

XAM120



Introduction to Xamarin.Forms

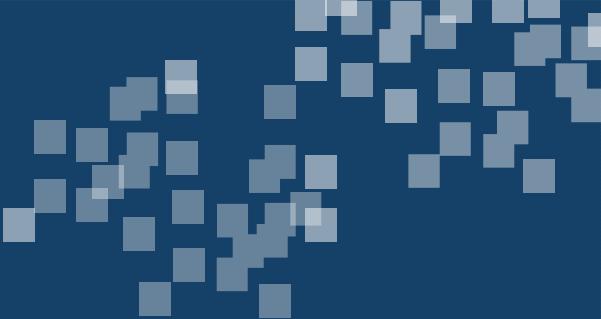


Xamarin University

Objectives

1. Create a single screen cross-platform application
2. Arrange the UI using Layouts
3. Use platform-specific features in shared code

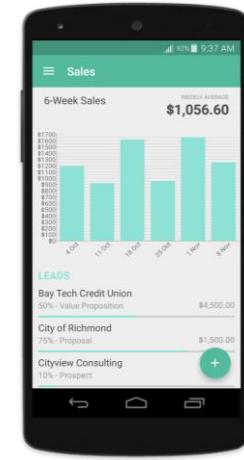
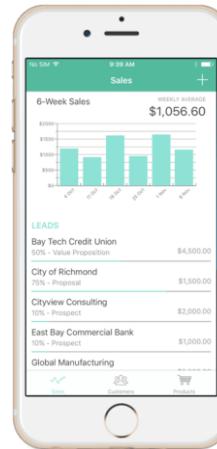




Create a single screen cross-platform application

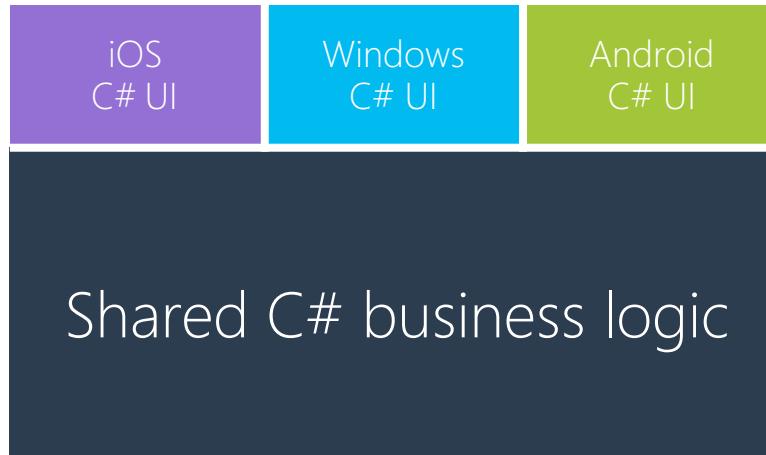
Tasks

- ❖ Compare traditional development to Xamarin.Forms
- ❖ Understand Xamarin.Forms project structure
- ❖ Use application components
- ❖ Create a Xamarin.Forms app



Xamarin.iOS and Xamarin.Android

- ❖ Traditional Xamarin approach allows for shared business logic and non-sharable platform-specific code for the UI layer



Xamarin.iOS and Xamarin.Android

Xamarin.Forms

- ❖ Xamarin.Forms allows you to describe the UI once using a shared set of elements which create a native UI at runtime



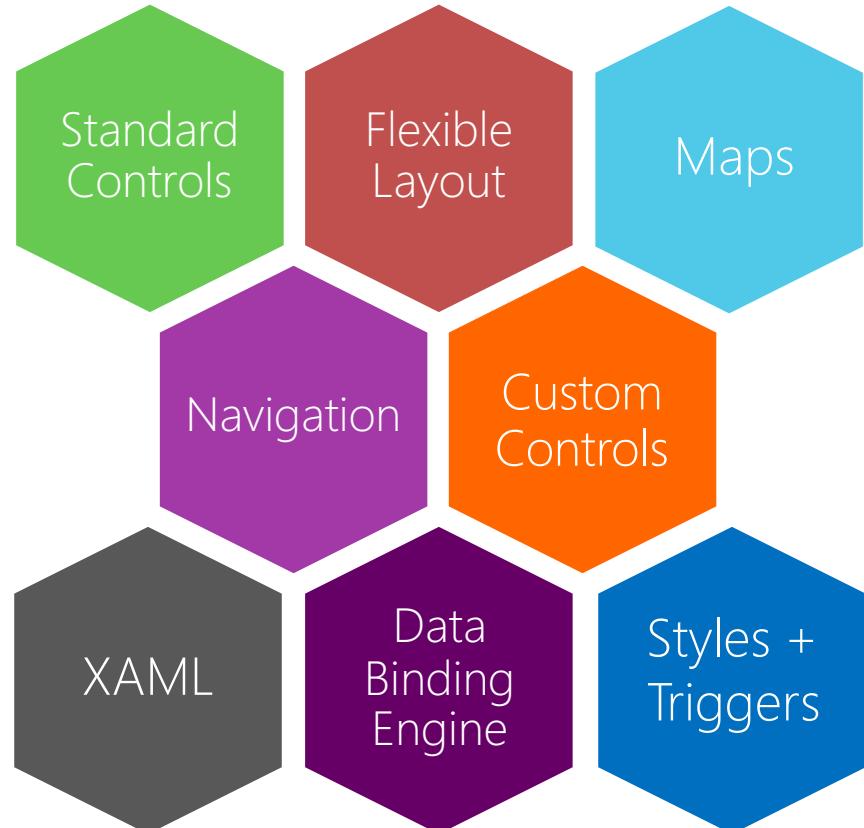
Xamarin.iOS and Xamarin.Android



Xamarin.Forms

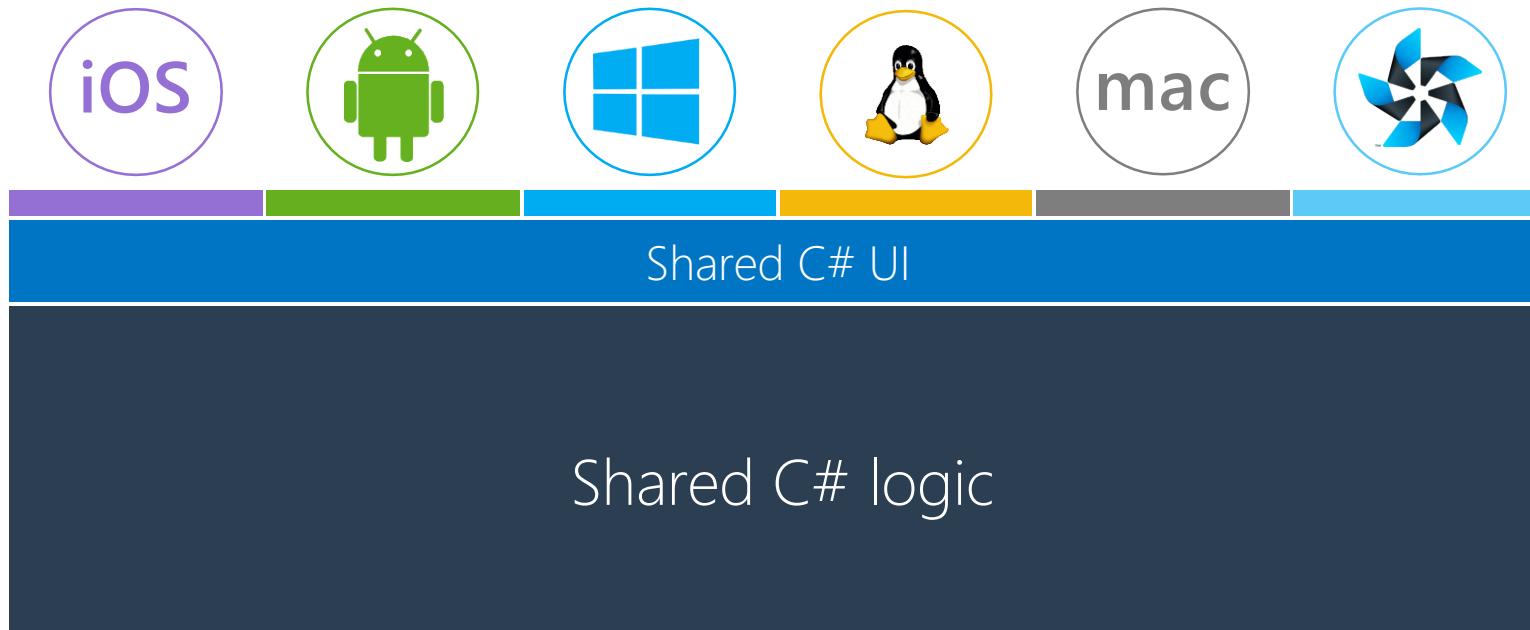
What is Xamarin.Forms?

