XAM 120

Introduction to Xamarin.Forms

- Lecture will begin shortly
- Download class materials from university.xamarin.com

Xamarin University

Xamarin University

Objectives

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

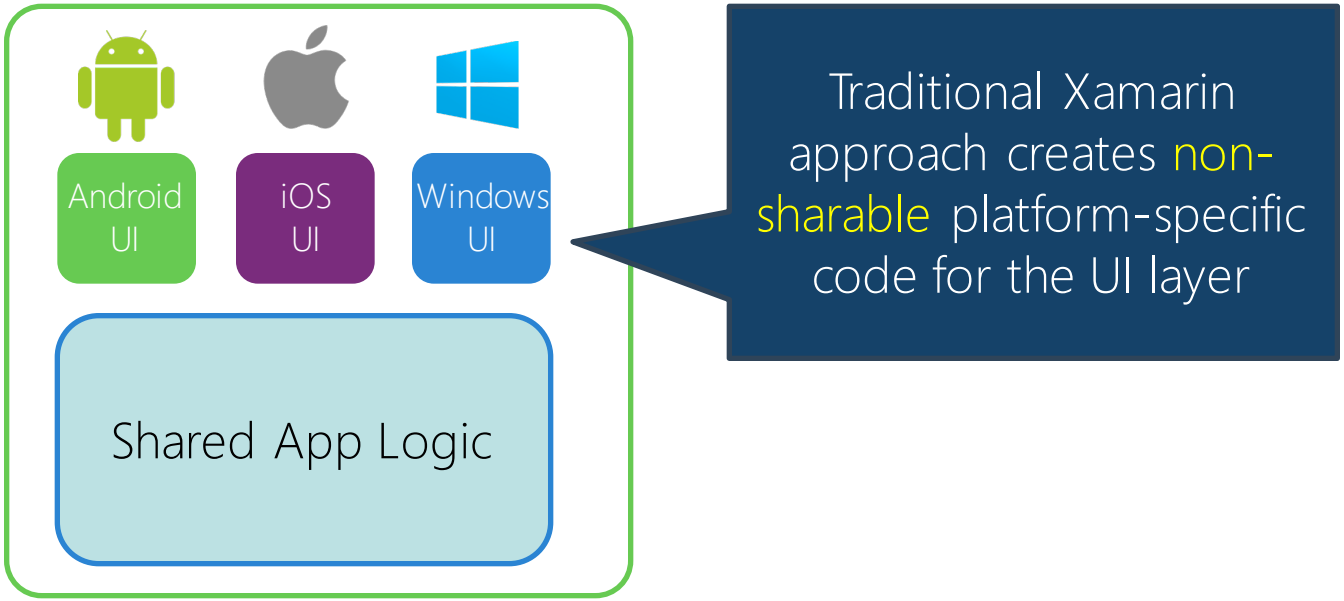
What is Xamarin.Forms?

