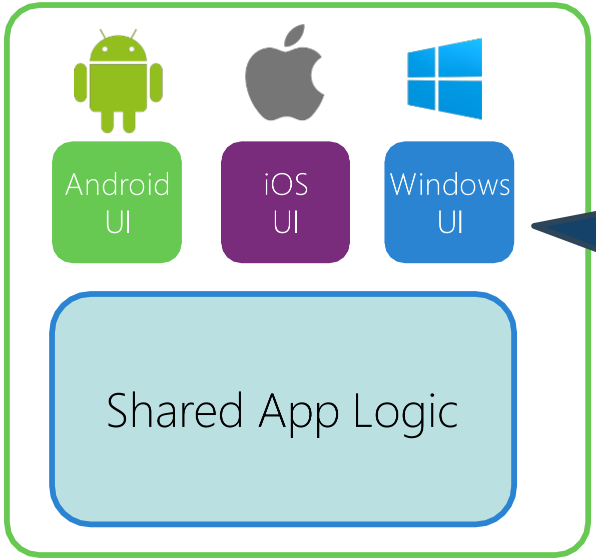
Introduction to Xamarin.Forms

Objectives

1. What is Xamarin.Forms?
2. Xamarin.Forms App Structure
3. Pages, Controls, and Layout
4. Using Platform-Specific Features

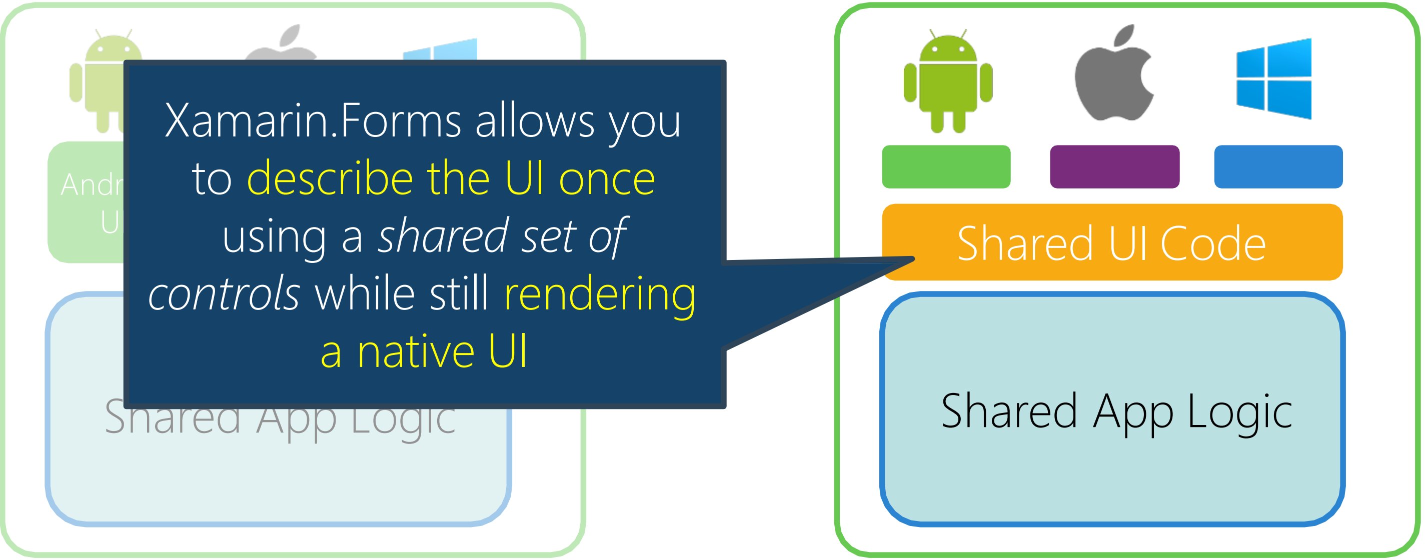
Cross-Platform UI Strategies

Traditional approach vs. Xamarin.Forms



Traditional Xamarin approach creates non-sharable platform-specific code for the UI layer

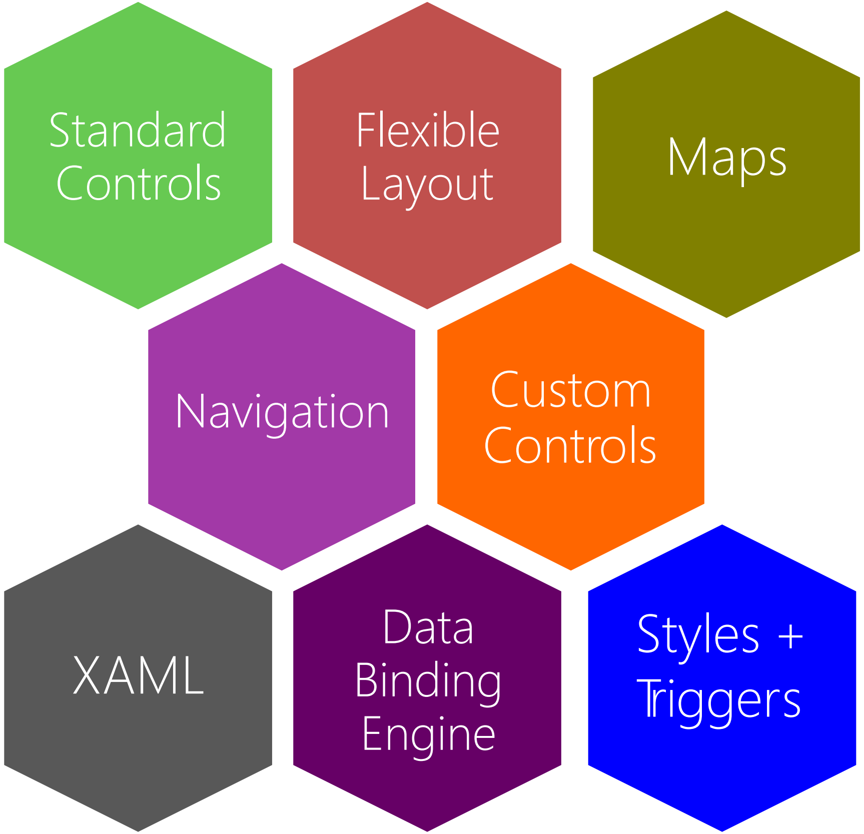
Traditional approach vs. Xamarin.Forms



Xamarin.Forms allows you to describe the UI once using a *shared set of controls* while still rendering a native UI

What is Xamarin.Forms?

- Xamarin.Forms is a cross-platform UI framework to create mobile apps for:
  - Android 4.0+
  - iOS 6.1+
  - Windows 10



Standard Controls

Flexible Layout

Maps

Navigation

Custom Controls

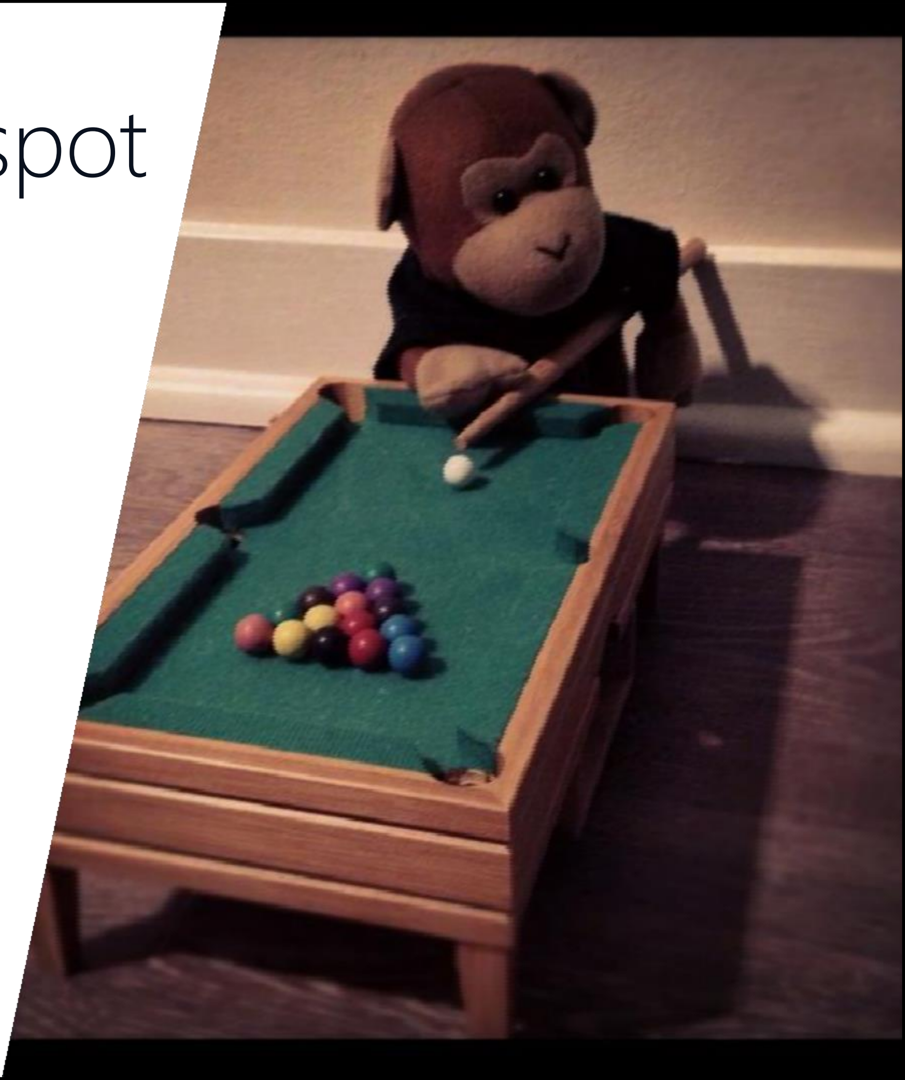
XAML

Data Binding Engine

Styles + Triggers

Xamarin.Forms sweet spot

- ❖ Xamarin.Forms is not suitable for all types of apps
  - ✓ Great for data-driven (forms) and utility applications
  - x Not ideal if your UI will be highly customized to the platform
- ❖ Can be used for quick prototyping even if you do not utilize it for the final app