- ❖ Xamarin.Forms is a cross-platform UI framework to create mobile apps for:
 - Android 4.0+
 - iOS 8.0+
 - Windows 10



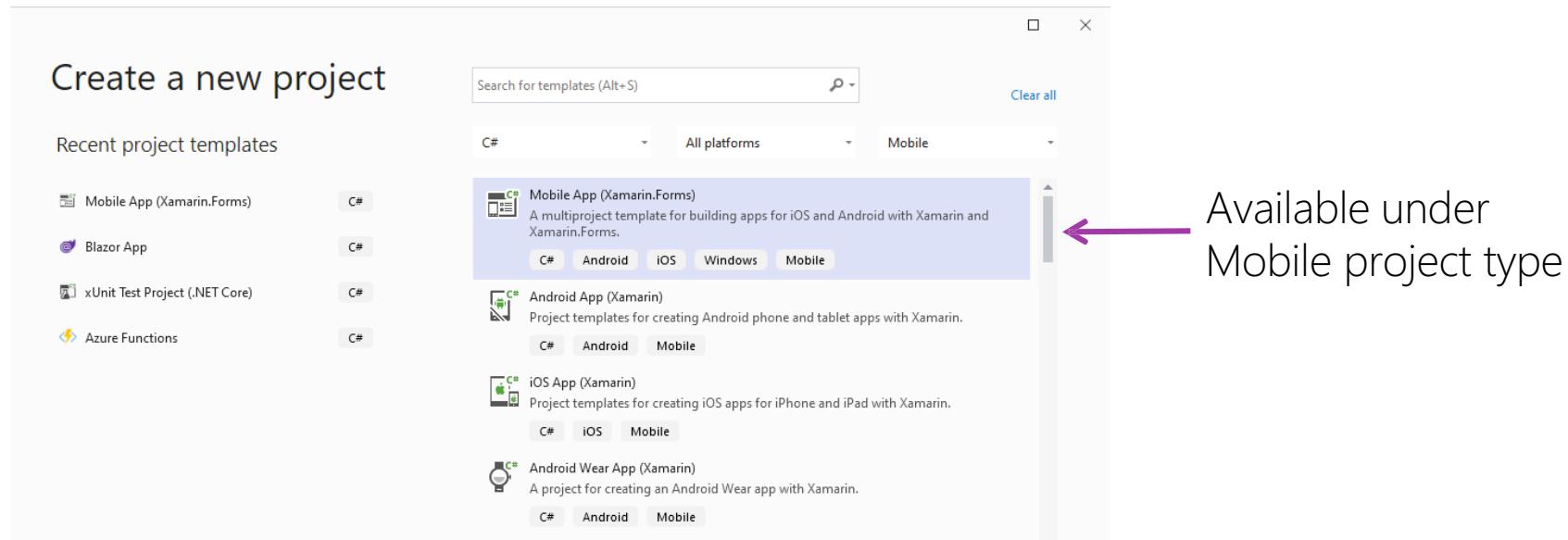
Xamarin.Forms platform support

- ❖ Xamarin.Forms supports a broad selection of mobile and desktop platforms and UI frameworks



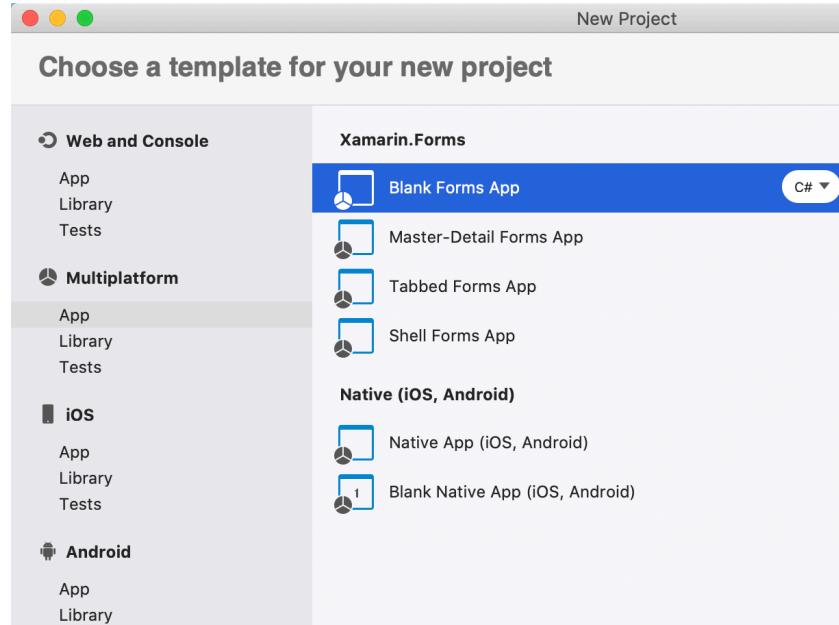
Creating a Xamarin.Forms App [Windows]

- ❖ Visual Studio for Windows includes built-in project templates for Xamarin.Forms applications



Creating a Xamarin.Forms App [Mac]

- ❖ Visual Studio for Mac includes built-in project templates for Xamarin.Forms applications

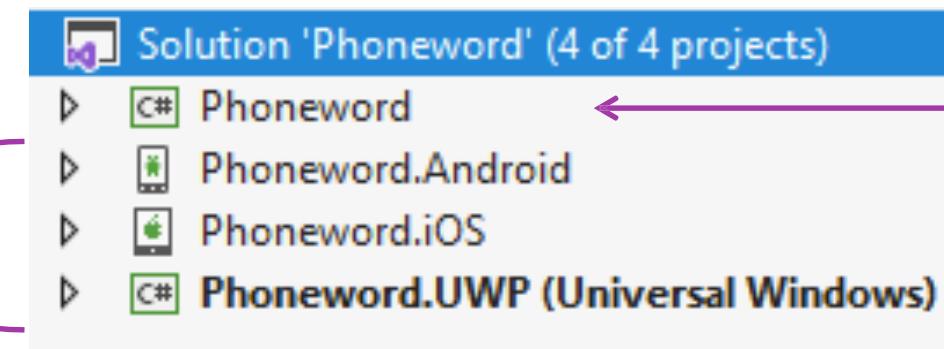


← After selecting your template, a project wizard walks through the available options

Project Structure

- ❖ The Xamarin Cross Platform App project template creates several related projects

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

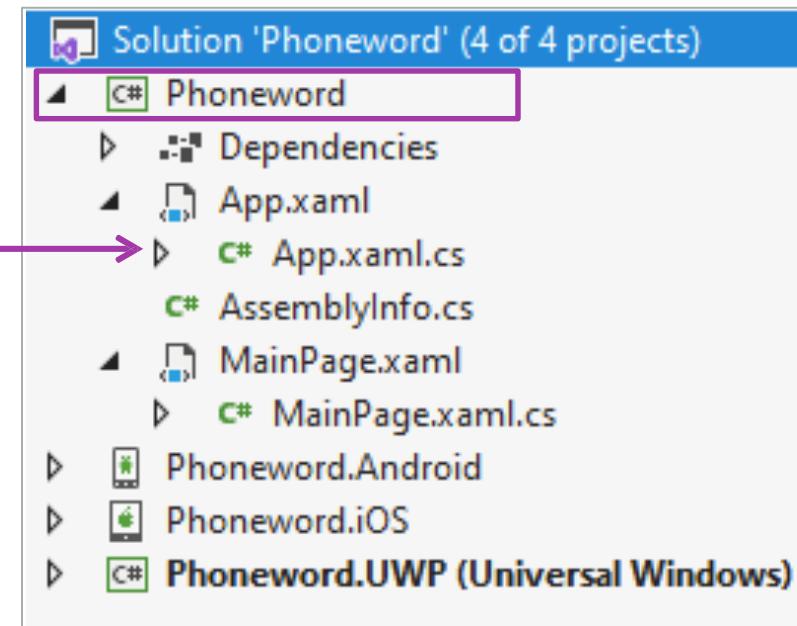


.NET Standard project used to hold shared code that defines UI and logic

Project Structure - .NET Standard

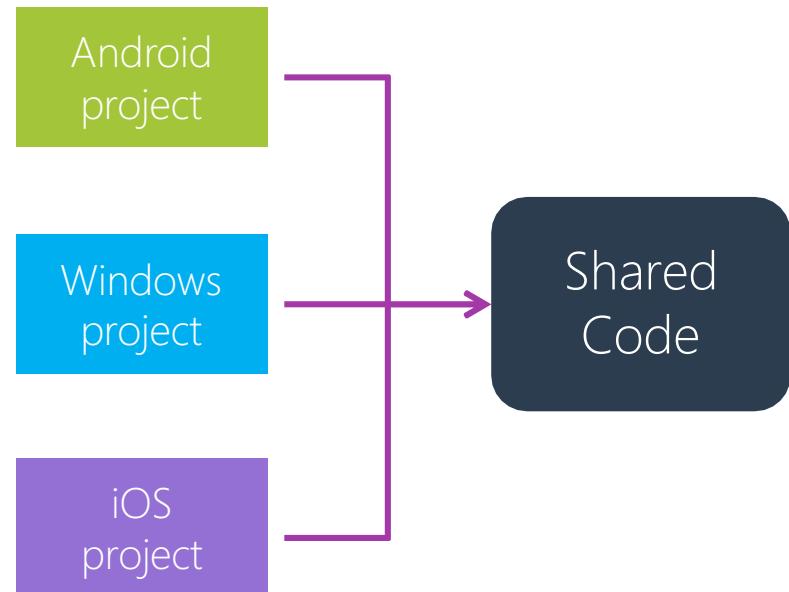
- ❖ Most of your code will go into the .NET Standard used for shared logic + UI

Default template creates an App class which decides the initial screen for the application