Xamarin University

Tasks

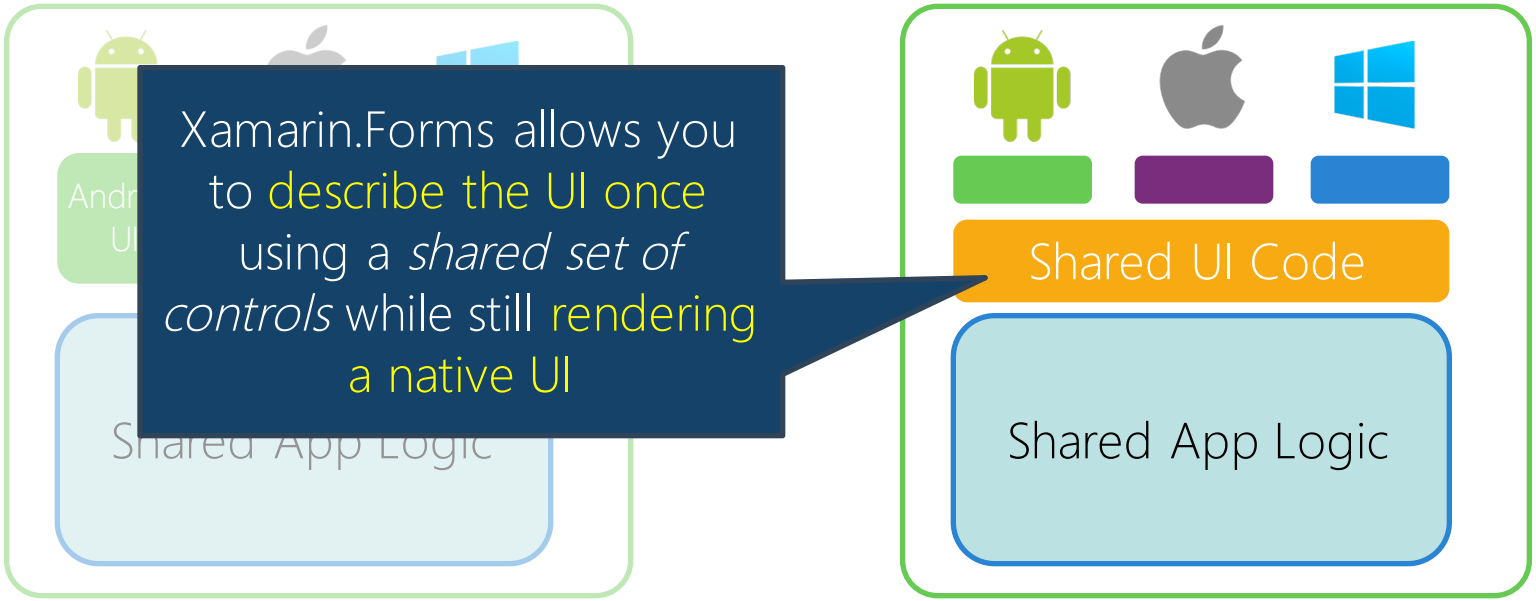
- Traditional vs. Xamarin.Forms
- Xamarin.Forms project structure
- Application Components
- "Hello, Forms!"

