LIVE! 360
TECH EVENTS WITH PERSPECTIVE

Visual Studio LIVE! | SQL Server LIVE! | TECHMENTOR | Office & SharePoint LIVE! | Modern Apps LIVE!

VSS02 – HOL Xamarin
Xamarin.Forms

The Ultimate Education Destination

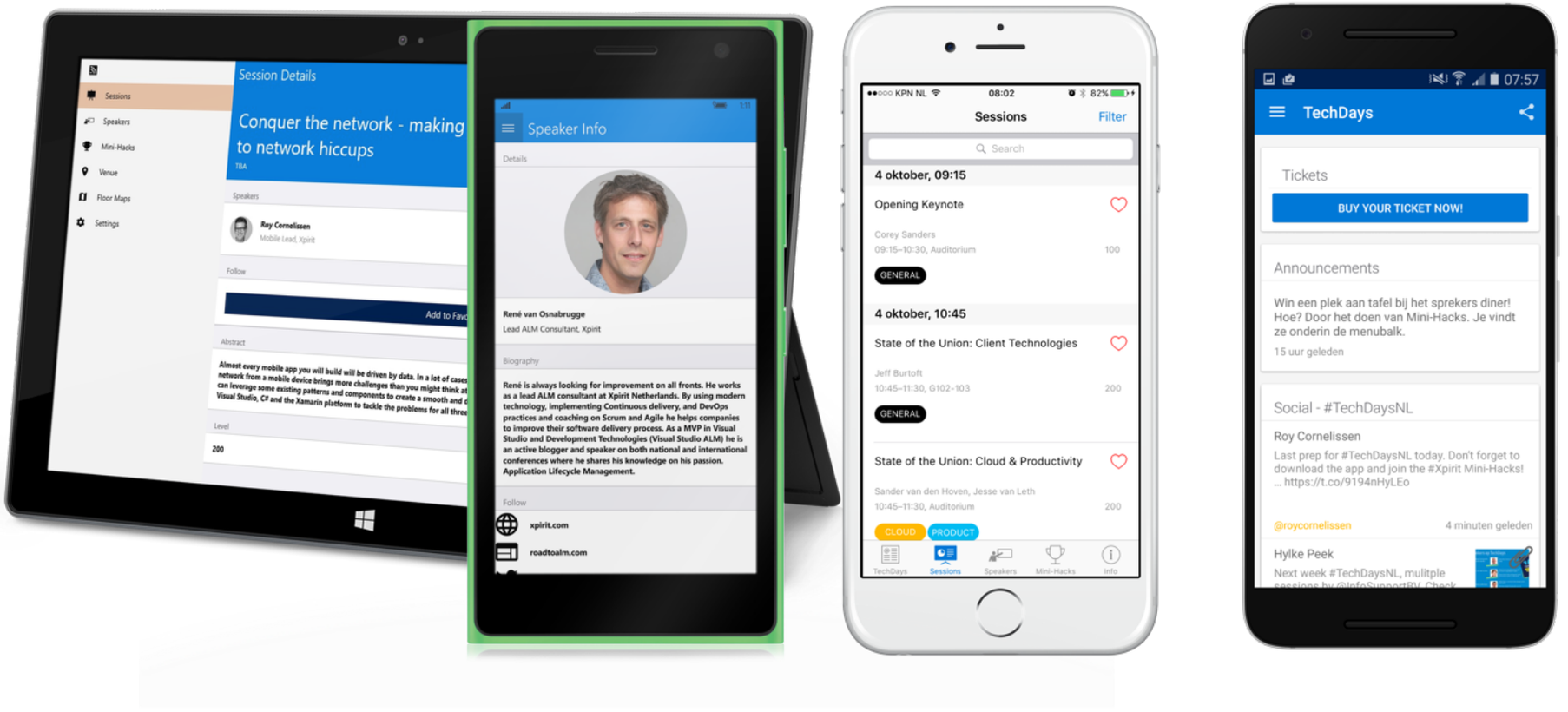
2017 Orlando

Roy Cornelissen & Marcel de Vries

Xpirit

Level: Introductory - Intermediate

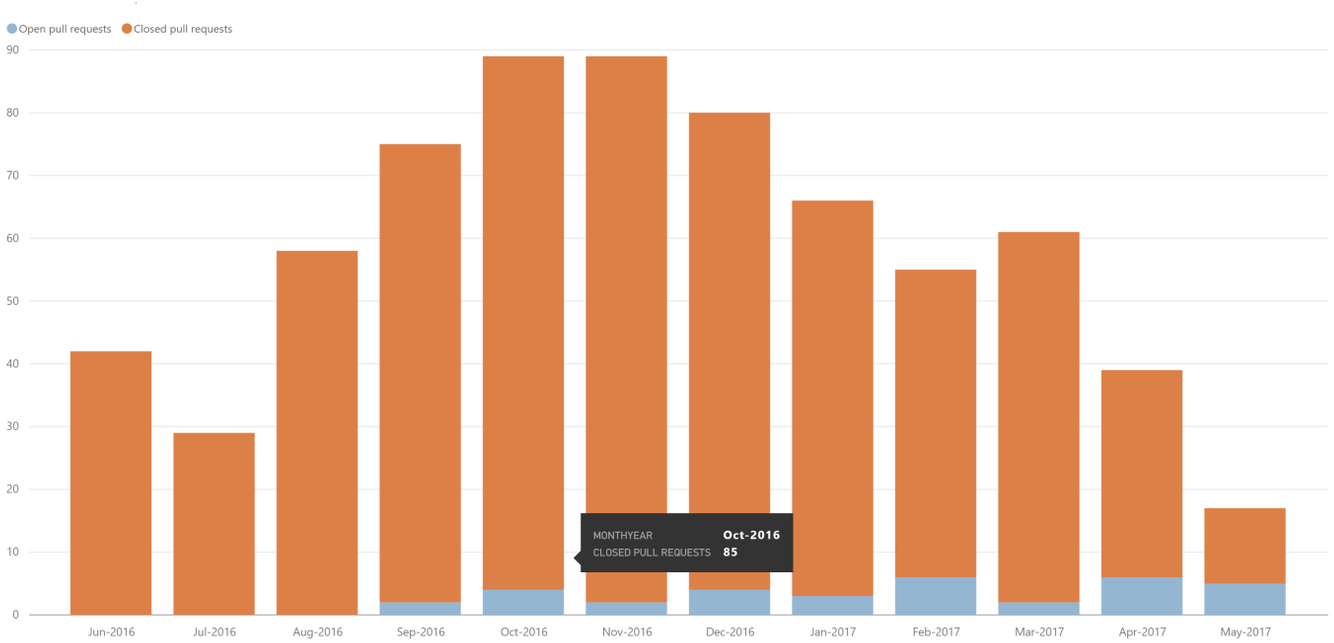
Meet Xamarin.Forms



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

Community

- Public Roadmap: aka.ms/xfroadmap
- 70 contributors
- 700 merged pull requests
- 37 evolution proposals