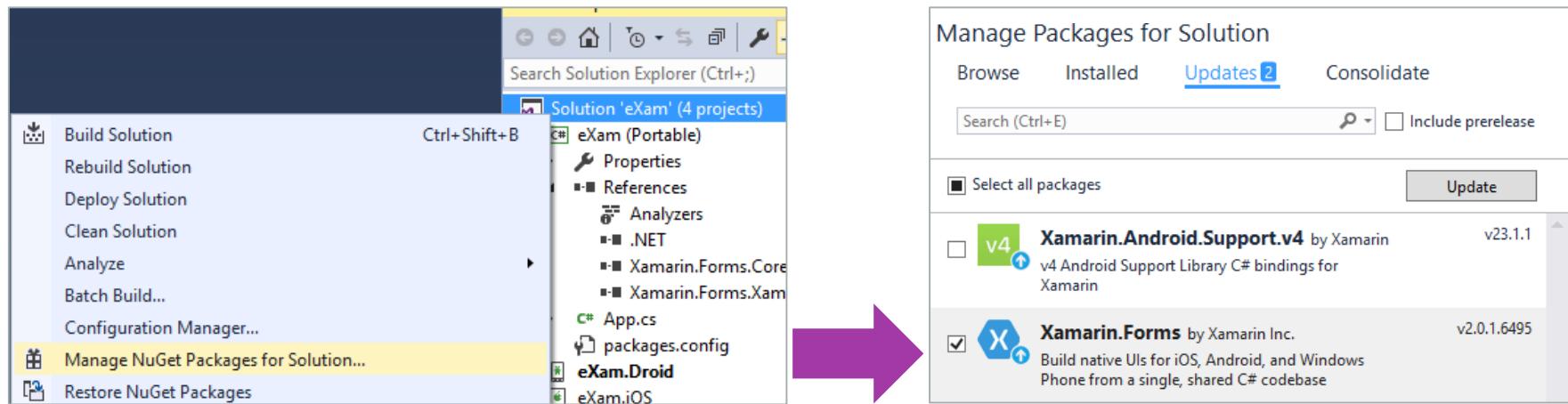
Project Structure - Dependencies

- ❖ Platform-specific projects use the shared code (.NET Standard or SAP), but *not* the other way around
- ❖ Xamarin.Forms defines the UI and behavior in the .NET Standard or SAP (shared) and then calls it from each platform-specific project



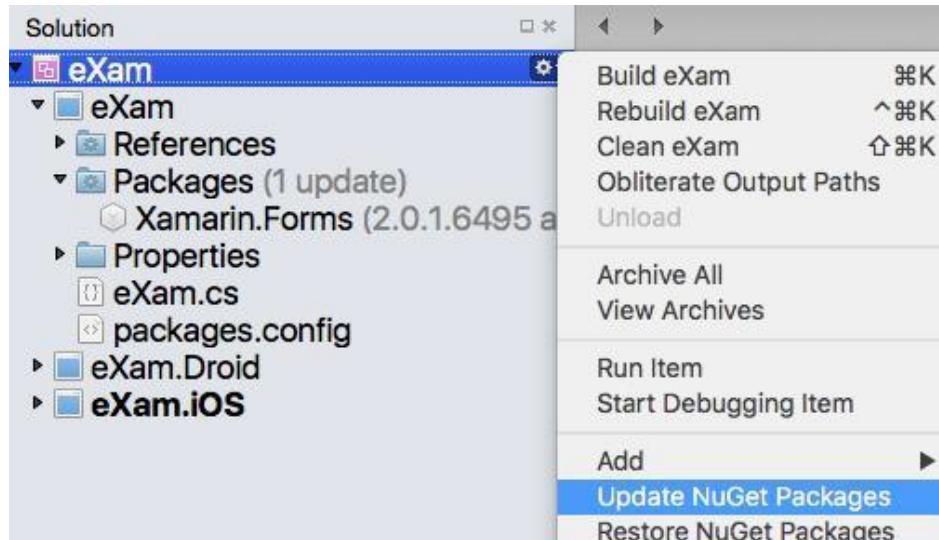
Xamarin.Forms updates [Windows]

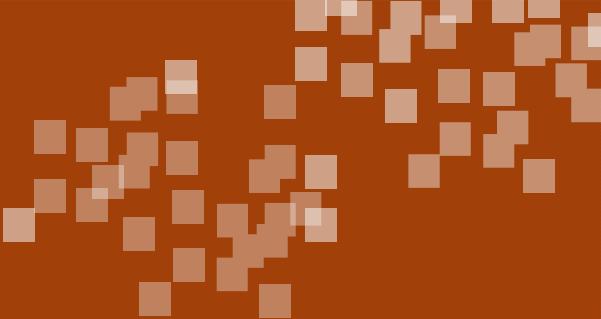
- ❖ Should update Xamarin.Forms Nuget package when starting a new project



Xamarin.Forms updates [Mac]

- ❖ Should update Xamarin.Forms Nuget package when starting a new project





Demonstration

Creating a Xamarin.Forms application



Xamarin.Forms app anatomy

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



Application

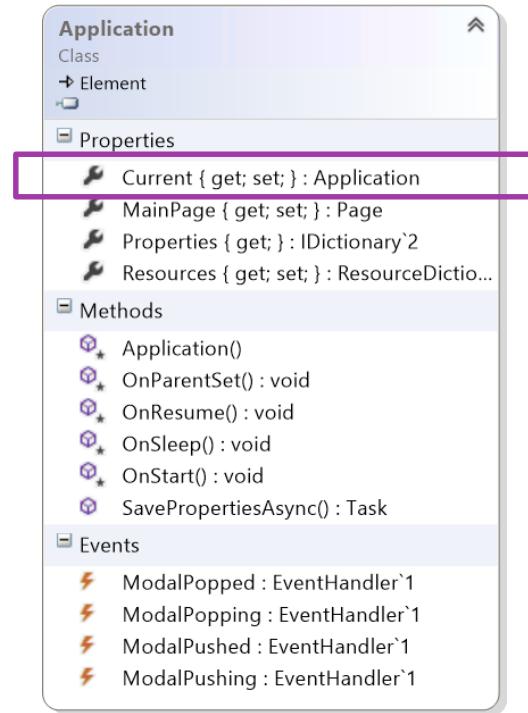
Provides initialization for
the application

Page(s
)

Represents a single
screen to display

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  
{ ...  
    protected override void OnStart() {}  
    protected override void OnSleep() {}  
    protected override void OnResume() {}  
}
```

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

Use **OnSleep** to save changes or persist information

Use **OnResume** to refresh your displayed data

Persisting information

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

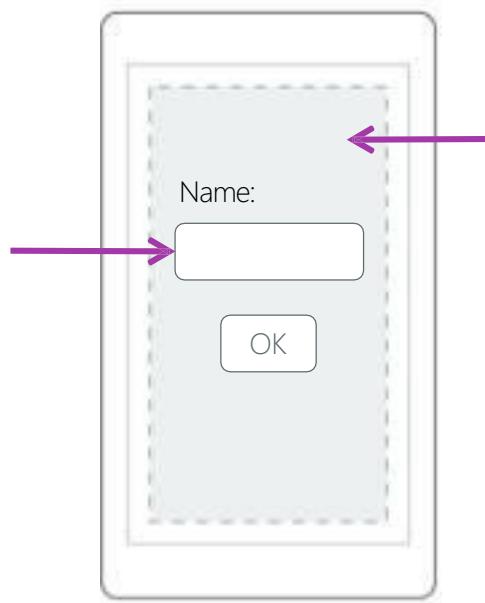
```
// Save off username in global property bag
Application.Current.Properties["username"] = username.Text;
```