Reminder: Traditional Xamarin approach



Xamarin University

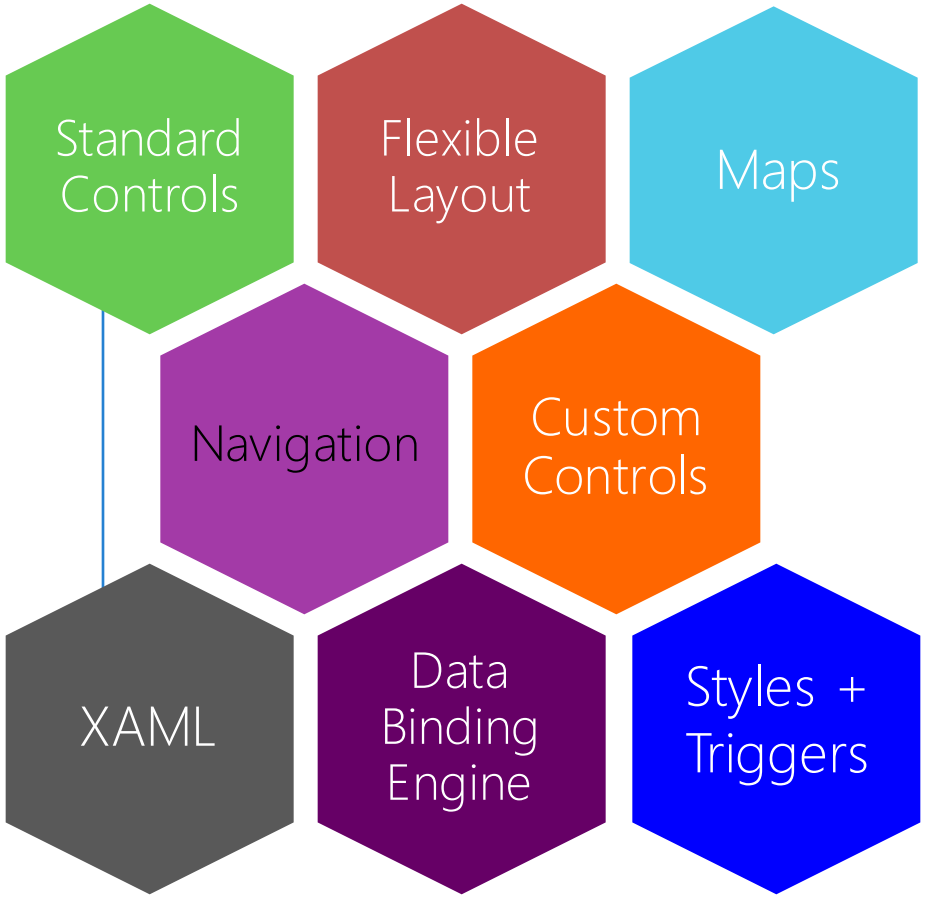
Traditional vs. Xamarin.Forms



Xamarin University

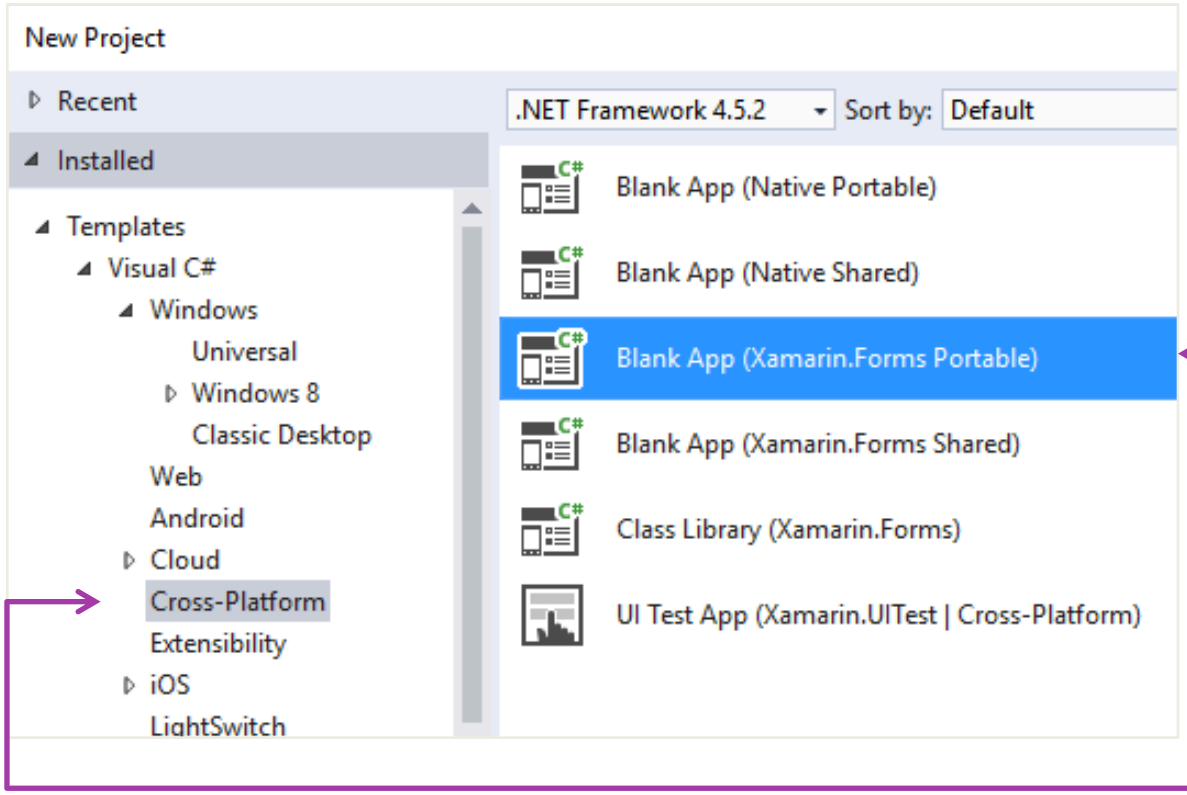
What is Xamarin.Forms?