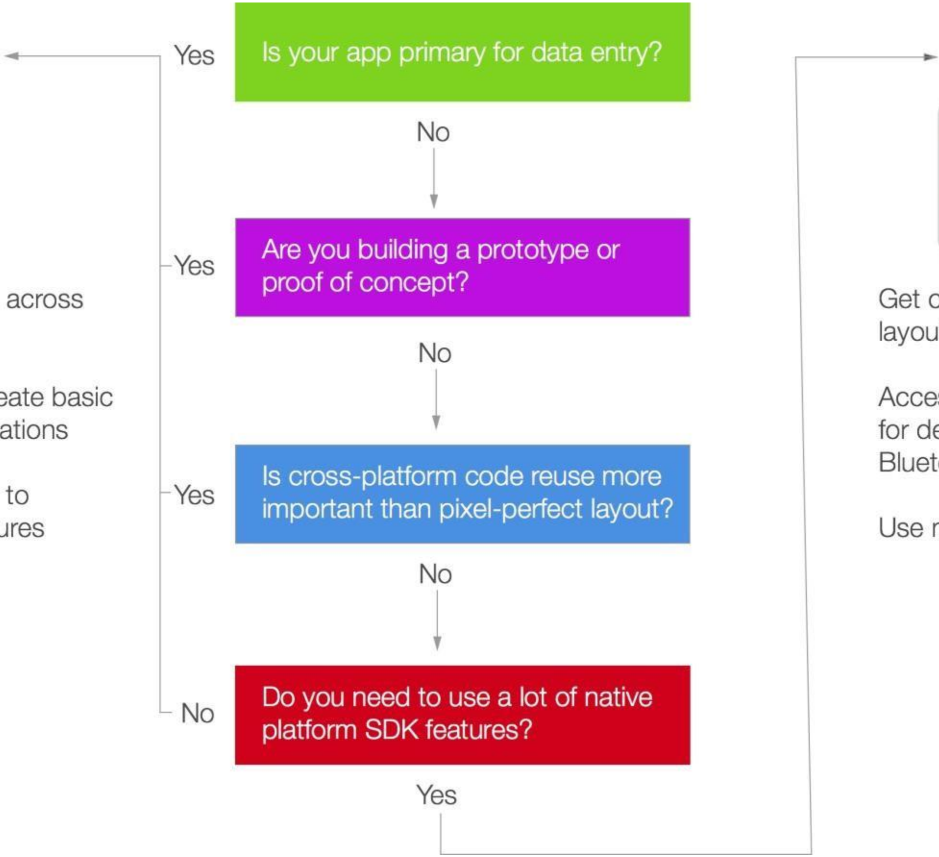


Use Xamarin.Forms

Reuse 90%+ of your UI code across iOS, Android and Windows.

Use Custom Renderers to create basic platform-specific UI customizations

Use the Dependency Service to access platform-specific features



Yes
Is your app primary for data entry?
No
Yes
Are you building a prototype or proof of concept?
No
Yes
Is cross-platform code reuse more important than pixel-perfect layout?
No
No
Do you need to use a lot of native platform SDK features?
Yes

Use Xamarin.iOS and Xamarin.Android

Get complete control of the UI, animations, layout and special effects

Access 100% of the platform features and SDK for deep integration with the platform (camera, Bluetooth, NFC, etc.)

Use native 3rd party controls

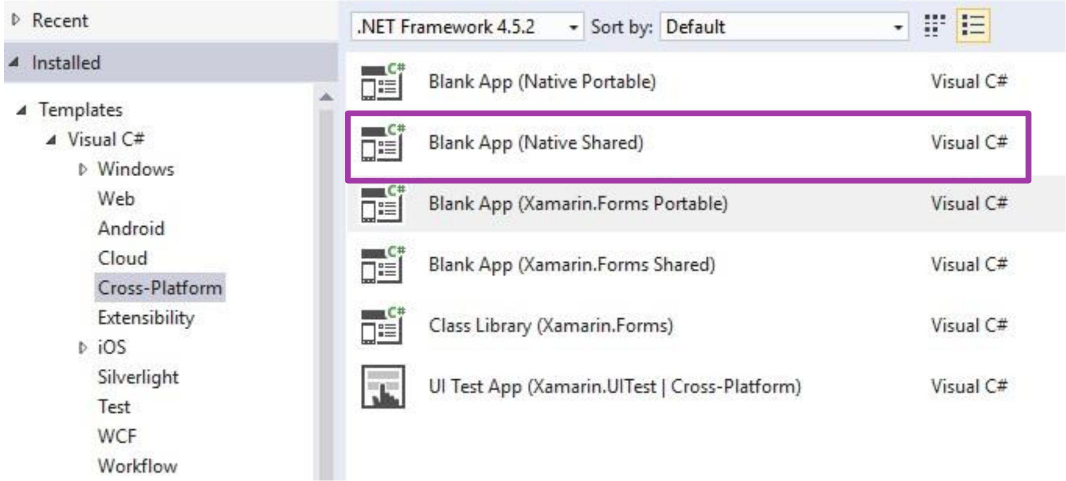
Xamarin Forms Application Structure

Tasks

- Xamarin.Forms project structure
- Application Components
- "Hello, Forms!"

Creating a Xamarin.Forms App

- Built-in project templates for Xamarin.Forms applications available under Cross-Platform
  - Blank App to create a new application
  - Class Library to create a PCL for use with Xamarin.Forms

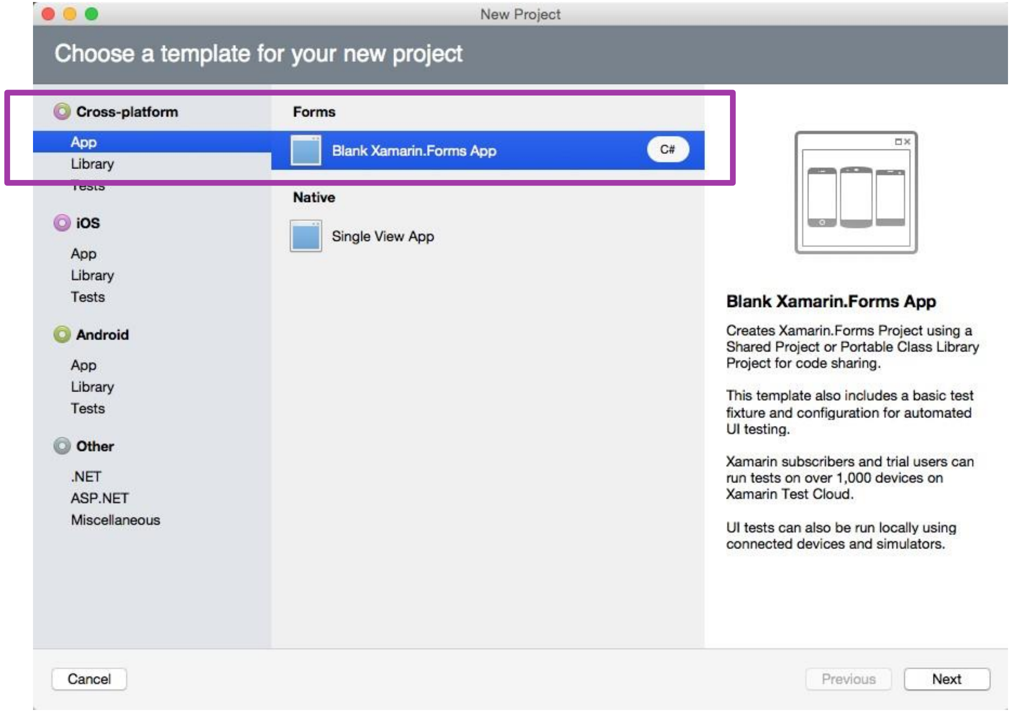


.NET Framework 4.5.2 Sort by: Default

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

Creating a Xamarin.Forms App

- ❖ Xamarin Studio on the Mac supports Android + iOS
- ❖ Xamarin Studio on Windows supports only Android
- ❖ Project wizard lets you select code sharing technique (PCL vs. Shared Project)



Blank Xamarin.Forms App

Creates Xamarin.Forms Project using a Shared Project or Portable Class Library Project for code sharing.