```
// 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*

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

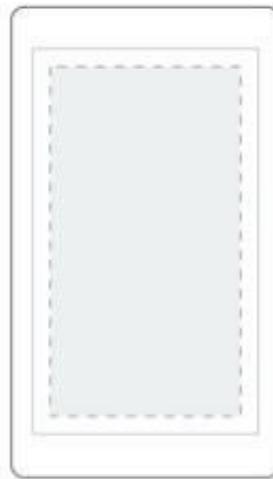


Content

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

Pages

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



Content

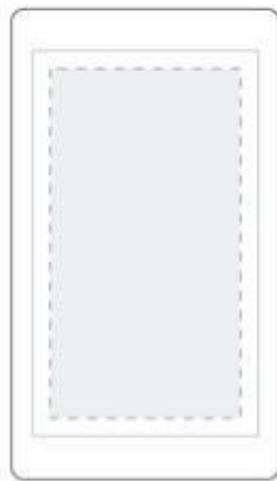


Master Detail

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

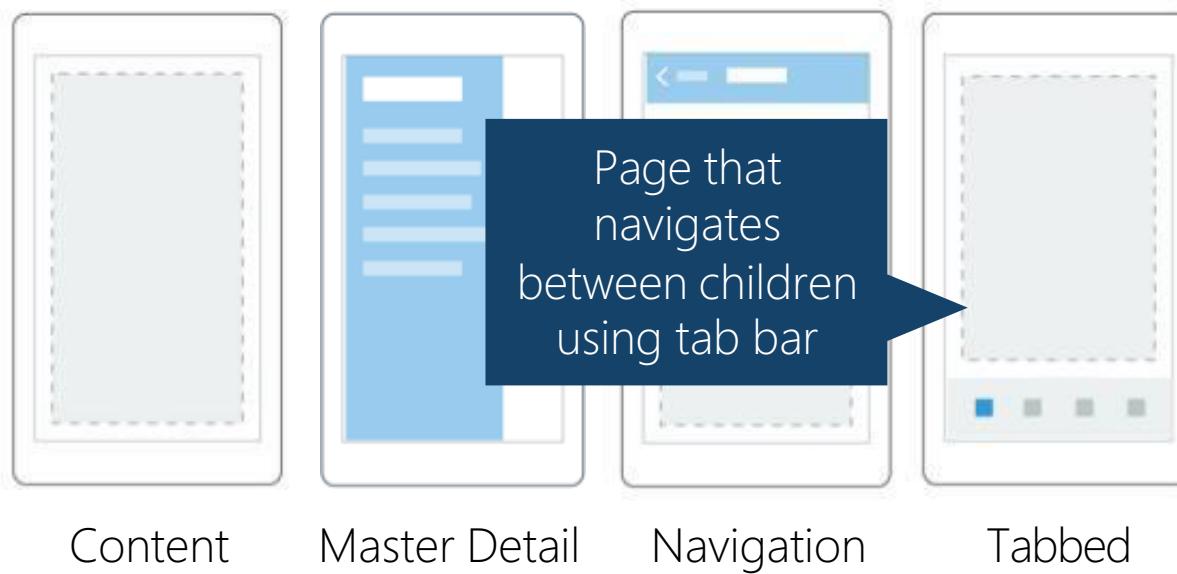


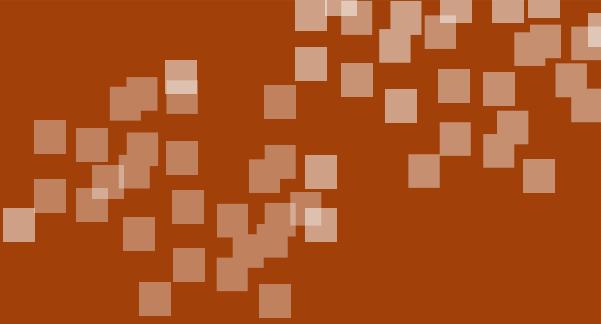
Navigation

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





Demonstration

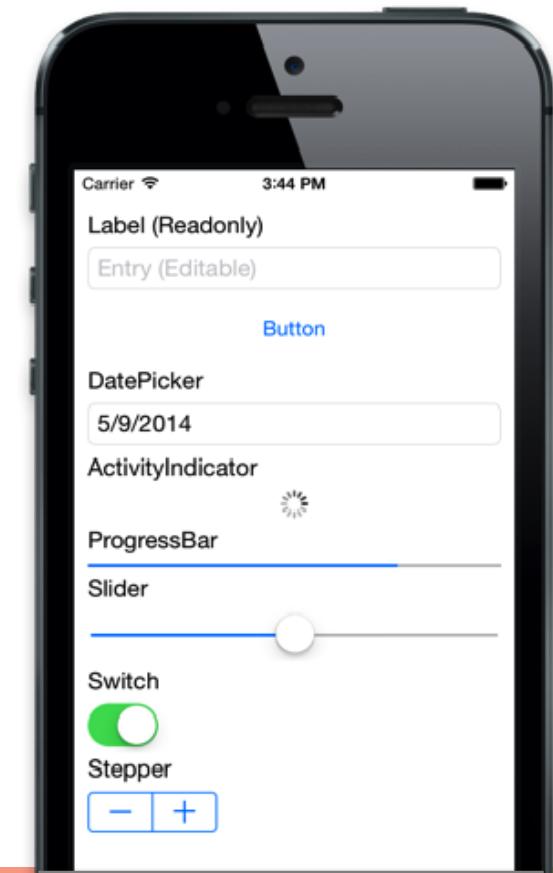
Adding a new ContentPage to a Xamarin.Forms application



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	CollectionView
BoxView	TimePicker	
Frame	Picker	



Views - Button

- ❖ **Button** provides a clickable surface with text



OK

```
var okButton = new Button() {  
    Text = "OK"  
};  
okButton.Clicked += OnClick;
```

```
void OnClick(object sender, EventArgs e) {  
    ...  
}
```

Views - Label

- ❖ Use a **Label** to display read-only text blocks

Hello, Forms!

```
var hello = new Label() {  
    Text = "Hello, Forms!",  
    HorizontalTextAlignment = TextAlignment.Center,  
    TextColor = Color.Blue,  
    FontFamily = "Arial"  
};
```

Views - Entry

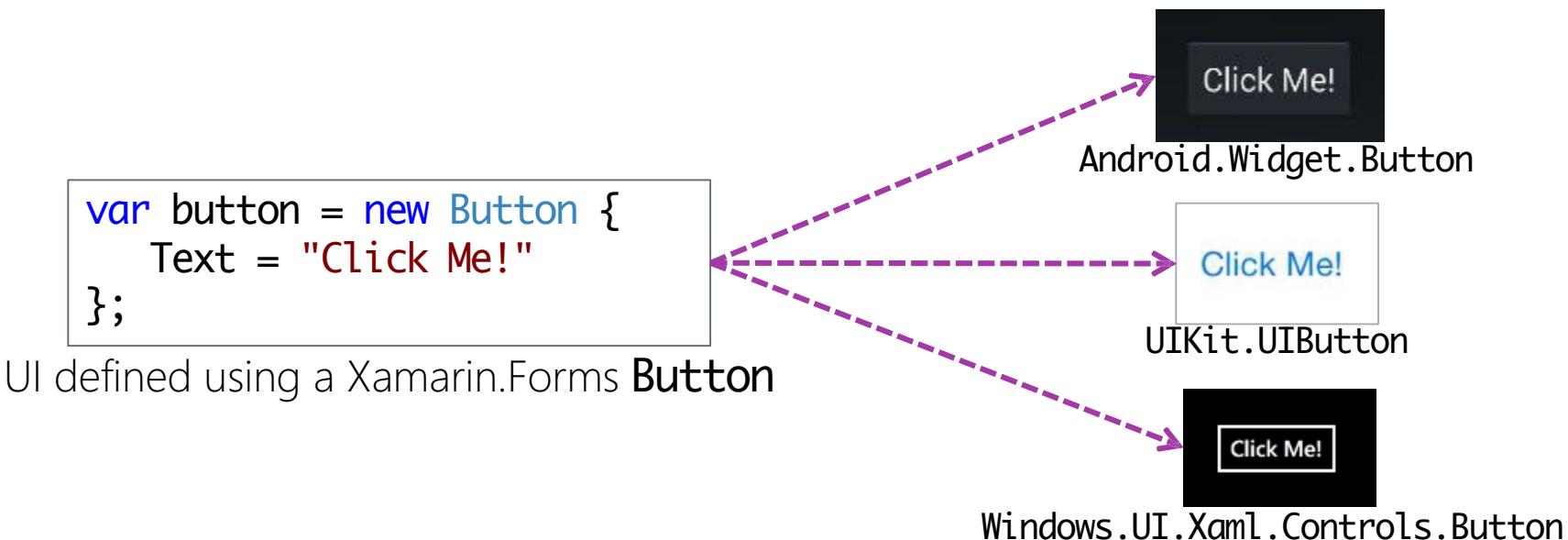
- ❖ Use an **Entry** control if you want the user to provide input with an on-screen or hardware keyboard



```
var edit = new Entry() {  
    Keyboard = Keyboard.Text,  
    PlaceholderText = "Enter Text"  
};
```

Rendering views

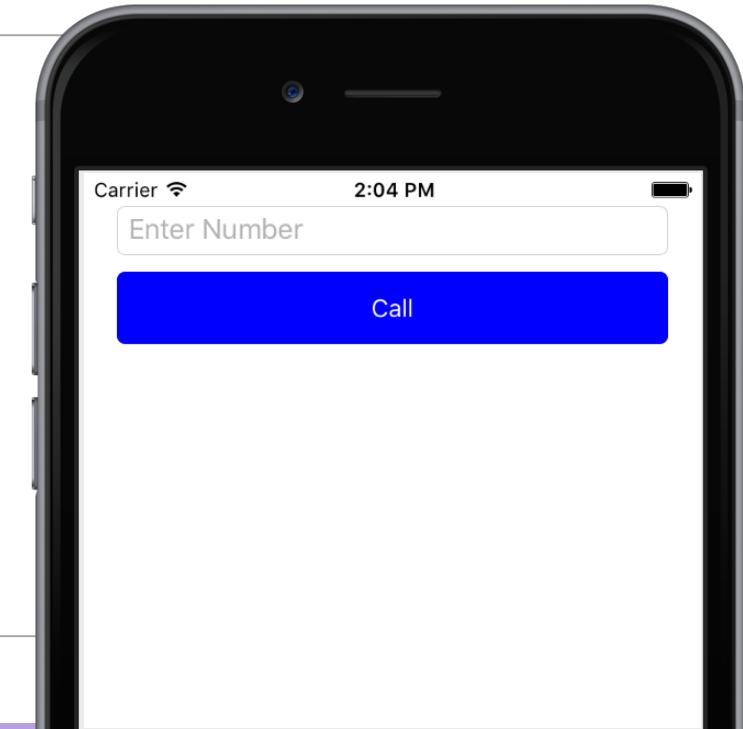
- ❖ Platform defines a *renderer* for each view that turns each view into the appropriate platform-specific control



Visual adjustments

- ❖ Views utilize properties to adjust visual appearance and behavior

```
var numEntry = new Entry {  
    Placeholder = "Enter Number",  
    Keyboard = Keyboard.Numeric  
};  
  
var callButton = new Button {  
    Text = "Call",  
    BackgroundColor = Color.Blue,  
    TextColor = Color.White  
};
```



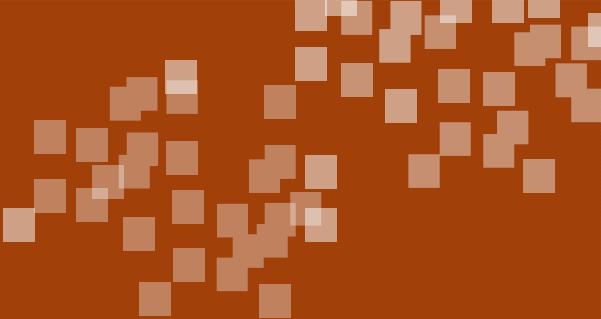
Providing Behavior

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

```
var numEntry = new Entry { ... };
numEntry.TextChanged += OnTextChanged;
...
void OnTextChanged (object sender, string newValue)
{
    ...
}
```



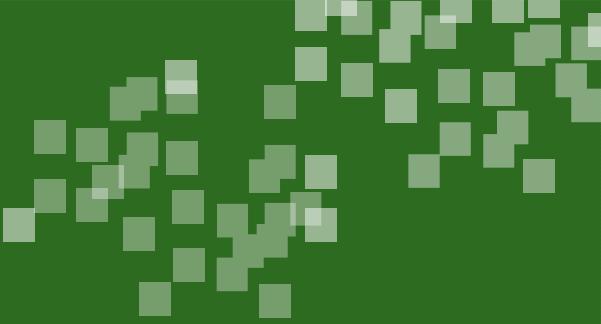
You can use traditional delegates, anonymous methods, or lambdas to handle events