- Xamarin.Forms is a cross-platform UI framework to create mobile apps for:
  - Android 4.0+
  - iOS 6.1+
  - Windows Phone 8.x (SL)
  - Windows Phone 8.1 (RT)
  - Windows 10 (UWP)



Xamarin University

Creating a Xamarin.Forms App [VS]



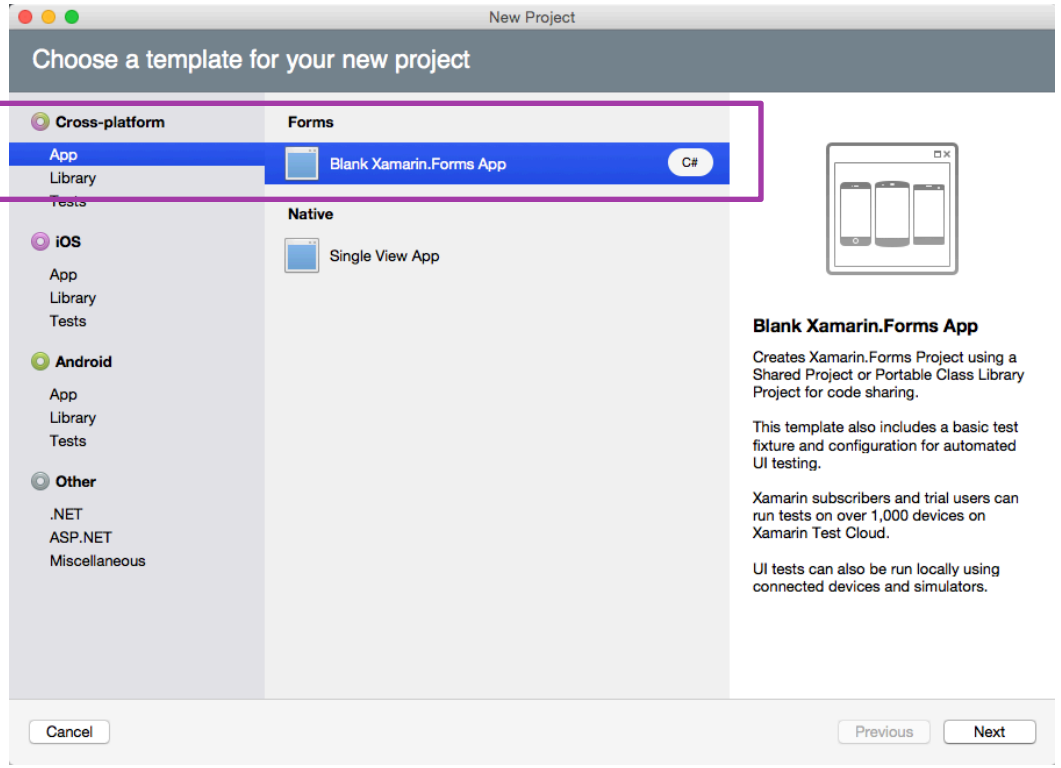
Built-in project templates for Xamarin.Forms applications available under **Cross-Platform**

Xamarin University

Creating a Xamarin.Forms App [XS]

