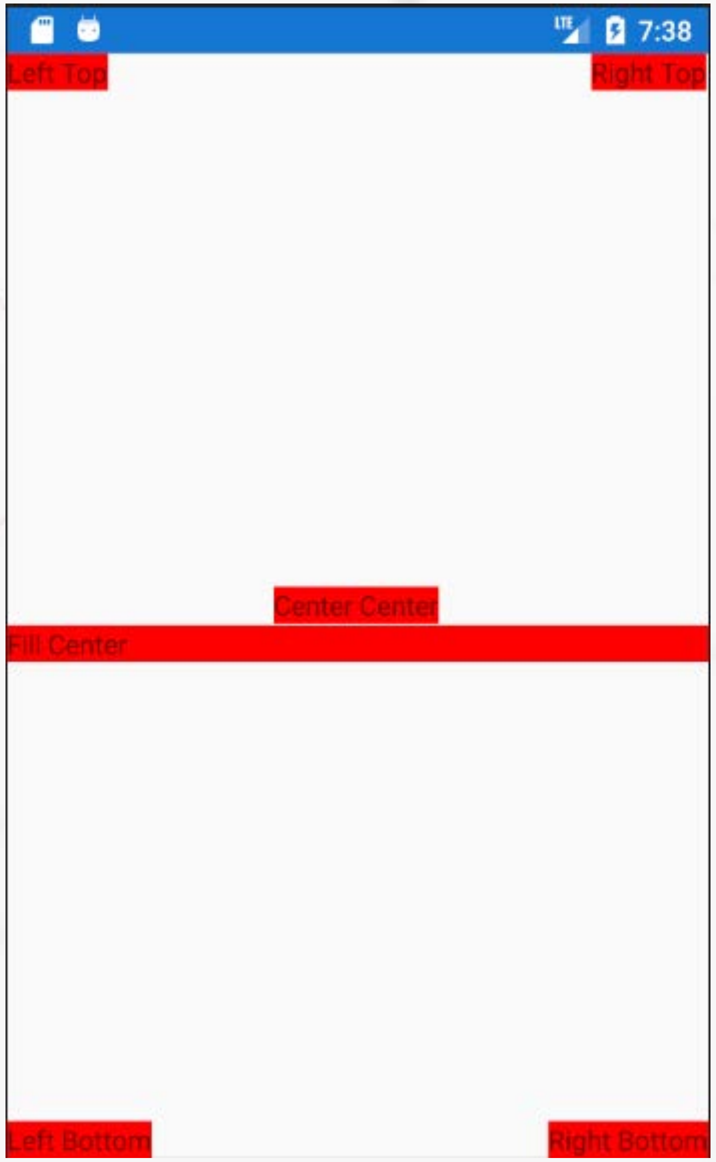
Alignements

```
<Grid>
        <Label Text="Left Top"
               HorizontalOptions="Start"
               VerticalOptions="Start"
               BackgroundColor="Red"/>

        <Label Text="Right Top"
               HorizontalOptions="End"
               VerticalOptions="Start"
               BackgroundColor="Red"/>

        <Label Text="Left Bottom"
               HorizontalOptions="Start"
               VerticalOptions="End"
               BackgroundColor="Red"/>

        <Label Text="Right Bottom"
               HorizontalOptions="End"
               VerticalOptions="End"
               BackgroundColor="Red"/>

        <Label Text="Center Center"
               HorizontalOptions="Center"
               VerticalOptions="Center"
               BackgroundColor="Red"/>

        <Label Text="Fill Center"
               HorizontalOptions="Fill"
               VerticalOptions="Center"
               Margin="0, 40, 0, 0"
               BackgroundColor="Red"/>
    </Grid>
```



Marge et padding

```
<Sta>
        <Label Text="Left Top"
               HorizontalOptions="Start"
               VerticalOptions="Start"
               BackgroundColor="Red"/>

        <Label Text="Right Top"
               HorizontalOptions="End"
               VerticalOptions="Start"
               BackgroundColor="Red"/>

        <Label Text="Left Bottom"
               HorizontalOptions="Start"
               VerticalOptions="End"
               BackgroundColor="Red"/>

        <Label Text="Right Bottom"
               HorizontalOptions="End"
               VerticalOptions="End"
               BackgroundColor="Red"/>

        <Label Text="Center Center"
               HorizontalOptions="Center"
               VerticalOptions="Center"
               BackgroundColor="Red"/>

        <Label Text="Fill Center"
               HorizontalOptions="Fill"
               VerticalOptions="Center"
               Margin="0, 40, 0, 0"
               BackgroundColor="Red"/>
    </Grid>
```



Dimensionnement

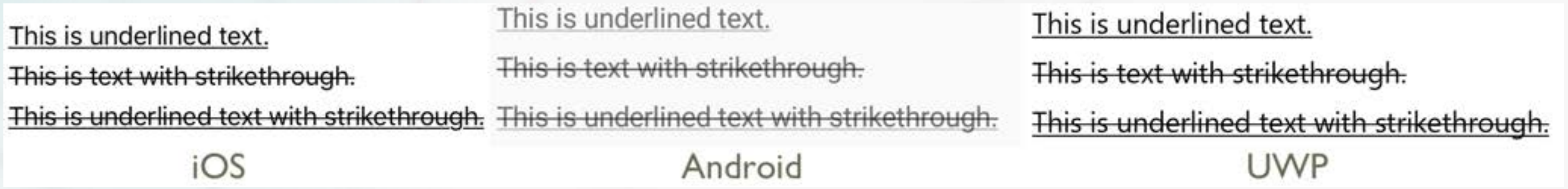
- Eviter de définir des hauteurs et des largeurs fixe
    - Préférez les alignements et le padding
- WidthRequest et HeightRequest

Interface utilisateurs

PRÉSENTATION D’UNE PARTIE DES VIEWS DE XAMARIN.FORMS

Label

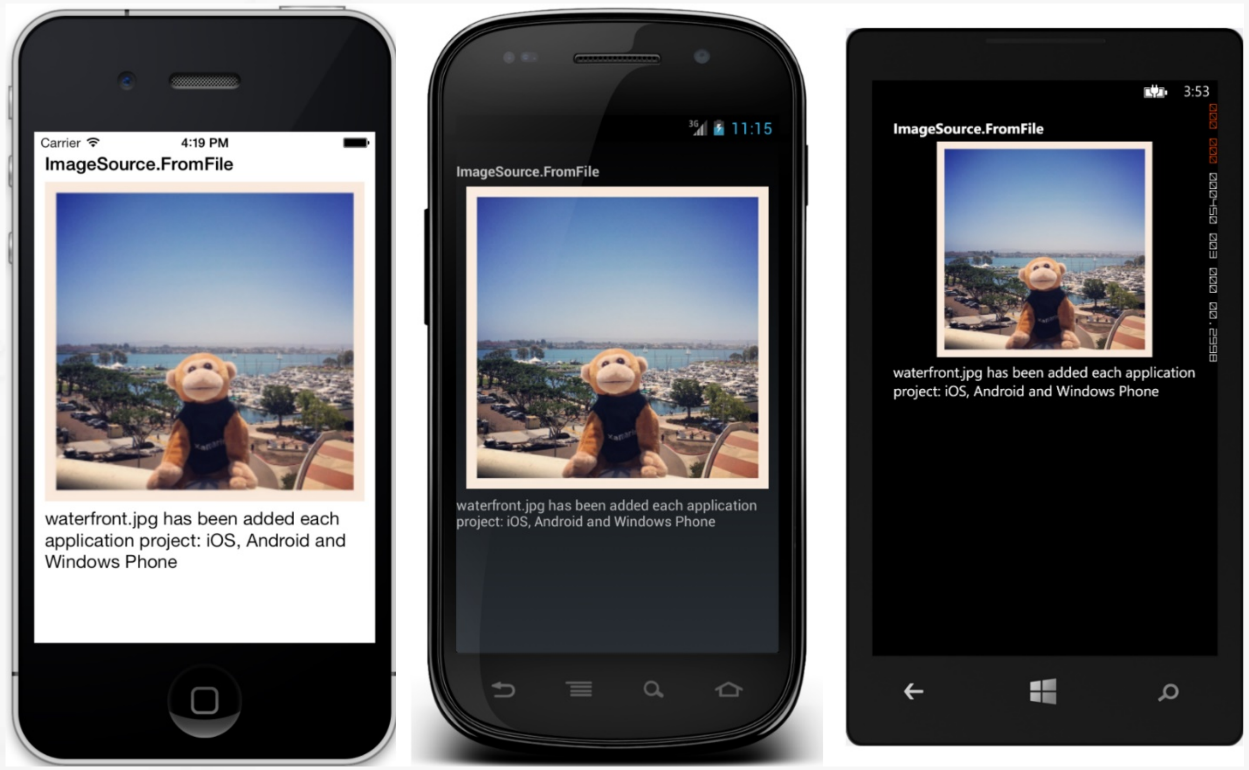
```
<Label Text="This is underlined text." TextDecorations="Underline"  />
<Label Text="This is text with strikethrough." TextDecorations="Strikethrough" />
<Label Text="This is underlined text with strikethrough." TextDecorations="Underline,
Strikethrough" />
```


This is underlined text.
~~This is text with strikethrough.~~
~~This is underlined text with strikethrough.~~
iOS
This is underlined text.
~~This is text with strikethrough.~~
~~This is underlined text with strikethrough.~~
Android
This is underlined text.
~~This is text with strikethrough.~~
~~This is underlined text with strikethrough.~~
UWP

Images

```
<Image Source="waterfront.jpg"
       Aspect="AspectFit|Fill|AspectFill" />
```

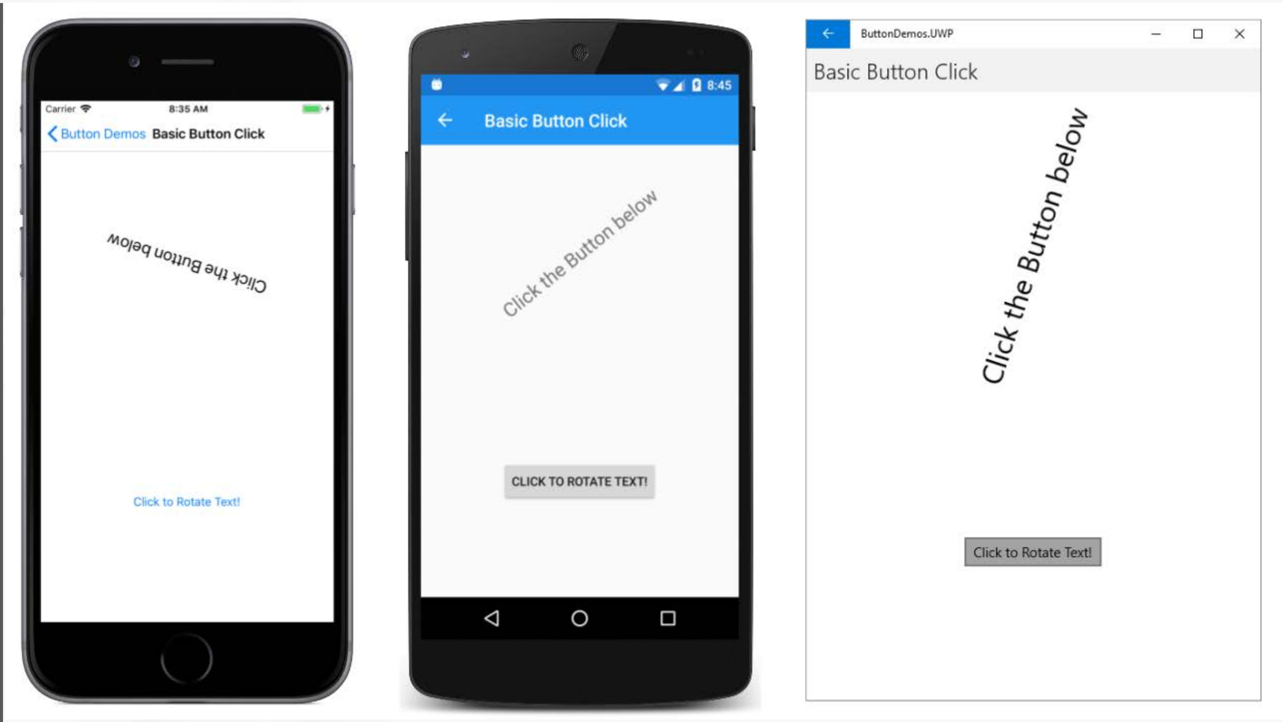
- Chargement depuis :
  - Image locale,
  - Une image en resource
  - Une Uri
  - Un Stream



waterfront.jpg has been added each application project: iOS, Android and Windows Phone

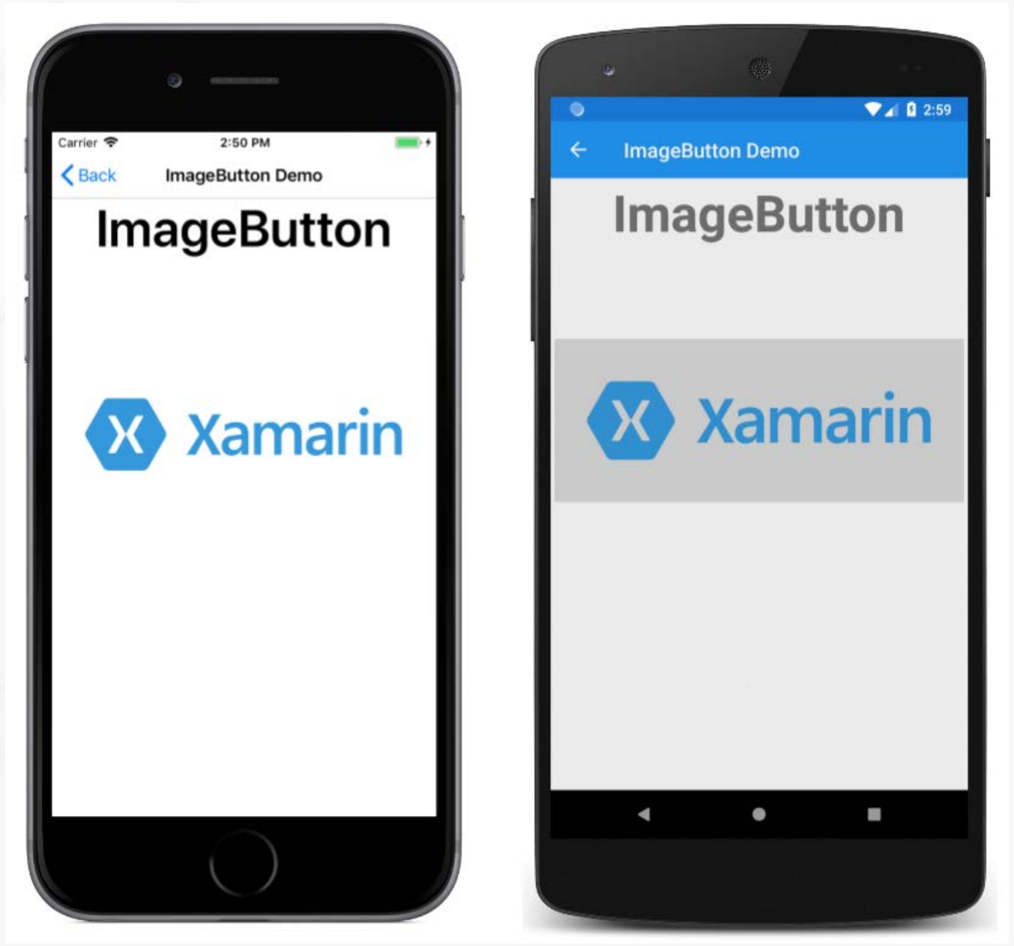
Button

```
<StackLayout>

       <Label x:Name="label"
              Text="Click the Button below"
              FontSize="Large"
              VerticalOptions="CenterAndExpand"
              HorizontalOptions="Center" />

       <Button Text="Click to Rotate Text!"
              VerticalOptions="CenterAndExpand"
              HorizontalOptions="Center"
              Clicked="OnButtonClicked" />

</StackLayout>
```



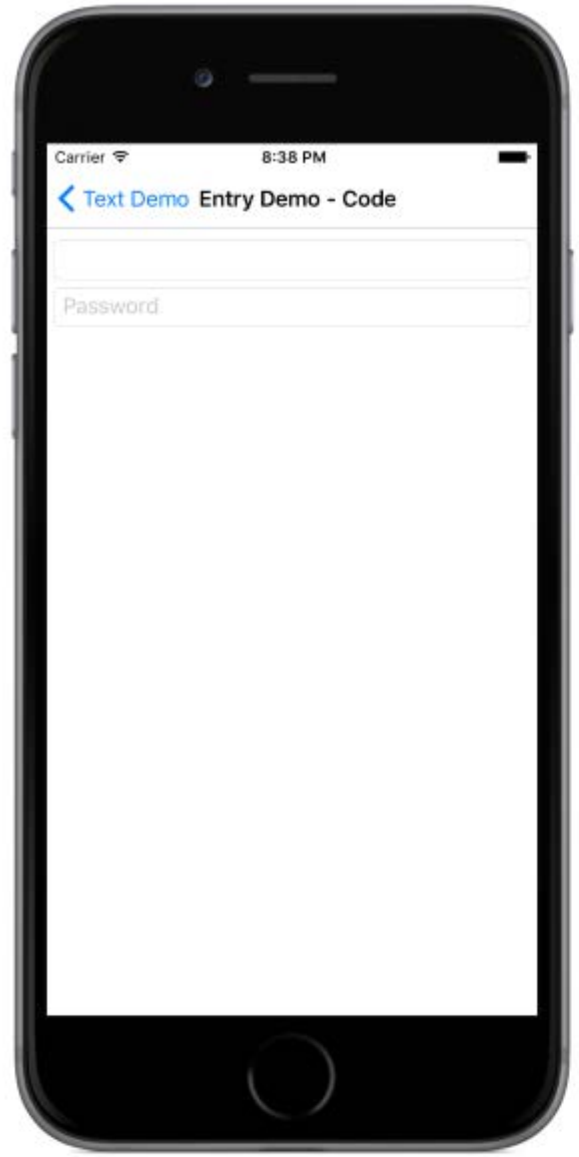
ImageButton

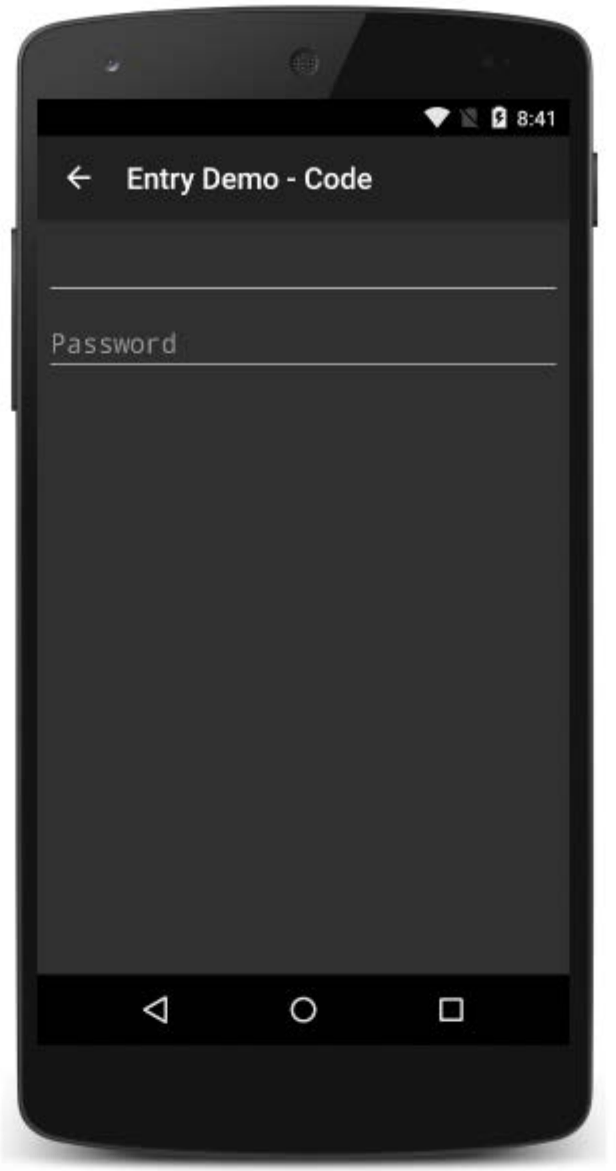
```
<ImageButton Source="XamarinLogo.png"/>
```



Entry

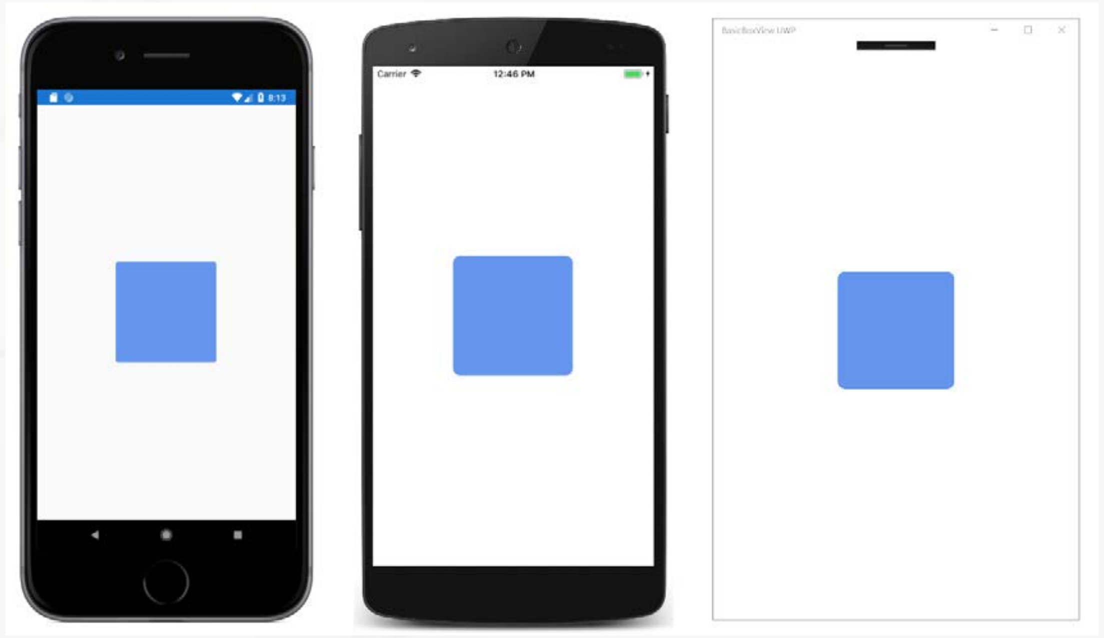
```
<Entry />
<Entry IsPassword="true" Placeholder="Password"
/>
```