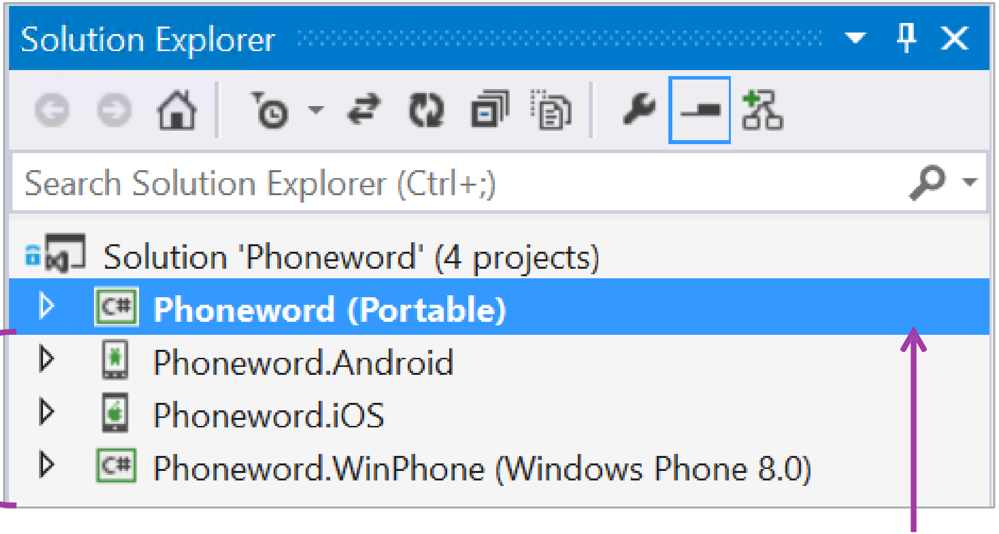
This template also includes a basic test fixture and configuration for automated UI testing.

Xamarin subscribers and trial users can run tests on over 1,000 devices on Xamarin Test Cloud.

UI tests can also be run locally using connected devices and simulators.

Project Structure

- Blank App project template creates several related projects



Solution 'Phoneword' (4 projects)

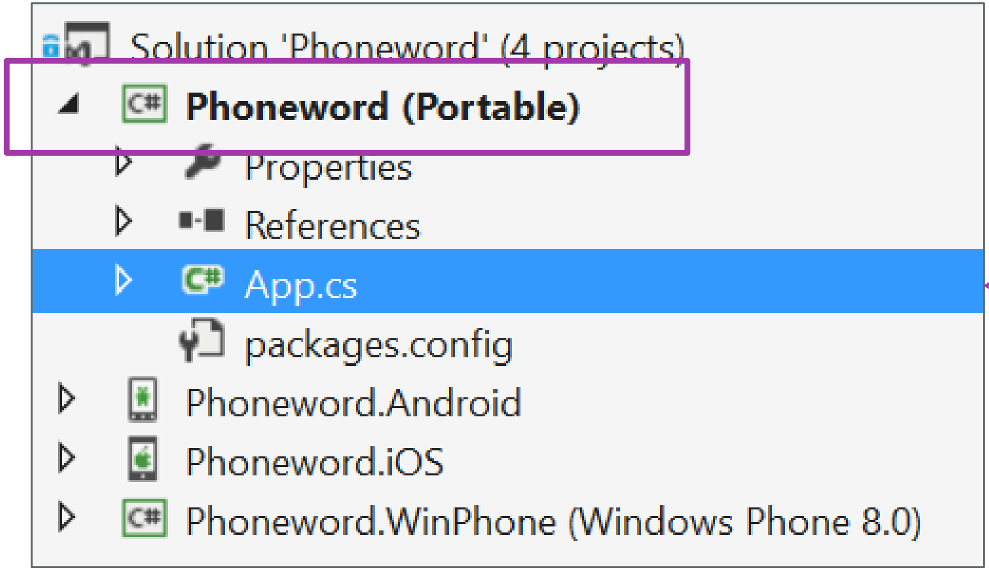
- Phoneword (Portable)
- Phoneword.Android
- Phoneword.iOS
- Phoneword.WinPhone (Windows Phone 8.0)

Platform-specific projects act as "host" to create native application

Portable Class Library used to hold shared code that defines UI and logic

Project Structure - PCL

- Most of your code will go into the PCL used for shared logic + UI



Solution 'Phoneword' (4 projects)

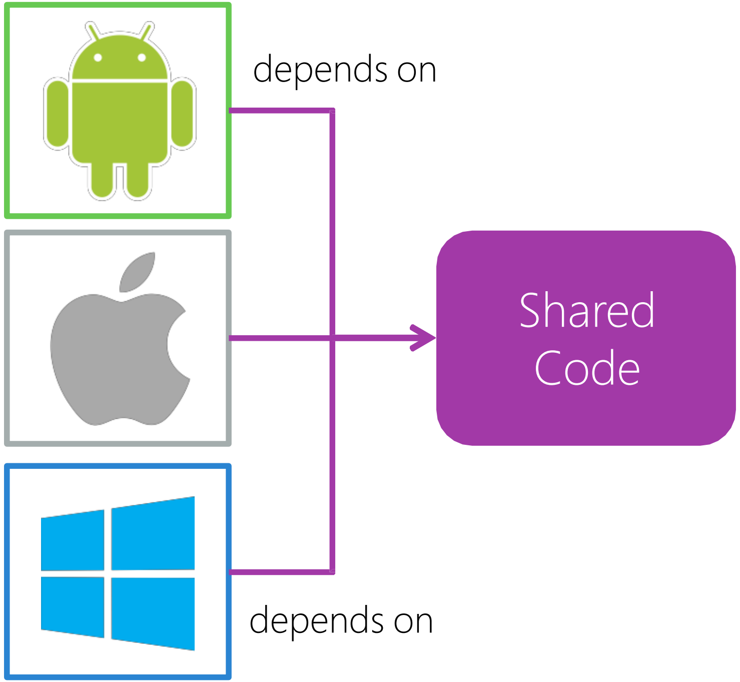
- Phoneword (Portable)
  - Properties
  - References
  - App.cs
  - packages.config
- Phoneword.Android
- Phoneword.iOS
- Phoneword.WinPhone (Windows Phone 8.0)

Default template creates a single **`App.cs`** file which decides the initial screen for the application

Project Structure - Dependencies

- Platform-specific projects depend on the shared code (PCL or SAP), but *not* the other way around

- Xamarin.Forms defines the UI and behavior in the PCL or SAP (shared) and then calls it from each platform-specific project



Xamarin.Forms app anatomy

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

Application

Provides initialization for the application

Page(s)

Represents a single screen to display

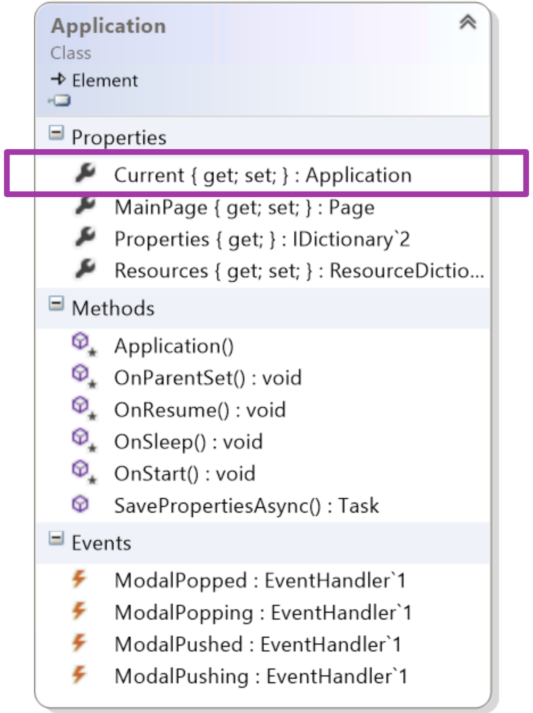
Demonstration

Creating a Xamarin.Forms application

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`**



Application
Class
→ Element

Properties
- Current { get; set; } : Application
- MainPage { get; set; } : Page
- Properties { get; } : IDictionary`2
- Resources { get; set; } : ResourceDictio...