- ❖ Project wizard walks through the available options

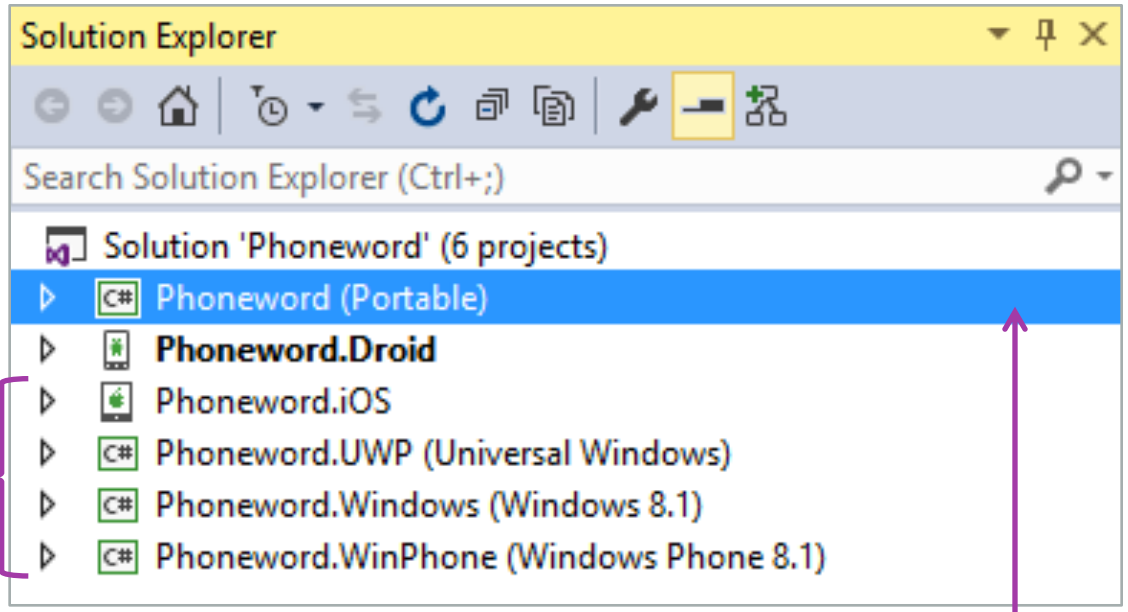
- ✓ Mac supports Android + iOS
- ✓ Windows supports Android



Xamarin University

Project Structure

- Blank App project template creates several related projects



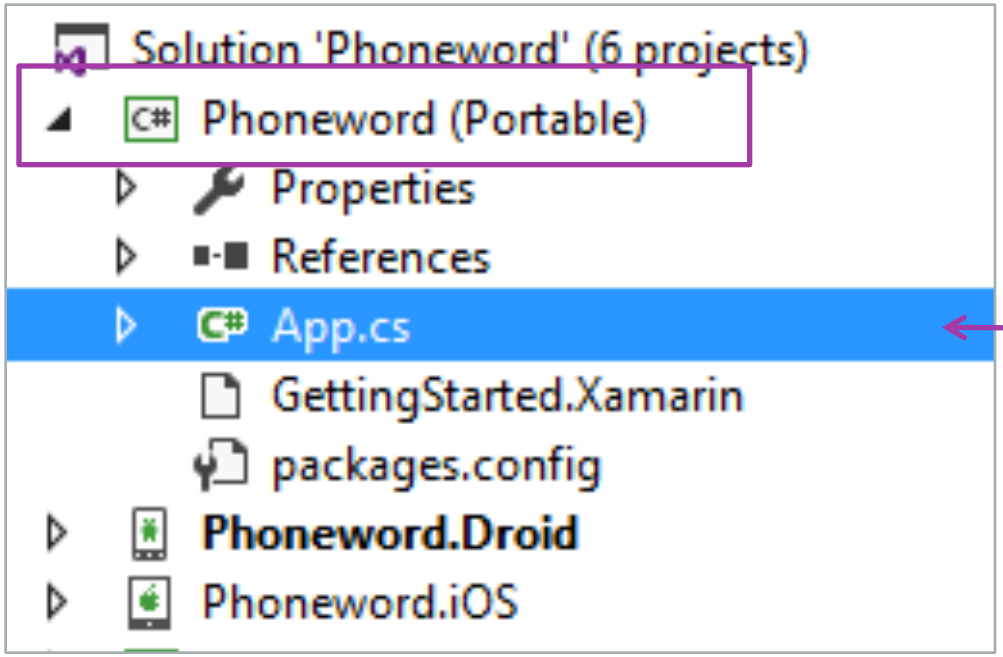
Platform-specific projects act as "host" to create native application