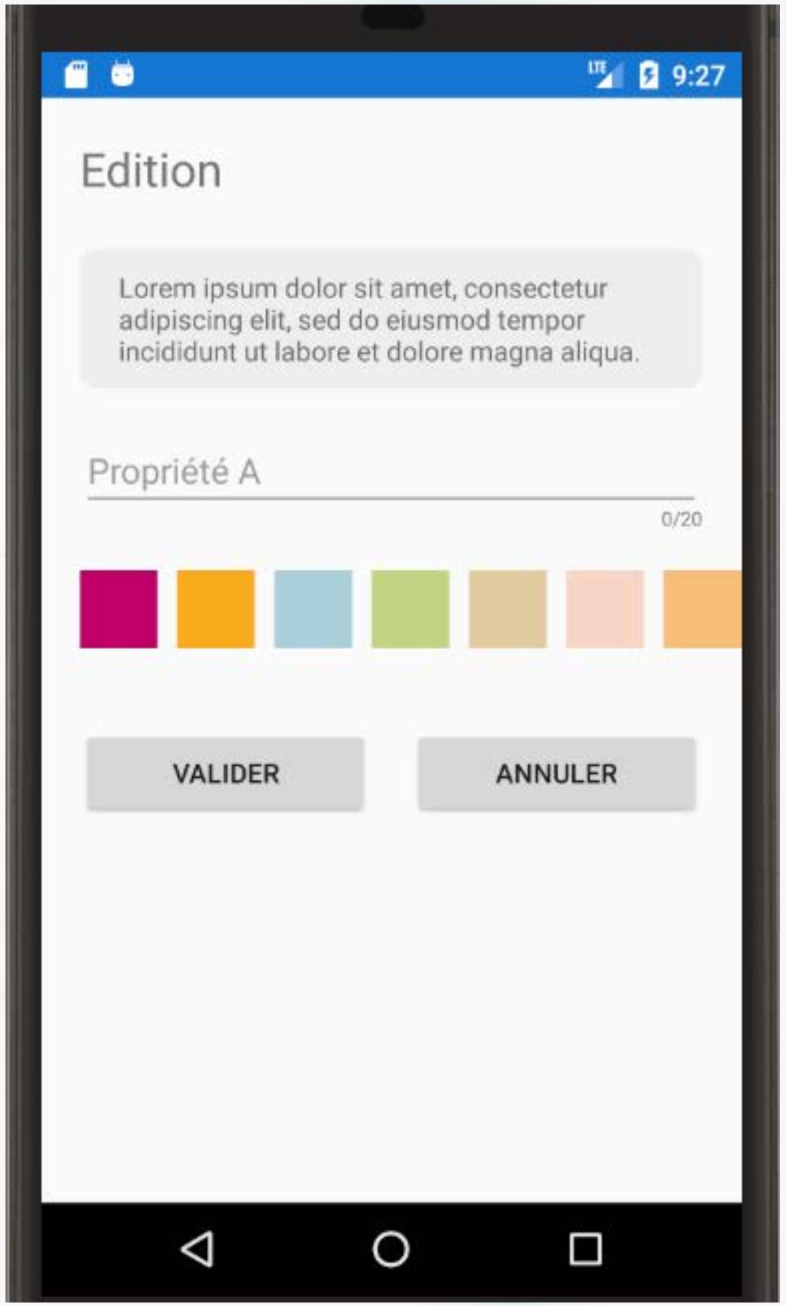
BoxView

```
<ContentPage>

    <BoxView Color="CornflowerBlue"
             CornerRadius="10"
             WidthRequest="160"
             HeightRequest="160"
             VerticalOptions="Center"
             HorizontalOptions="Center" />

</ContentPage>
```



Atelier Layouting

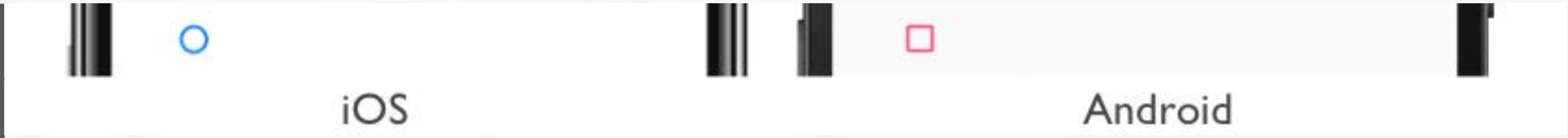
- Utiliser les différentes notions suivantes:
  - StackLayout
  - Grid
  - ScrollView
  - BoxView
  - Button
  - Entry

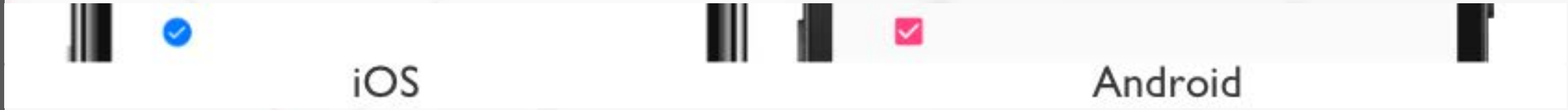


Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

CheckBox

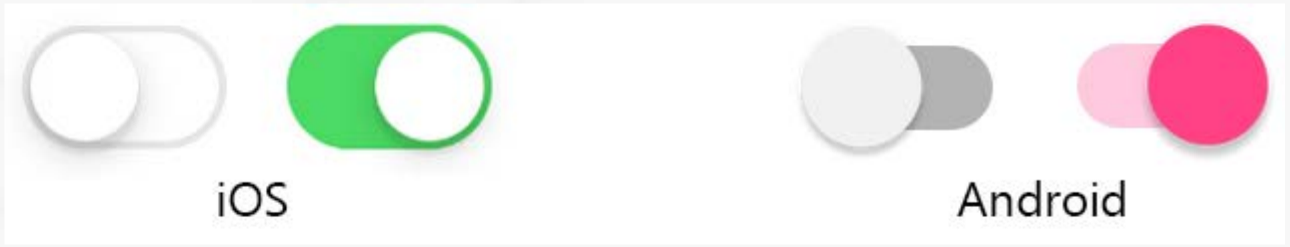
```
<CheckBox />


<CheckBox IsChecked="true" />
```





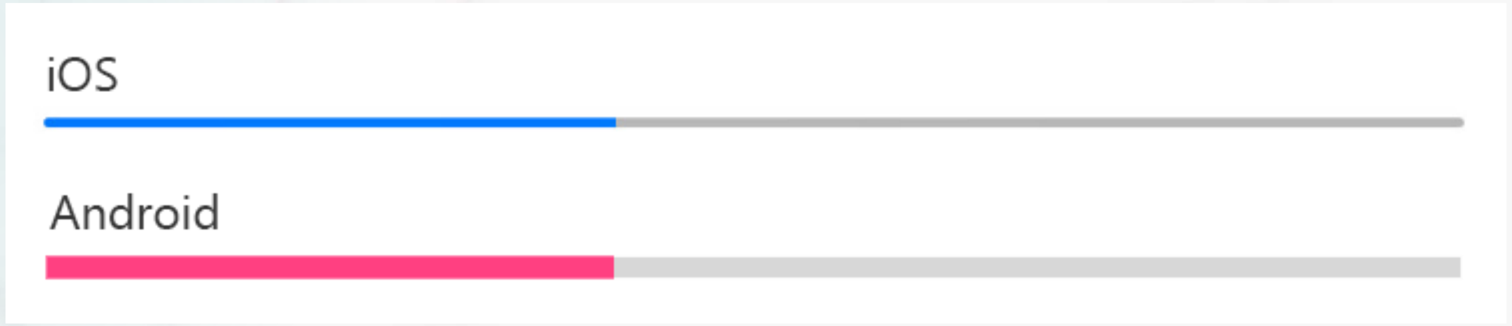
Switch

```
<Switch IsToggled="true"/>
```



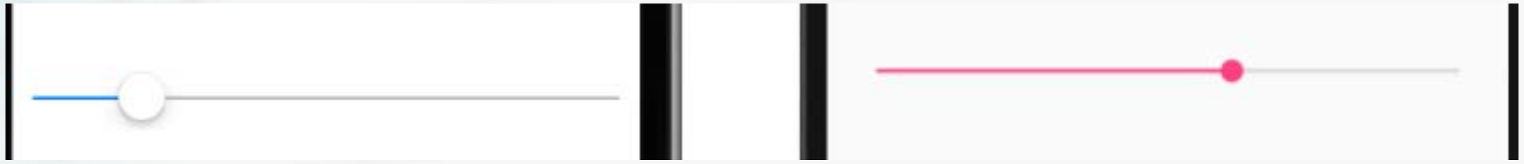
ProgressBar

```
<ProgressBar Progress="0.5" />
```



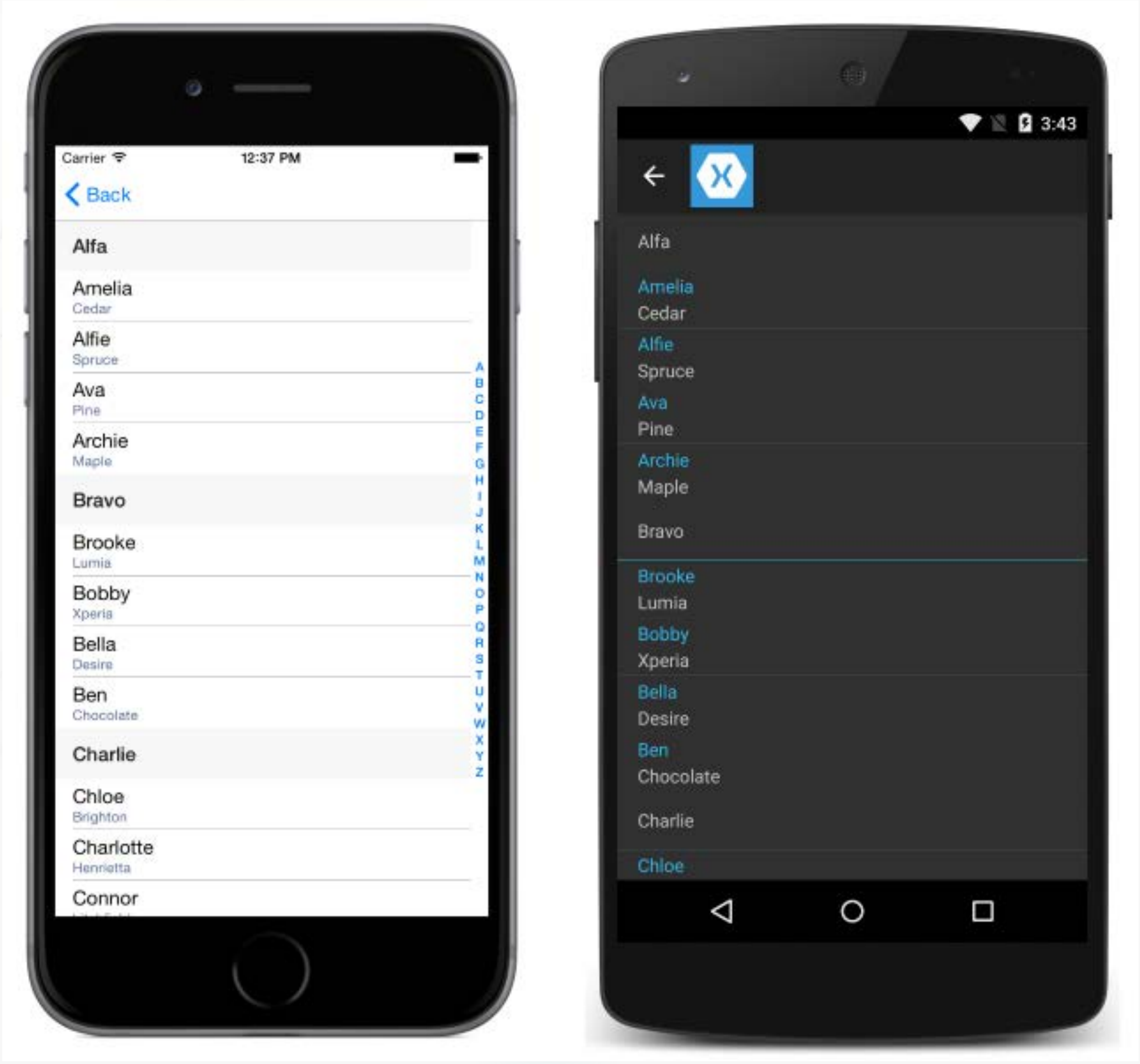
Slider

```
<Slider Value=«50»  Minimum=« 0 »
Maximum=« 100 » />
```



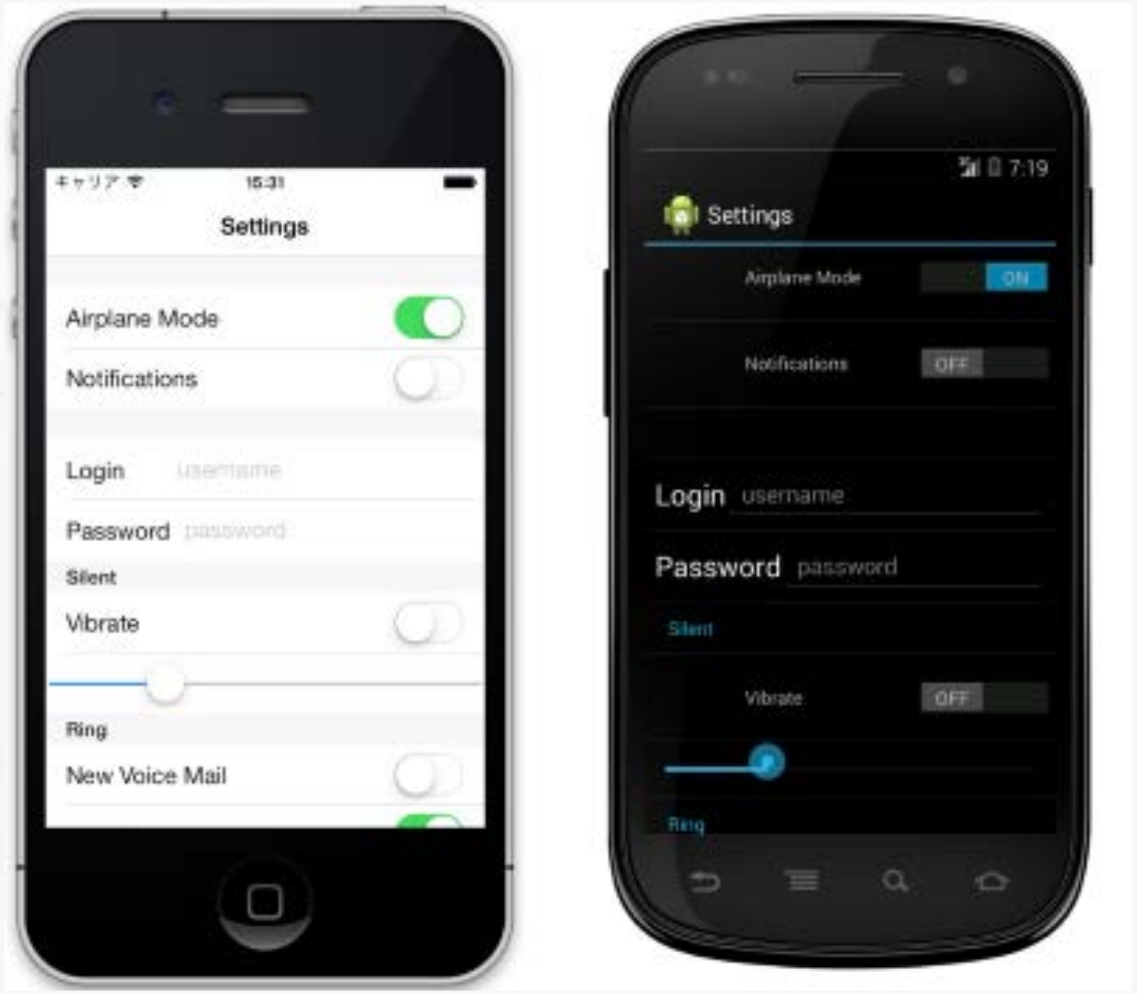
ListView

- Affichage de liste scrollable
  - Texte,
  - Image / Texte,
  - Personnalisé
- Sélection d'une ligne
- Click sur une ligne
- Header / Footer



TableView

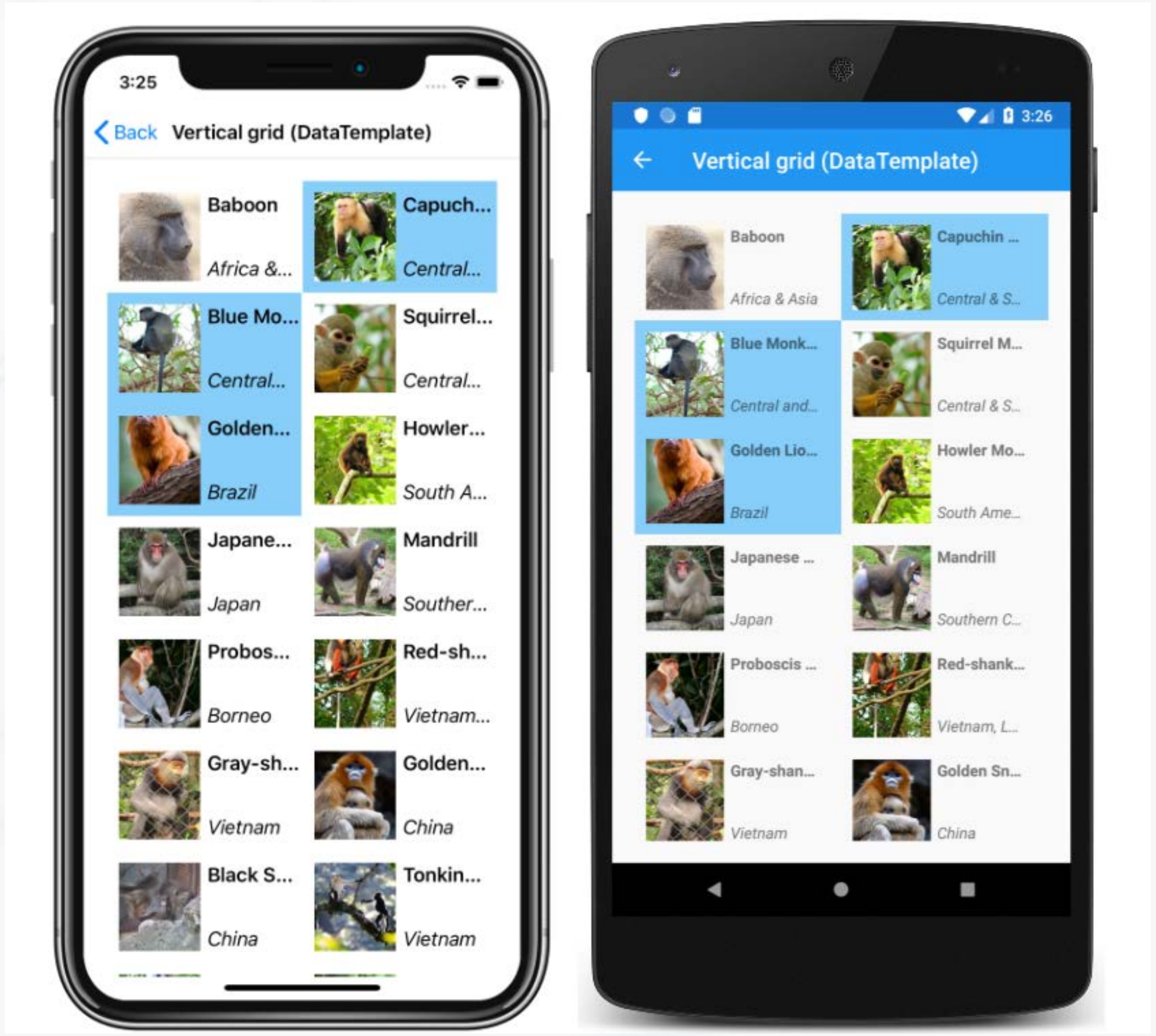
```
<TableView Intent="Settings">
    <TableRoot>
        <TableSection Title="Ring">
            <SwitchCell Text="New Voice Mail" />
            <SwitchCell Text="New Mail"
On="true" />
        </TableSection>
    </TableRoot>
</TableView>
```