Features and Enhancements by Community

Fast Renderers Preview

Memory Improvements

Native Control Binding

Bindable Picker

Map Extensibility

XAMLC

Startup Time

Forms Previewer

Android Support Libraries

FlexLayout Preview

Frame Corners

Enhanced GestureRecognizers

Tizen

Blur Behind

macOS Preview

Themes

ListView Scrolling

Embedded Native Controls

Deep Linking

.netstandard support

Fast Scroll

Styles

NavigationPage Translucency

Xamarin.Forms Embedding

PlatformSpecifics

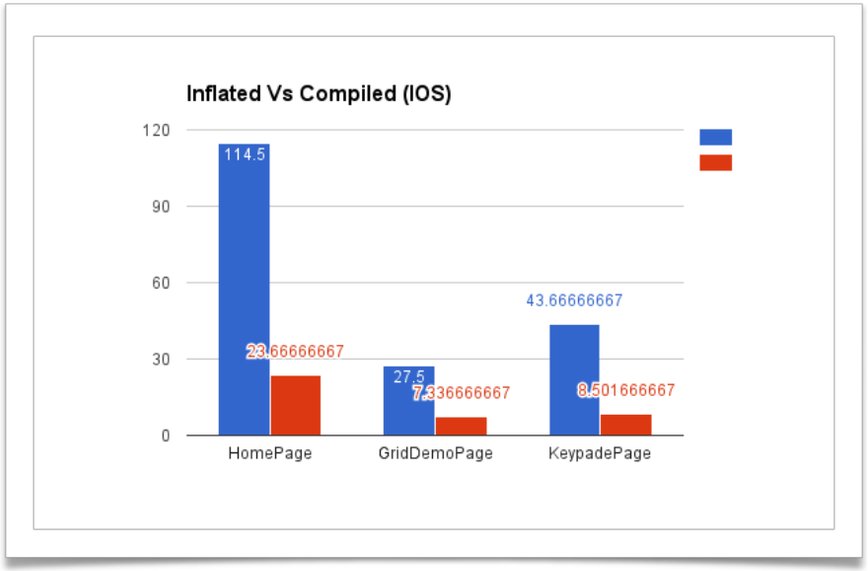
Accessibility

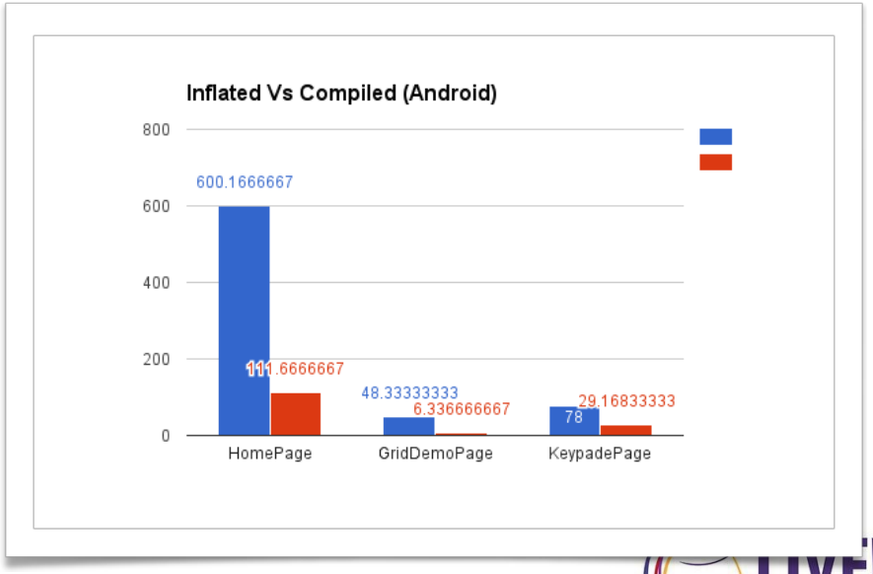
Partial Collapse MasterPage

DataPages

Quality and Performance

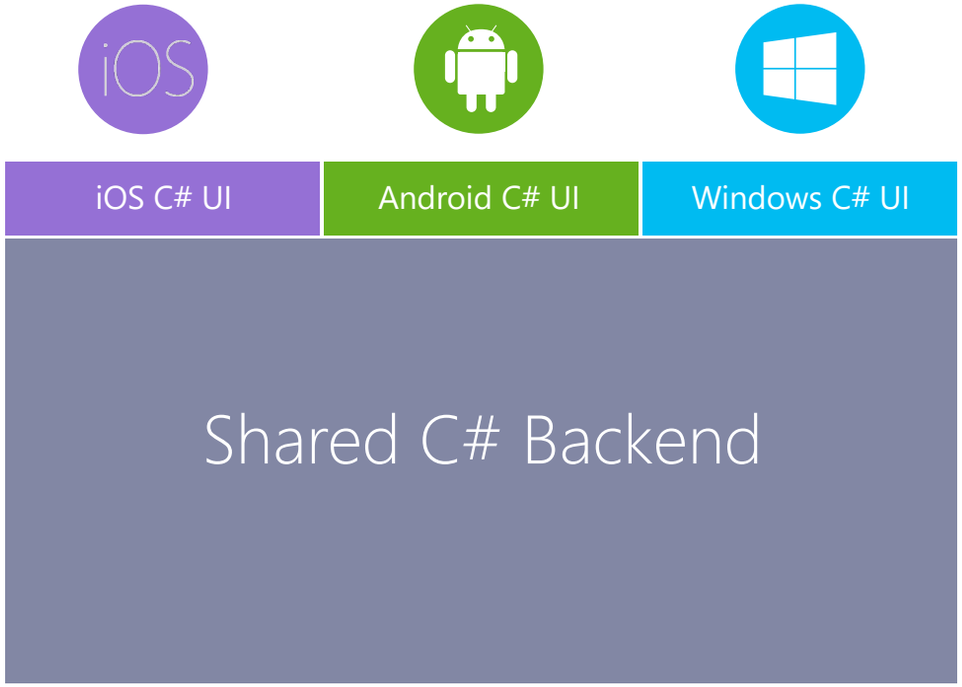
- Aggressive bug triage
  - New issues reviewed within 24 hrs
  - Reproduction projects for quick resolution
- Performance Improvements
  - Startup Time
  - Fast Renderers for Android
  - ListView improvements, Fast Scroll
  - XAMLC – 5x faster
  - Discussion: aka.ms/xfperformance