Group Exercise

Creating our first Xamarin.Forms application





Flash Quiz

Flash Quiz

- ① Xamarin.Forms creates a single binary that can be deployed to Android, iOS or Windows
 - a) True
 - b) False

Flash Quiz

- ① Xamarin.Forms creates a single binary that can be deployed to Android, iOS or Windows
 - a) True
 - b) False

Flash Quiz

- ② You must call _____ before using Xamarin.Forms
- a) Forms.Initialize
 - b) Forms.Init
 - c) Forms.Setup
 - d) No setup call necessary

Flash Quiz

- ② You must call _____ before using Xamarin.Forms
- a) Forms.Initialize
 - b) Forms.Init
 - c) Forms.Setup
 - d) No setup call necessary

Flash Quiz

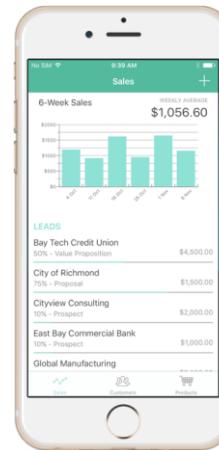
- ③ To supply the initial page for the application, you must set the _____ property.
- a) Application.FirstPage
 - b) Application.PrimaryPage
 - c) Application.MainPage
 - d) Application.MainView

Flash Quiz

- ③ To supply the initial page for the application, you must set the _____ property.
- a) Application.FirstPage
 - b) Application.PrimaryPage
 - c) Application.MainPage
 - d) Application.MainView

Summary

- ❖ Compare traditional development to Xamarin.Forms
- ❖ Understand Xamarin.Forms project structure
- ❖ Use application components
- ❖ Create a Xamarin.Forms app

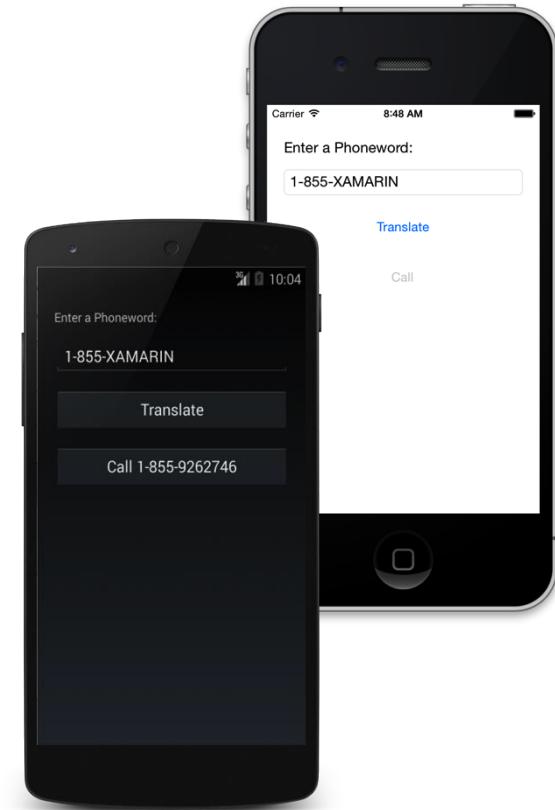




Arrange the UI using Layouts

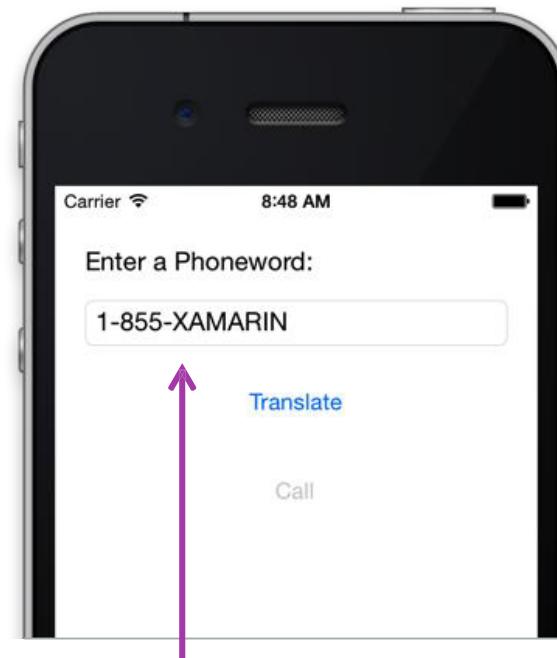
Tasks

- ❖ Choose a layout container to structure your UI
- ❖ Add views to a layout container



Organizing content

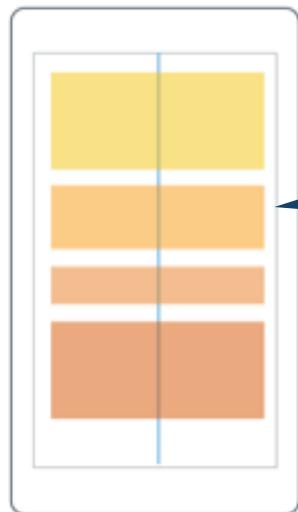
- ❖ Rather than specifying positions with coordinates (pixels, dips, etc.), you use layout containers to control how views are positioned relative to each other
- ❖ This provides for a more *adaptive* layout which is not as sensitive to dimensions and resolutions



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

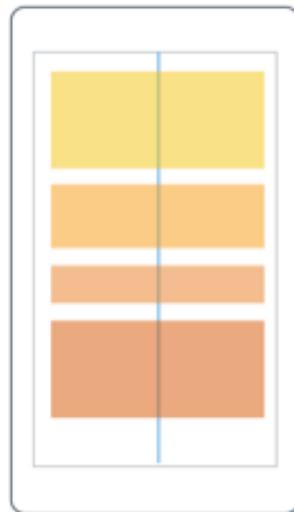


StackLayout

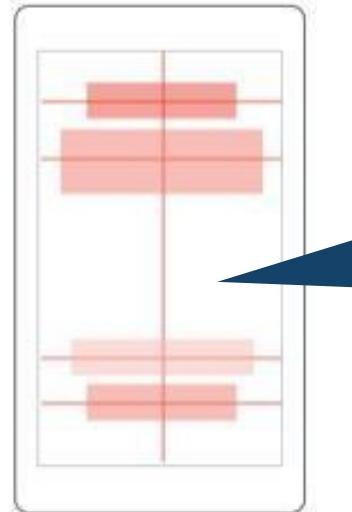
StackLayout places children top-to-bottom (default) or left-to-right based on **Orientation** property setting

Layout containers

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



StackLayout

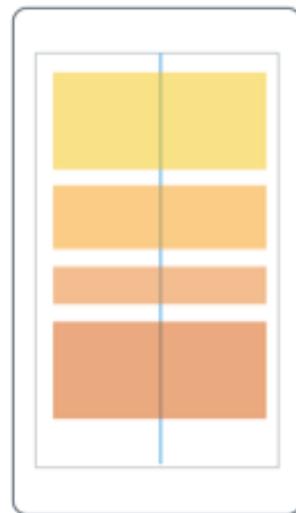


AbsoluteLayout

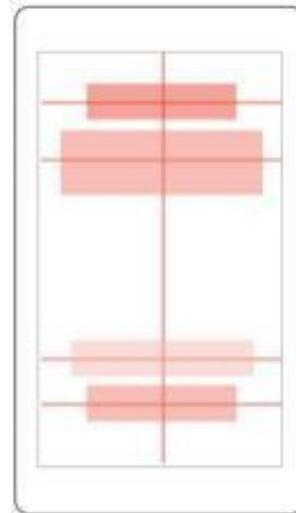
AbsoluteLayout places children
in absolute requested positions
based on anchors and bounds

Layout containers

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



StackLayout



Absolute Layout

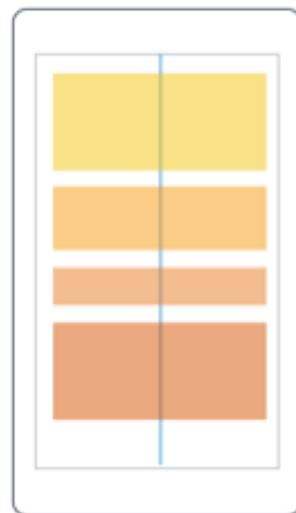


RelativeLayout

RelativeLayout
uses constraints to
position the children

Layout containers

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



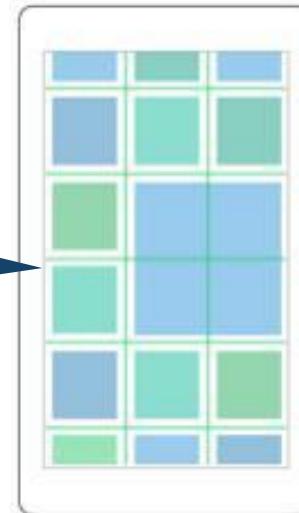
StackLayout



Absolute Layout



Relative Layout

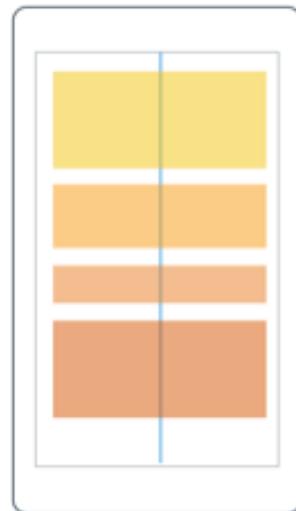


Grid

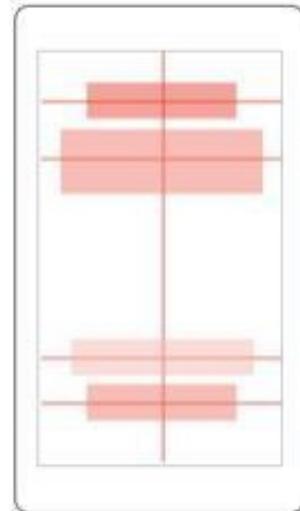
Grid places
children in
defined rows and
columns