CollectionView

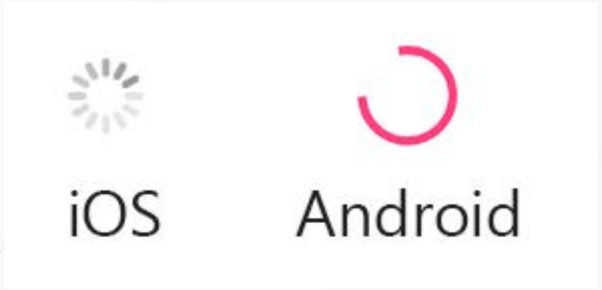
Preview

- Remplaçante de la ListView
  - Plus flexible
  - Plus performante



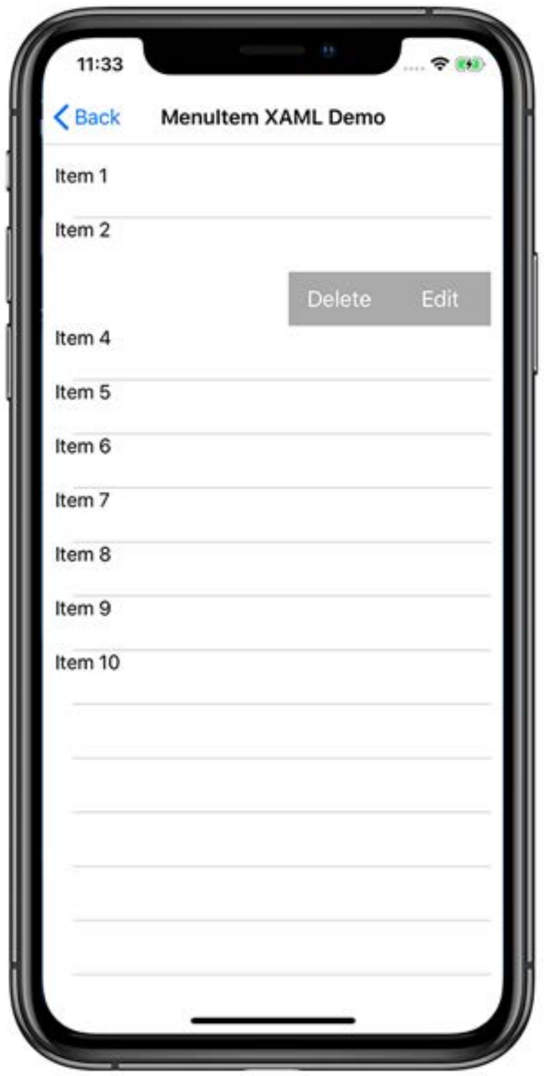
ActivityIndicator

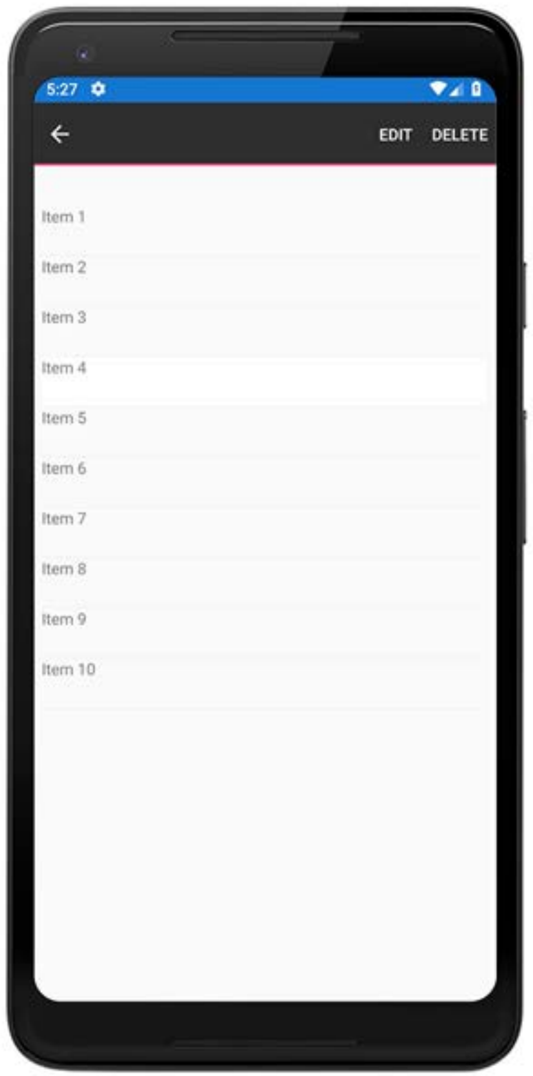
```
<ActivityIndicator IsRunning="true" />
```



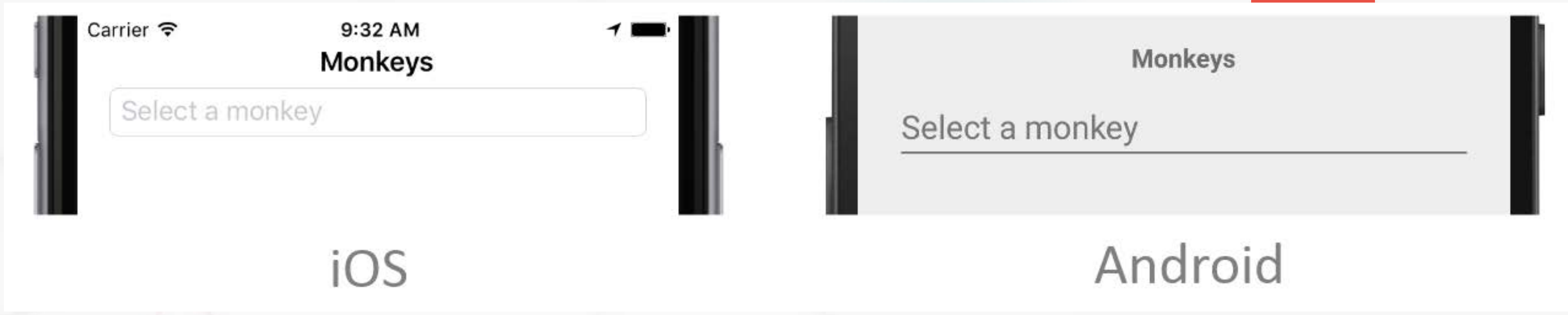
MenuItem

```
<ListView>
    <ListView.ItemTemplate>
        <DataTemplate>
            <ViewCell>
                <ViewCell.ContextActions>
                    <MenuItem Text="Context Menu
Option" />
                </ViewCell.ContextActions>
                <Label Text="{Binding .}" />
            </ViewCell>
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>
```

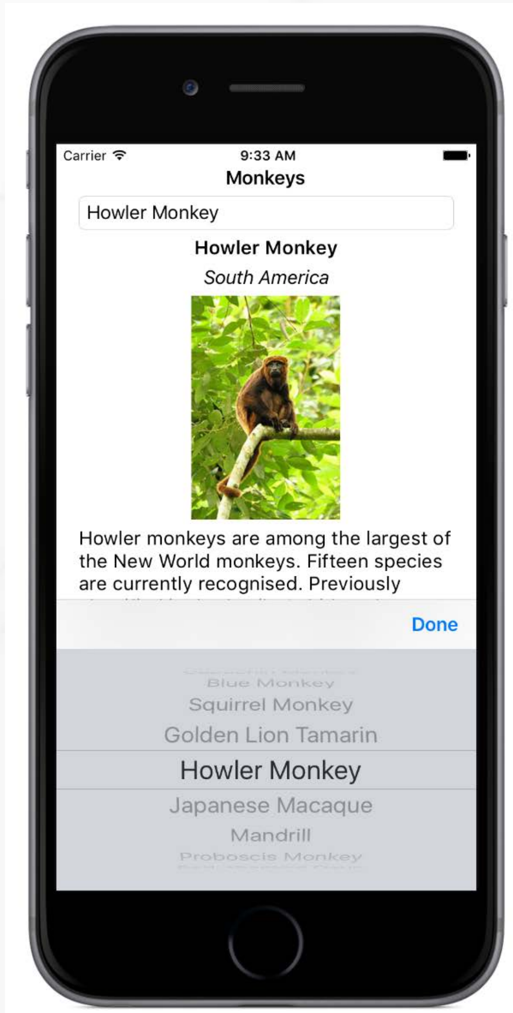




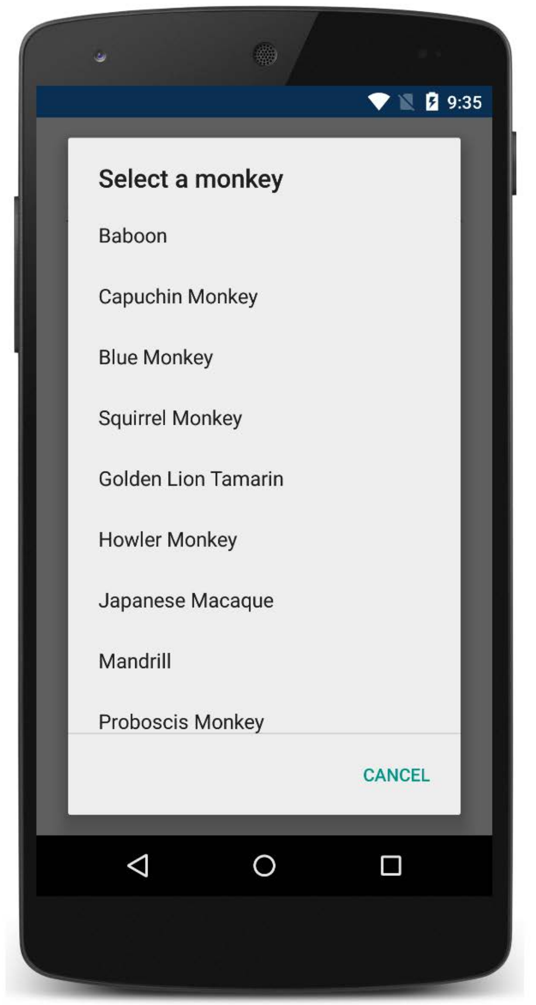
Picker



```
<Picker Placeholder=« select a monkey »>
```

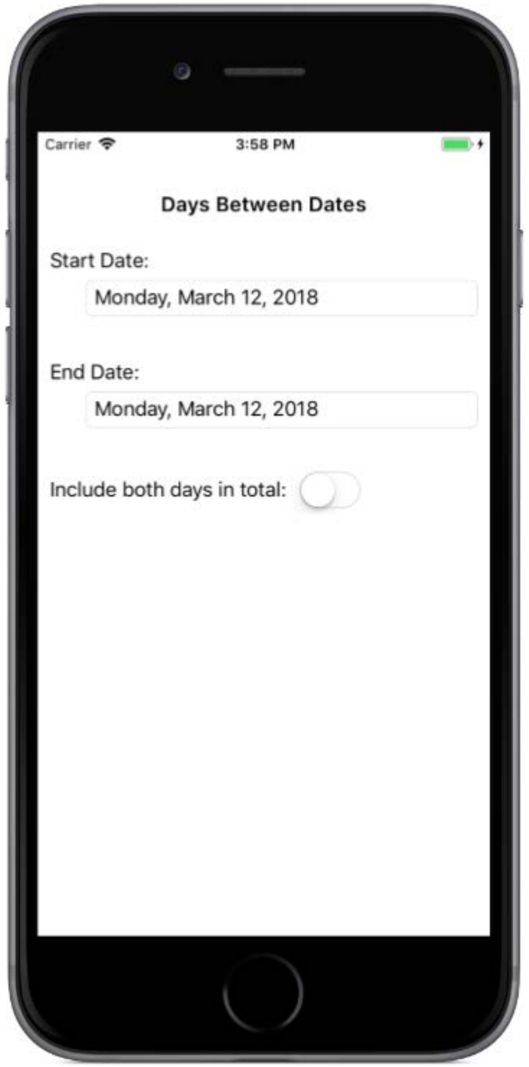


Howler monkeys are among the largest of the New World monkeys. Fifteen species are currently recognised. Previously



DatePicker / TimePicker

```
<DatePicker MinimumDate="01/01/2018"
            MaximumDate="12/31/2018"
            Date="06/21/2018" />
```

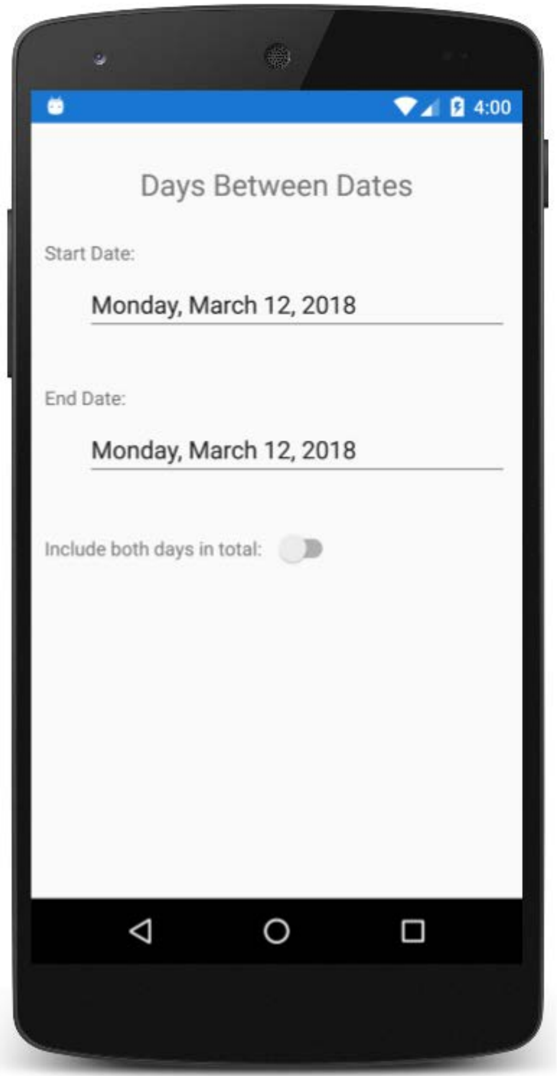

Carrier 3:58 PM

Days Between Dates

Start Date:
Monday, March 12, 2018

End Date:
Monday, March 12, 2018

Include both days in total:

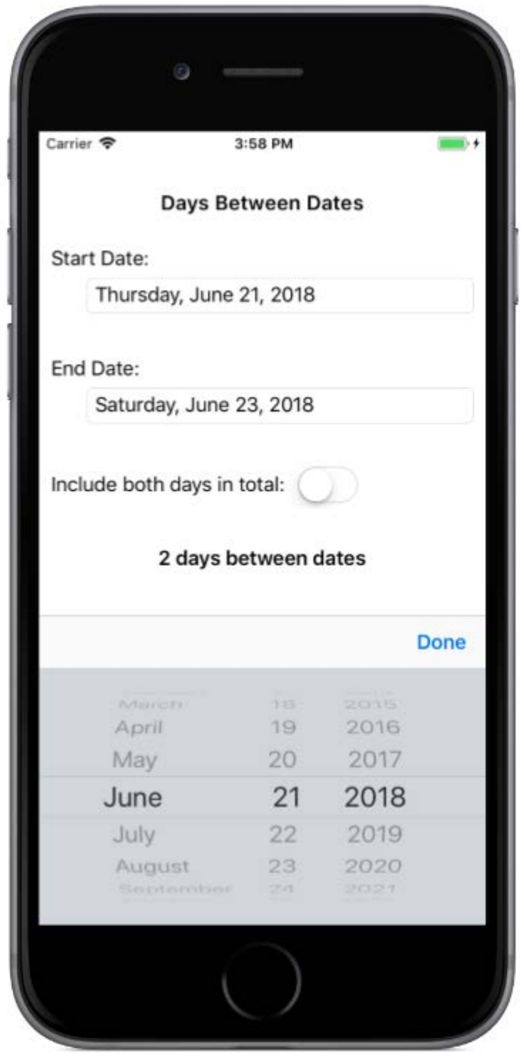

4:00

Days Between Dates

Start Date:
Monday, March 12, 2018

End Date:
Monday, March 12, 2018

Include both days in total:


Carrier 3:58 PM

Days Between Dates

Start Date:
Thursday, June 21, 2018

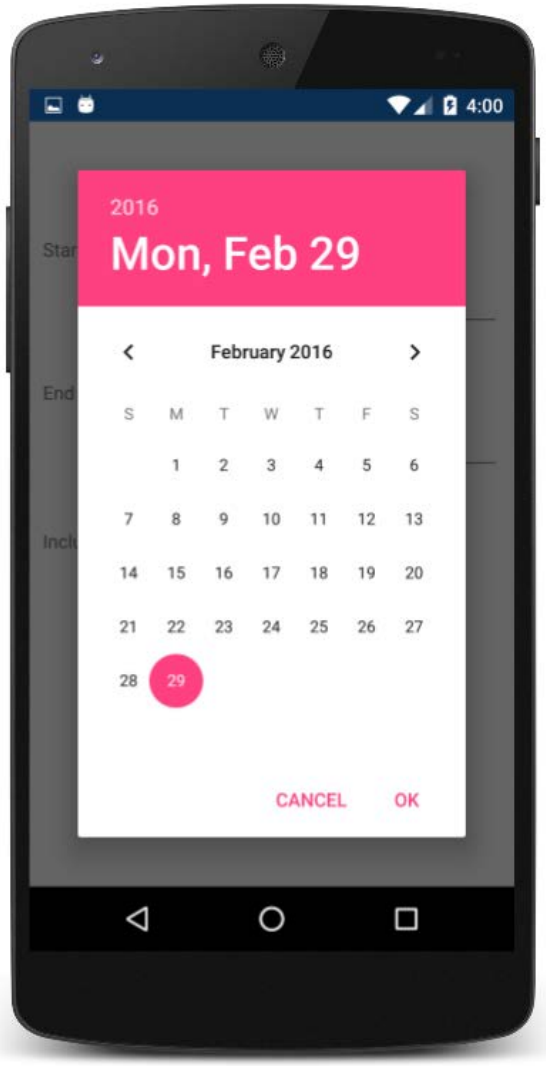
End Date:
Saturday, June 23, 2018

Include both days in total:

2 days between dates

Done

```
March      18   2015
April      19   2016
May        20   2017
June       21   2018
July       22   2019
August     23   2020
September  24   2021
```


4:00

2016
Mon, Feb 29

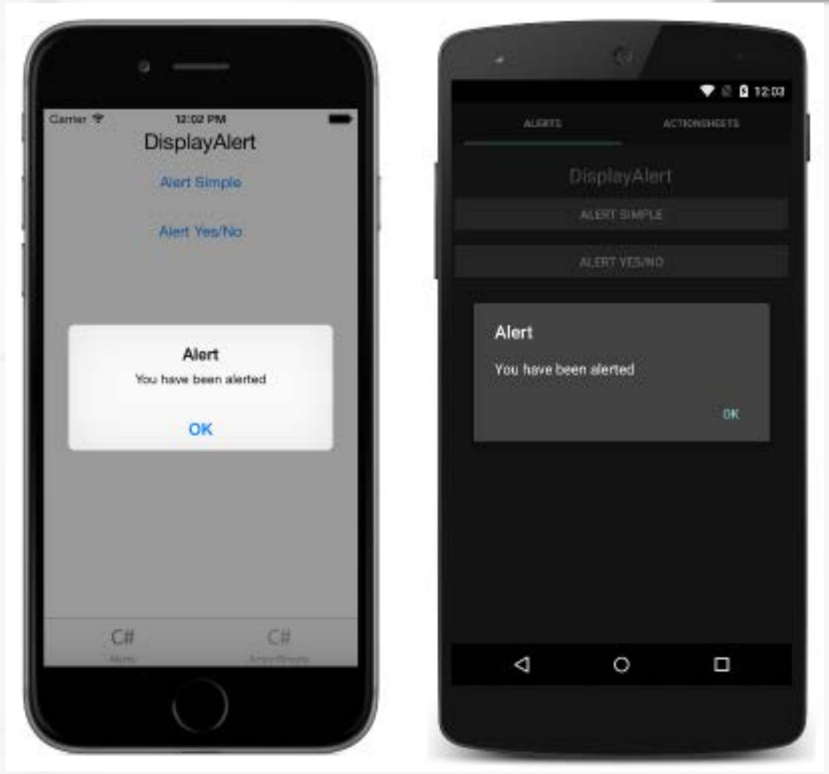
February 2016

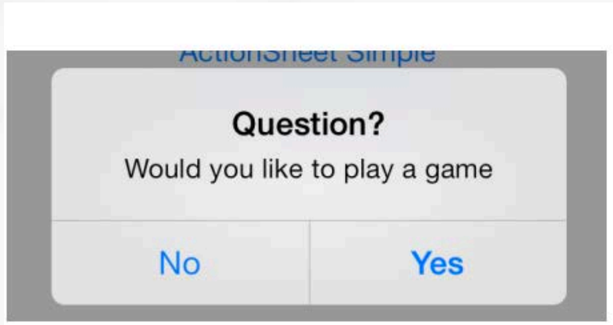
S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29					

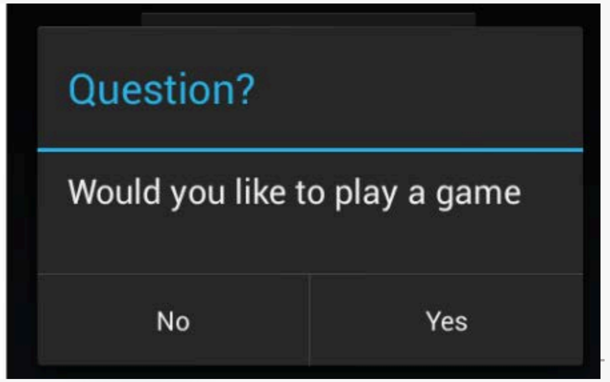
CANCEL OK

Pop-ups

```
DisplayAlert ("Alert", "You have been alerted", "OK");

bool answer = await DisplayAlert ("Question?", "Would
you like to play a game", "Yes", "No");
```