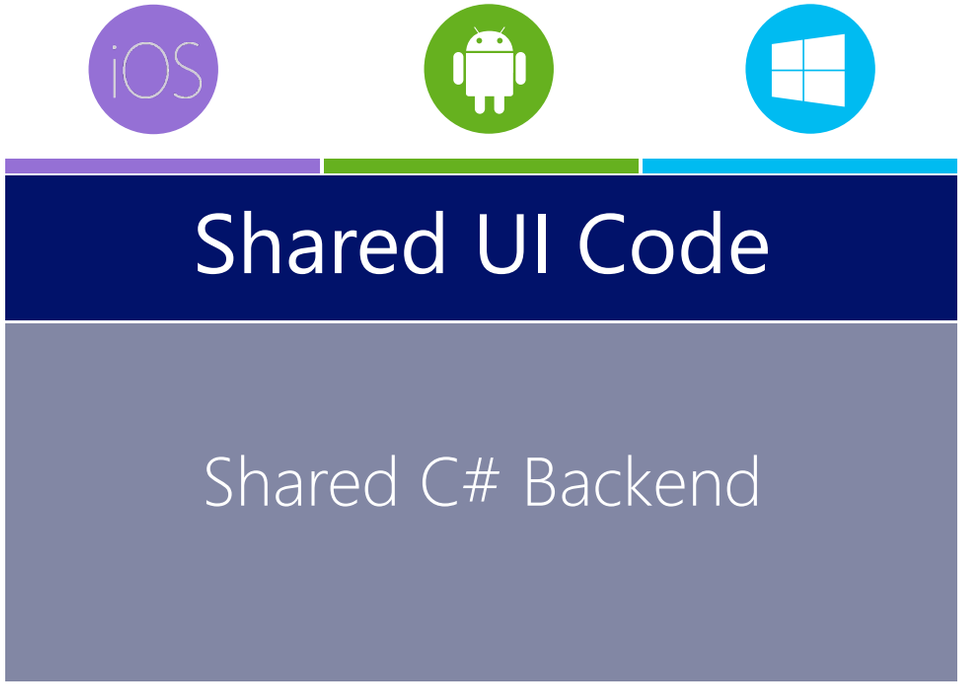


Source: Matthew Robbins - MFractor

Xamarin + Xamarin.Forms

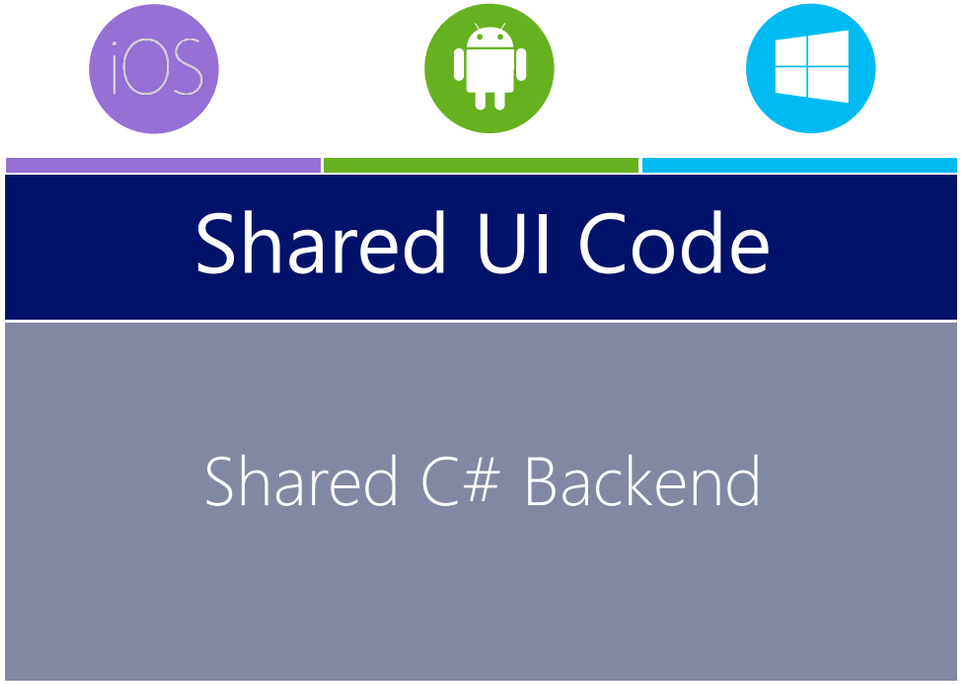


Traditional Xamarin Approach



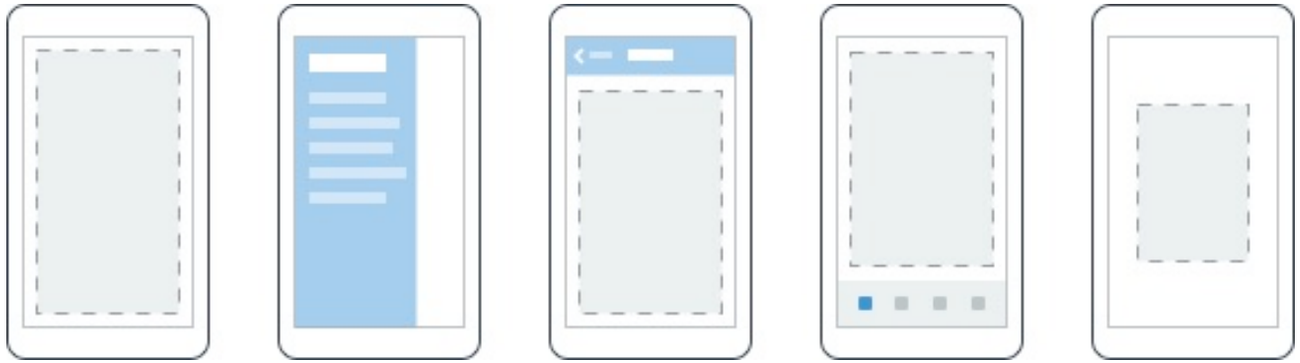
With Xamarin.Forms:
More code-sharing, all native

What's included



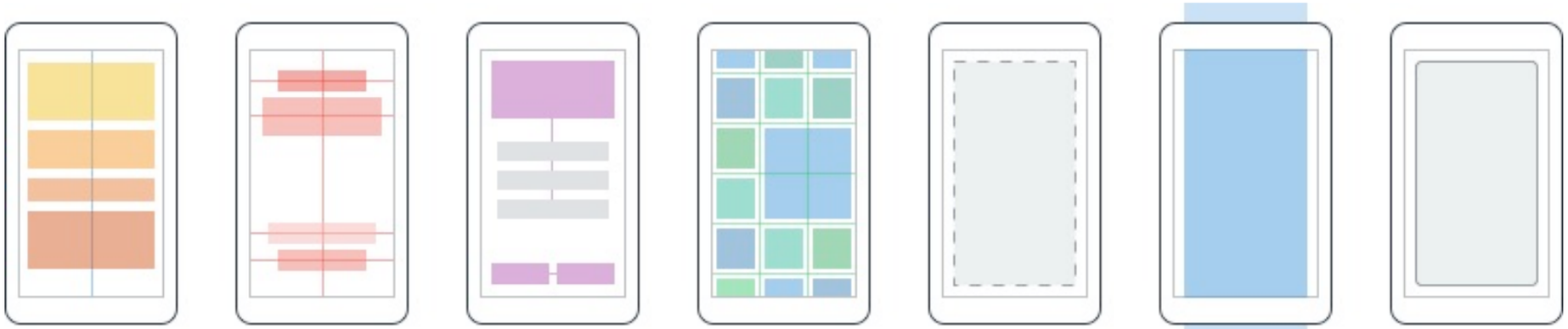
- ✓ 40+ Pages, layouts, and controls
  (Build from code behind or XAML)
- ✓ Two-way data binding
- ✓ Navigation
- ✓ Animation API
- ✓ Dependency Service
- ✓ Messaging Center

Pages



Content MasterDetail Navigation Tabbed Carousel

Layouts

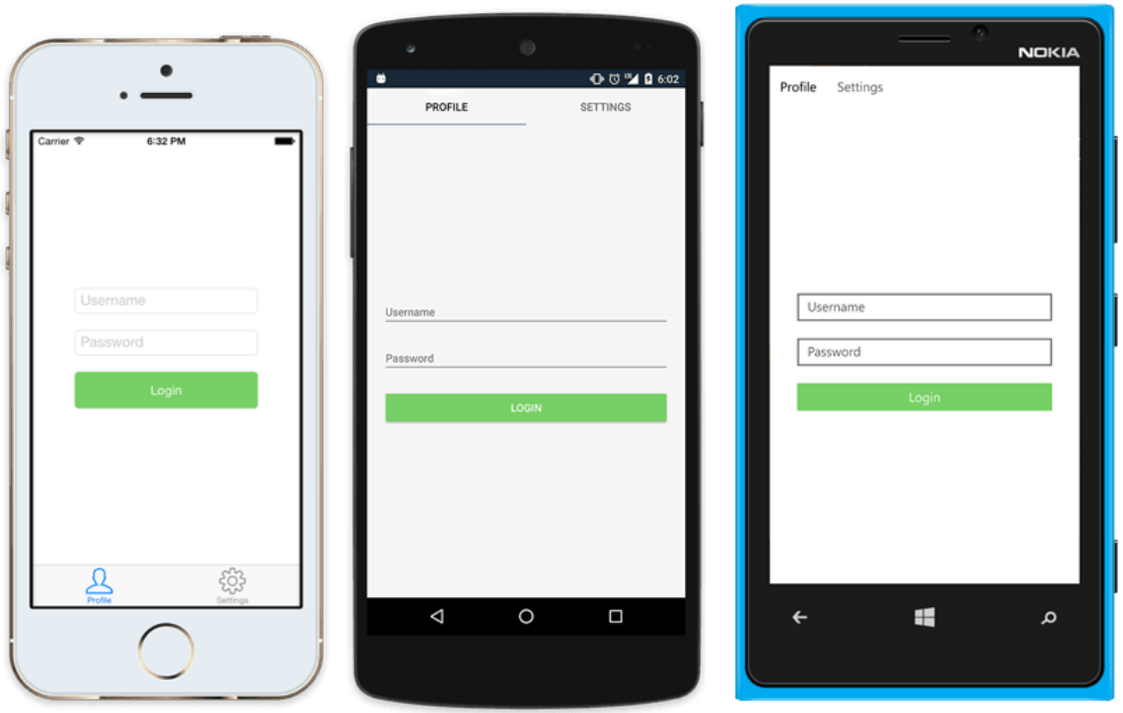


Stack Absolute Relative Grid ContentView ScrollView Frame

Controls

ActivityIndicator	BoxView	Button	DatePicker	Editor
Entry	Image	Label	ListView	Map
OpenGLView	Picker	ProgressBar	SearchBar	Slider
Stepper	TableView	TimePicker	WebView	EntryCell
ImageCell	SwitchCell	TextCell	ViewCell	

Native UI from shared code




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

Xamarin.Forms Ecosystem

Control Comparison

Windows	Xamarin.Forms
StackPanel	StackLayout
TextBox	Entry
ListBox	ListView
CheckBox	Switch
ProgressBar	ActivityIndicator
Grid	Grid
Label	Label
Button	Button
Image	Image
Date/TimePicker	Date/TimePicker

Binding Comparison

Windows	Xamarin.Forms
DataContext	BindingContext
{Binding Property}	{Binding Property}
ItemsSource	ItemsSource
ItemTemplate	ItemTemplate
DataTemplate	DataTemplate

```
<Label Text="{Binding Color.R,
                      Converter={StaticResource intConverter},
                      ConverterParameter=255,
                      StringFormat='R={0:X2}'}" />
```

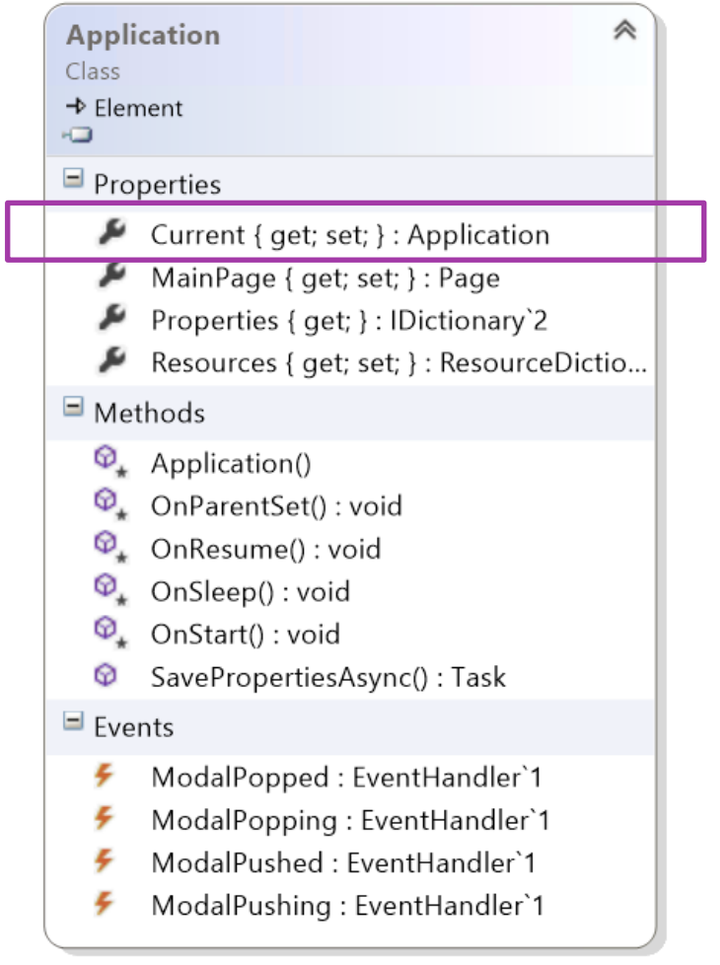
Xamarin.Forms app anatomy

- Xamarin.Forms applications have two required components which are provided by the template