Layout containers

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



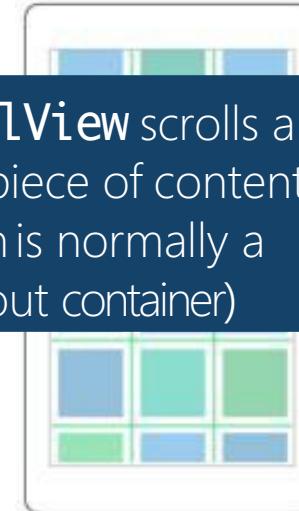
StackLayout



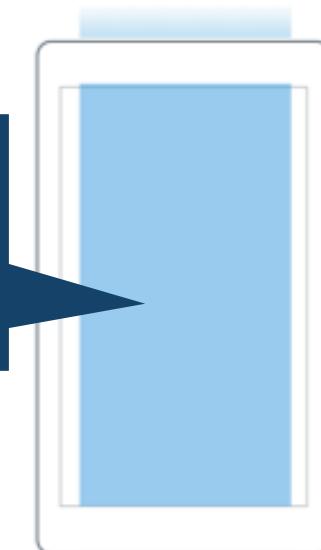
Absolute Layout



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);

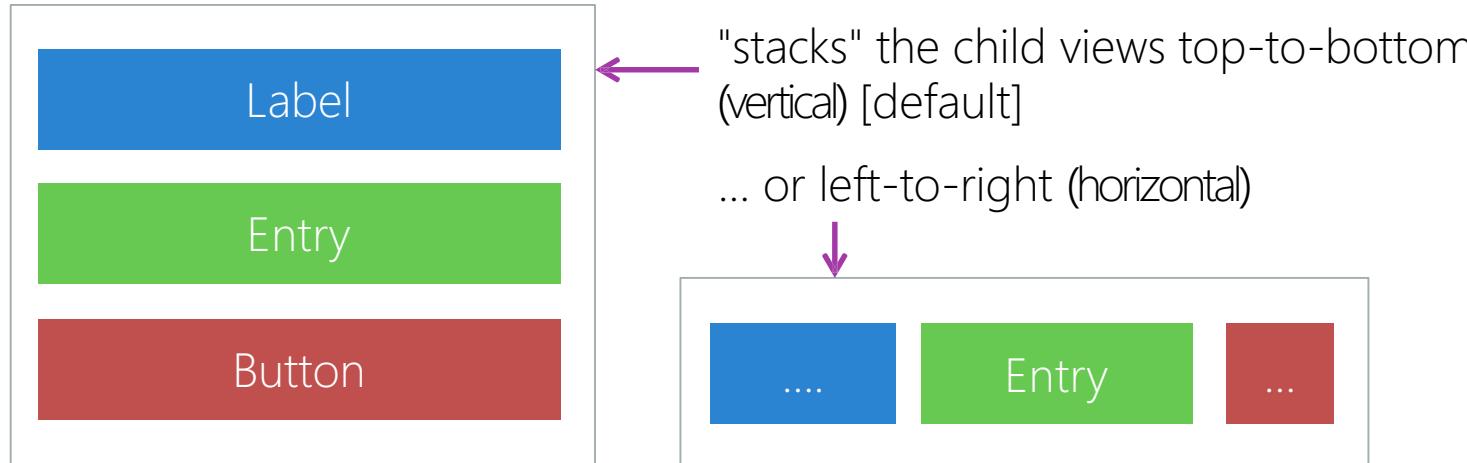
this.Content = layout;
```



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



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

Working with StackLayout

- ❖ **StackLayout** is used to create typical form style layout, **Orientation** property decides the direction that children are stacked

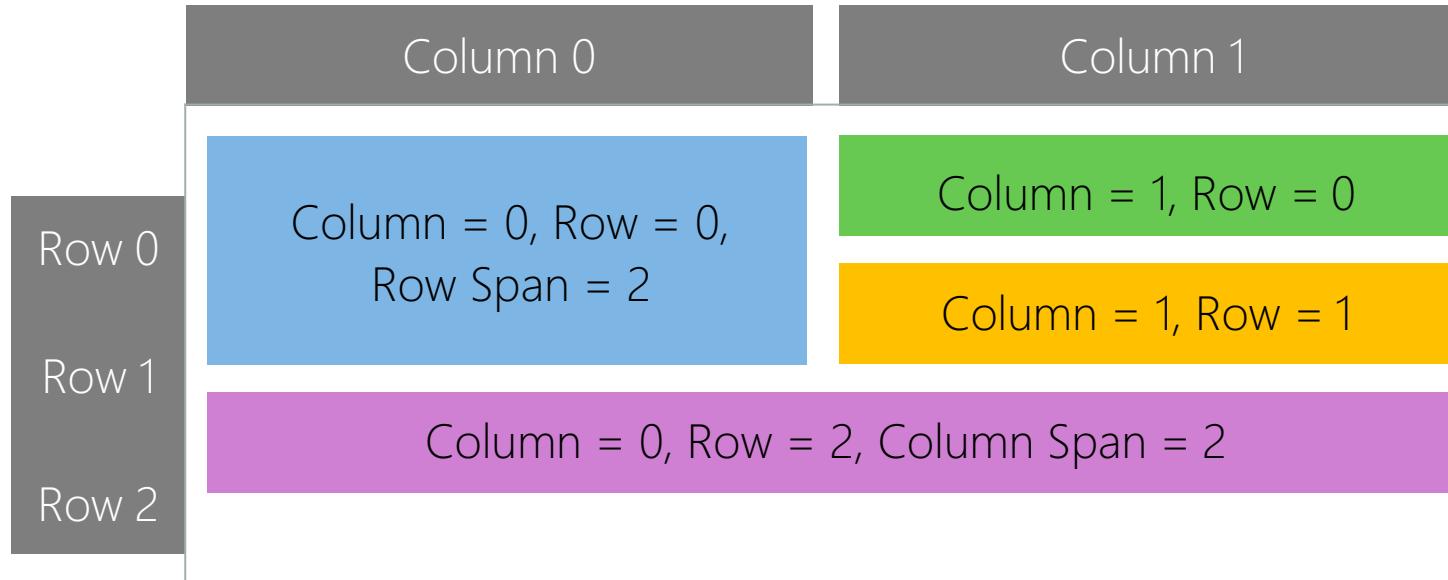
```
var layout = new StackLayout {
    Orientation = StackOrientation.Vertical
};

layout.Children.Add(new Label { Text = "Enter your name:" });
layout.Children.Add(new Entry());
layout.Children.Add(new Button { Text = "OK" });
```



Working with Grid

- ❖ **Grid** is a layout panel used to create rows and columns of views, children identify specific column, row and span



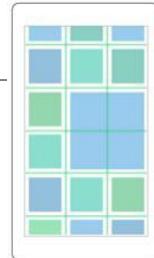
Adding items to a Grid

- ❖ Children in **Grid** must specify the layout properties, or they will default to the first column/row

```
Label label = new Label { Text = "Enter Your Name" };
```

```
Grid layout = new Grid();  
layout.Children.Add(label);
```

```
Grid.SetColumn(label, 1);  
Grid.SetRow(label, 1);  
Grid.SetColumnSpan(label, 2);  
Grid.SetRowSpan(label, 1);
```



Use static methods defined on **Grid** to set layout properties

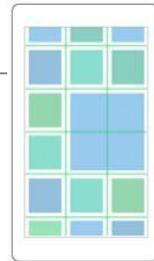
Adding items to a Grid

- ❖ Children in **Grid** must specify the layout properties, or they will default to the first column/row

```
Grid layout = new Grid();
```

```
...
```

```
layout.Children.Add(label, 0, 1);           // Left=0 and Top=1  
layout.Children.Add(button, 0, 2, 2, 3);    // L=0, R=2, T=2, B=3
```



Can also specify row/column as Left/Right/Top/Bottom values to **Add** method

Controlling the shape of the grid

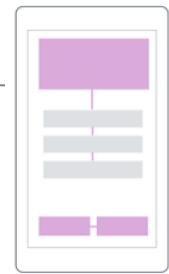
- ❖ Can influence the determined shape and size of the columns and rows

```
Grid layout = new Grid();
layout.RowDefinitions.Add(new RowDefinition {
    Height = new GridLength(100, GridUnitType.Absolute) // 100px
});
layout.RowDefinitions.Add(new RowDefinition {
    Height = new GridLength(1, GridUnitType.Auto) // "Auto" size
});
layout.ColumnDefinitions.Add(new ColumnDefinition {
    Width = new GridLength(1, GridUnitType.Star) // "Star" size
});
```

Working with RelativeLayout

- ❖ **RelativeLayout** allows you to position child views relative to two other views, or to the panel itself using constraint-based rules

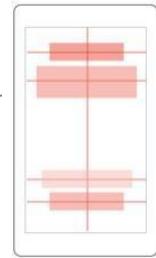
```
var layout = new RelativeLayout();  
...  
layout.Children.Add(label,  
    Constraint.RelativeToParent(  
        parent => (0.5 * parent.Width) - 25), // X  
    Constraint.RelativeToView(button,  
        (parent, sibling) => sibling.Y + 5), // Y  
    Constraint.Constant(50), // Width  
    Constraint.Constant(50)); // Height
```



Working with AbsoluteLayout

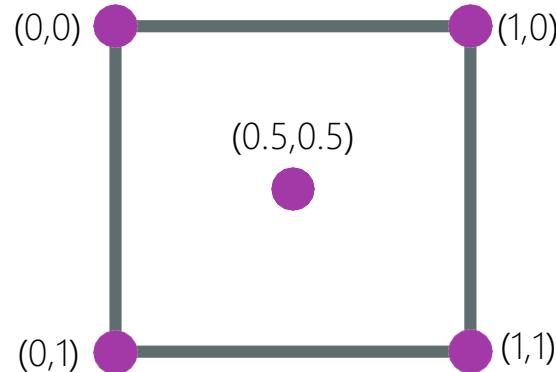
- ❖ **AbsoluteLayout** positions and sizes children by absolute values through either a coordinate (where the view determines its own size), or a bounding box