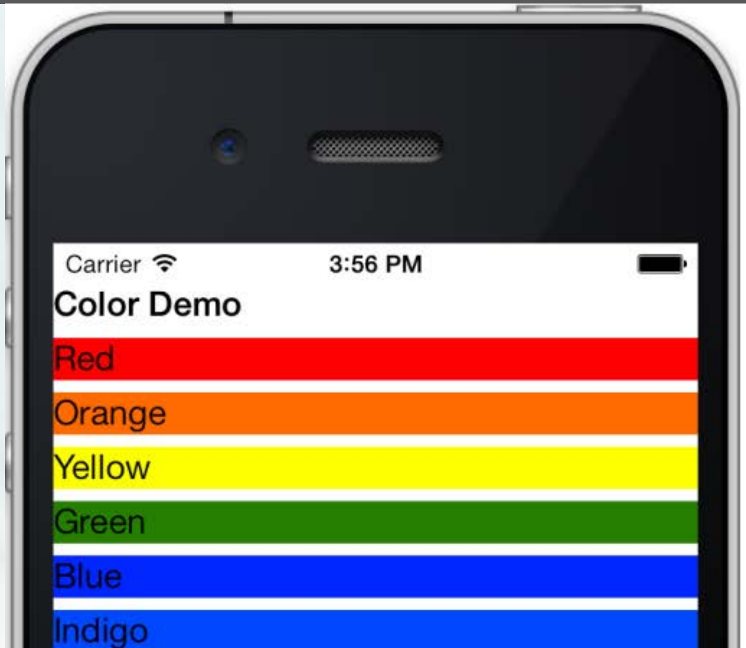


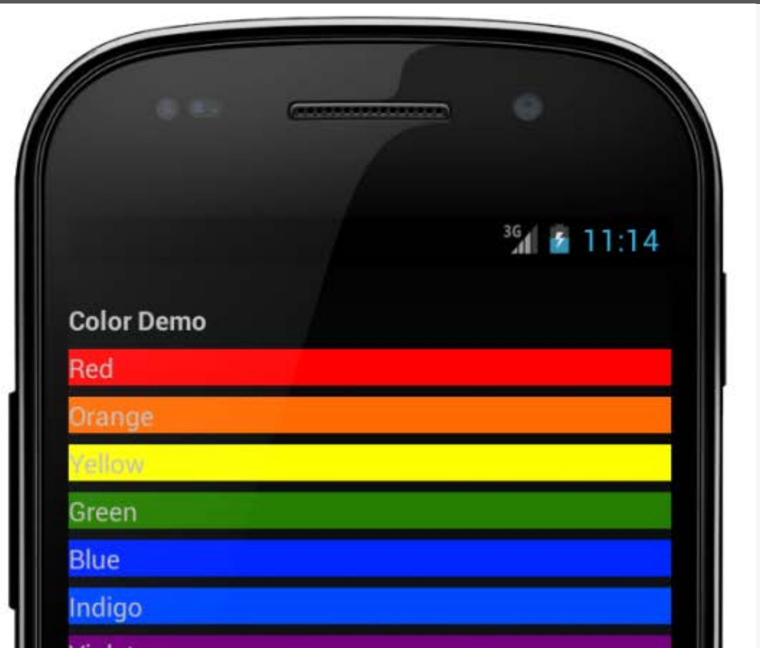




Couleurs

```
var red    = new Label { Text = "Red",   BackgroundColor = Color.Red };
var orange = new Label { Text = "Orange",BackgroundColor = Color.FromHex("FF6A00") };
var yellow = new Label { Text = "Yellow",BackgroundColor = Color.FromHsla(0.167, 1.0, 0.5, 1.0) };
var green  = new Label { Text = "Green", BackgroundColor = Color.FromRgb (38, 127, 0) };
var blue   = new Label { Text = "Blue",  BackgroundColor = Color.FromRgba(0, 38, 255, 255) };
var indigo = new Label { Text = "Indigo",BackgroundColor = Color.FromRgb (0, 72, 255) };
var violet = new Label { Text = "Violet",BackgroundColor = Color.FromHsla(0.82, 1, 0.25, 1) };

var transparent = new Label { Text = "Transparent",BackgroundColor = Color.Transparent };
var @default = new Label    { Text = "Default",    BackgroundColor = Color.Default };
var accent = new Label      { Text = "Accent",     BackgroundColor = Color.Accent };
```



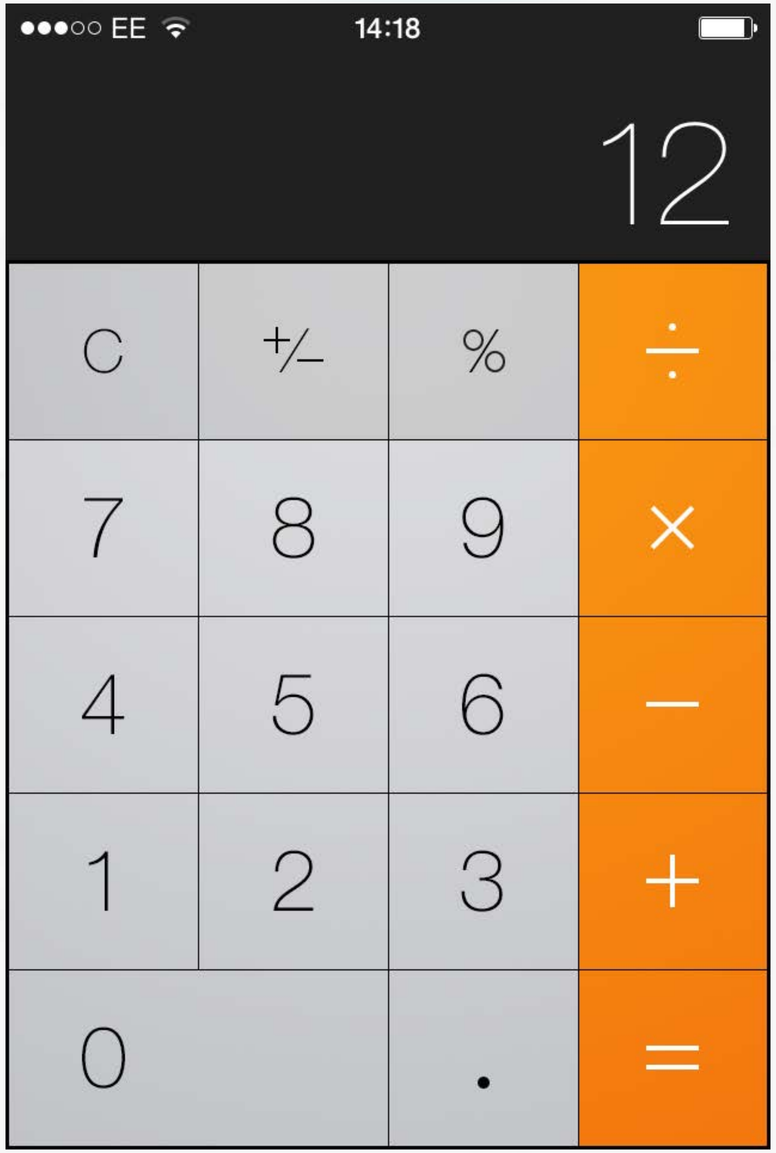


SkiaSharp

- Librairie de dessin
- Permet de créer des visuels très complexes
- Graphique, ombrage, effet visuel …

Atelier Layout + Code métier

- Créer un projet
- Créer une page
- Mettre en place les contrôles
- Ajouter le code métier (4 opérations)



Avoid duplicate XAML with Resources

Motivation

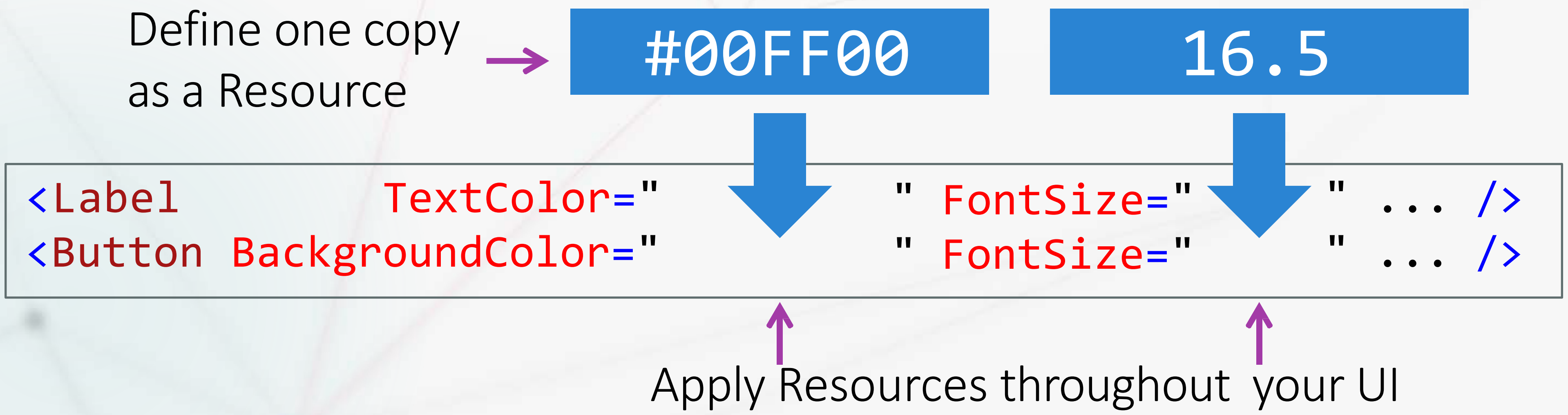
- Duplicate XAML values are error prone and difficult to maintain

```
<StackLayout  BackgroundColor= "#FFFFFF">
  <Label            TextColor=  "#00FF00" FontSize="16.5"  . . .   />
  <Entry      BackgroundColor= "#FFFFFF"                   . . .   />
  <BoxView    BackgroundColor= "#00FF00"                   . . .   />
  <Button     BackgroundColor= "#00FF00"                   . . .   />
                               "#00FF00" FontSize="16.5"

</StackLayout>
```

Common to use the same colors and sizes across the UI

What is a Resource?

- A *Resource* is an object that can be used in multiple places in your UI



```
<Label        TextColor="          " FontSize="          " ... />
<Button BackgroundColor="          " FontSize="          " ... />
```

What is a ResourceDictionary?

- `ResourceDictionary` is a key/value dictionary that is customized for use with UI Resources

Mostly has standard dictionary operations →

```
public sealed class ResourceDictionary : ...
{ ...
  public object this[string index] { get; set; }

  public void Add(string key, object value);
  public void Add(Style implicitStyle);
}
```

↑ Some added UI-specific functionality

Page-level Resources

v Every page can have a resource dictionary, must be set in code or XAML

You must create the dictionary

```
<ContentPage  . . . >

  <ContentPage.Resources>
    <ResourceDictionary>
      ...
    </ResourceDictionary>
  </ContentPage.Resources>

</ContentPage>
```

Assign the dictionary you create to the page's `Resources` property

Creating Resources

- Resources created in XAML must use the XAML-language keyword `x:Key` to set the key

Create inside the page's dictionary

```
<ContentPage ...>

  <ContentPage.Resources>
    <ResourceDictionary>
      <Thickness x:Key="myKey">10,20,40,80</Thickness>
    </ResourceDictionary>
  </ContentPage.Resources>

</ContentPage>
```

Value

Key

Choose Resource names based on use, not value; e.g. use **`bgColor`**, not **`redColor`**.

Using static Resources

- The **`StaticResource`** markup extension retrieves a resource, the value is applied once when the target object is created

```
          <ContentPage ... >

            <ContentPage.Resources>
              <ResourceDictionary>
Define ——→      <Thickness x:Key="myKey">10,20,40,80</Thickness>
              </ResourceDictionary>
            </ContentPage.Resources>

Use ——→     <StackLayout Padding="{StaticResource myKey}">
              ...
            </StackLayout>

          </ContentPage>
```

XAML intrinsic types

- The XAML spec defines many types you can use for XAML Resources

`String` and `Double` are useful since many UI properties use those types

```
<ResourceDictionary>
  <x:String   x:Key="...">Hello</x:String>
  <x:Char     x:Key="...">X</x:Char>
  <x:Single   x:Key="...">31.4</x:Single>
  <x:Double   x:Key="...">27.1</x:Double>
  <x:Byte     x:Key="...">8</x:Byte>
  <x:Int16    x:Key="...">16</x:Int16>
  <x:Int32    x:Key="...">32</x:Int32>
  <x:Int64    x:Key="...">64</x:Int64>
  <x:Decimal  x:Key="...">12345</x:Decimal>
  <x:TimeSpan x:Key="...">1.23:5959</x:TimeSpan>
  <x:Boolean  x:Key="...">True</x:Boolean>
</ResourceDictionary>
```

Platform dependencies

- Can use **`OnPlatform`** objects in your resource dictionaries to handle platform-specific values

```
<ResourceDictionary>
  <OnPlatform x:Key="textColor"
    x:TypeArguments="Color"
    iOS="Red"
    Android="Blue"
    WinPhone="Green" />
</ResourceDictionary>
```

```
<Label TextColor="{StaticResource textColor}" ... />
```

How to update Resources

- Can update resource values from code, useful when you download new values or let the user select preferred colors, font sizes, etc.

Define a default in XAML →

```
<ResourceDictionary>
  <Color x:Key="bg">Blue</Color>
</ResourceDictionary>
```

Update to new value →

```
void OnChangeColor()
{
  this.Resources["bg"] = Color.Green;
}
```

Using dynamic Resources

- The **`DynamicResource`** markup extension retrieves a resource when the target object is created and updates it as the value changes

`BackgroundColor` set to **`Blue`** initially →

```
<ResourceDictionary>
  <Color x:Key="bg">Blue</Color>
</ResourceDictionary>

<StackLayout BackgroundColor="{DynamicResource bg}">
  ...
</StackLayout>
```

`BackgroundColor` changes to **`Green`** →

```
void OnChangeColor()
{
  this.Resources["bg"] = Color.Green;
}
```

Key not found is OK

- `DynamicResource` leaves the property unset if the key is not found, it is not an error and no exception is generated

Key not defined

No value assigned to `BackgroundColor`

```
<ContentPage ... >

   <ContentPage.Resources>
     <ResourceDictionary>
     </ResourceDictionary>
   </ContentPage.Resources>

   <StackLayout BackgroundColor="{DynamicResource bg}">
     ...
   </StackLayout>

</ContentPage>
```

Applying Resources in code

- Resources can be set in code using **`SetDynamicResource`**, allows logic to apply different resources based on runtime knowledge

```
var name = new Label { Text = "Name" };

if (Device.OS == TargetPlatform.iOS)
{
   name.SetDynamicResource(Label.TextColorProperty, "hlColor");
}
```

The **`BindableProperty`** to assign

The Resource key to apply

Create consistent UI with Styles

Motivation [repeated code]

- Resources let you avoid duplicate values, but you still have to set each property individually which creates clutter and yields repeated code

The property settings must be repeated on each view

```
<Button
   BackgroundColor="{StaticResource  highlightColor}"
   BorderColor    ="{StaticResource  edgeColor}"
   BorderRadius   ="{StaticResource  edgeRadius}"
   BorderWidth    ="{StaticResource  edgeSize}"
   TextColor      ="{StaticResource  textColor}"
   Text           ="OK"  />
```

```
<Button
   BackgroundColor="{StaticResource  highlightColor}"
   BorderColor    ="{StaticResource  edgeColor}"
   BorderRadius   ="{StaticResource  edgeRadius}"
   BorderWidth    ="{StaticResource  edgeSize}"
   TextColor      ="{StaticResource  textColor}"
   Text           ="Cancel"  />
```

Motivation [efficiency]

- Resource lookup can increase the startup time of your app since the lookup takes longer than assigning a literal value

```
<Button
   TextColor="{StaticResource textColor}"
   ... />
```

Slower

```
<Button
   TextColor="White"
   ... />
```

Faster

What is a Setter?

v A **`Setter`** is a container for a property/value pair

```
<Setter Property="TextColor" Value="White" />
```

↑ A bindable property

↑ A value appropriate for the property

What is a Style?

- A **`Style`** is a collection of setters for a particular type of view
  - **`TargetType`** must be set (or runtime exception)

```
<Style TargetType="Button">
  <Setter Property="BackgroundColor"  Value="#2A84D3"  />
  <Setter Property="BorderColor"      Value="#1C5F9B"  />
  <Setter Property="BorderRadius"     Value="10"       />
  <Setter Property="BorderWidth"      Value="3"        />
  <Setter Property="TextColor"        Value="White"    />
</Style>
```

The properties must be members of the `TargetType` class (or runtime exception)

Styles as Resources

- Styles are shareable, so they are generally defined as Resources

Define in a dictionary

```
<ContentPage.Resources>
  <ResourceDictionary>

    <Style x:Key="MyButtonStyle" TargetType="Button">
      ...
    </Style>

  </ResourceDictionary>
</ContentPage.Resources>
```

Using a Style

- Styles are set on a control through the **`Style`** property, this applies all the setters in the style to that control

```
<Button Text="OK"     Style="{StaticResource MyButtonStyle}" />
<Button Text="Cancel" Style="{StaticResource MyButtonStyle}" />
```

↑

The **`Style`** property is defined in the **`VisualElement`** base class so it is available in all views

Combining Styles and Resources

- Can use a resource as the **`Value`** for a setter, this lets it share a value with other styles

```
<Color x:Key="bgColor">White</Color>
<Color x:Key="fgColor">Black</Color>

<Style TargetType="Button" x:Key="AllButtons">
  <Setter Property="BackgroundColor" Value="{StaticResource  bgColor}" />
  <Setter Property="TextColor"       Value="{DynamicResource fgColor}" />
  ...
</Style>
```

Can use either static or dynamic lookup

Implicit Styles

- Styles can be automatically applied to all controls of a target type by omitting `x:Key` and placing the style into an accessible dictionary

```
<ContentPage.Resources>
   <ResourceDictionary>
      <Style TargetType="Button">
         <Setter Property="BackgroundColor" Value="Blue" />
         <Setter Property="BorderColor"     Value="Navy" />
           ...
      </Style>
   </ResourceDictionary>
</ContentPage.Resources>
```

The target type is still specified and is matched exactly, this style will be applied to all buttons in this page

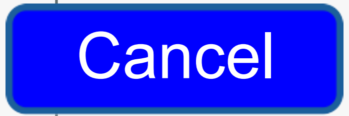
Overriding a setter

v Styles provide the *default* values, explicit property values on the control are applied *after* the style and take precedence

```
<Style x:Key="MyButtonStyle" TargetType="Button">
   <Setter Property="BackgroundColor" Value="Red" />
</Style>
```

```
<Button
   Style="{StaticResource MyButtonStyle}"
   BackgroundColor="Blue" ✓
   Text="Cancel"
   ... />
```

Value set directly overrules the style value



Background is blue, not red

Ancestor targeting

v A `Style` can target a base type of the object to which it is applied

This style targets `VisualElement`

```
<Style x:Key="MyVisualElementStyle" TargetType="VisualElement">
  <Setter Property="BackgroundColor" Value="#2A84D3" />
</Style>
```

```
<Button Style="{StaticResource MyVisualElementStyle}" ... />
```

Can apply to a button since the `Button` class is derived from `VisualElement`

Creating a Style in code

- Styles can be created in code to allow runtime customizations

```
var s = new Style(typeof(Button));

s.Setters.Add(new Setter {Property = Button.BackgroundColorProperty, Value = Color.Red});
s.Setters.Add(new Setter {Property = Button.BorderRadiusProperty,     Value = 4        });
```

Can then apply `Style` to a `Button` directly, or add it to the resources to apply in XAML

Motivation [repeated code]

v Styles often have duplicate Setters which are then hard to maintain

Repeated

```
<Style x:Key="MyButtonStyle" TargetType="Button">
  <Setter Property="BackgroundColor" Value="Blue"  />
  <Setter Property="BorderColor"     Value="Navy"  />
  <Setter Property="BorderWidth"     Value="5"     />
</Style>

<Style x:Key="MyEntryStyle" TargetType="Entry">
  <Setter Property="BackgroundColor" Value="Blue"  />
  <Setter Property="TextColor"       Value="White" />
</Style>
```

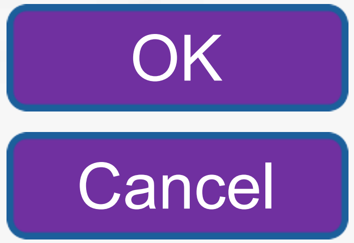
Motivation [customization]

- A provided Style might need some adjustment to meet your needs

```
<Style x:Key="MyButtonStyle" TargetType="Button">
  <Setter Property="BackgroundColor" Value="Blue" />
  ...
</Style>
```

Color might not be right for current use

```
<Button Style="{StaticResource MyButtonStyle}" Text="OK"      BackgroundColor="Purple" />
<Button Style="{StaticResource MyButtonStyle}" Text="Cancel" BackgroundColor="Purple" />
```



It is tedious to manually set properties that don't fit the current situation

Style inheritance

- A style can inherit from a base style

Base's **TargetType** must be the same or a base class

```
<Style x:Key="MyButtonStyle"      TargetType="Button" >
  ...
</Style>

<Style x:Key="DiscoButtonStyle"  TargetType="Button"  BasedOn="{StaticResource  MyButtonStyle}" >
  ...
</Style>
```

Indicates which style this will inherit from

Only **StaticResource** is allowed to set the base style

Inherited properties

v The new style can modify existing property values and/or add new ones

```
<Style x:Key="MyButtonStyle" TargetType="Button">
  <Setter Property="BackgroundColor"  Value="Blue"  />
  <Setter Property="BorderColor"      Value="Navy"  />
</Style>

<Style x:Key="DiscoButtonStyle" TargetType="Button" BasedOn="{StaticResource MyButtonStyle}">
   <Setter Property="BackgroundColor"  Value="Purple"  />
   <Setter Property="Rotation"         Value="30"      />
</Style>
```

Add new setter   Replace inherited setter

Motivation

- You will often need to share resources across multiple pages of your app; however, page-level resources are only available on one page

```
<ContentPage ... >

  <ContentPage.Resources>
    <ResourceDictionary>
      <Font x:Key="codeFont" FontFamily="..." />
    </ResourceDictionary>
  </ContentPage.Resources>
  ...
  <Label Font="{StaticResource codeFont}" />
  ...
</ContentPage>
```

```
<ContentPage ... >
    ...
    ...
    ...
    ...
    ...
    ...
    ...
    <Button Font="{StaticResource codeFont}" />
    ...
</ContentPage>
```

Resources defined in one page are not available in a different page

Available dictionaries

- `VisualElement` and `Application` have built-in resource dictionaries – these are initialized to `null` by default

```
public class VisualElement : ...
{ ...
   public ResourceDictionary Resources
   {
      get;
      set;
   }
}
```

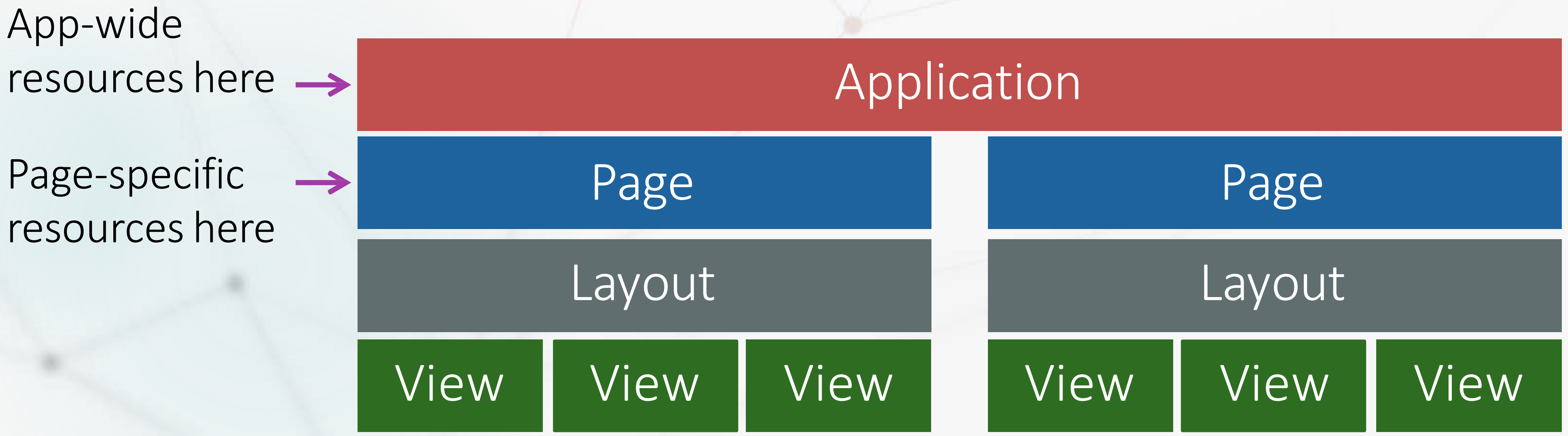
```
public class Application : ...
{ ...
   public ResourceDictionary Resources
   {
      get;
      set;
   }
}
```