Xamarin.Forms Application

- **`Application`** class provides a *singleton* which manages:
  - Lifecycle methods
  - Modal navigation notifications
  - Currently displayed page
  - Application state persistence

- New projects will have a derived implementation named **`App`**



Note: Windows apps *also* have an **`Application`** class, make sure not to confuse them!

Xamarin.Forms Application

- **`Application`** class provides lifecycle methods which can be used to manage persistence and refresh your data

```
public class App : Application
{
    // Handle when your app starts
    protected override void OnStart() {}
    // Handle when your app sleeps
    protected override void OnSleep() {}
    // Handle when your app resumes
    protected override void OnResume() {}
}
```

Use **`OnStart`** to initialize and/or reload your app's data

Xamarin.Forms Application

- **`Application`** class provides lifecycle methods which can be used to manage persistence and refresh your data

```
public class App : Application
{
    // Handle when your app starts
    protected override void OnStart() {}
    // Handle when your app sleeps
    protected override void OnSleep() {}
    // Handle when your app resumes
    protected override void OnResume() {}
}
```

Use **`OnSleep`** to save changes or persist information the user is working on

Xamarin.Forms Application

- **`Application`** class provides lifecycle methods which can be used to manage persistence and refresh your data

```
public class App : Application
{
    // Handle when your app starts
    protected override void OnStart() {}
    // Handle when your app sleeps
    protected override void OnSleep() {}
    // Handle when your app resumes
    protected override void OnResume() {}
}
```

Use **`OnResume`** to refresh your displayed data

Persisting information

- **`Application`** class also includes a **`string`** >> **`object`** property bag which is persisted between app launches

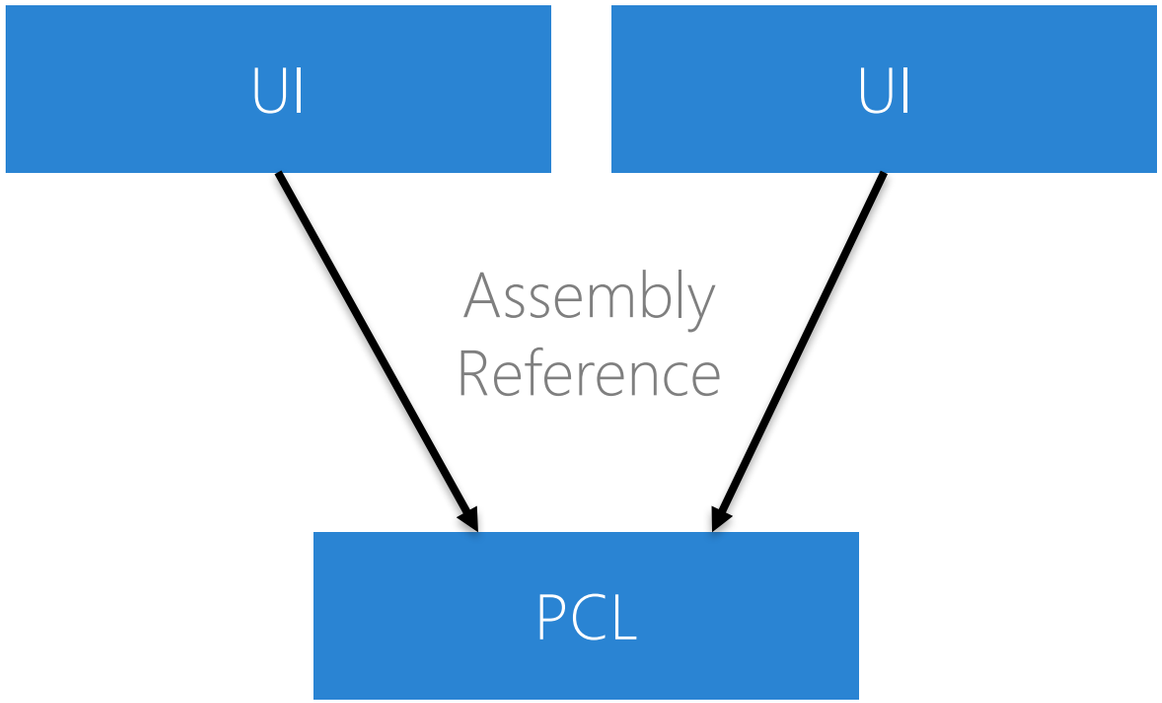
```
// Save off username in global property bag
Application.Current.Properties["username"] = username.Text;
```

```
// Restore the username before it is displayed
if (Application.Current.Properties.ContainsKey("username")) {
    var uname = Application.Current.Properties["username"] as string
                    ?? "";
    username.Text = uname;
}
```

Demo

- First Xamarin Forms Application

Portable Class Library



Runtime concept

Lowest common denominator across target profiles

Binary reuse

Portable Class Libraries

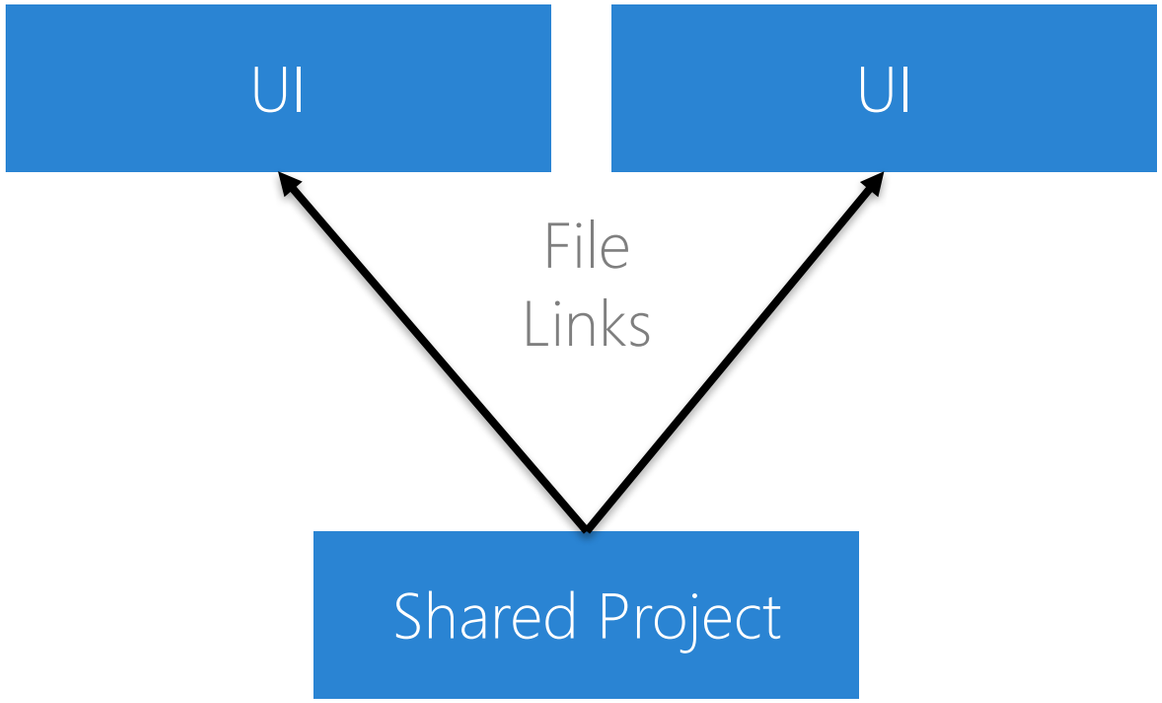
Pros

- One class library project
- Use well-known references
- Reuse DLL across platforms

Cons

- Lowest common denominator
- Can’t use compiler directives
- Platform-specific code complex