```
var layout = new AbsoluteLayout();  
...  
// Can do absolute positions by coordinate point  
layout.Children.Add(label1, new Point(100, 100));  
  
// Or use a specific bounding box  
layout.Children.Add(label2, new Rectangle(20, 20, 100, 25));
```



Working with AbsoluteLayout

- ❖ **AbsoluteLayout** can also position and size children proportional to its own size using coordinates based on a 1x1 unit square which represents a percentage of the container's size



Working with AbsoluteLayout

- ❖ **AbsoluteLayout** can also position and size children proportional to its own size using coordinates based on a 1x1 unit square which represents a percentage of the container's size

```
var layout = new AbsoluteLayout();  
...  
// Center at the bottom of the container, take up ½ the space  
layout.Children.Add(bottomLabel, new Rectangle (.5, 1, .5, .1),  
    AbsoluteLayoutFlags.All );
```



Here we center the label (.5) at the bottom of the container (1) and take up ½ the space (.5) width and 1/10 the space height (.1)

Working with AbsoluteLayout

- ❖ **AbsoluteLayout** can also position and size children proportional to its own size using coordinates based on a 1x1 unit square which represents a percentage of the container's size

```
var layout = new AbsoluteLayout();  
...  
// Stretch image across entire container  
layout.Children.Add(fillImage, new Rectangle (0, 0, 1, 1),  
    AbsoluteLayoutFlags.All );
```



Here we "fill" the container with an image
[0,0] – [1,1]

Fine-tuning AbsoluteLayout

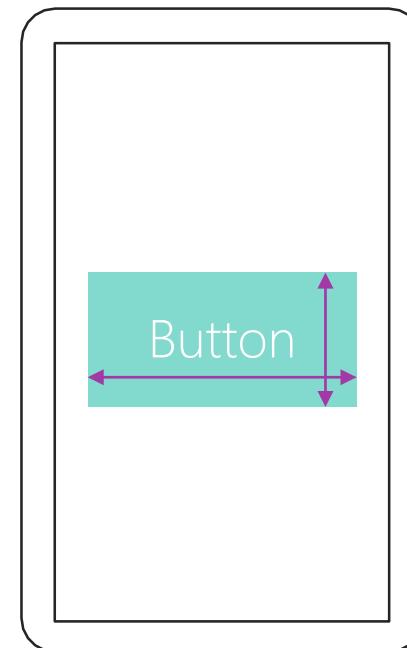
- ❖ Can use either **Add** method, or specific static methods to control the bounding box and layout flags for children in **AbsoluteLayout** – this allows for "runtime" adjustments

```
Label bottomLabel;  
  
void MoveLabelToTopRight(object sender, EventArgs e)  
{  
    AbsoluteLayout.SetLayoutBounds(bottomLabel,  
        new Rectangle(1, 0, .5, .1));  
    AbsoluteLayout.SetLayoutFlags(bottomLabel,  
        AbsoluteLayoutFlags.All);  
}
```

Element size

- ❖ Use **WidthRequest** and **HeightRequest** to ask the layout panel for a specific size for your views

```
var button = new Button();  
button.WidthRequest = 300;  
button.HeightRequest = 150;
```



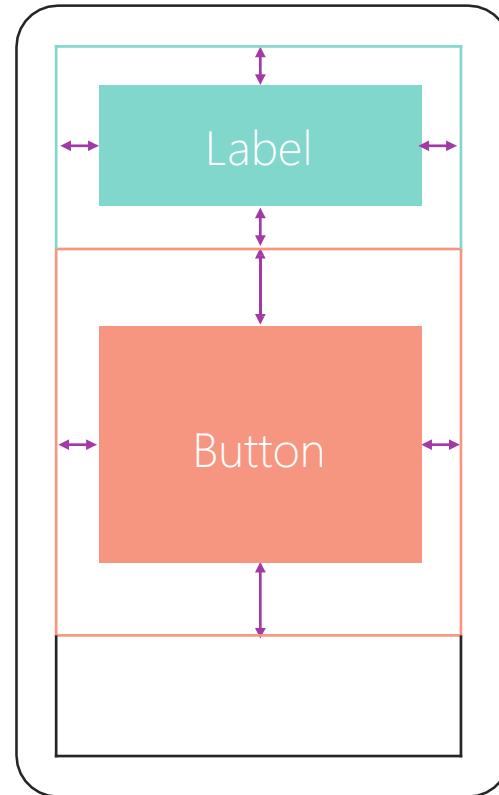
View Margin

- ❖ **Margin** adds distance from a view to adjacent views within a managed layout