Pages, layouts, and views inherit from `VisualElement`

Your app class inherits from `Application`

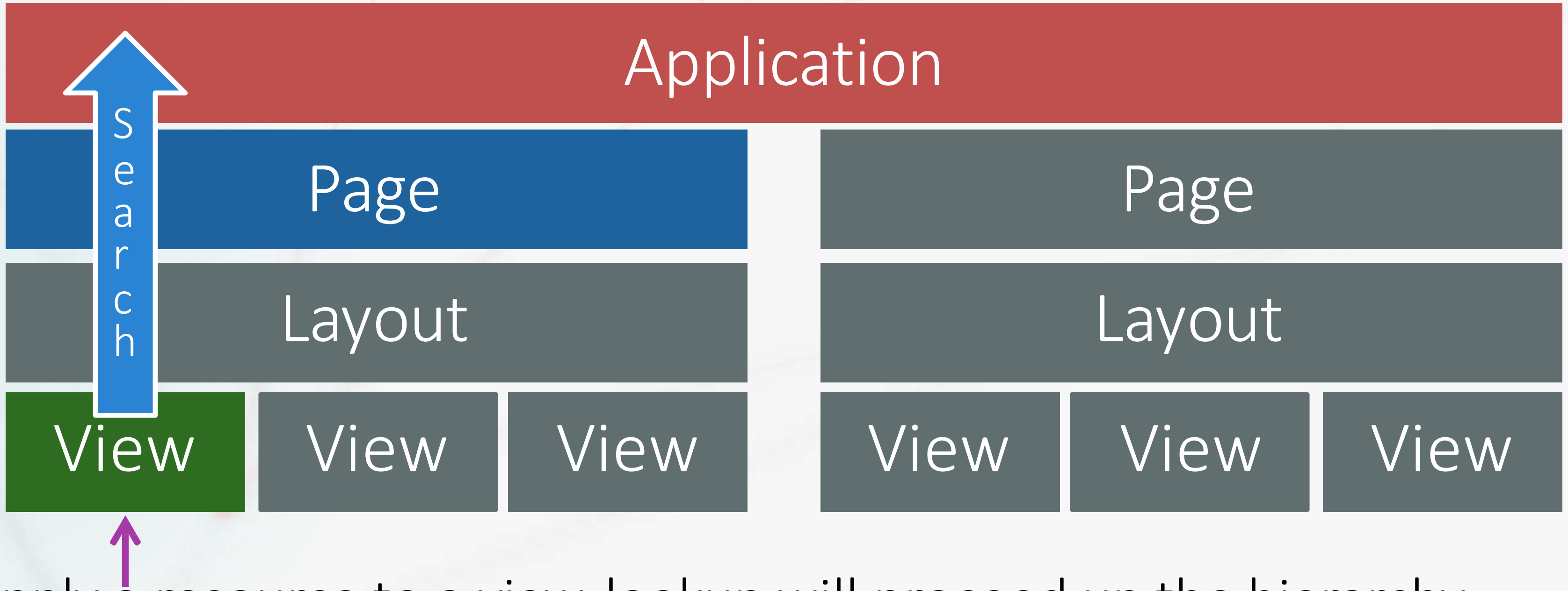
Resource scope

- Resources can be defined at different levels so they are scoped to a specific usage area in the application



Lookup rules

- Dictionaries are searched starting at the point a resource is applied, then up the visual tree to the Page, and finally to the App



Apply a resource to a view, lookup will proceed up the hierarchy

Place resources close to where they are used to minimize lookup cost

Defining application-level resources

- You code **App.xaml** and **App.xaml.cs** files in order to get an application-wide resource dictionary

App.xaml

```
<Application
  xmlns ="http://xamarin.com/schemas/2014/forms"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  x:Class="MyApp.App">

  <Application.Resources>
    <ResourceDictionary>
      <Font x:Key="codeFont" FontFamily="Courier New" />
    </ResourceDictionary>
  </Application.Resources>

</Application>
```

App.xaml.cs

```
namespace MyApp
{
  public partial class App : Application
  {
    public App()
    {
      InitializeComponent();
      MainPage = new MyPage();
    }
  }
}
```

Using application-level resources

- Can use either **`StaticResource`** or **`DynamicResource`** to apply an application-level resource

```
<ContentPage ...>
  ...
  <Label Font="{StaticResource codeFont}" />
  ...
</ContentPage>
```

```
<ContentPage ...>
  ...
  <Button Font="{StaticResource codeFont}"  />
  ...
</ContentPage>
```

The resource will be available in all pages of the app

Duplicate keys

- Keys can be repeated in different dictionaries, the first matching key on the search path is used

```
<Application.Resources>
  <ResourceDictionary>
    <x:String x:Key="msg">Two</x:String>
  </ResourceDictionary>
</Application.Resources>
```

App.xaml

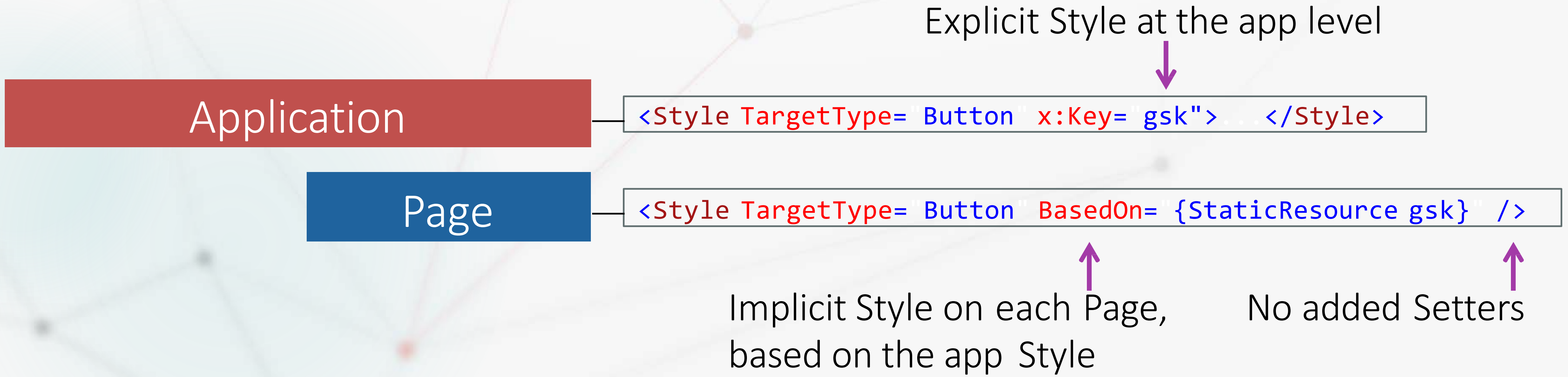
```
<ContentPage.Resources>
  <ResourceDictionary>
    <x:String x:Key="msg">One</x:String>
  </ResourceDictionary>
</ContentPage.Resources>

<Label Text="{StaticResource msg}">
```

MainPage.xaml

Text set to **One** →

Guideline for global styles

v Use explicit styles at the application level and then put an implicit style in each page that uses `BasedOn`

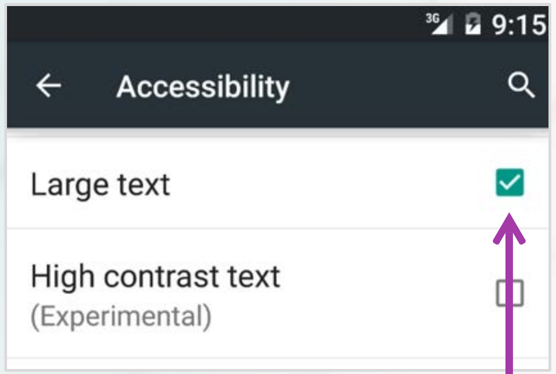

Explicit Style at the app level
Application
Page
Implicit Style on each Page, based on the app Style
No added Setters

```
<Style TargetType="Button" x:Key="gsk"> ... </Style>
<Style TargetType="Button" BasedOn="{StaticResource gsk}" />
```

This approach makes it clear which implicit Styles will be applied on each page

Why using device appearance ?

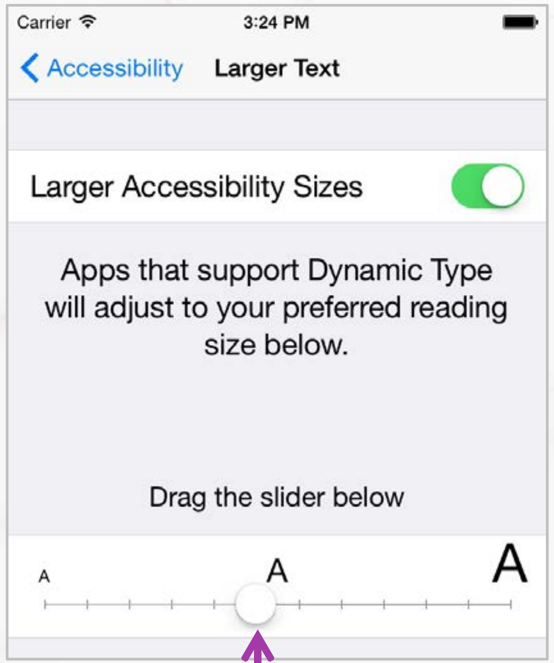
- Apps should respect the user's device-wide preferences for appearance and accessibility; ideally, apps update their UI when settings change


9:15

Accessibility

Large text ✓

High contrast text
(Experimental)

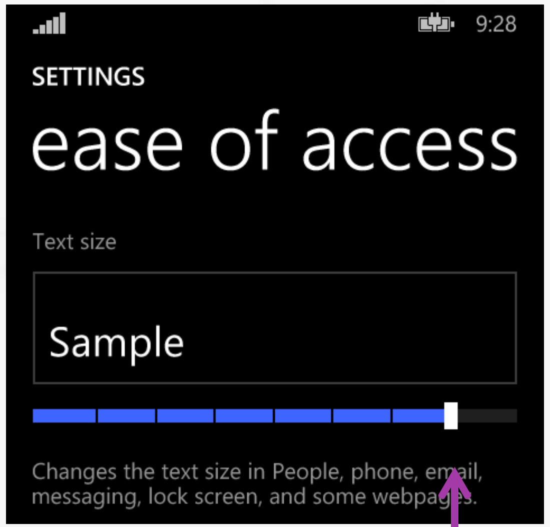

Carrier 3:24 PM

Accessibility Larger Text

Larger Accessibility Sizes

Apps that support Dynamic Type will adjust to your preferred reading size below.

Drag the slider below


9:28

SETTINGS

ease of access

Text size

Sample

Changes the text size in People, phone, email, messaging, lock screen, and some webpages.

Apps should try to use the text size the user requested

Control Template

ControlTemplate

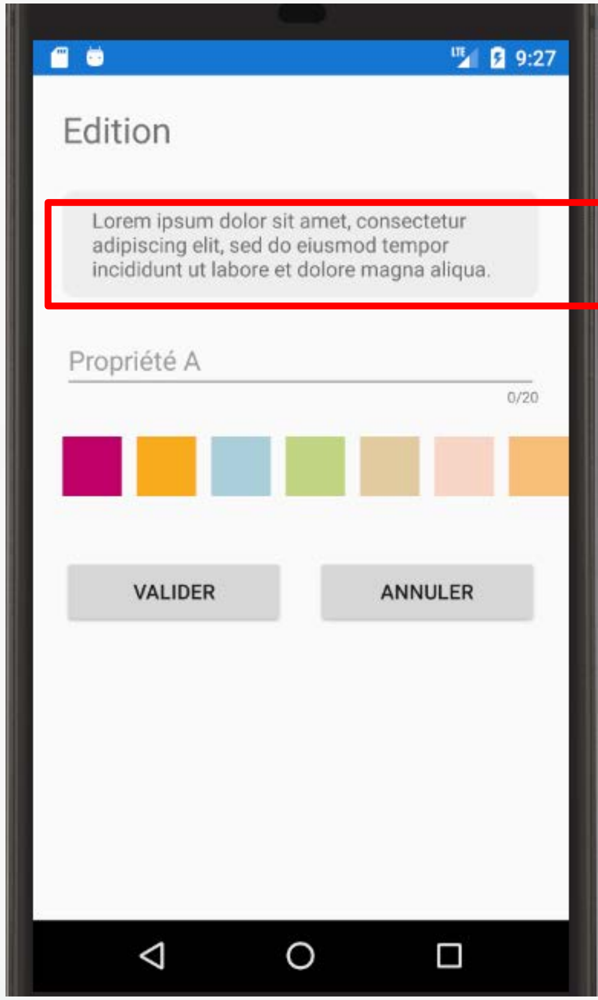
- Fonctionnalité avancée XAML
- Uniquement disponible sur certains contrôles
  - ContentView, ContentPage
- Un ControlTemple permet de modifier l'apparence d'un contrôle
  - Le contenu reste inchanger
- Eviter le copier/coller
- ContentPresenter

```xml
<ContentView ControlTemplate="{StaticResource HelpTextControlTemplate}">

</ContentView>
```

ControlTemplate

```
<Grid>
  <BoxView BackgroundColor="{StaticResource LightGrayColor}"
           CornerRadius="8" />
  <Label Text="..."
         LineBreakMode="WordWrap"
         Margin="20, 10"/>
</Grid>
```

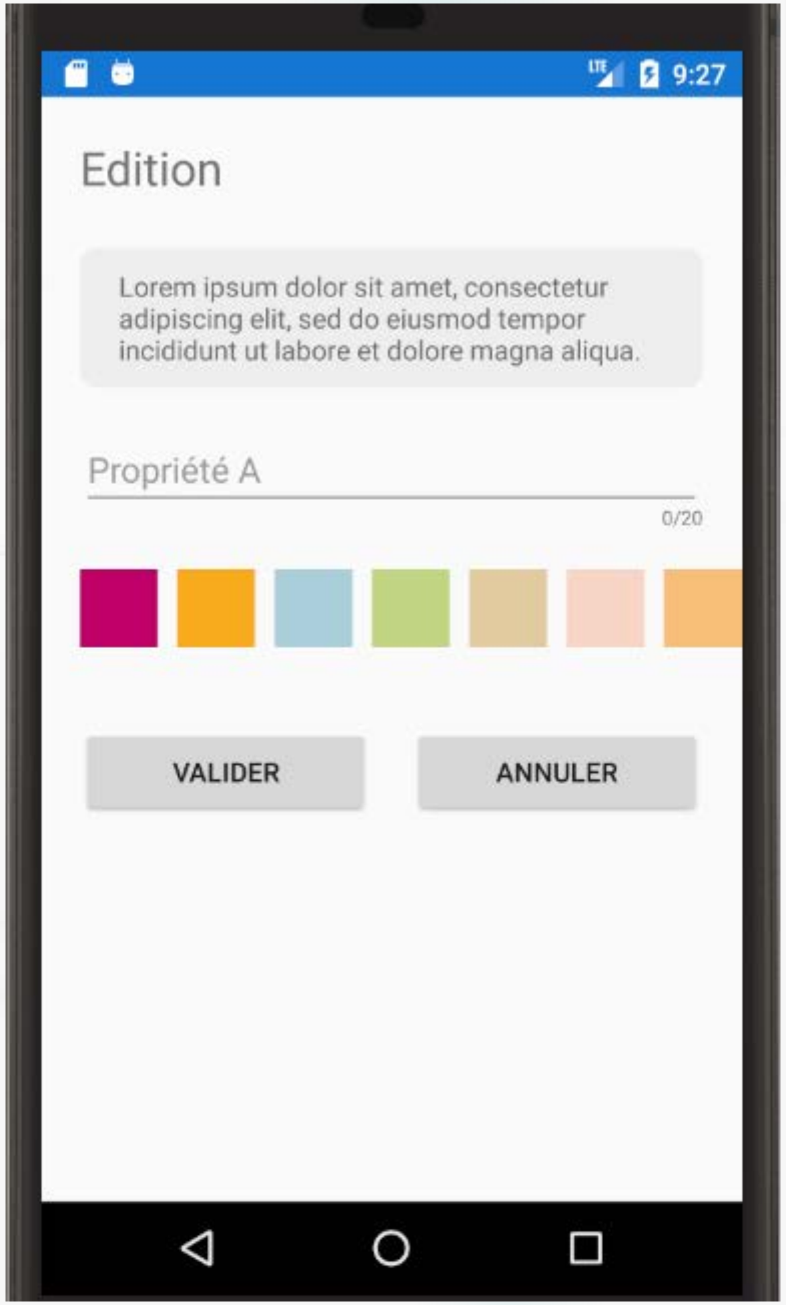
```
<ControlTemplate x:Key="MonControlTemplate">
  <Grid>
    <BoxView BackgroundColor="{StaticResource LightGrayColor}"
             CornerRadius="8" />
    <ContentPresenter Padding="20, 10" />
  </Grid>
</ControlTemplate>

<ContentView ControlTemplate="{StaticResource MonControlTemplate}">
  <Label Text="…" LineBreakMode="WordWrap"/>
</ContentView>
```



Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Atelier Style

- Reprendre l’atelier Layout et inclure :
  - Resource
  - Style
  - ControlTemplate

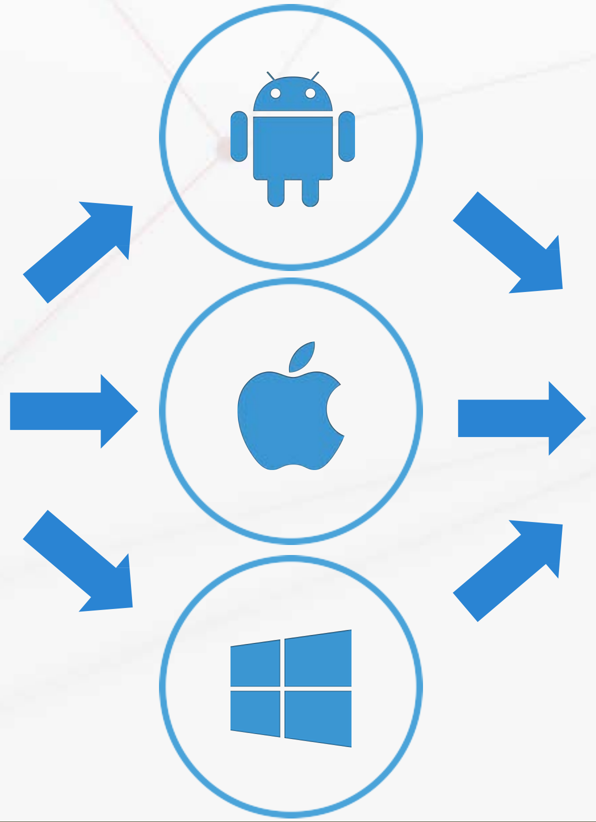


Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

What is a built-in Style?

- Xamarin.Forms maps the user's device-wide preferencesto Styles, it keeps those Styles updated as the user changes their settings

Xamarin.Forms asks the OS for user preferences



It loads the results into code-generated `Styles` which you can use

Built-in Styles are under development, please expect changes and additions.