Shared Project



Compile time concept

Files are **compiled into** target platform binary

Shared Project

Pros

- One code library project
- Code is reused
- Can use compiler directives
- Platform-specific code easy

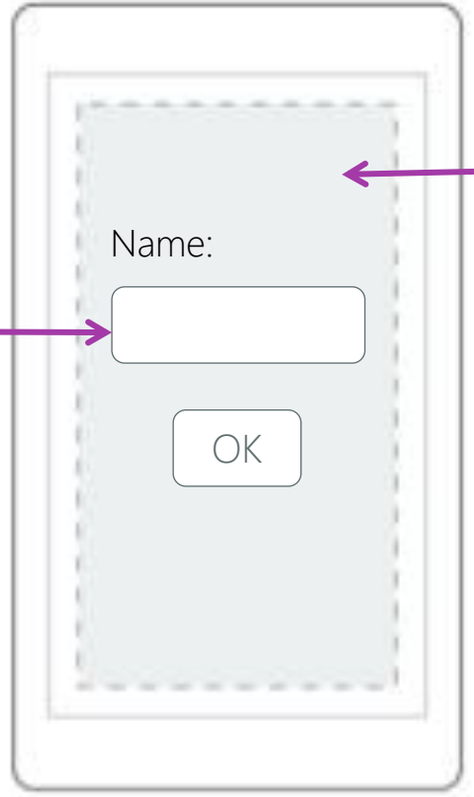
Cons

- No assembly to reuse
- Uses less known file linking
- Testing potentially harder

Creating the application UI

- Application UI is defined in terms of *pages* and *views*

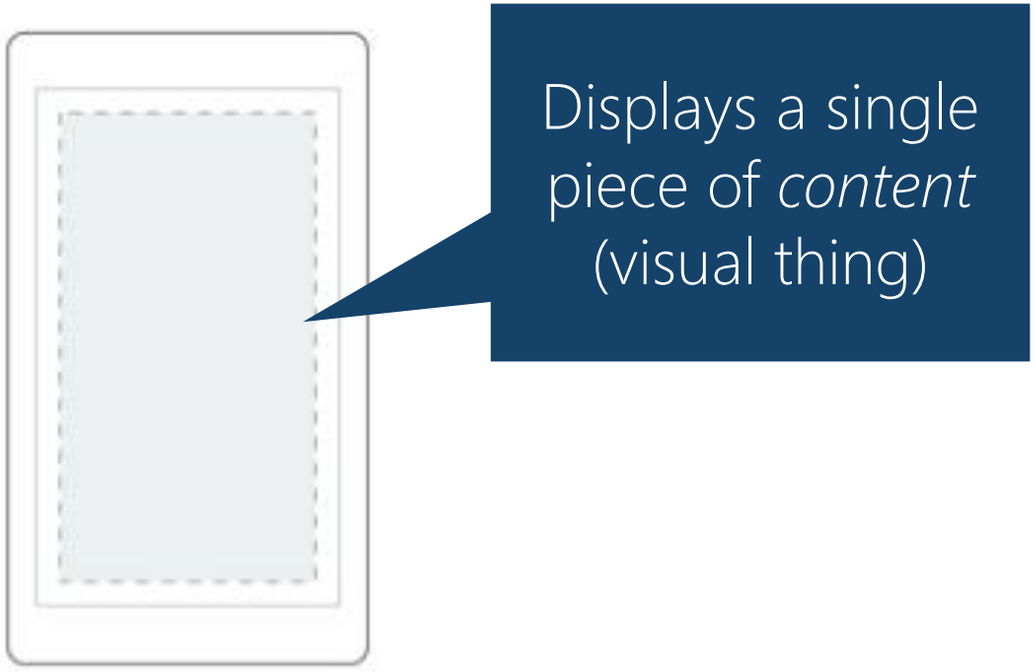
Views are the UI controls the user interacts with



Page represents a single screen displayed in the app

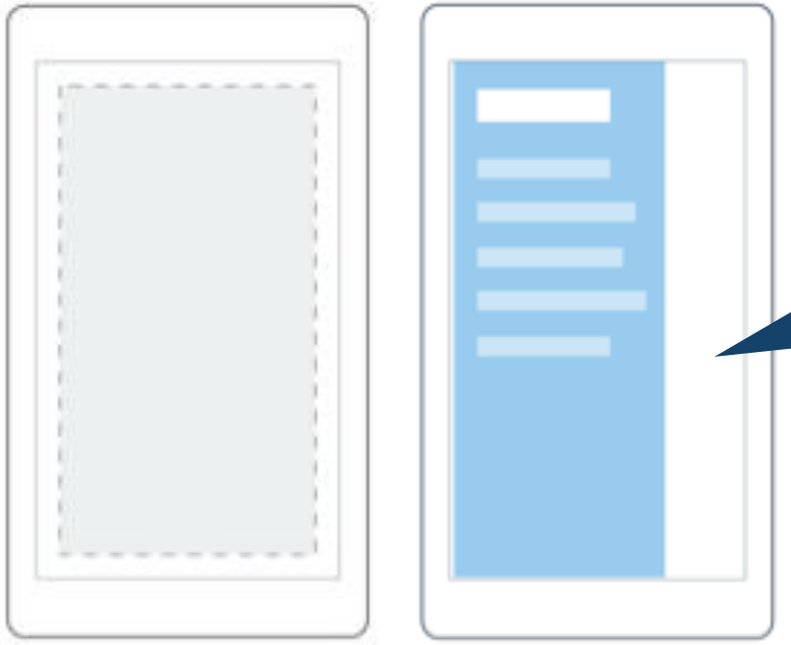
Pages

- **Page** is an abstract class used to define a single screen of content
  - derived types provide specific visualization / behavior

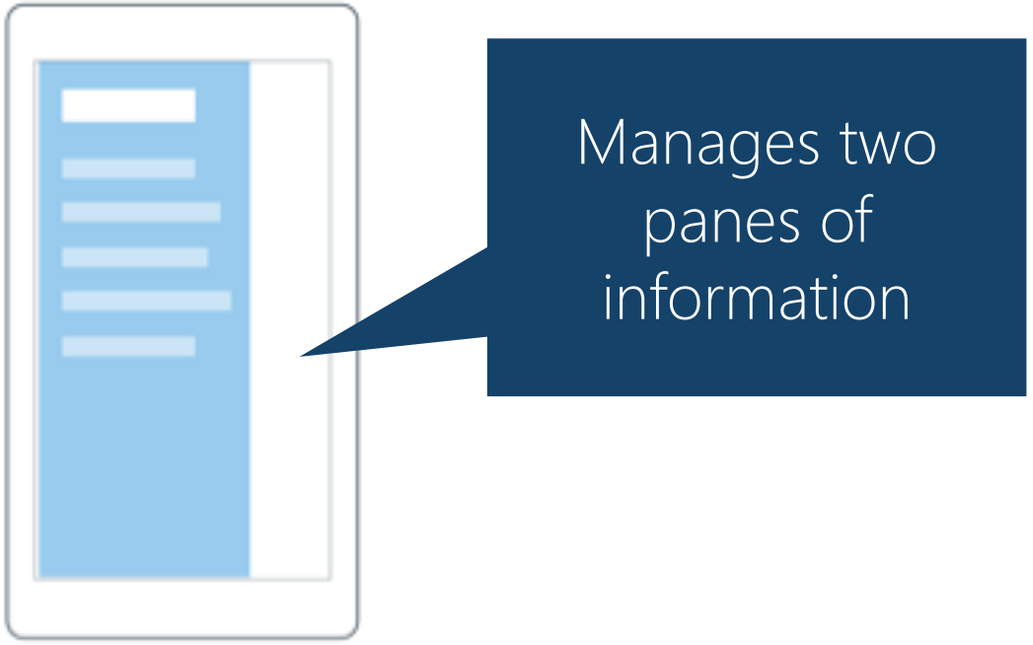


Pages

- **Page** is an abstract class used to define a single screen of content
  - derived types provide specific visualization / behavior