PCL or SAP used to hold shared code that defines UI and logic

Xamarin University

Project Structure - PCL

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

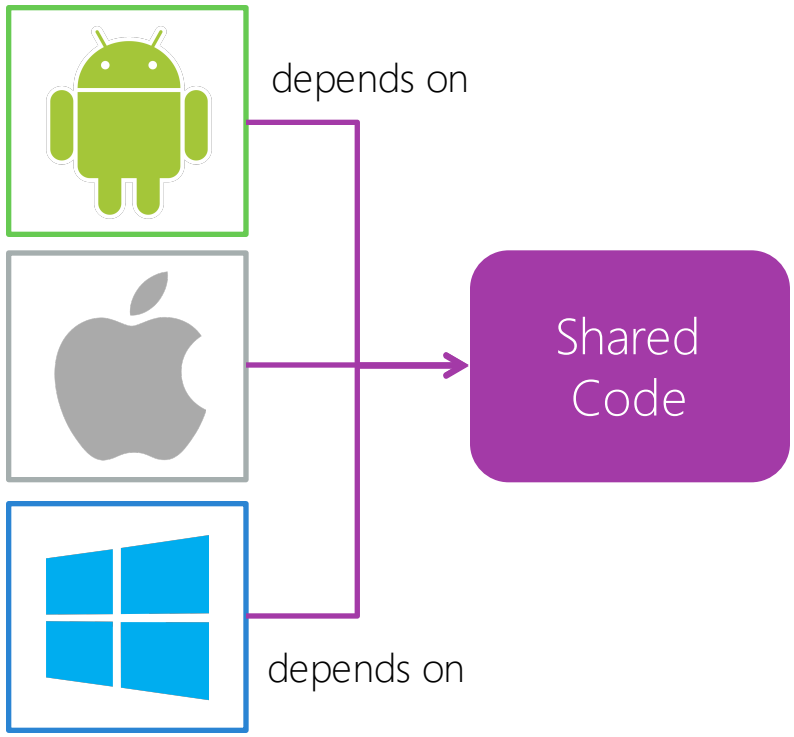


Default template creates a single **`App.cs`** file which decides the initial screen for the application