Implementation

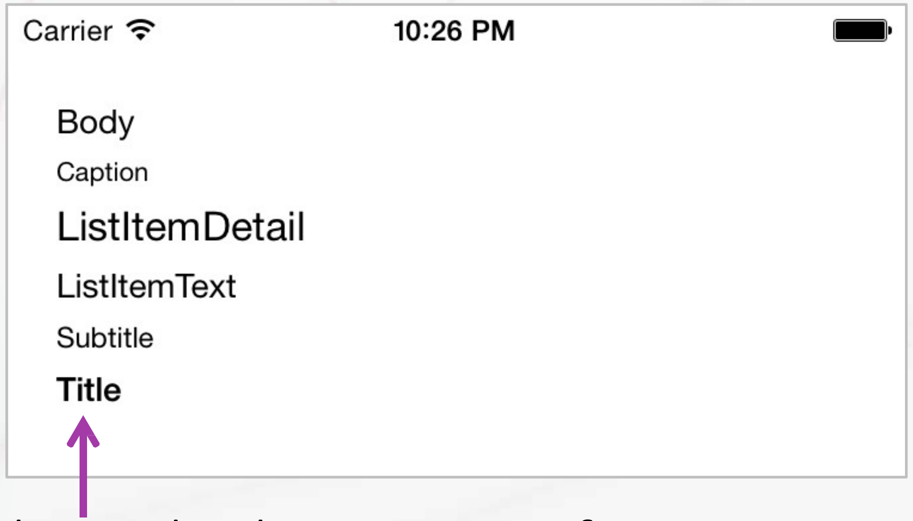
- The built-in styles are provided as **`Style`** objects in **`Device.Styles`**

```
public static class Styles
{ ...
  public static readonly Style BodyStyle;
  public static readonly Style CaptionStyle;
  public static readonly Style ListItemDetailTextStyle;
  public static readonly Style ListItemTextStyle;
  public static readonly Style SubtitleStyle;
  public static readonly Style TitleStyle;
}
```

Styles are for common UI like titles, body text, and lists

Targets

- The built-in Styles use a `TargetType` of `Label`


Carrier
10:26 PM
Body
Caption
ListItemDetail
ListItemText
Subtitle
Title

The Styles have setters for common properties such as fonts and colors

Resource keys

- Symbolic constants from **`Device.Styles`** identify the built-in Styles in XAML

```
public static class Styles
{ ...
  public static readonly string BodyStyleKey               = "BodyStyle";
  public static readonly string CaptionStyleKey            = "CaptionStyle";
  public static readonly string ListItemDetailTextStyleKey = "ListItemDetailTextStyle";
  public static readonly string ListItemTextStyleKey       = "ListItemTextStyle";
  public static readonly string SubtitleStyleKey           = "SubtitleStyle";
  public static readonly string TitleStyleKey              = "TitleStyle";
}
```

You use these in your XAML

Using a built-in Style

- Must use **DynamicResource** to access a built-in Style

```
public static class Styles
{ ...
  public static readonly string TitleStyleKey = "TitleStyle";
}
```

Use the predefined string resource key

```
<Label Text="Welcome" Style="{DynamicResource TitleStyle}" />
```

DynamicResource is required because these styles are generated via code and can change at runtime if the user changes their preferences

Customizing built-in Styles

- **BaseResourceKey** lets you use a built-in Style as a base, it performs a dynamic lookup which keeps the property values synchronized to the user preferences

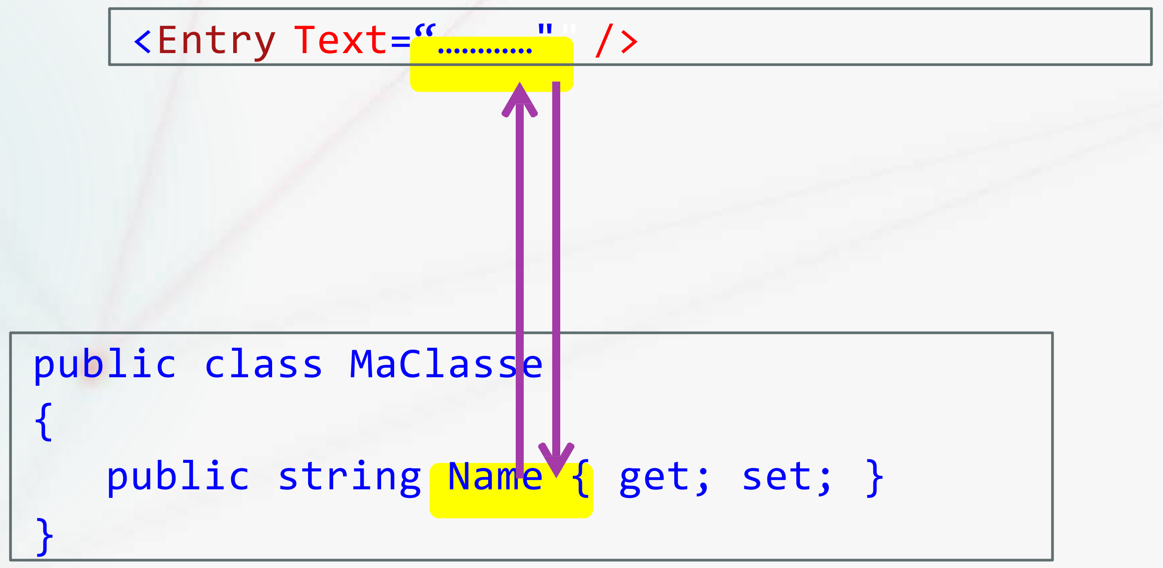
```
<Style BaseResourceKey="TitleStyle" TargetType="Label" x:Key="MyTitleStyle">
  ...
</Style>
```

Property identifies the Resource to use as the **BasedOn** style
(i.e. you are supplying a key that will be used for Resource lookup)

Binding & MVVM

Binding

- Le Binding est une notion centrale pour les applications XAML
  - On les utilise PARTOUT !
- Un binding est une liaison entre :
  - Une propriété d’un objet (la source de données)
  - Une propriété d’un contrôle (Page, Layout, View…)



```
<Entry Text=“............" /> 
```

```
public class MaClasse
{
    public string Name { get; set; }
}
```

Binding

- Toutes les propriétés des contrôles sont « bindables »

```
<Entry Text="{Binding Name}" />

<ListView ItemsSource="{Binding ListeElements}"
          SelectedItem="{Binding ElementSelectionne, Mode=TwoWay" />

<Button IsEnabled="{Binding IsBusy}" />

<Label Text="{Binding ErrorMessage}" IsVisible="{Binding IsBusy}"
```

- La propriété BindingContext permet de définir la source de données d’un contrôle (héritage)

```
var maSourceDeDonnees = new Book();
maSourceDeDonnees.Author = "AZZSD QSdQSD";

monText.BindingContext = maSourceDeDonnees;
```

Et comment on utilise tout ca ?

- Explication par l’exemple
  - MVVM
  - Comment marche le binding birectionnel ?
  - Utilisation des commandes
  - Dependency Property
  - Découpage d’une application
  - …

Binding bi-directionnel

- Mode de Binding

```
<Entry Text="{Binding FirstName, Mode=TwoWay}" >
```

- La source de données **doit** implémenter l’interface INotifyPropertyChanged

```
public class ViewModelBase : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    protected void OnPropertyChanged([CallerMemberName] string propertyName = null)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }
}

public class MonViewModel : ViewModelBase
{
    private string _firstName;

    public string FirstName
    {
        get { return _firstName; }
        set
        {
            if (_firstName != value)
            {
                _firstName = value;
                OnPropertyChanged();
            }
        }
    }
}
```

Source Binding

- Utiliser un autre élément comme source de données

```
<Entry x:Name="monEntry" />


<Label Text="{Binding Text, Source={Reference monEntry}" />
```

DependencyProperty

- Seule les DependencyProperty sont bindables

```
public static readonly BindableProperty BorderWidthProperty =
BindableProperty.Create("BorderWidth", typeof(int), typeof(SliderImage), 1, BindingMode.OneWay,
null);

public int BorderWidth
{
   get { return (int)GetValue(BorderWidthProperty); }
   set { SetValue(BorderWidthProperty, value); }
}
```

- Ecrire un Binding en C#

```
<Label IsVisible="{Binding HasError, Mode=OneWay}" />

_label.SetBinding(IsVisibleProperty, new Binding("HasError", BindingMode.OneWay));
```

Converter

```
public class StringToBooleanConverter : IValueConverter
    {
        public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
        {
            if (value == null)
                return false;

            return String.IsNullOrWhiteSpace((string)value) == false;
        }

        public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
        {
            throw new NotImplementedException();
        }
    }
```

```
<tc:StringToBooleanConverter x:Key="StringToBooleanConverter" />

<Label Text="{Binding Message}"
       IsVisible="{Binding Message, Converter={StaticResource StringToBooleanConverter}}" />
```

Les commandes

- Les commandes sont des DependencyProperty qui sont appelées lorsqu’un évènement est appelé

```
<Button Text="Valider" Command="{Binding SaveCommand}" />
```

```
public class MonViewModel
{
    public string FirstName { get; set; }

    public ICommand SaveCommand { get; private set; }

    public MonViewModel()
    {
        SaveCommand = new Command(Save, CanSave);
    }

    private void Save(object obj)
    {
        // ...
    }

    private bool CanSave(object arg)
    {
        return string.IsNullOrEmpty(FirstName) == false;
    }
}
```

La classe Xamarin.Forms.Command est l’implémentation par défaut de Xamarin de l’interface System.Windows.Input.ICommand

Définir un paramètre à une commande

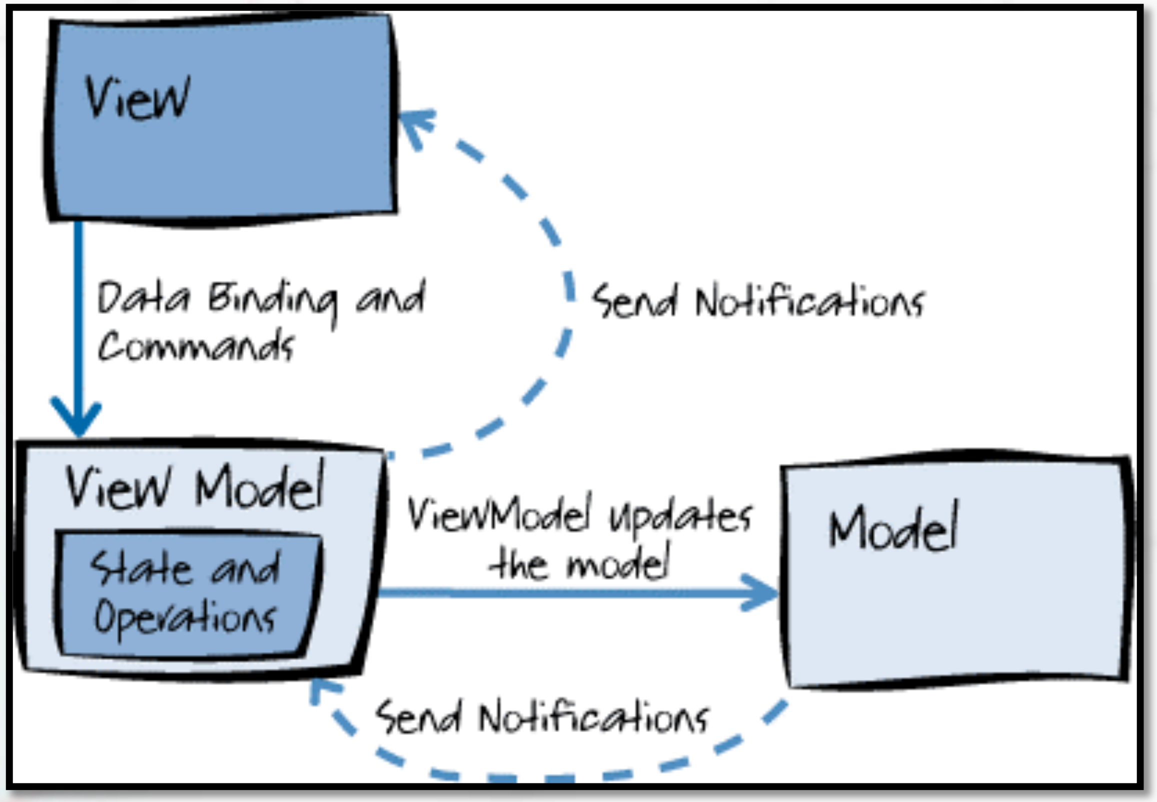
- Les commandes peuvent avoir un paramètre

```
<ListView x:Name="maListe">
            <!--...-->
</ListView>

<Button  Text="Supprimer"
         Command="{Binding RemoveCommand}"
         CommandParameter="{Binding SelectedItem, Source={Reference maListe}}" />
```

```
private void Remove(object selectedItem)
{
            // ...
}
```

Le paramètre est la valeur de la propriété CommandParameter, donc l'élément sélectionné de la liste

Pattern MVVM



Utiliser un package MVVM

- MVVM Light
- MvvmCross

Atelier Binding et MVVM

- Créer un formulaire de Login en MVVM
  - Deux champs de saisie : Login et Password
  - Un bouton « Se connecter »
  - Un message d’erreur si le login ou le password ne sont pas correctement renseignés
  - Pour simuler que tout ce passe bien, on affichera un message dans une popup.

Affichage de liste

- Il existe plusieurs manières d’afficher une liste d’éléments :
  - ListView (prochainement remplacée par CollectionView)
  - BindableLayout
  - Contrôle personnalisé
- ListView vs BindableLayout
  - ListView
    - Avantages : groupe, sélection, évènement tap, virtualisation, interaction, reload, header, footer
    - Inconvénients : performances, Stack uniquement customisation graphique
  - BindableLayout
    - Avantages : très léger, flexible, marche avec tous les types de Layout
    - Inconvénients : pas de virtualisation, pas de sélection, pas d’évènement tap

Utilisation d’une ListView

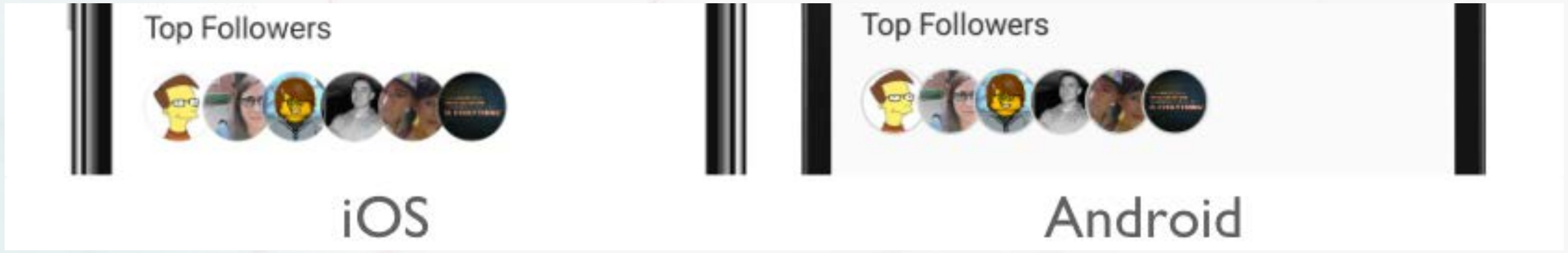
```
<ListView ItemsSource="{Binding Employes}"
        SelectionMode="None">
    <ListView.ItemTemplate>
        <DataTemplate>
            <TextCell Text="{Binding FirstName}" />
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>
```

```
<ListView ItemsSource="{Binding Employes}"
    SelectionMode="None">
    <ListView.ItemTemplate>
        <DataTemplate>
            <ViewCell>
                <StackLayout>
                    <Label Text="{Binding FirstName}" />
                    <Label Text="{Binding LastName}" />
                </StackLayout>
            </ViewCell>
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>
```

```
public class EmployeListViewModel : ViewModelBase
{
    private List<Employe> _employes;

    public List<Employe> Employes
    {
        get { return _employes; }
        private set { Set(ref _employes, value); }
    }

    public EmployeListViewModel()
    {
        var employes = new List<Employe>();
        employes.Add(new Employe { FirstName = "Lois",…});
        employes.Add(new Employe { FirstName = "Cyril",… });
        employes.Add(new Employe { FirstName = "Guillaume",… });
        employes.Add(new Employe { FirstName = "Anael", … });
        employes.Add(new Employe { FirstName = "Julien", … });

        Employes = employes;
    }
}
```

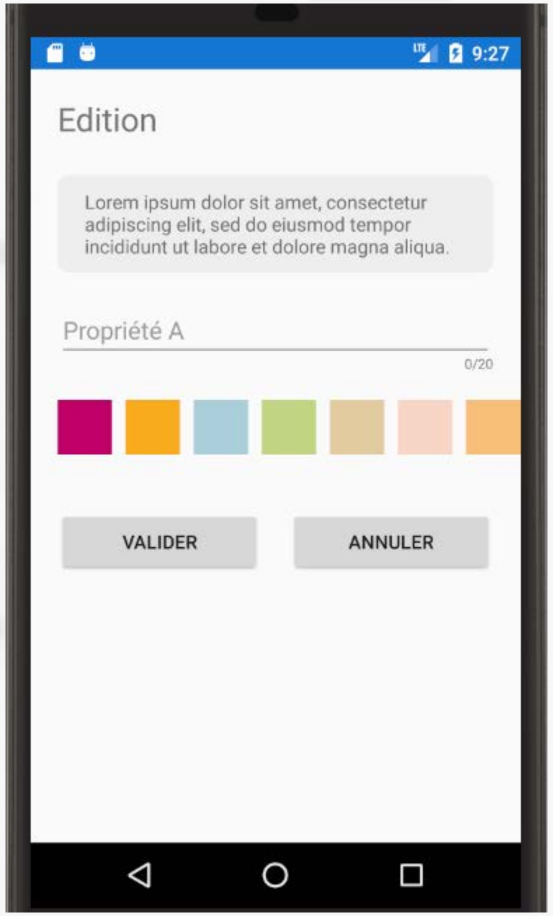
Utilisation du BindableLayout

```
<StackLayout BindableLayout.ItemsSource="{Binding User.TopFollowers}" Orientation="Horizontal">
    <BindableLayout.ItemTemplate>
        <DataTemplate>
            <controls:CircleImage Source="{Binding}"
                                  Aspect="AspectFill" WidthRequest="44" HeightRequest="44" />
        </DataTemplate>
    </BindableLayout.ItemTemplate>
</StackLayout>
```



Atelier

- Reprendre l’atelier Layout
- Utiliser un BindableLayout pour afficher la liste des Couleurs

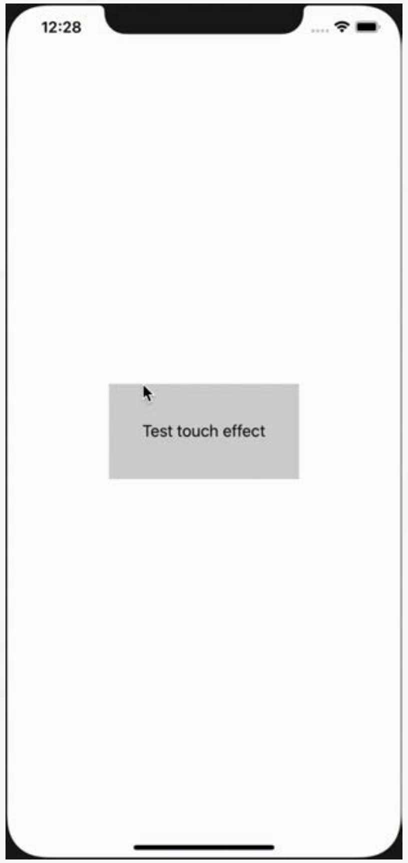

9:27
Edition
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Propriété A
0/20
VALIDER
ANNULER

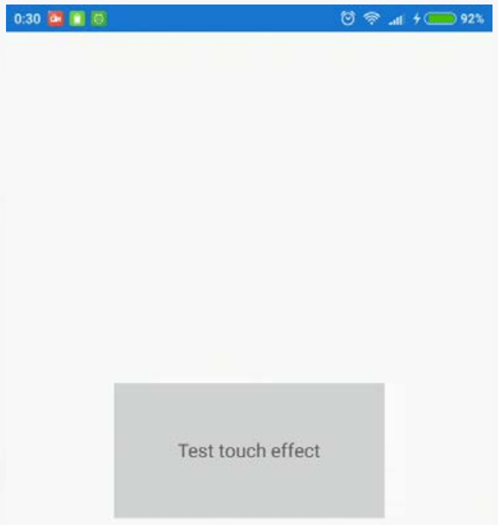
XamEffects

XamEffects

- Package nugget XamEffects
- Permet d'ajouter de gérer le Tap et le feedback utilisateur
  - iOS et Android

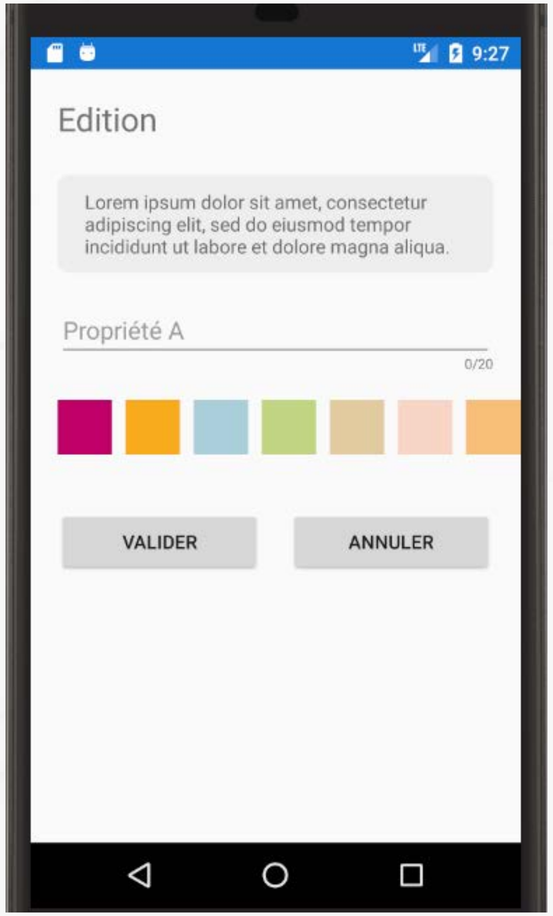
```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        xmlns:local="clr-namespace:XamEffects.Sample"
        xmlns:xe="clr-namespace:XamEffects;assembly=XamEffects"
        x:Class="XamEffects.Sample.MainPage">
    <Grid HorizontalOptions="Center"
    VerticalOptions="Center"
    HeightRequest="100"
    WidthRequest="200"
    BackgroundColor="LightGray"
    xe:TouchEffect.Color="Red">
        <Label Text="Test touch effect"
        HorizontalOptions="Center"
        VerticalOptions="Center"/>
    </Grid>
</ContentPage>
```





Démo

- Ajouter la sélection sur le BindableLayout



Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

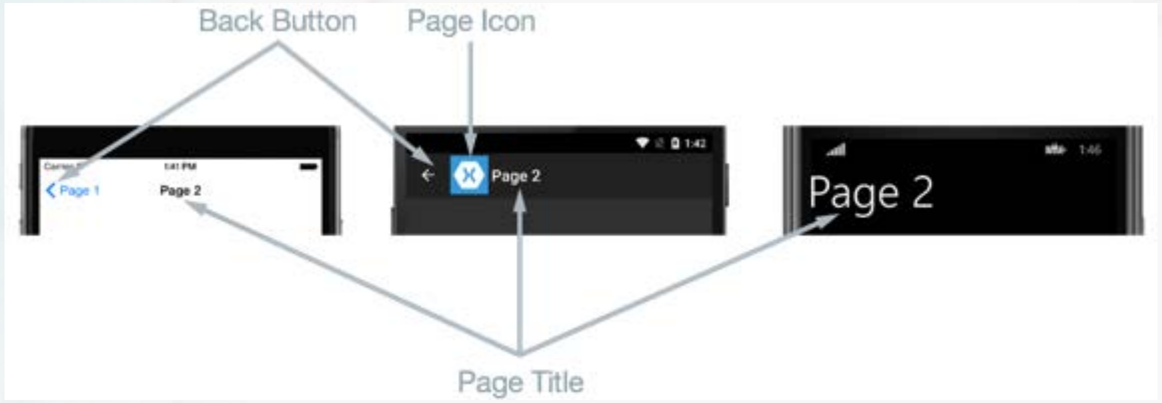
Navigation

Navigation hiérarchique

- Xamarin propose un service de navigation via les NavigationPage

```
MainPage = new NavigationPage(new MainPage());
```

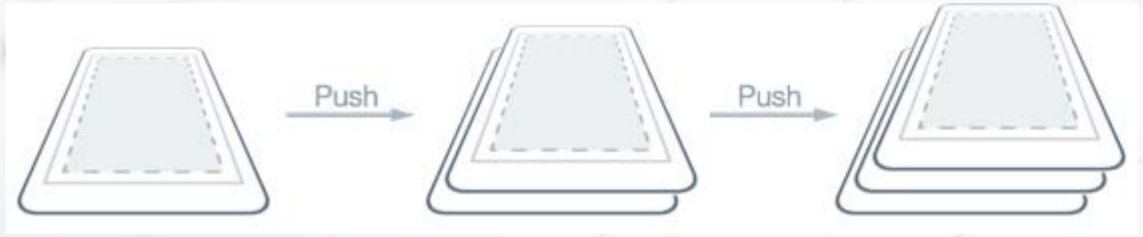
- La NavigationPage permet de
  - Naviguer entre les pages
  - Gère le Back
  - Ajoute un header



Naviguer

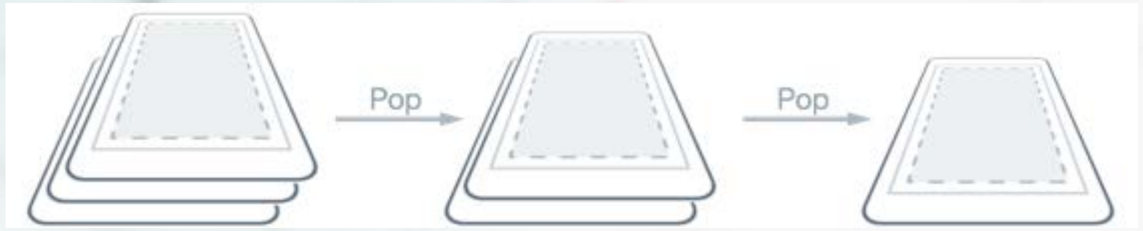
- Naviguer vers une page

```
App.Current.MainPage.Navigation.PushAsync (new MaPage())
```

- 

- Revenir à la page précédente

```
App.Current.MainPage.Navigation.PopAsync (…)
```



Atelier

- Reprendre l’atelier précédent
- Créer deux nouvelles pages
  - Une page avec une ListView que vous remplissez avec des Employes (nom/prénom)
    - Lorsque l’utilisateur clique sur un employé, l’application navigue ver la page d’édition
  - Une page avec l’édition d’un employé (nom/prénom et un bouton enregistrer)
    - Le Back retour à la liste
- Discussion & problématiques
  - Il manquerait pas une Commande ?
  - Comment naviguer depuis un ViewModel
  - Comment passer des paramètres lors de la navigation

Créer un service de navigation

- Customiser la ListView
- ViewModel First

Xamarin Essential

Xamarin Essential

- Fournit un tas de fonctionnalités cross-plateforme

Accelerometer – Retrieve acceleration data of the device in three dimensional space.