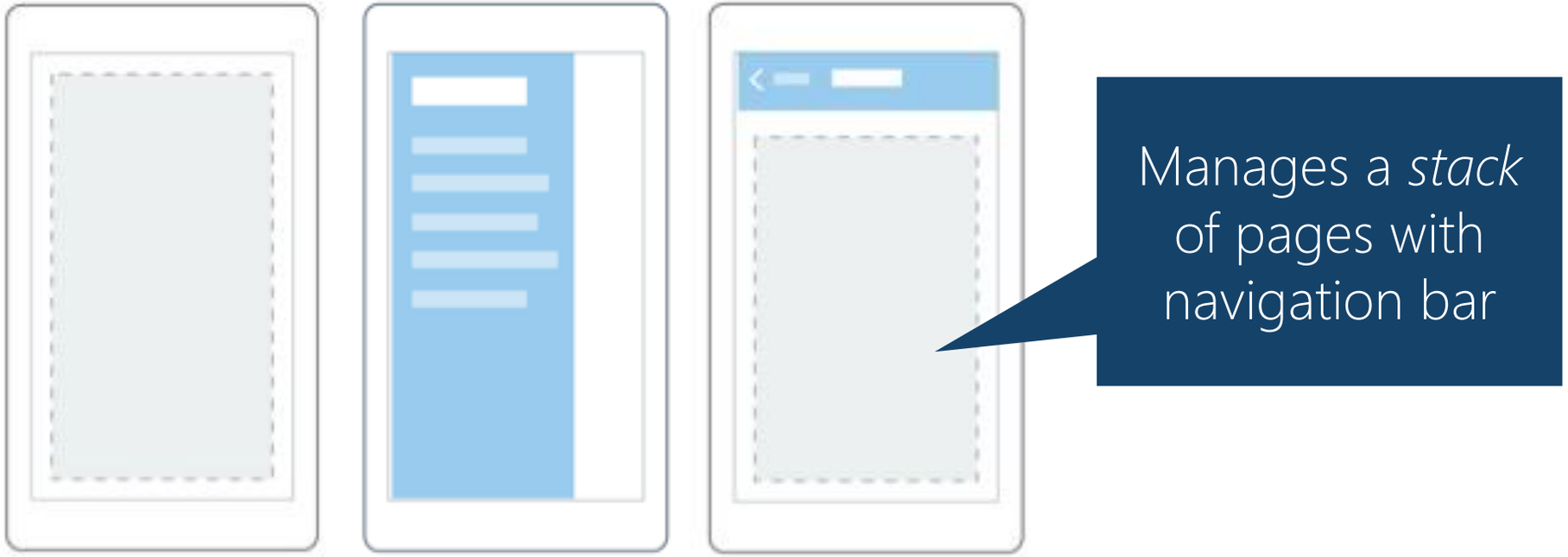
Content



Master Detail

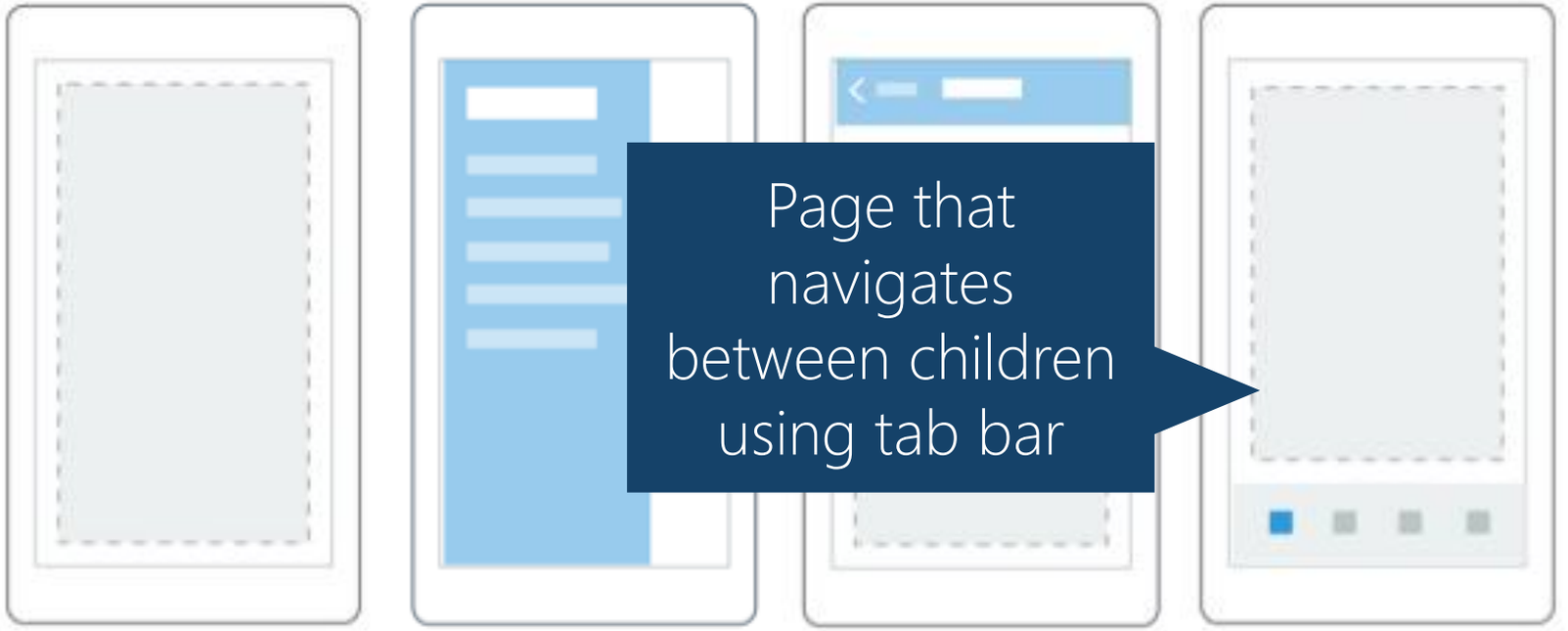
Pages

- **Page** is an abstract class used to define a single screen of content
  - derived types provide specific visualization / behavior



Pages

- **Page** is an abstract class used to define a single screen of content
  - derived types provide specific visualization / behavior

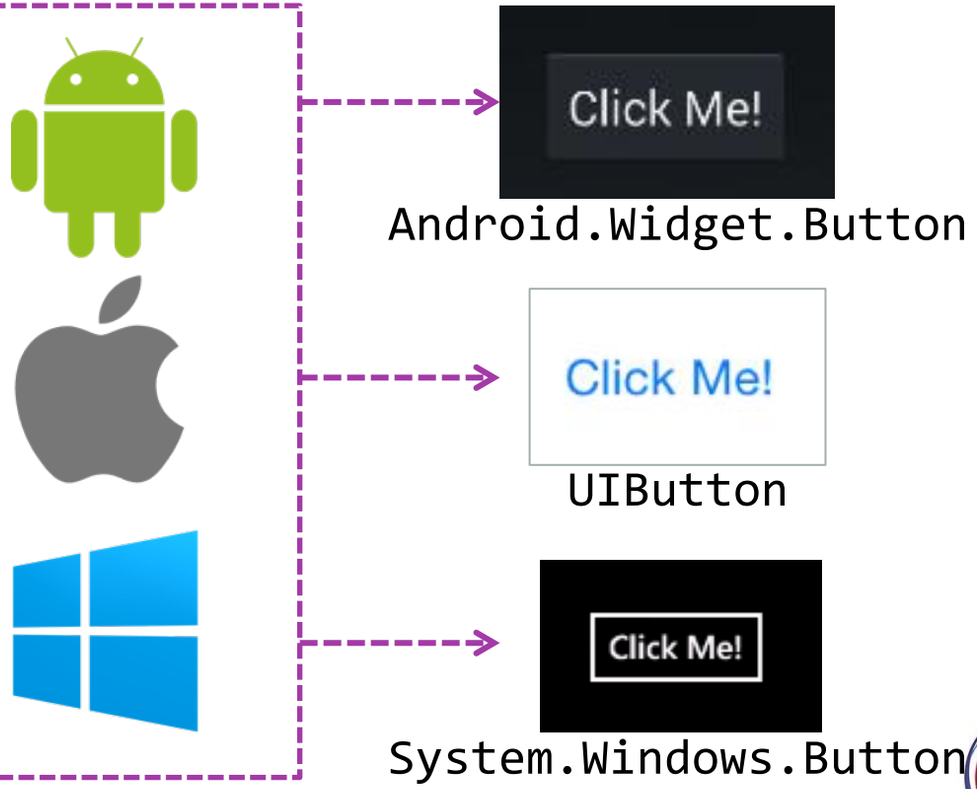


Rendering views

- Platform defines a *renderer* for each view that creates a native representation of the UI

UI uses a Xamarin.Forms `Button`

```
Button button = new Button {
    Text = "Click Me!"
};
```



Platform Renderer takes view and turns it into **platform-specific control**

Visual adjustments

- Views utilize **properties** to adjust visual appearance and behavior

```
Entry numEntry = new Entry {
    Placeholder = "Enter Number",
    Keyboard = Keyboard.Numeric
};

Button callButton = new Button {
    Text = "Call",
    BackgroundColor = Color.Blue,
    TextColor =  Color.White
};
```