Methods
- Application()
- OnParentSet() : void
- OnResume() : void
- OnSleep() : void
- OnStart() : void
- SavePropertiesAsync() : Task

Events
- ModalPopped : EventHandler`1
- ModalPopping : EventHandler`1
- ModalPushed : EventHandler`1
- ModalPushing : EventHandler`1

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

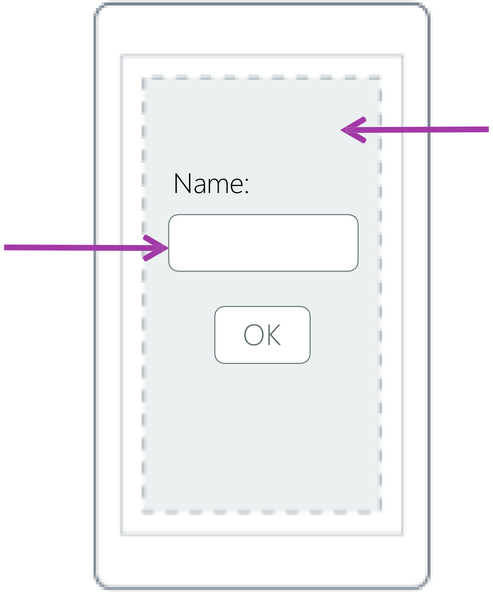
- **Application** class also includes a **string** >> **object** property bag which is persisted between app launches

```
// 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;
}
```

Creating the application UI

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

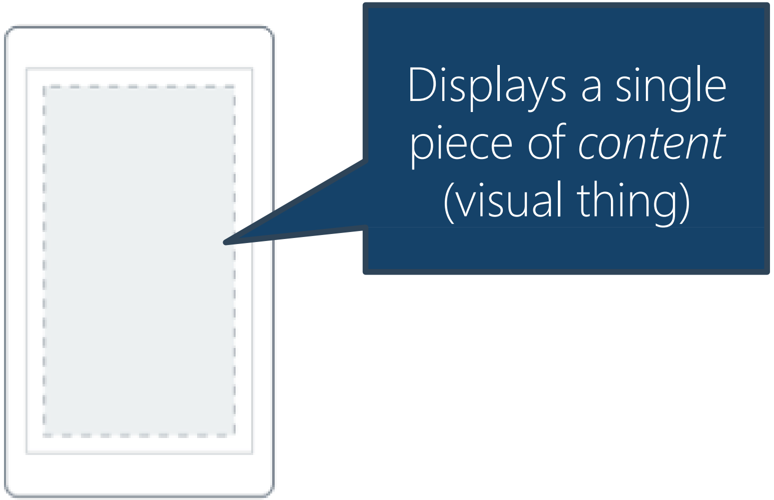


Page represents a single screen displayed in the app

Views are the UI controls the user interacts with

Pages

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

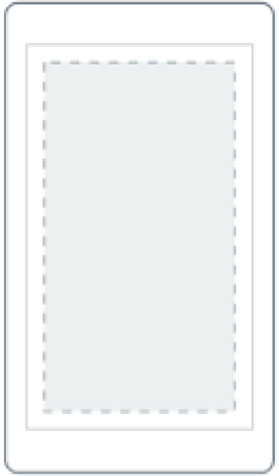


Displays a single piece of *content* (visual thing)