App Information – Find out information about the application.

Barometer – Monitor the barometer for pressure changes.

Battery – Easily detect battery level, source, and state.

Clipboard – Quickly and easily set or read text on the clipboard.

Color Converters – Helper methods for System.Drawing.Color.

Compass – Monitor compass for changes.

Connectivity – Check connectivity state and detect changes.

Detect Shake – Detect a shake movement of the device.

Device Display Information – Get the device's screen metrics and orientation.

Device Information – Find out about the device with ease.

Email – Easily send email messages.

File System Helpers – Easily save files to app data.

Flashlight – A simple way to turn the flashlight on/off.

Geocoding – Geocode and reverse geocode addresses and coordinates.

Geolocation – Retrieve the device's GPS location.

Gyroscope – Track rotation around the device's three primary axes.

Launcher – Enables an application to open a URI by the system.

Magnetometer – Detect device's orientation relative to Earth's magnetic field.

MainThread – Run code on the application's main thread.

Maps – Open the maps application to a specific location.

Open Browser – Quickly and easily open a browser to a specific website.

Orientation Sensor – Retrieve the orientation of the device in three dimensional space.

Phone Dialer – Open the phone dialer.

Platform Extensions – Helper methods for converting Rect, Size, and Point.

Preferences – Quickly and easily add persistent preferences.

Secure Storage – Securely store data.

Share – Send text and website uris to other apps.

SMS – Create an SMS message for sending.

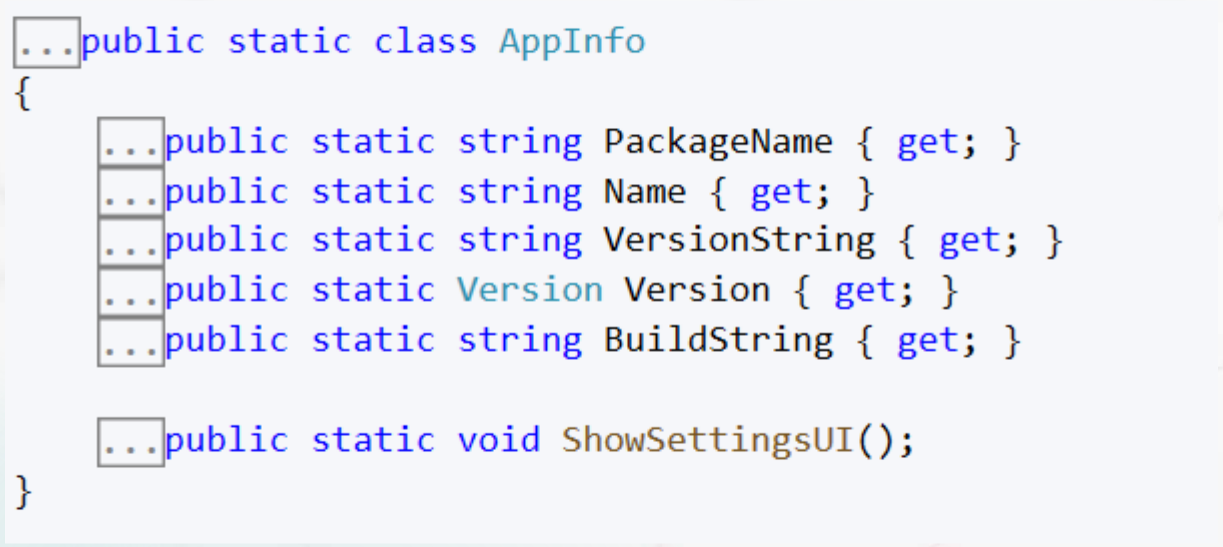
Text-to-Speech – Vocalize text on the device.

Unit Converters – Helper methods to convert units.

Version Tracking – Track the applications version and build numbers.

Vibrate – Make the device vibrate.

Xamarin.Essentials.AppInfo



```
...public static class AppInfo
{
    ...public static string PackageName { get; }
    ...public static string Name { get; }
    ...public static string VersionString { get; }
    ...public static Version Version { get; }
    ...public static string BuildString { get; }

    ...public static void ShowSettingsUI();
}
```

Etat de la connexion

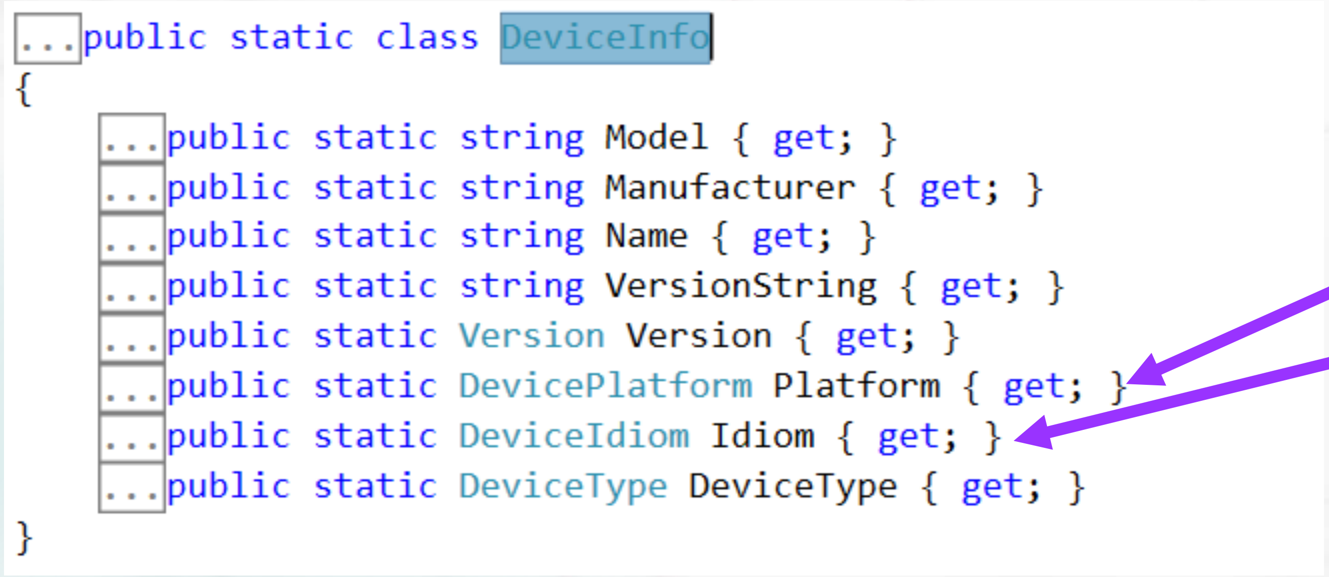
- Permet de connaitre l’état de la connexion Data

```
if (Xamarin.Essentials.Connectivity.NetworkAccess == NetworkAccess.Internet)
{
    // Internet
}
```

- Permet de savoir lorsque l’état de la connexion Data change

```

Xamarin.Essentials.Connectivity.ConnectivityChanged += Connectivity_ConnectivityChanged;

private void Connectivity_ConnectivityChanged(object sender,
Xamarin.Essentials.ConnectivityChangedEventArgs e)
{
    if (e.NetworkAccess == Xamarin.Essentials.NetworkAccess.None)
    {
            // Not Internet
    }
}
```

Device Information



```
...public static class DeviceInfo
{
    ...public static string Model { get; }
    ...public static string Manufacturer { get; }
    ...public static string Name { get; }
    ...public static string VersionString { get; }
    ...public static Version Version { get; }
    ...public static DevicePlatform Platform { get; }
    ...public static DeviceIdiom Idiom { get; }
    ...public static DeviceType DeviceType { get; }
}
```

iOS, Android …

Phone / Tablet / TV / Desktop / Watch

En vrac

- Ouvrir le navigateur

```
Launcher.OpenAsync("http://www.ai3.fr")
```

- Envoyer un mail

```
Email.ComposeAsync("Sujet", "Bonjour, ...")
```

- Exécuter une action sur le ThreadPrincipal

```
MainThread.BeginInvokeOnMainThread(() => MonAction())
```

- Appeler un numéro, envoyer des sms

```
PhoneDialer.Open("0642515648");
Sms.ComposeAsync(…)
```

- Ajouter / Modifier des préférences

```
Preferences.Set("PreviousLoggedUser", "Cyril");
var previousUser = Preferences.Get("PreviousLoggedUser");
```

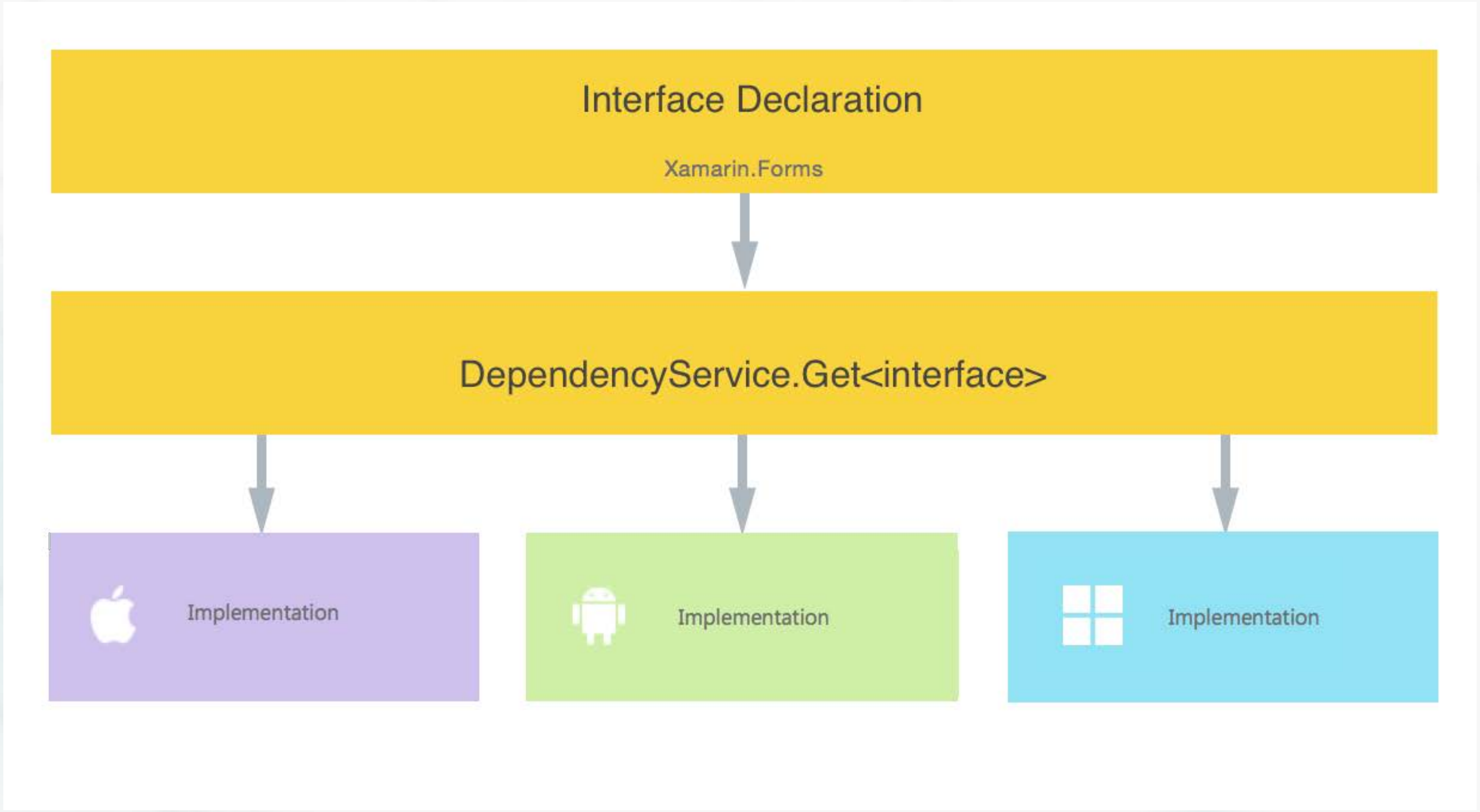
- SecureStorage, Location, Niveau de baterie, Boussole, Lampe torche, Vibration ….

Atelier

- Reprendre l’atelier sur la navigation
- Dans la page de Login
  - Enregistrer le userName dans les Préférences
  - Au chargement, remplir le champs userName avec celui enregistré dans le Préférences.

Injection de dépendances

Architecture



```
public interface ITextToSpeech
{
      void Speak(string text);
}
```

PCL

```
[assembly: Xamarin.Forms.Dependency(typeof(TextToSpeechImplementation))]
namespace App6.UWP
{
   public class TextToSpeechImplementation : ITextToSpeech
   {
      public TextToSpeechImplementation() { }

      public async void Speak(string text)
      {
                    //
      }
   }

}
```

Windows

```
private void Usage(string result)
{
   DependencyService.Get<ITextToSpeech>().Speak("My result is : " + result);
}
```

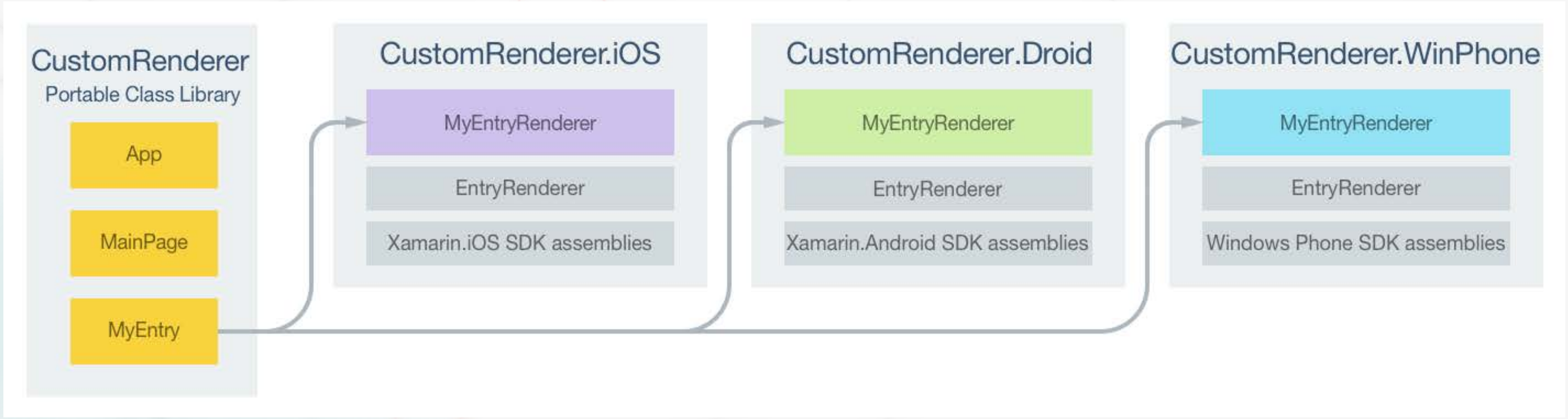
PCL

Custom Renderer

Custom Renderer

Views	Renderer	iOS	Android	Android (AppCompat)	Windows Phone 8	WinRT / UWP
ActivityIndicator	ActivityIndicatorRenderer	UIActivityIndicator	ProgressBar		ProgressBar	ProgressBar
BoxView	BoxRenderer (iOS and Android) BoxViewRenderer (Windows Phone and WinRT)	UIView	ViewGroup		Rectangle	Rectangle
Button	ButtonRenderer	UIButton	Button	AppCompatButton	Button	Button
CarouselView	CarouselViewRenderer	UIScrollView	RecyclerView		FlipView	FlipView
DatePicker	DatePickerRenderer	UITextField	EditText		DatePicker	DatePicker
Editor	EditorRenderer	UITextView	EditText		TextBox	TextBox
Entry	EntryRenderer	UITextField	EditText		PhoneTextBox/PasswordBox	TextBox
Image	ImageRenderer	UIImageView	ImageView		Image	Image

Exemple CustomEntry



Exemple CustomEntry

```
using Xamarin.Forms.Platform.Android;

[assembly: ExportRenderer (typeof(MyEntry), typeof(MyEntryRenderer))]
namespace CustomRenderer.Android
{
    class MyEntryRenderer : EntryRenderer
    {
        protected override void OnElementChanged (ElementChangedEventArgs<Entry> e)
        {
            base.OnElementChanged (e);

            if (Control != null) {
                Control.SetBackgroundColor (global::Android.Graphics.Color.LightGreen);
            }
        }
    }
}
```

Asynchronisme

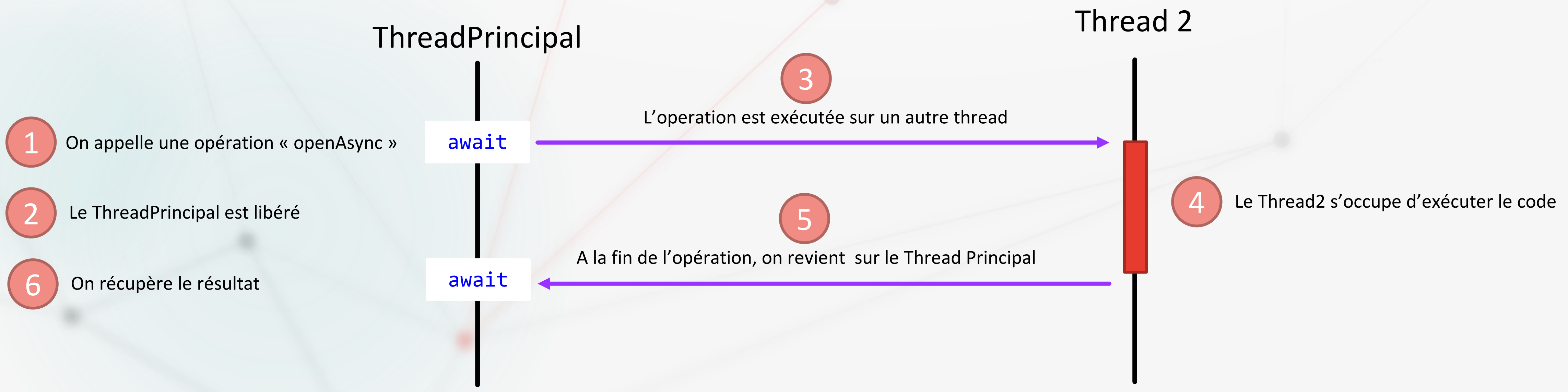
Asynchronisme

- Une application s'exécute sur un thread : le thread principal
- Toute opération qui bloque ou ralenti le thread principal va inévitablement bloquer l'interface utilisateur
  - L'application se fige voir peut planter
  - Expérience utilisateur catastrophique
- L'asynchronisme
  - Si une opération pénalise le thread principal, alors on l'exécute sur un autre Thread
  - Exemple faire une requête Web peut prendre plusieurs secondes
  - D'une manière générale toutes les entrées/sorties sont problématiques
  - La plupart des développeurs ne sont pas sensibles à cette problématique

Async / Await : le rêve !