Providing Behavior

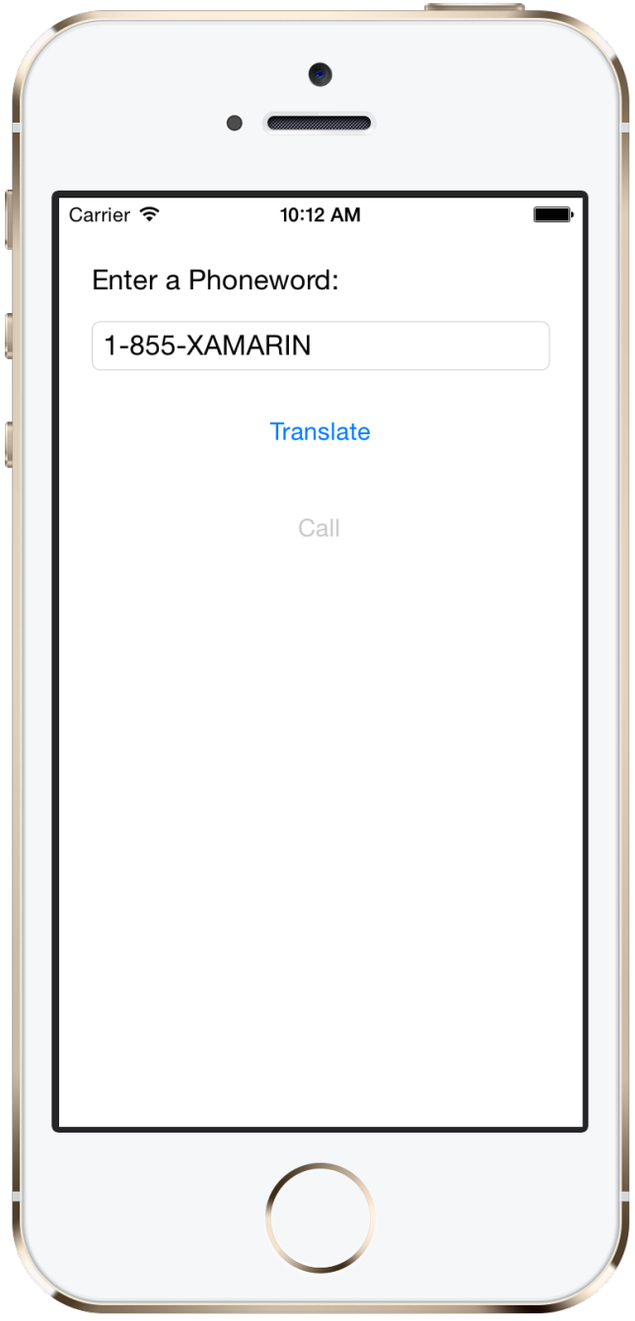
- Controls use events to provide interaction behavior, should be very familiar model for most .NET developers

```
Entry numEntry = new Entry { ... };
numEntry.TextChanged += OnTextChanged;
...

void OnTextChanged (object sender, string newValue)
{
    ...
}
```

You can use traditional delegates, anonymous methods, or lambdas to handle events

XAML

- Our goal is to build the UI for this screen
  - Label (Enter a Phoneword:)
  - Entry (1-855-XAMARIN)
  - Button (Translate)
  - Button (Call)



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

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

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

- 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>
```

Attributes set properties or events

Describing a screen in XAML

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

Child nodes used to establish relationship

```
<?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>
```

My first Xamarin.Forms app

Lab – app-quotes

Lab01 – exercises 1-3

Messaging center

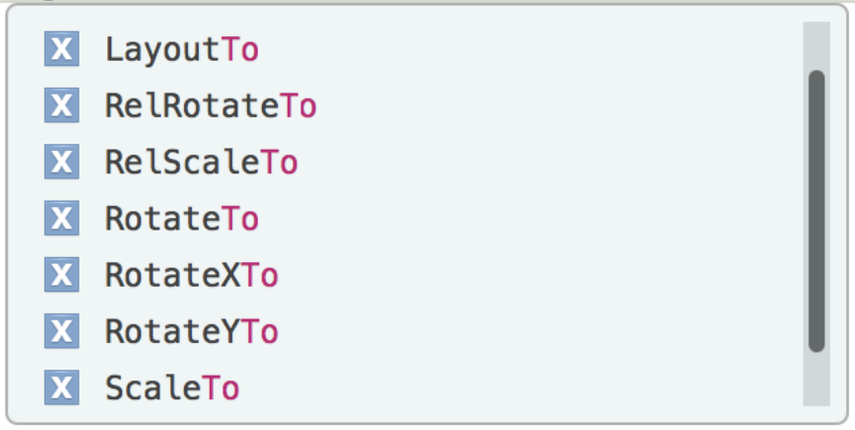
```
MessagingCenter.Send<Detail>(this, “SomeMessage");
```


```
MessagingCenter.Subscribe<Detail>(this, “SomeMessage", async (sender) =>
{
    await DoSomething();
});
```

Animations

- Cross-platform animations
- Translates to native animations as much as possible
- Async / Await API

```
ProfileImage.FadeTo (1, 1000);
ProfileImage.RotateTo (360, 1000, Easing.CubicOut);
ProfileImage.to
```



Platform tweaks

Use Device class in shared code

- Device.Styles
  Styles for controls, e.g. labels & lists
- Device.GetNamedSize
  Font sizes
- Device.Idiom
  Tablet or phone specific code / behavior
- Device.OS
  Target platform specific code
- Device.OnPlatform()
  Target platform specific values
- Device.OpenUri()
- Device.StartTimer()
- Device.BeginInvokeOnMainThread()

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 a .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`**

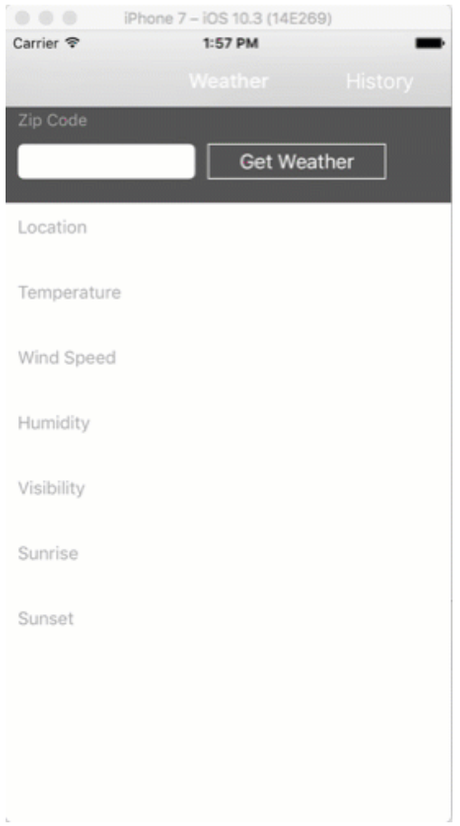
XAML Standard 1.0

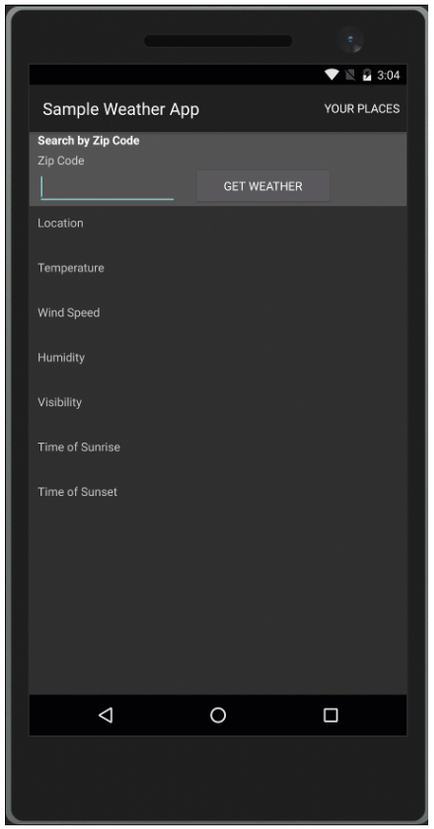
- Attempt to standardize all XAML variations
  - UWP, Forms, WPF, …
- Open source project
  - https://github.com/Microsoft/xaml-standard

The Future: Xamarin.Forms 3.0

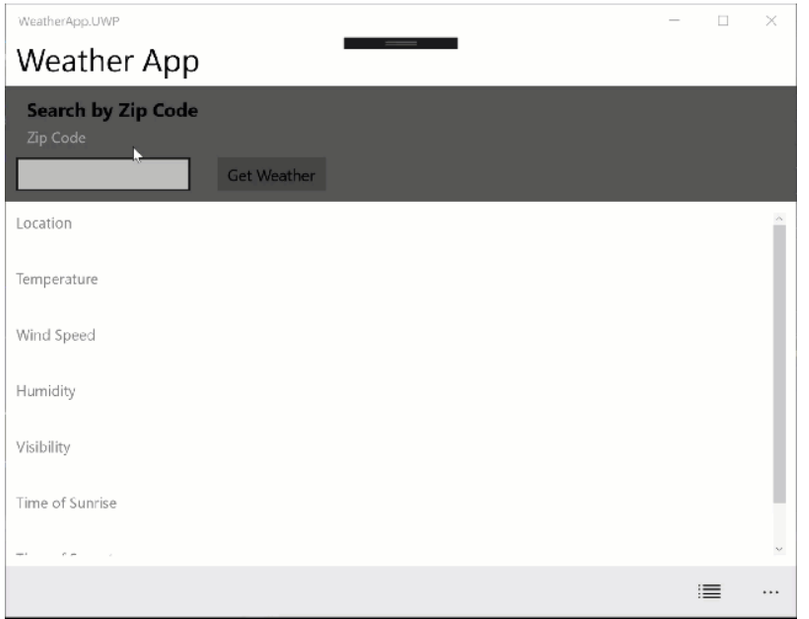
Xamarin.Forms Embedding

- Easily embed any page into a Xamarin Native Application



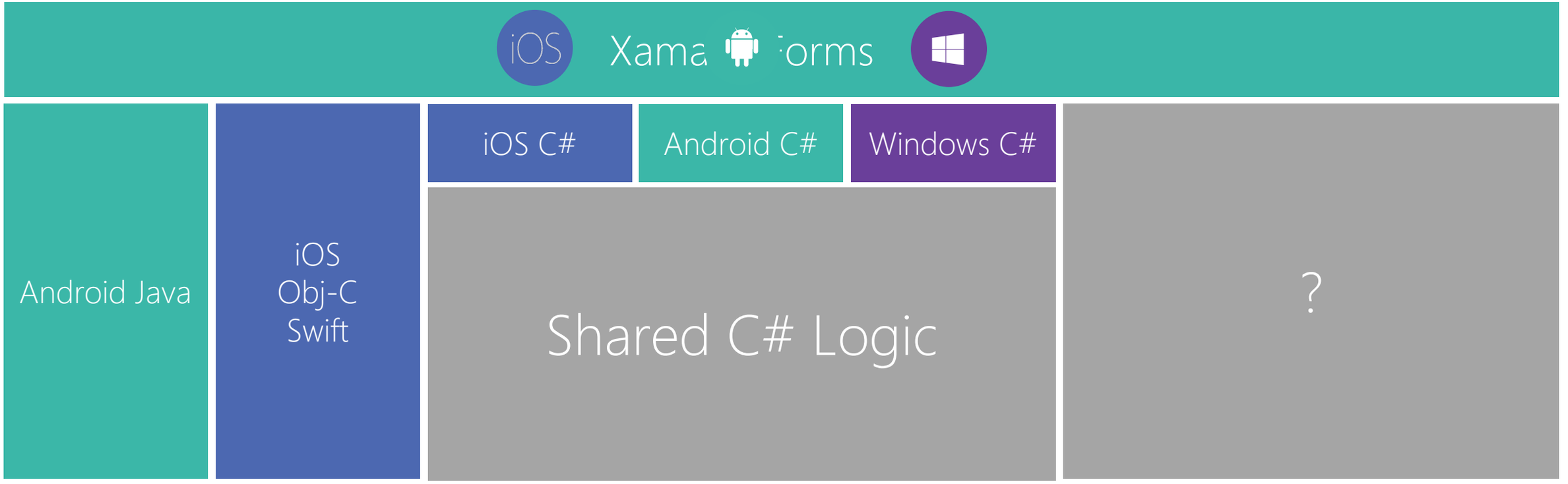