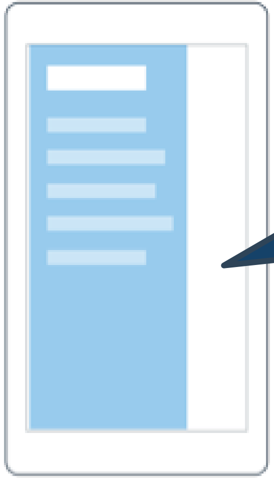
Content

Pages

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



Content

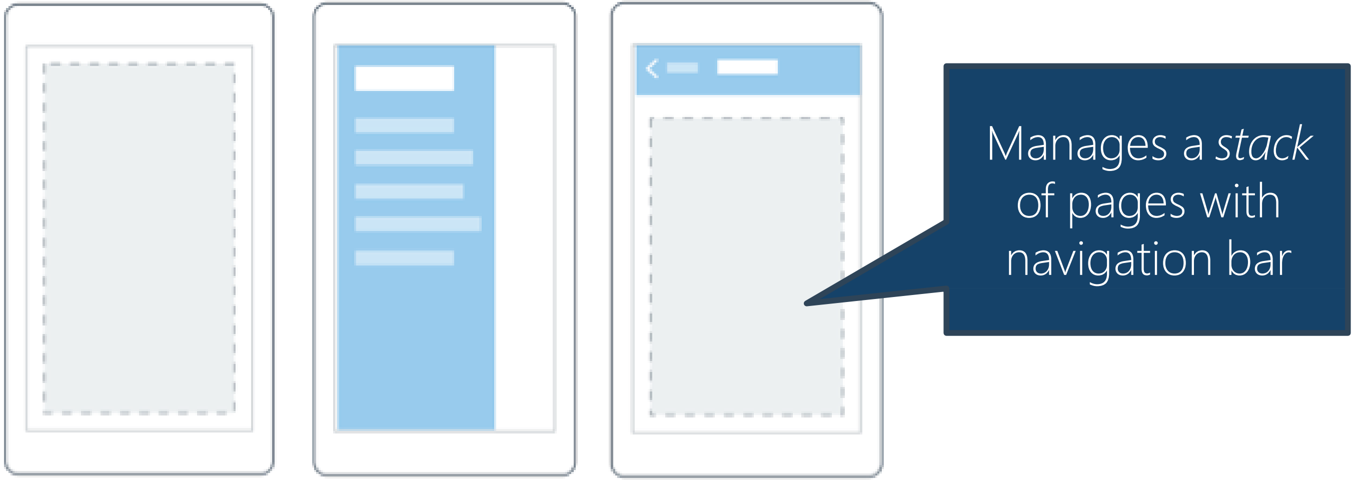


Master Detail

Manages two panes of information

Pages

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



Content

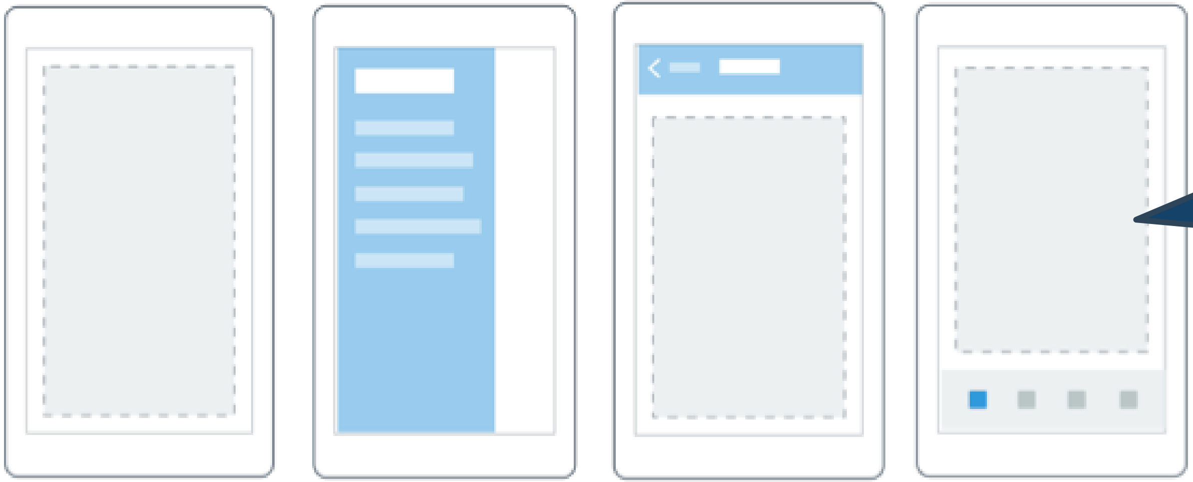
Master Detail

Navigation

Manages a *stack* of pages with navigation bar

Pages

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



Content

Master Detail

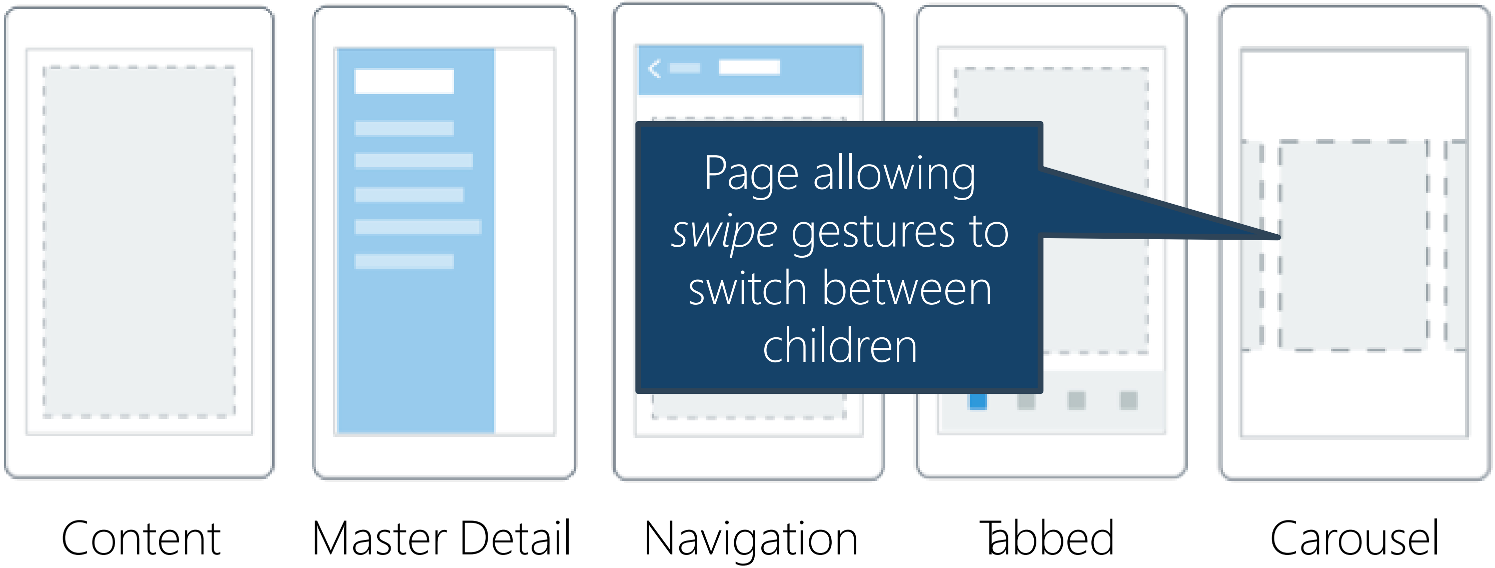
Navigation

Tabbed

Page that navigates between children using tab bar

Pages

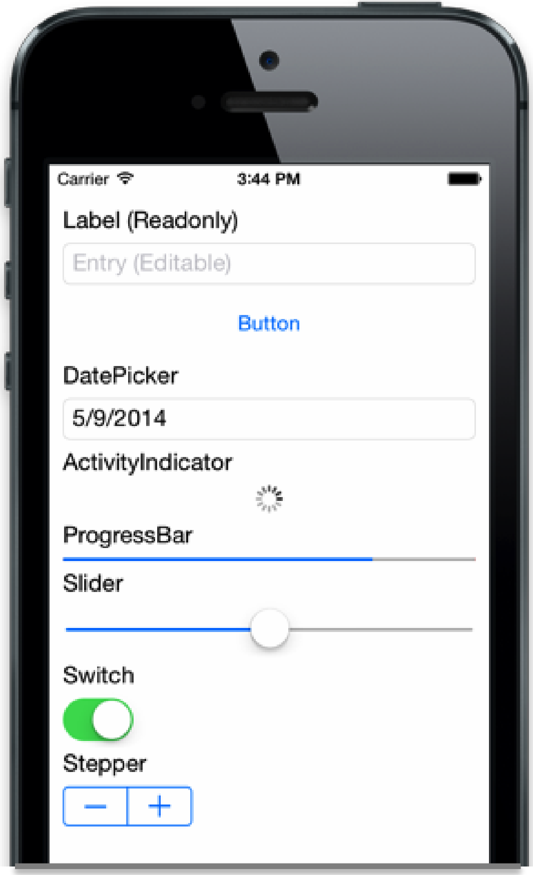
- **Page** is an abstract class used to define a single screen of content
  - derived types provide specific visualization / behavior



Views

- View is the base class for all visual controls, most standard controls are present

Label	Image	SearchBar
Entry	ProgressBar	ActivityIndicator
Button	Slider	OpenGLView
Editor	Stepper	WebView
DatePicker	Switch	ListView
BoxView	TimePicker	
Frame	Picker	



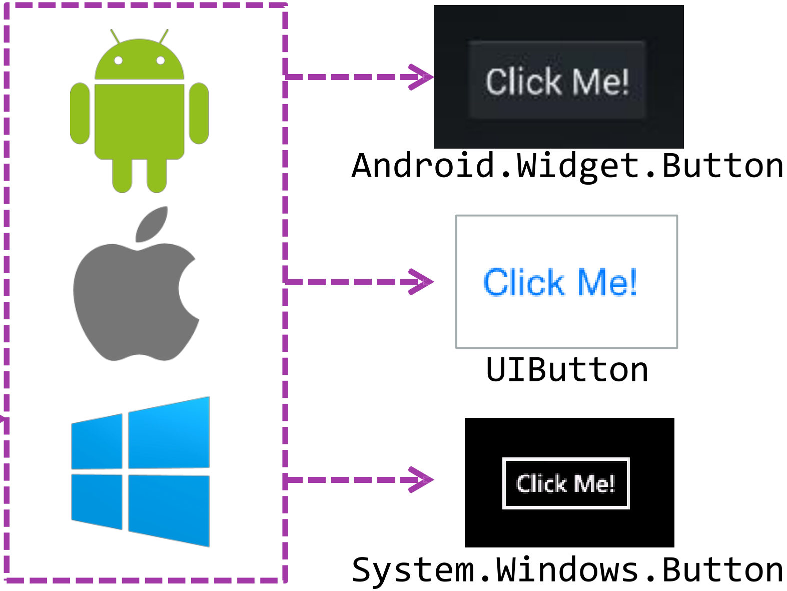
Rendering views

- 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 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

Group Exercise

Creating our first Xamarin.Forms application

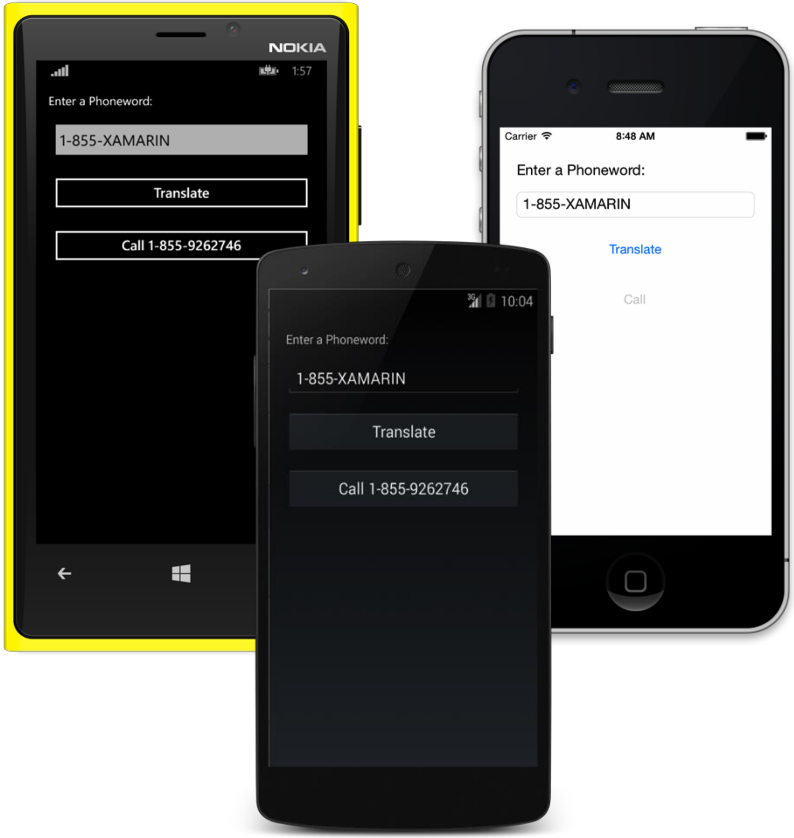
Summary

1. Xamarin.Forms project structure
2. Application Components
3. "Hello, Forms!"

Creating Phoneword in Xamarin.Forms

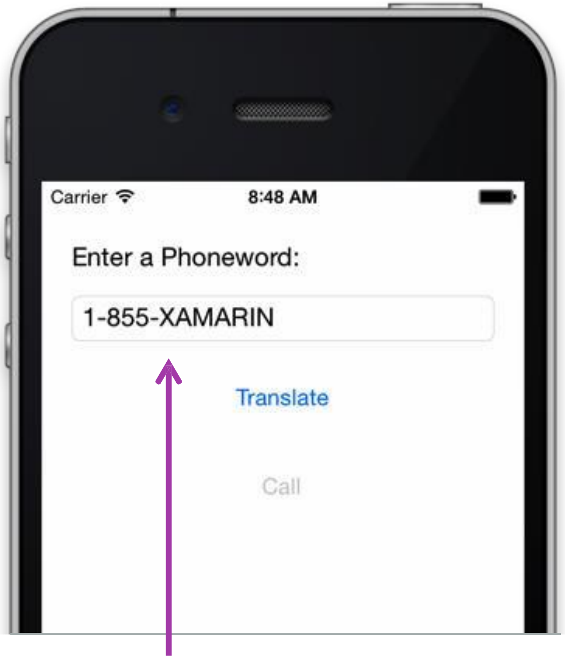
Tasks

1. Layout containers
2. Adding views
3. Fine-tuning layout



Organizing content

- Rather than specifying positions with coordinates (pixels, dips, etc.), you use layout containers to control how views are positioned relative to each other; this provides for a more *adaptive* layout which is not as sensitive to dimensions and resolutions



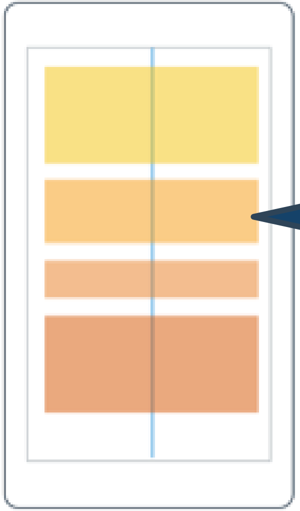
Enter a Phoneword:

1-855-XAMARIN

For example, "stacking" views on top of each other with some spacing between them