Xamarin University

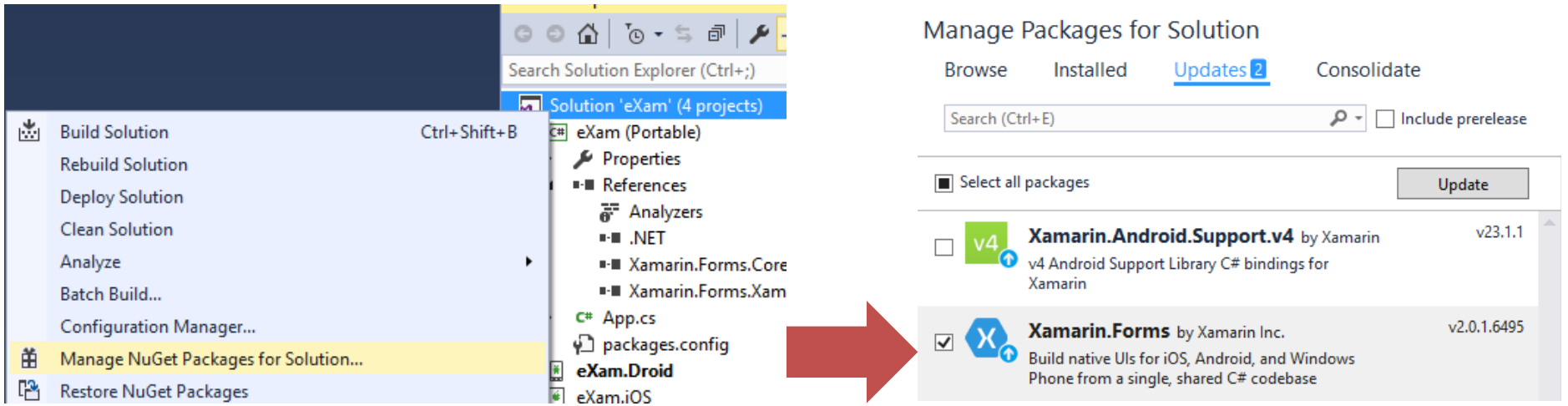
Project Structure - Dependencies

- 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 updates

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

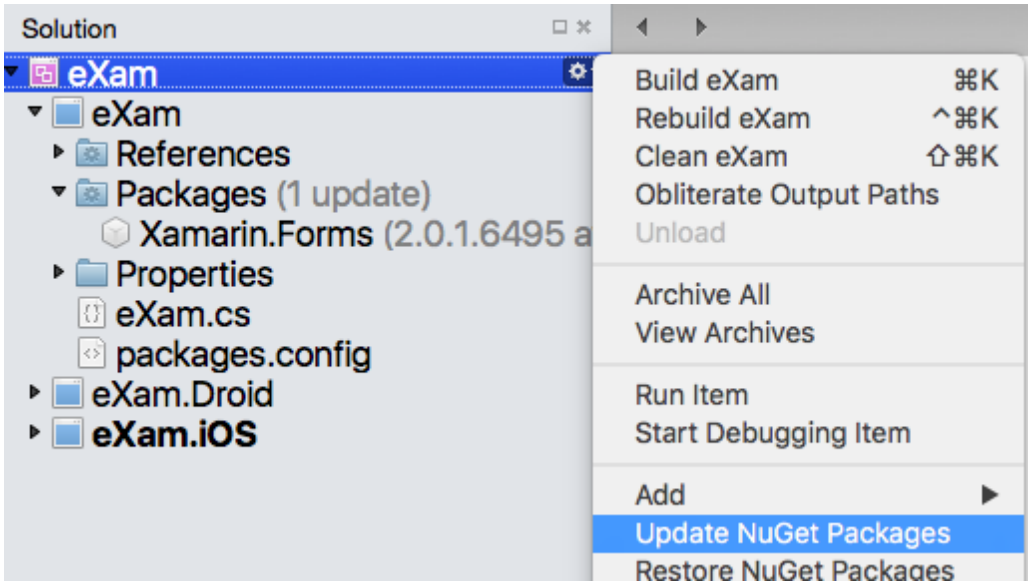


Visual Studio 2015

Xamarin University

Xamarin.Forms updates

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



Xamarin Studio

Demonstration

Creating a Xamarin.Forms application

Xamarin University

Xamarin University

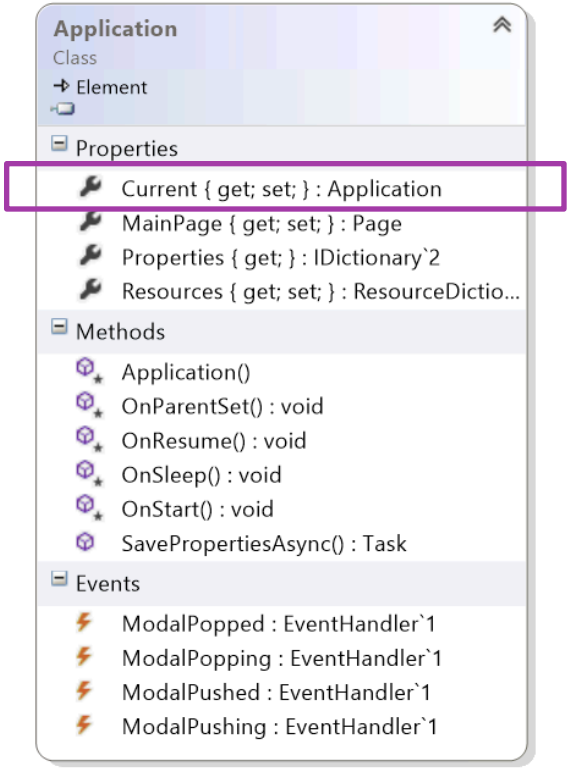
Xamarin.Forms app anatomy

- 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 University

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 University