```
Label label = ...  
Button button = ...
```

```
label.Margin = new Thickness(10);  
button.Margin = new Thickness(10,20);
```

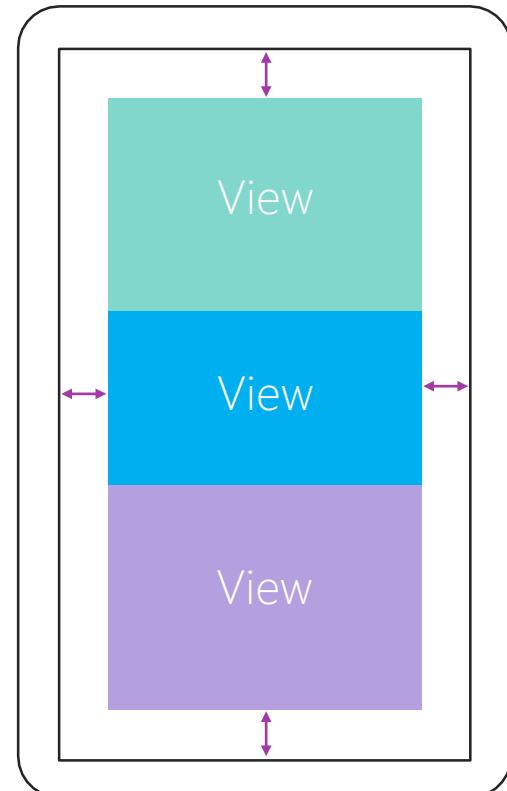
↑
Overloaded constructors give you several options, including the ability to set a separate value on each side



Layout Padding

- ❖ **Padding** adds distance between the inside edges of a layout container and its children (only available in layouts)

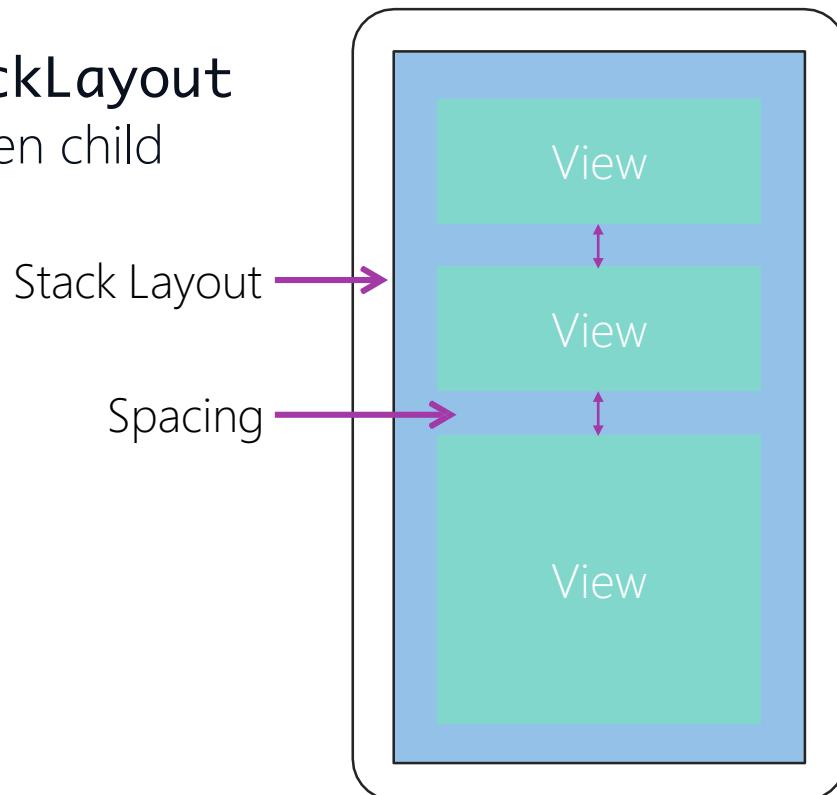
```
Grid grid = ...;  
grid.Padding = new Thickness(10);
```

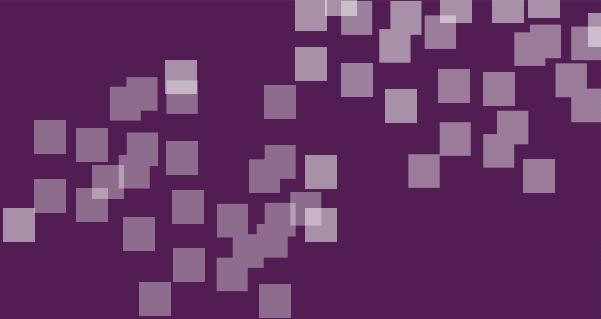


StackLayout Spacing

- ❖ The **Spacing** property of **StackLayout** and controls the distance between child elements

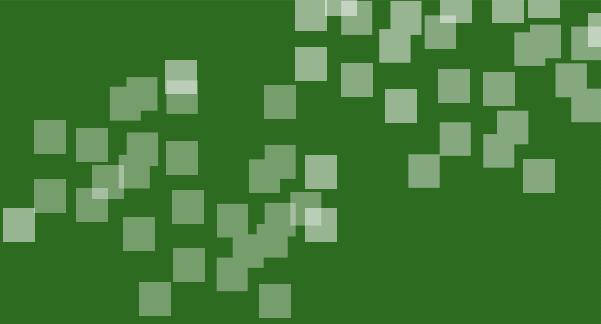
```
StackLayout panel = ...;  
panel.Spacing = 20;
```





Individual Exercise

Creating Xamarin.Forms Phoneword



Flash Quiz

Flash Quiz

- ① The direction (left-to-right or top-to-bottom) a **StackLayout** organizes content is controlled by which property?
- a) Style
 - b) Direction
 - c) Orientation
 - d) LayoutDirection

Flash Quiz

- ① The direction (left-to-right or top-to-bottom) a **StackLayout** organizes content is controlled by which property?
- a) Style
 - b) Direction
 - c) Orientation
 - d) LayoutDirection

Flash Quiz

- ② Which of these controls is not available in Xamarin.Forms?
- a) Button
 - b) DatePicker
 - c) ListBox
 - d) ListView

Flash Quiz

- ② Which of these controls is not available in Xamarin.Forms?
- a) Button
 - b) DatePicker
 - c) ListBox
 - d) ListView

Flash Quiz

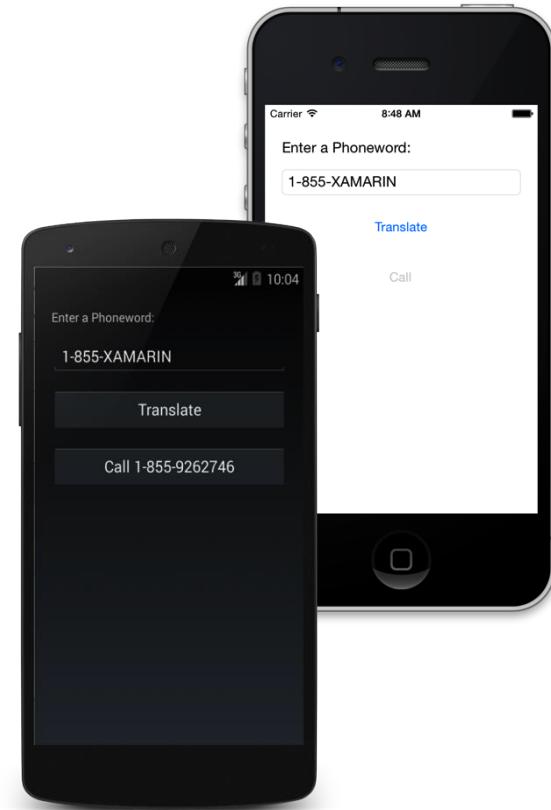
- ③ To adjust spacing between children when using the **StackLayout** container we can change the _____ property on the stack layout.
- a) Margin
 - b) Padding
 - c) Spacing

Flash Quiz

- ③ To adjust spacing between children when using the **StackLayout** container we can change the _____ property on the stack layout.
- a) Margin
 - b) Padding
 - c) Spacing

Summary

- ❖ Choose a layout container to structure your UI
- ❖ Add views to a layout container

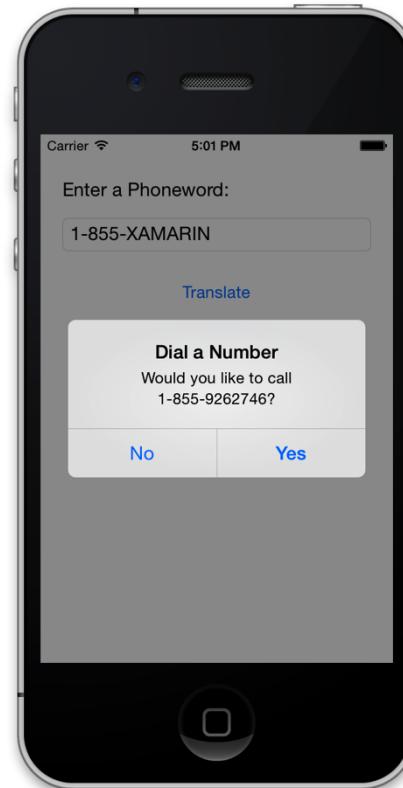




Use platform-specific features in shared code

Tasks

- ❖ Change the UI per-platform
- ❖ Use platform-specific features
- ❖ Use the **DependencyService**



Recall: Xamarin.Forms architecture

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



- *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.RuntimePlatform` allows your app to determine the executing platform

```
switch(Device.RuntimePlatform)
{
    case Device.iOS:
        ...
        break;
    case Device.Android:
        ...
        break;
    case Device.UWP:
        ...
        break;
    case Device.macOS:
        ...
        break;
}
```

Detecting the platform

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

```
if (Device.Idiom == TargetIdiom.Tablet) {  
    // Code for tablets only  
    if (Device.RuntimePlatform == Device.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 in your shared code

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



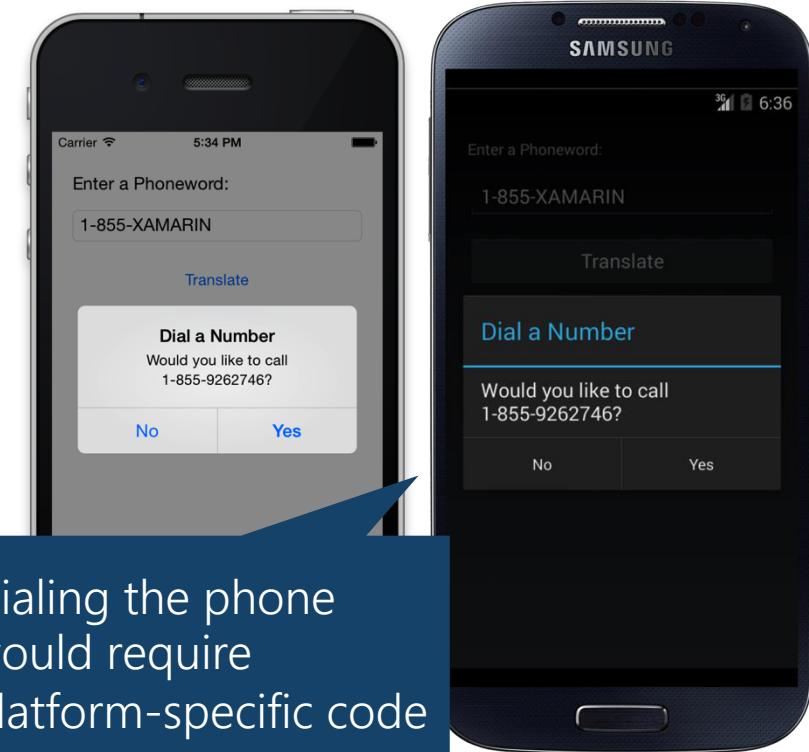
UI Thread
marshaling with
`Device.BeginInvokeOnMainThread`



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



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

PhoneDialerWin

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 (.NET Standard)

```
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

- Exposure 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

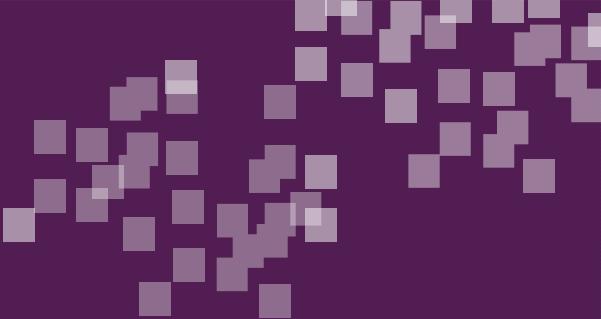
- ❖ Xamarin.Forms includes a *service locator* called **DependencyService** which can be used to register platform-specific implementations and then locate them through the abstraction in your shared code

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

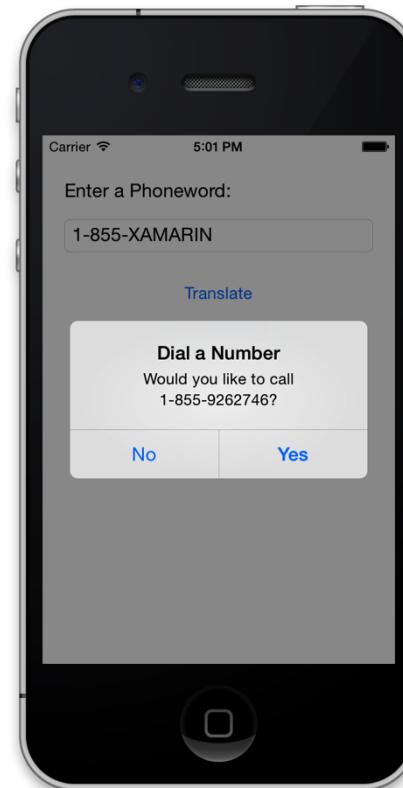


Individual Exercise

Adding support for dialing the phone

Summary

- ❖ Change the UI per-platform
- ❖ Use platform-specific features
- ❖ Use the **DependencyService**



Thank You!