Layout containers

- *Layout Containers* organize child elements based on specific rules

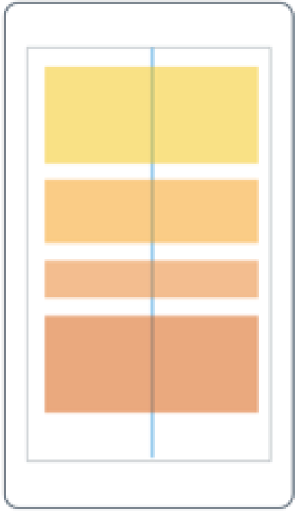


StackLayout

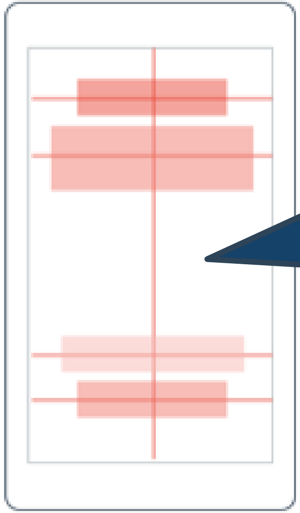
StackLayout places children top-to-bottom (default) or left-to-right based on **Orientation** property setting

Layout containers

- *Layout Containers* organize child elements based on specific rules



StackLayout

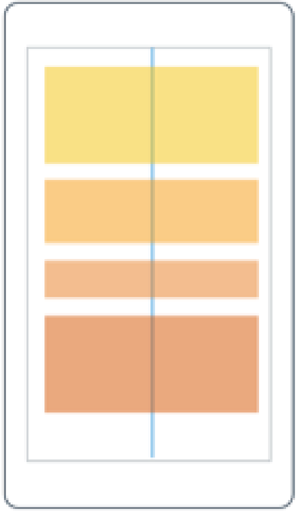


AbsoluteLayout

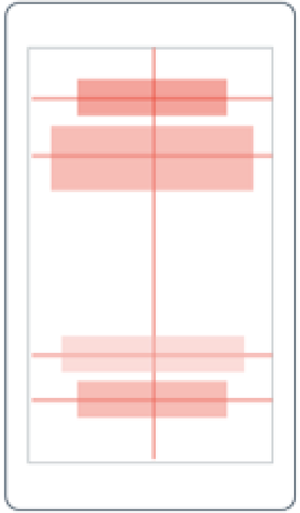
`AbsoluteLayout` places children in absolute requested positions based on anchors and bounds

Layout containers

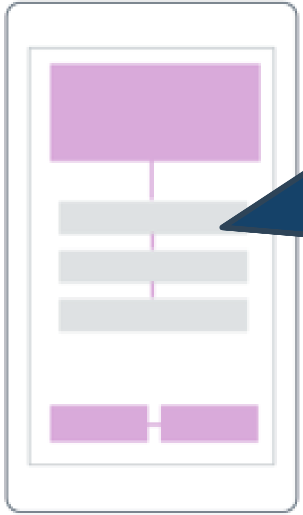
❖ *Layout Containers* organize child elements based on specific rules



StackLayout



Absolute Layout

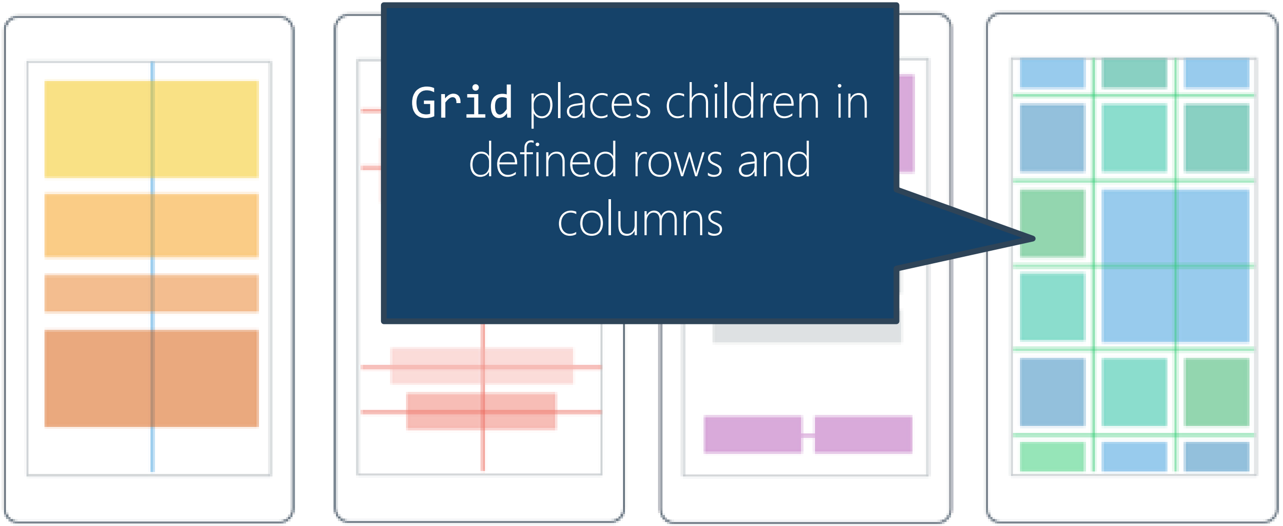


Relative Layout

`RelativeLayout` uses constraints to position the children

Layout containers

- *Layout Containers* organize child elements based on specific rules



`Grid` places children in defined rows and columns

StackLayout

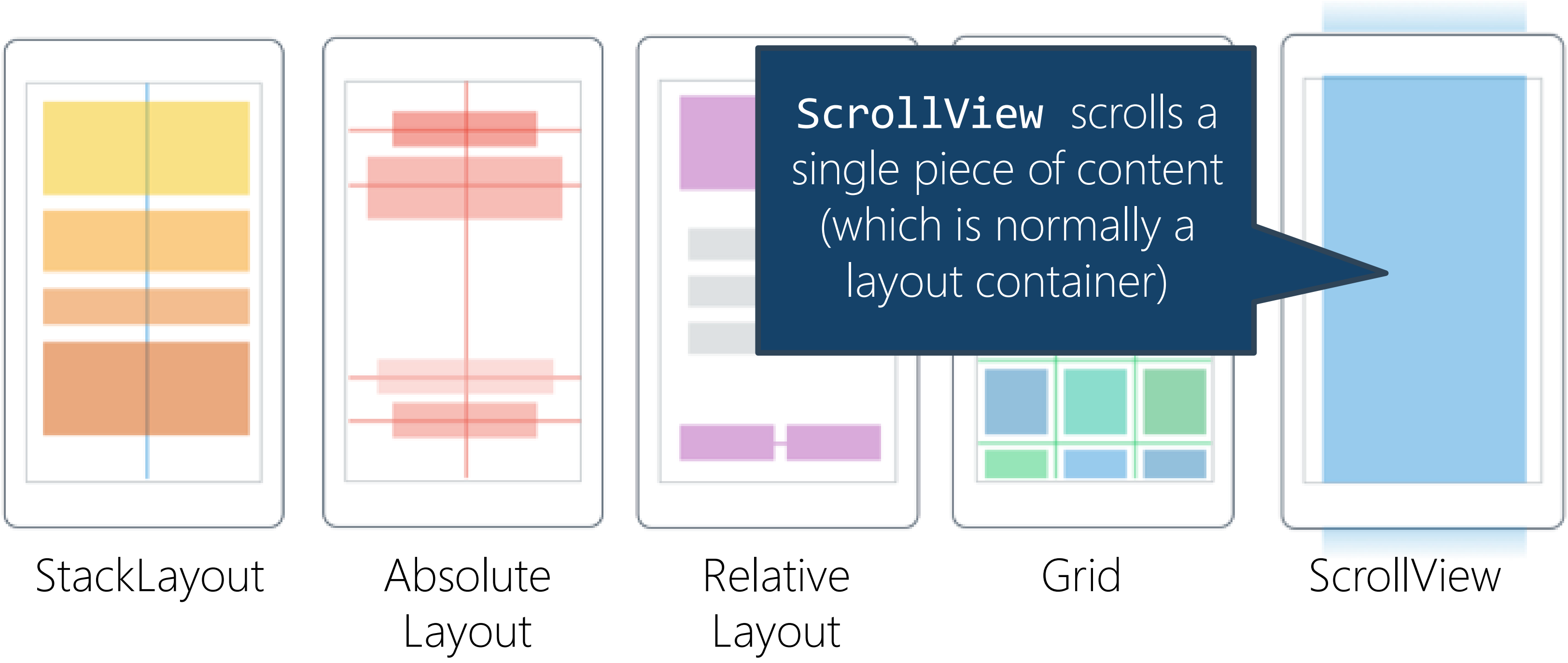
Absolute Layout

Relative Layout

Grid

Layout containers

❖ *Layout Containers* organize child elements based on specific rules



StackLayout

Absolute Layout

Relative Layout

Grid

ScrollView

`ScrollView` scrolls a single piece of content (which is normally a layout container)