- Les mots clés async/await :
  - Simplifient la vie des développeurs en « asynchronisant » certaines tâches
  - .NET Standard toutes les I/O utilisent ce mécanisme (HTTP, File, Stream…)

```
var response = await client.GetAsync($"http://www.monapi.com/api/employe/{employeId}"));
```


ThreadPrincipal
Thread 2
1 On appelle une opération « openAsync »
2 Le ThreadPrincipal est libéré
6 On récupère le résultat
await
await
3 L'operation est exécutée sur un autre thread
5 A la fin de l'opération, on revient sur le Thread Principal
4 Le Thread2 s'occupe d'exécuter le code

Async / Await : la réalité

- NON ! Les mots-clé async/await ne permettent pas de lancer une opération en async
- Ils permettent de simplifier l'appel à une opération async
  - Masque le changement de contexte, la gestion des erreurs et le retour sur le ThreadPrincipal
  - Rend la programmation linéaire (évite les callbacks liés à l'utilisation de thread)
  - L'opération n'est en fait asynchrone que lorsque le développeur lance une System.Theading.Task

Démo

- Await Async

Les bonnes pratiques

```
public async Task<TResult> AsyncRun<TResult>(Func<TResult> action)
{
    IsBusy = true;

    try
    {
        return await Task.Run(() =>
        {
            return action();
        });
    }
    catch (Exception ex)
    {
        // Gérer les erreurs ici !
        return default(TResult);
    }
    finally
    {
        IsBusy = false;
    }
}
```

Gérer l'état Busy

Utiliser une Task afin de garantir que tout le traitement est async

Intercepter les exceptions

```
var result = await AsyncRun(() =>
{
    Thread.Sleep(1000);
    return true;
});
```

Attention

- Il est interdit de modifier l'interface utilisateur depuis un autre Thread que le thread principal

```
var result = await AsyncRun(() =>
{
    Thread.Sleep(1000);

    IsBusy = false;

    return true;
});
```

Si la propriété IsBusy est bindé, votre application peut lever une erreur fatale

- Deux solutions
  1. Mettre à jour les propriétés de ViewModel après le traitement asynchrone
  2. Utiliser Xamarin.Essentials.MainThread.BeginInvokeOnMainThread(() => IsBusy = false);

Gestion des fichiers

Limites du Storage

- Votre application s'exécute dans une SandBox
  - Elle ne peut pas accéder n'importe quoi
  - Nécessite des permissions spécifiques
- Chemin d'accès au Storage réservé à l'Application
  - Xamarin.Essentials.FileSystem.AppDataDirectory
  - Xamarin.Essentials.FileSystem.CacheDirectory
- Ouvrir un asset de votre application
  - Xamarin.Essentials.FileSystem.OpenAppPackageFileAsync(…)
  - Android : AndroidAsset
  - iOS : BundledResource
  - UWP : Content
- Ouvrir une resource de votre application
  - EmbeddedResource

System.IO

- .NET standard

```
// Ecrire un fichier
File.WriteAllText(fileName, text);

// Lire un fichier
string text = File.ReadAllText(fileName);

// Vérifier qu'un fichier existe
bool doesExist = File.Exists(fileName);

// Lister les fichiers d'un répertoire
string[] files = Directory.GetFiles(folder);

// Créer un répertoire
Directory.CreateDirectory(folder);
```

Stream, Writer, Reader

- Aucun changement par rapport à une application Framework .NET
  - FileStream
  - MemoryStream
  - StreamWriter / StreamReader
  - BinaryWriter / BinaryReader

Json

- Package nugget Json.NET – Newtonsoft

```
// Sérialise un objet en json
var monObjet = new Employe { ... };
var json = Newtonsoft.Json.JsonConvert.SerializeObject(monObjet);

// Désérialise un objet
var employe = Newtonsoft.Json.JsonConvert.DeserializeObject<Employe>(json);
```

HttpService

System.Net.Http

- .NET standard

```
using (var client = new System.Net.Http.HttpClient())
{
    client.DefaultRequestHeaders.Add("bearer", accessToken);

    using (var response = await client.GetAsync($"http://www.monapi.com/api/employe/{employeId}"))
    {
        if (response.IsSuccessStatusCode)
        {
            var json = await response.Content.ReadAsStringAsync();
            return JsonConvert.DeserializeObject<Employe>(json);
        }
    }
}
```

```
using (var client = new System.Net.Http.HttpClient())
{
    var json = JsonConvert.SerializeObject(employe);
    var content = new StringContent(json, Encoding.UTF8, "application/json");
    using (var response = await client.PostAsync("http://www.monapi.com/api/employe", content))
    {
        return response.IsSuccessStatusCode;
    }
}
```

iOS

Reminder: developmentsetup

v You must have the following to build iOS apps:

Mac running OS X

with the latest version of Xcode

Xamarin tools on all your development machines (both Mac and Windows)

What is included in Xamarin.iOS?

v Xamarin.iOS includes both compile-time and runtime components

C# compiler for Mac

Native compiler and linker

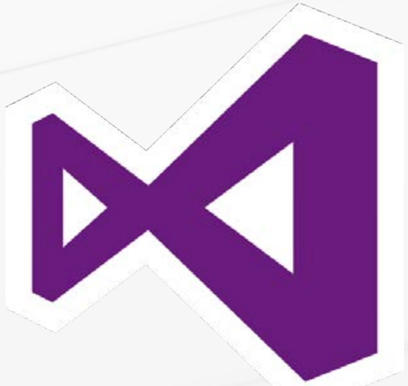
Runtime services (GC, type checking, etc.)

Core .NET Libraries

Choose your IDE

- Xamarin allows you to build iOS applications using C# / .NET with either

Visual Studio on Windows

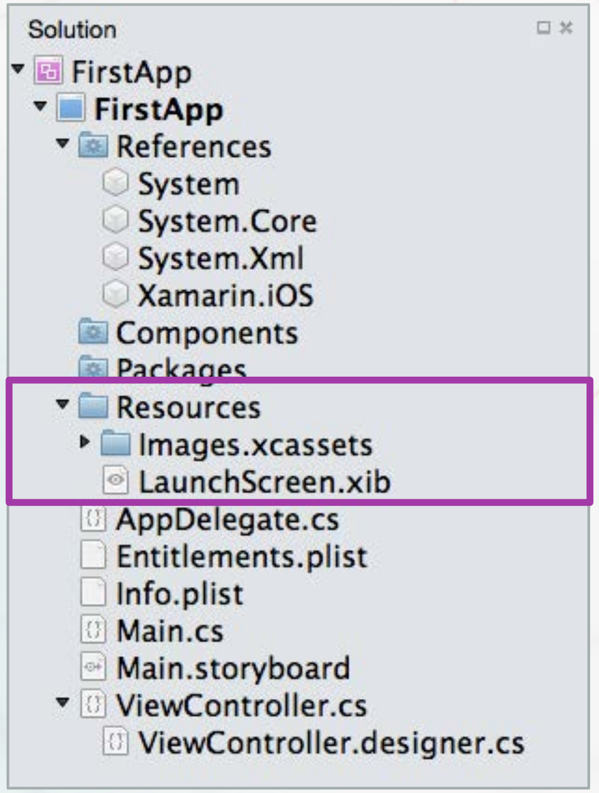

Visual Studio on Mac OS X

Note: even though Xamarin Studio is installed and runs on Windows, it does *not* support iOS application development

What about deploying to a device?

- To test on a device, you will need to register each device and get a set of signing certificates from Apple
- Must have a registered developer Apple account to deploy to a device
- Watch the lightning lecture on provisioning an iOS device for testing

Let's explore the created project

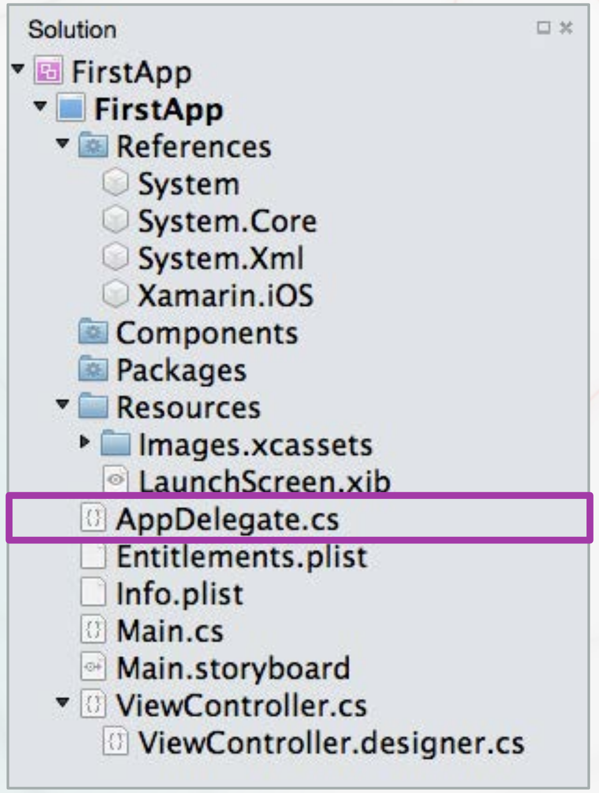

Solution
- FirstApp
  - FirstApp
    - References
      - System
      - System.Core
      - System.Xml
      - Xamarin.iOS
    - Components
    - Packages
    - Resources
      - Images.xcassets
      - LaunchScreen.xib
    - AppDelegate.cs
    - Entitlements.plist
    - Info.plist
    - Main.cs
    - Main.storyboard
    - ViewController.cs
      - ViewController.designer.cs

v Resources folder contains additional assets needed at runtime such as images

v Files in this folder should have a build action of `BundleResource` and are included with the generated application package to be installed on a device

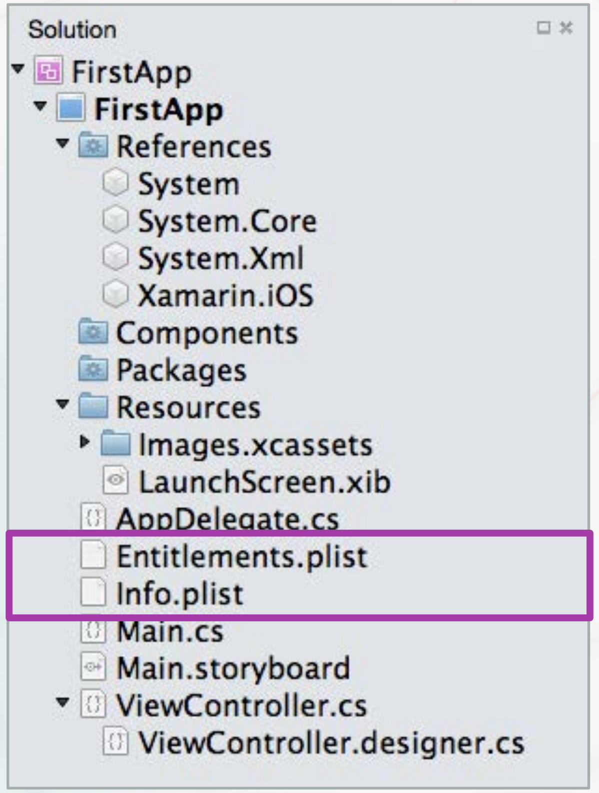
v Template creates some icon assets and a launch screen displayed while the app starts

Let's explore the created project


Solution
- FirstApp
  - FirstApp
    - References
      - System
      - System.Core
      - System.Xml
      - Xamarin.iOS
    - Components
    - Packages
    - Resources
      - Images.xcassets
      - LaunchScreen.xib
    - AppDelegate.cs
    - Entitlements.plist
    - Info.plist
    - Main.cs
    - Main.storyboard
    - ViewController.cs
      - ViewController.designer.cs

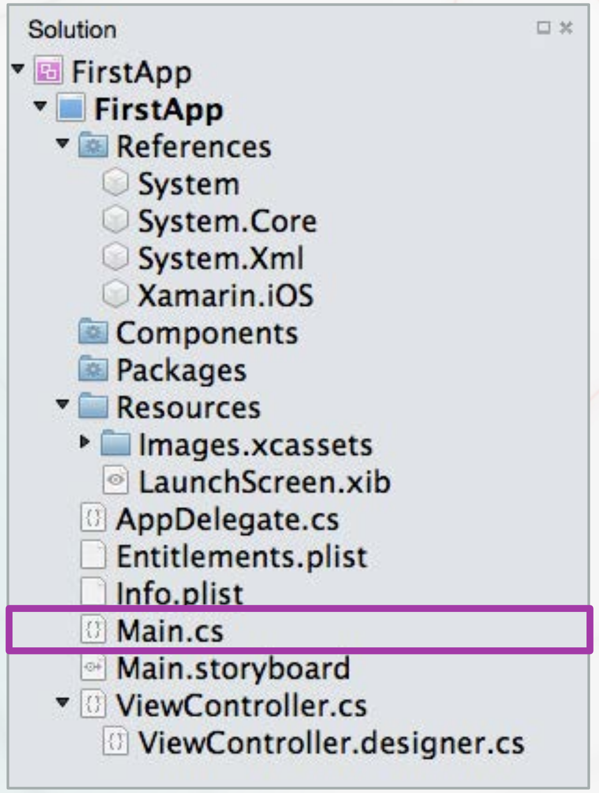
- `AppDelegate.cs` is responsible for creating the main window and listening to operating system events
- Contains a class implements that derives from iOS `UIApplicationDelegate`
- Must override virtual methods in class to process received operating system events

Let's explore the created project


Solution
- FirstApp
  - FirstApp
    - References
      - System
      - System.Core
      - System.Xml
      - Xamarin.iOS
    - Components
    - Packages
    - Resources
      - Images.xcassets
      - LaunchScreen.xib
    - AppDelegate.cs
    - Entitlements.plist
    - Info.plist
    - Main.cs
    - Main.storyboard
    - ViewController.cs
      - ViewController.designer.cs

- iOS uses *property list* files to store application metadata as key/value pairs
  - `Entitlements.plist` lists external Apple services your app wants to interact with such as in-app purchases, HealthKit or push notifications
  - `Info.plist` identifies app icons, version number, app name and other app details
- Both IDEs include a GUI editor for these files to edit the most common settings

Let's explore the created project



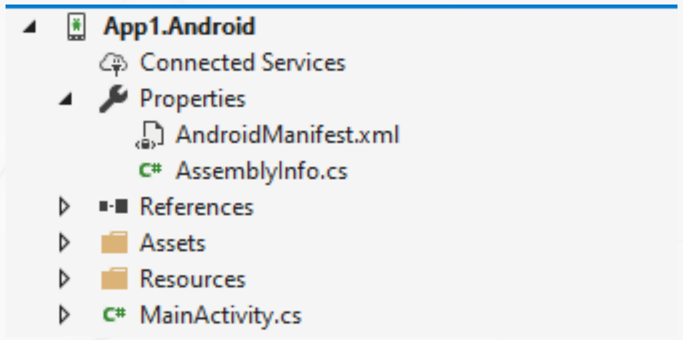
```
Solution
▾ FirstApp
  ▾ FirstApp
    ▾ References
        System
        System.Core
        System.Xml
        Xamarin.iOS
      Components
      Packages
    ▾ Resources
      ▸ Images.xcassets
        LaunchScreen.xib
      AppDelegate.cs
      Entitlements.plist
      Info.plist
      Main.cs
      Main.storyboard
    ▾ ViewController.cs
        ViewController.designer.cs
```

- `Main.cs` contains the main entry point for the application in the form of a standard .NET `static void Main()`
- It starts up the iOS UI framework (UIKit) and identifies the App Delegate, which will in turn bring up the initial screen for the application
- Be cautious about adding code into the `Main` method – iOS has time limits on app launches!

Android

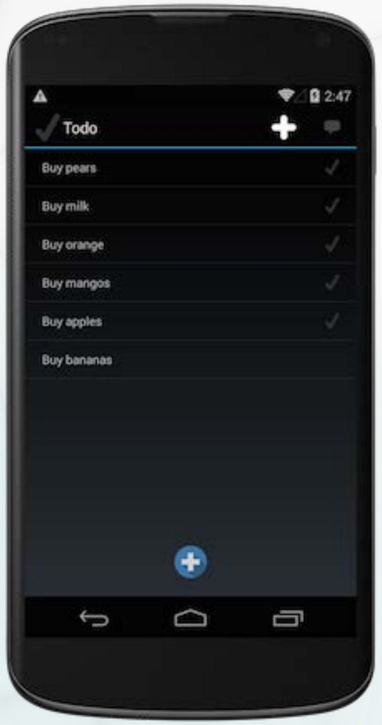
Android project

- Manifeste
- MainActivity

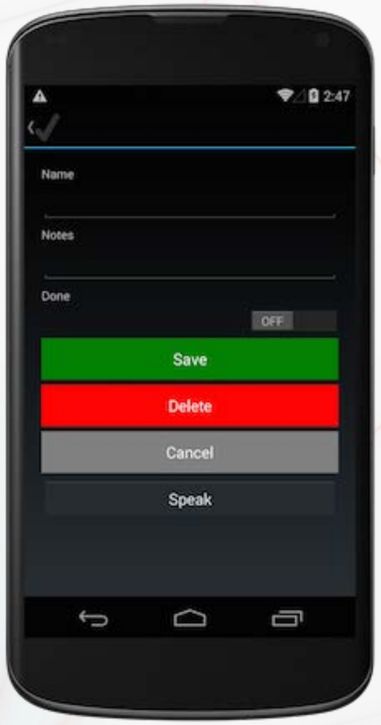


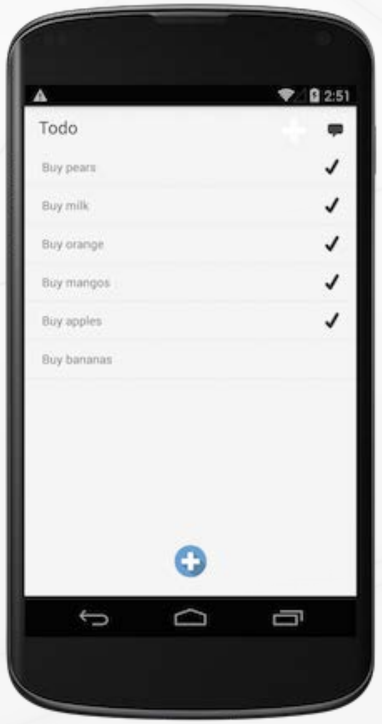
```
<manifest xmlns:android="http://schemas.android.com/apk/res/android" android:versionCode="1"
android:versionName="1.0"
          package="com.companyname.app1">
    <uses-sdk android:minSdkVersion="21" android:targetSdkVersion="28" />
    <application android:label="App1.Android"></application>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
</manifest>
```

App Compat

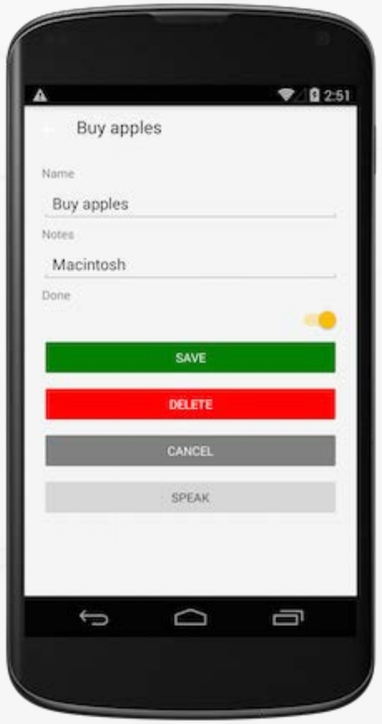


Buy pears
Buy milk
Buy orange
Buy mangos
Buy apples
Buy bananas



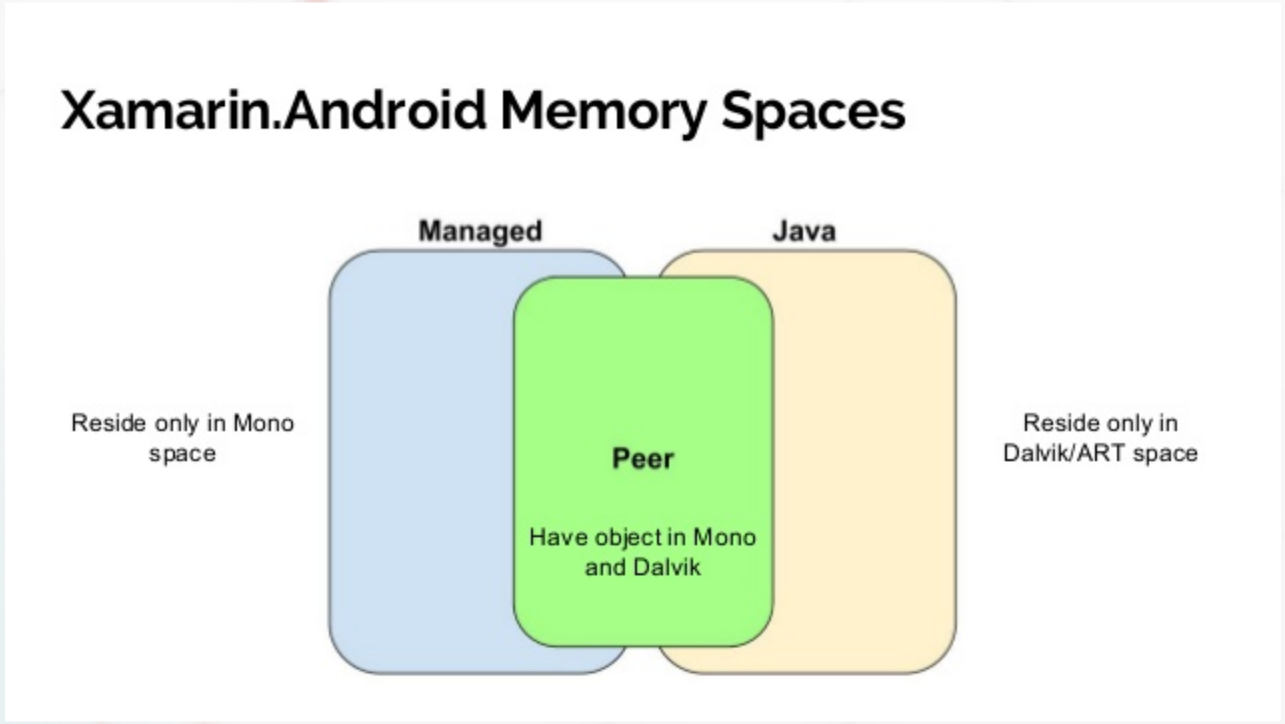


Buy pears
Buy milk
Buy orange
Buy mangos
Buy apples
Buy bananas



Name: Buy apples
Notes: Macintosh

Gestion de la mémoire


Xamarin.Android Memory Spaces
Managed
Java
Reside only in Mono space
Peer
Have object in Mono and Dalvik
Reside only in Dalvik/ART space

- Attention à la gestion des images!
- Penser à les traiter

Windows

App center