```
// Android
Forms.Init(this, null);
var androidFragment = new MyFormsPage().CreateFragment(this);

// iOS
Forms.Init()
var iosViewController = new MyFormsPage().CreateViewController();

// UWP
Forms.Init(e);
var uwpElement = new MyFormsPage().CreateFrameworkElement();
```



Embedding

- Works on ContentPages
- Full support for DependencyService and MessagingCenter



FlexLayout

- A CSS FlexBox inspired layout system
- Used for
  - Flowing items
  - Adaptive layout

FlexLayout Example

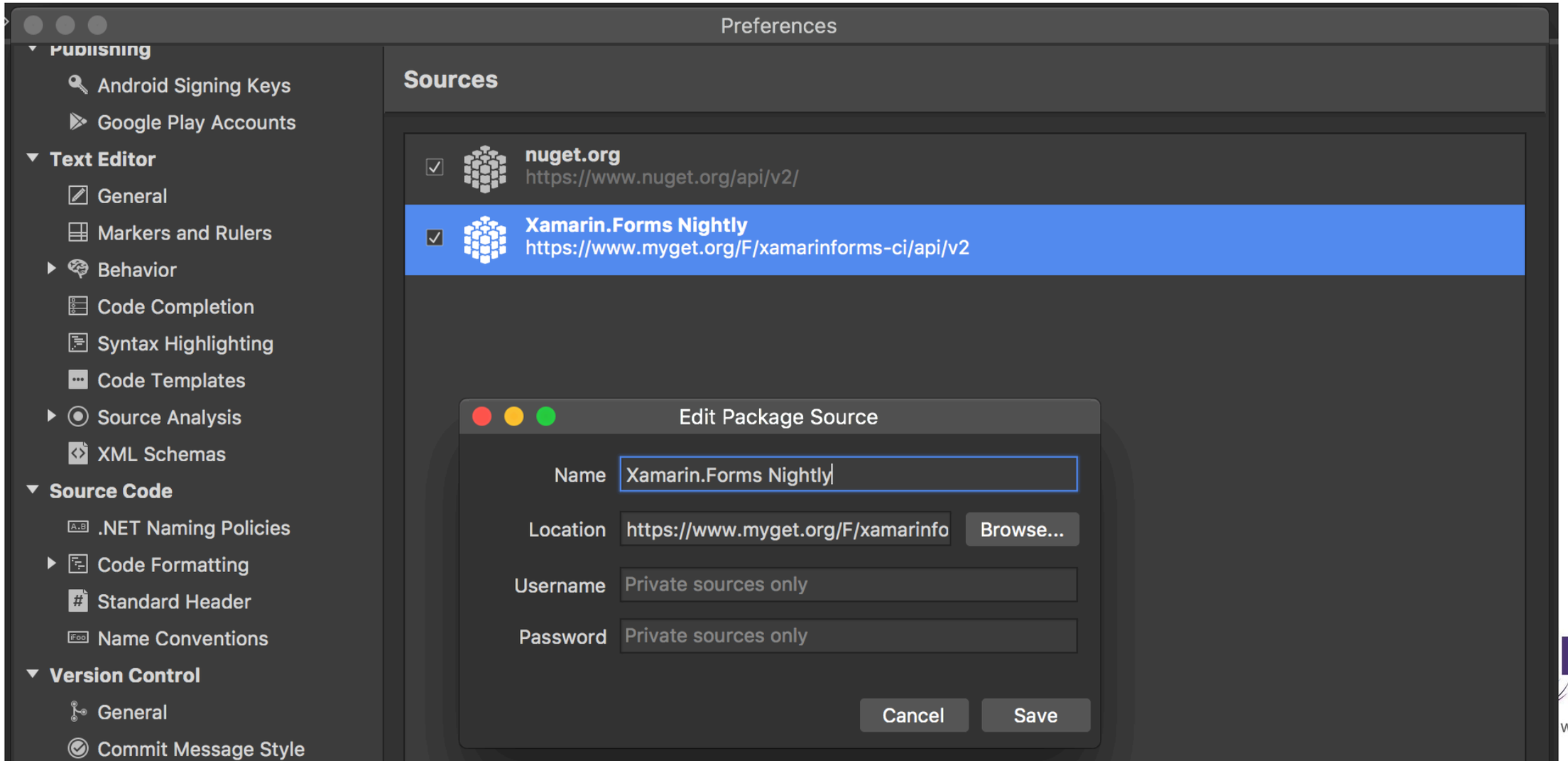
```
<?xml version="1.0" encoding="UTF-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x=http://schemas.microsoft.com/winfx/2009/xaml
             x:Class="FormsFlexLayoutDemo.FlexDemoPage">
    <FlexLayout x:Name="flexbox”>
        <Label Text="Flex Element 1" />
        <Label Text="Flex Element 2" />
        <Label Text="Flex Element 3" />
        <Label Text="Flex Element 4" />
        <Label Text="Flex Element 5" />
    </FlexLayout>
</ContentPage>
```

New Platforms

- Samsung Tizen
  - Televisions, Wearables, Mobile
- macOS
- WPF
- Linux: GTK#

Nightly Builds

https://github.com/xamarin/Xamarin.Forms/wiki/Nightly-Builds



Add XAML Compilation

Lab – app-quotes

Lab01 – exercise 4

Extending the app

Lab – app-quotes

Lab02