Adding views to layout containers

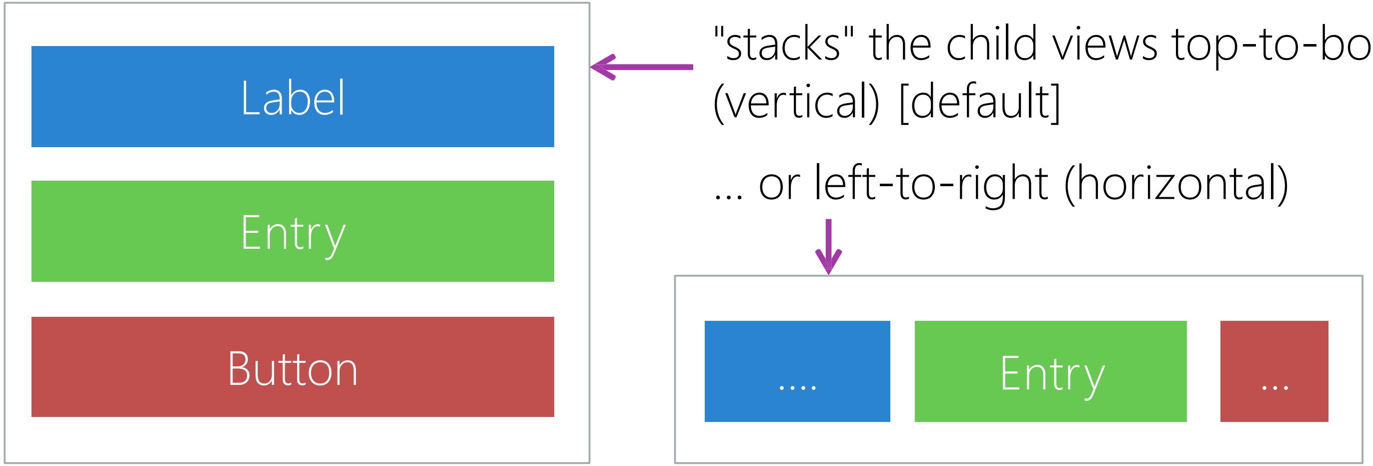
- Layout containers have a **`Children`** collection property which is used to hold the views that will be organized by the container

```
Label label = new Label { Text = "Enter Your Name" };
Entry nameEntry = new Entry();

StackLayout layout = new StackLayout();
layout.Children.Add(label);
layout.Children.Add(nameEntry);

this.Content = layout;
```

Views are laid out and rendered in the order they appear in the collection

Working with StackLayout

❖ `StackLayout` is used to create typical form style layout



"stacks" the child views top-to-bottom (vertical) [default]

... or left-to-right (horizontal)

The **`Orientation`** property can be set to either **`Horizontal`** or **`Vertical`** to control which direction the child views are stacked in

Element spacing

- Properties used to control sizing and spacing on managed layouts

Name	Purpose	Used On

`VerticalOptions,`
`HorizontalOptions`
	Determines how child content is stretched or positioned	Any **`View`** type, but most often set on the layout containers

`Spacing`	Spacing added between child elements, rendered in the platform measurement system	**`StackLayout`** container
`Padding`	Padding added around element	Any **`Page`** type – almost always set to inset page

Controlling Width and Height

- Can request a width / height for a view

Name	Purpose

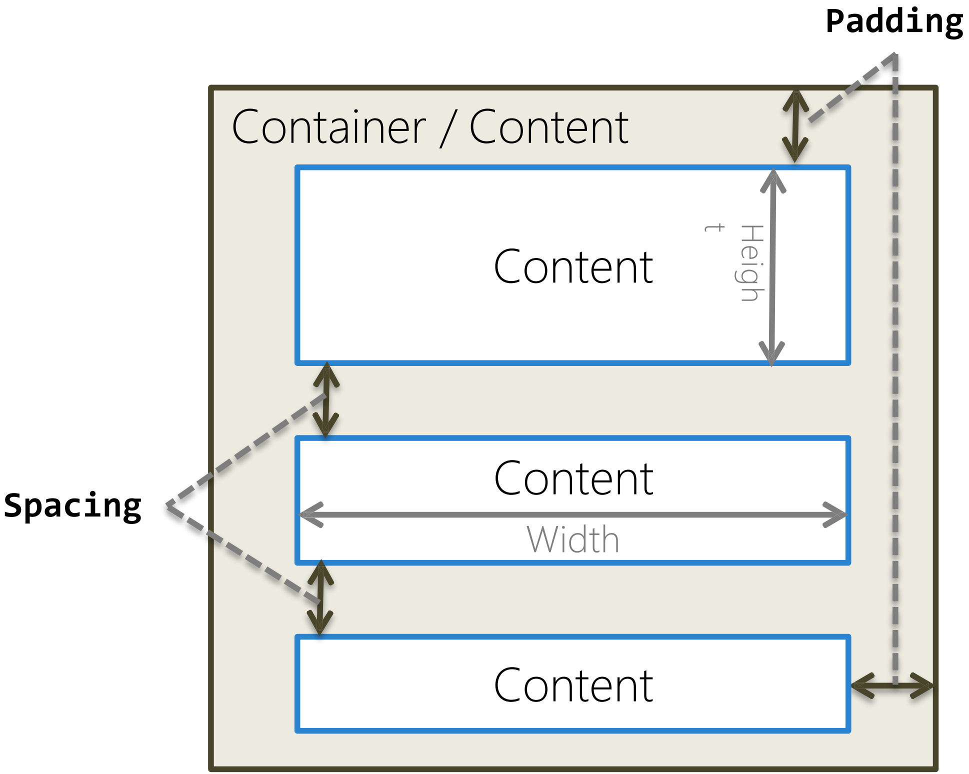
`WidthRequest,`
`HeightRequest`
	Request a specific width & height for the element. Overrides the measured size of the element.

`MinimumWidthRequest,`
`MinimumHeightRequest`
	Request a minimum width & height, can be made larger to fit content if necessary.

`Width, Height`	(read-only) Final, calculated width & height
`Bounds`	(read-only) Position and Size of the frame relative to the parent's coordinates.

Understanding Layout

- Layout uses the CSS Box Model (with no margin value)
- Content may itself be a container
- Use `WidthRequest` and `HeightRequest` to override the measured size

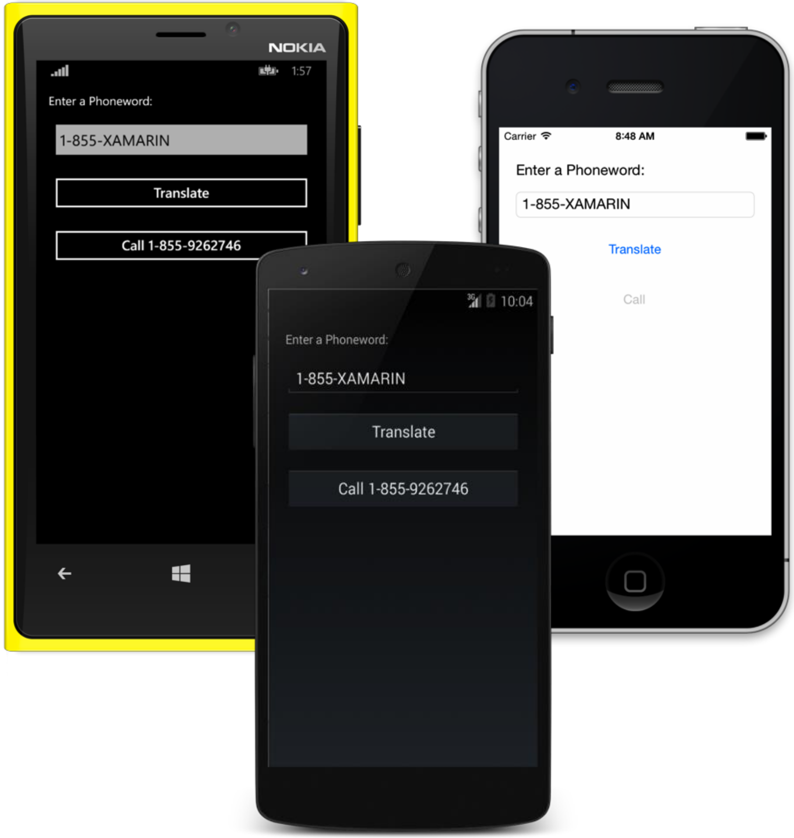


Individual Exercise

Creating Xamarin.Forms Phoneword

Summary

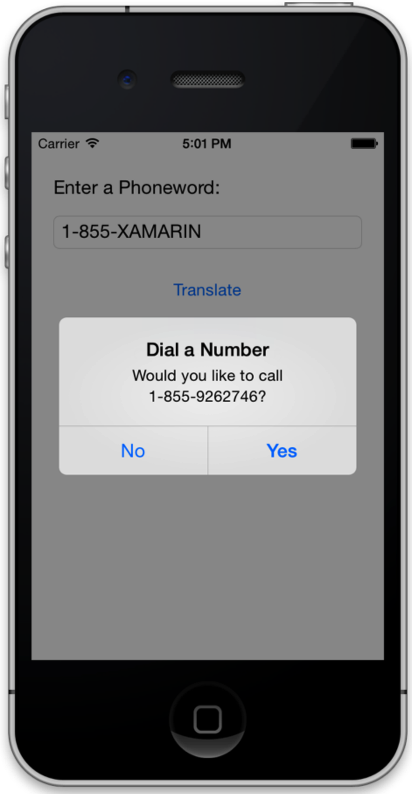
- ❖ Layout containers
- ❖ Adding views
- ❖ Fine-tuning layout



Using Platform-Specific Code

Tasks

1. Changing the UI per-platform
2. Using Platform features
3. Working with `DependencyService`



Recall: Xamarin.Forms architecture

- Xamarin.Forms applications have two projects that work together to provide the logic + UI for each executable

Portable
Class Library

Platform
Specific project

- *shared* across all platforms
- limited access to .NET APIs
- want most of our code here

- 1-per platform
- code is *not* shared
- full access to .NET APIs
- any platform-specific code must be located in these projects

Changing the UI per-platform

- **`Device.OnPlatform`** allows you to fine-tune the UI for each platform

```
Device.OnPlatform(
   iOS: () => { ... },
   Android: () => { ... },
   WinPhone: () => { ... },
   Default: () => { ... });
```

Can execute specific logic per-platform using delegates for each platform

```
new Thickness(5,
    Device.OnPlatform(20, 0, 0),
    5, 5);
```

Can return a different value per-platform (iOS, Android, WinPhone) using **`Device.OnPlatform<T>`**

This code is used in the shared code but only uses one of the supplied values or delegates when the code is executed on a specific platform

Detecting the platform

- Can use the static **Device** class to identify the platform and device style

```
if (Device.Idiom == TargetIdiom.Tablet) {
    // Code for tablets only
    if (Device.OS == TargetPlatform.iOS) {
        // Code for iPad only
    }
}
```

Note that this does not allow for *platform-specific code* to be executed, it allows runtime detection of the platform to execute a unique branch of code in your shared PCL

Using Platform Features

- Xamarin.Forms has support for dealing with a few, very common platform-specific features

`Device.OpenUri`
to launch external apps based on a URL scheme

`Page.DisplayAlert`
to show simple alert messages

Timer management using
`Device.StartTimer`

Using Platform Features

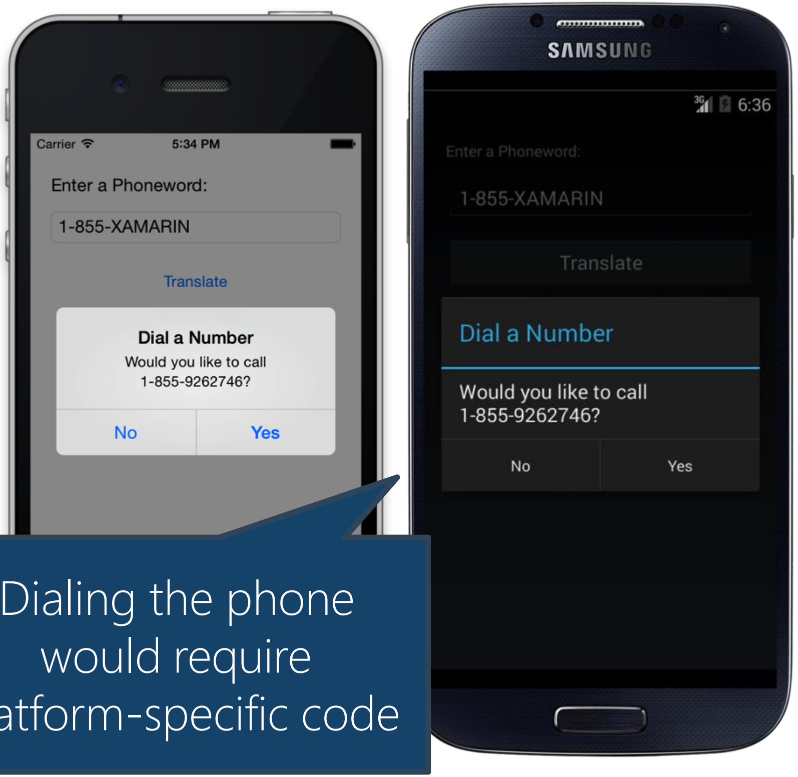
- Xamarin.Forms has support for dealing with a few, very common platform-specific features

UI Thread marshaling with Device.BeginInvokeOnMainThread

Mapping and Location through `Xamarin.Forms.Maps`

Other platform-specific features

- Platform features *not* exposed by Xamarin.Forms can be used, but will require some architectural design
  - code goes into platform-specific projects
  - often must (somehow) use code from your shared logic project



Enter a Phoneword:

1-855-XAMARIN

Dial a Number

Would you like to call 1-855-9262746?

Dialing the phone would require platform-specific code

Creating abstractions

- Best practice to build an *abstraction* implemented by the target platform which defines the platform-specific functionality

```
public interface IDialer
{
    bool MakeCall(string number);
}
```

Shared code defines **IDialer** interface to represent required functionality

PhoneDialerIOS

PhoneDialerDroid

PhoneDialerWP8

Platform projects implement the shared dialer interface using the platform-specific APIs

Locating dependencies

- Xamarin.Forms includes a *service locator* called **`DependencyService`** which can be used to register platform-specific implementations and then locate them through the abstraction in your shared code

1 Define an interface or abstract class in the shared code project (PCL)

```
public interface IDialer
{
    bool MakeCall(string number);
}
```

Locating dependencies

- Xamarin.Forms includes a *service locator* called **`DependencyService`** which can be used to register platform-specific implementations and then locate them through the abstraction in your shared code

2 Provide implementation of abstraction in each platform-specific project

```
class PhoneDialerIOS : IDialer
{
    public bool MakeCall(string number) {
      // Implementation goes here
    }
}
```

Locating dependencies

- Xamarin.Forms includes a *service locator* called **`DependencyService`** which can be used to register platform-specific implementations and then locate them through the abstraction in your shared code

3 Expose platform-specific implementation using assembly-level attribute in platform-specific project

```
[assembly: Dependency(typeof(PhoneDialerIOS))]
```

Implementation type is supplied to attribute as part of registration

Locating dependencies

- Xamarin.Forms includes a *service locator* called **`DependencyService`** which can be used to register platform-specific implementations and then locate them through the abstraction in your shared code

4 Retrieve and use the dependency anywhere using **`DependencyService.Get<T>`** (both shared and platform specific projects can use this API)

```
IDialer dialer = DependencyService.Get<IDialer>();
if (dialer != null) {
   ...
}
```

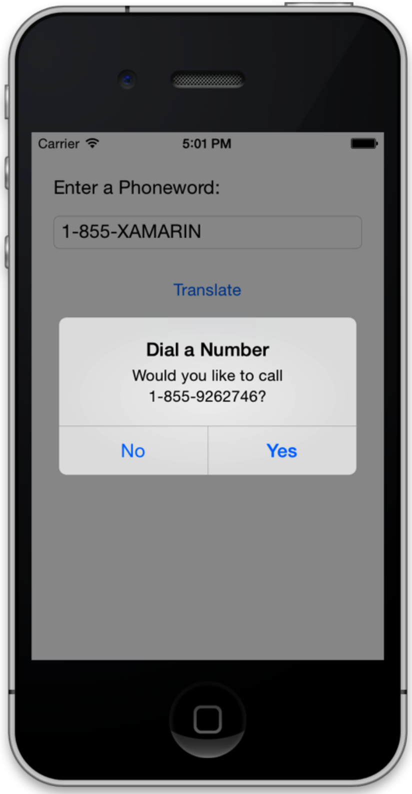
Request the *abstraction* and the implementation will be returned

Individual Exercise

Adding support for dialing the phone

Summary

- ❖ Changing the UI per-platform
- ❖ Using Platform features
- ❖ Working with `DependencyService`



Dial a Number

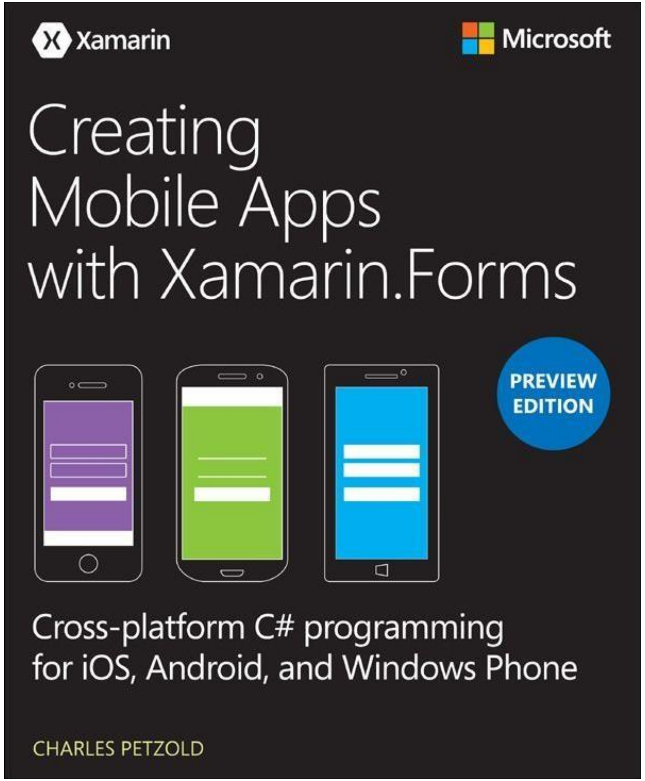
Would you like to call 1-855-9262746?

What's Next?

- XAM130 contines your exploration of Xamarin.Forms by diving into XAML

- For more in-depth information, download Charles Petzold's book online:

  bit.ly/xforms-book


Xamarin
Microsoft
Creating Mobile Apps with Xamarin.Forms
PREVIEW EDITION
Cross-platform C# programming for iOS, Android, and Windows Phone
CHARLES PETZOLD

Thank You!