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 University

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

Xamarin University

Persisting information

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

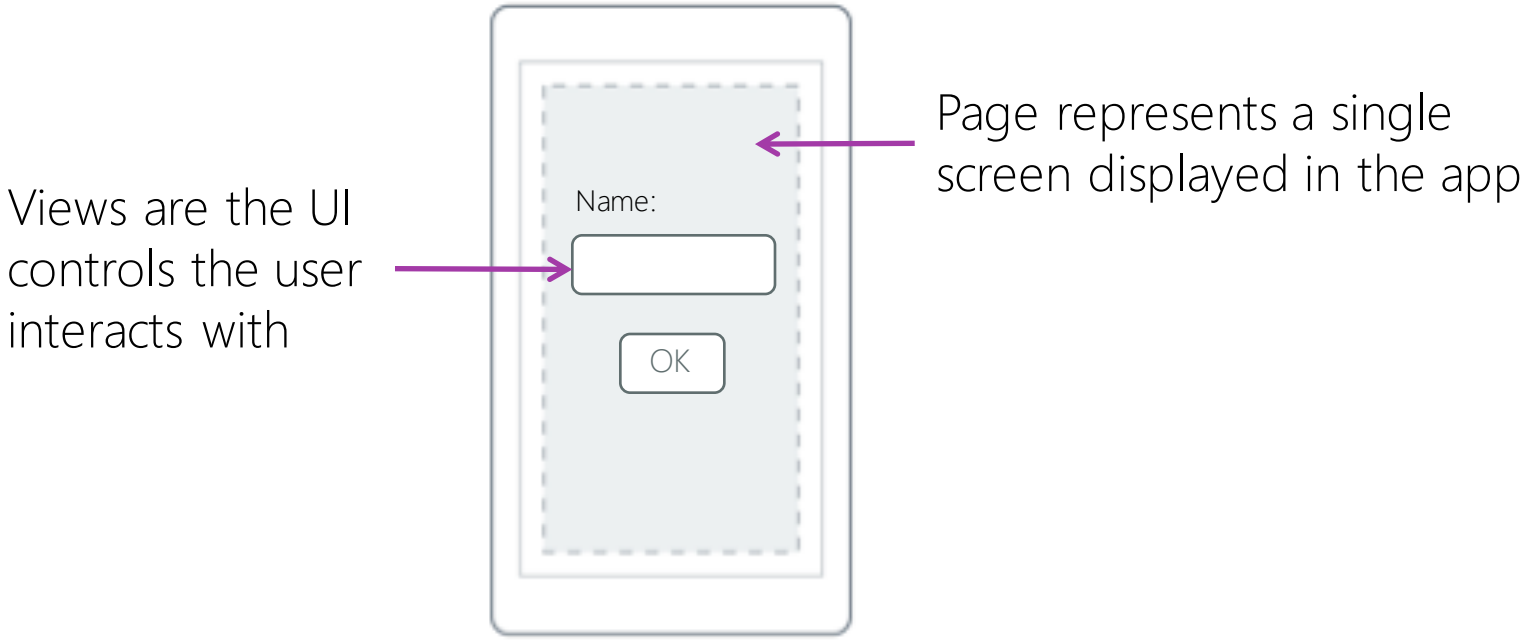
```
// Save off username in global property bag
Application.Current.Properties["username"] = username.Text;
```

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

Xamarin University

Creating the application UI

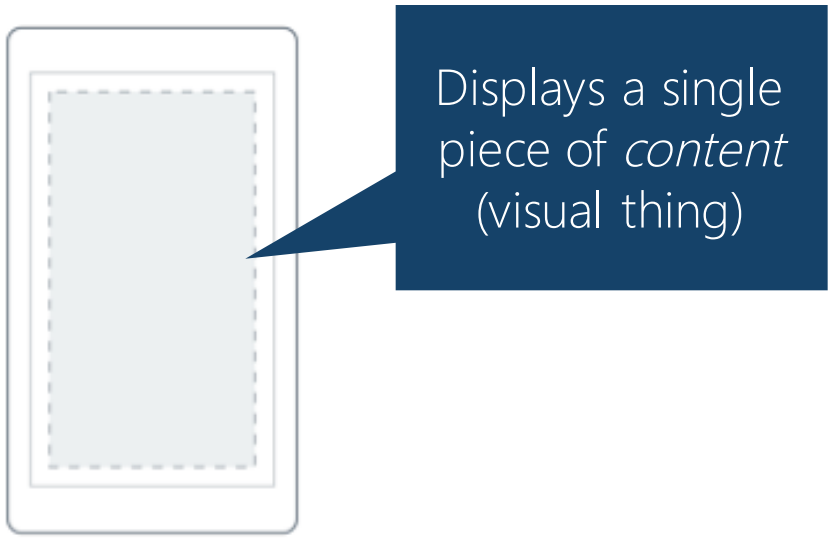
- Application UI is defined in terms of *pages* and *views*



Xamarin University

Pages

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



Xamarin University

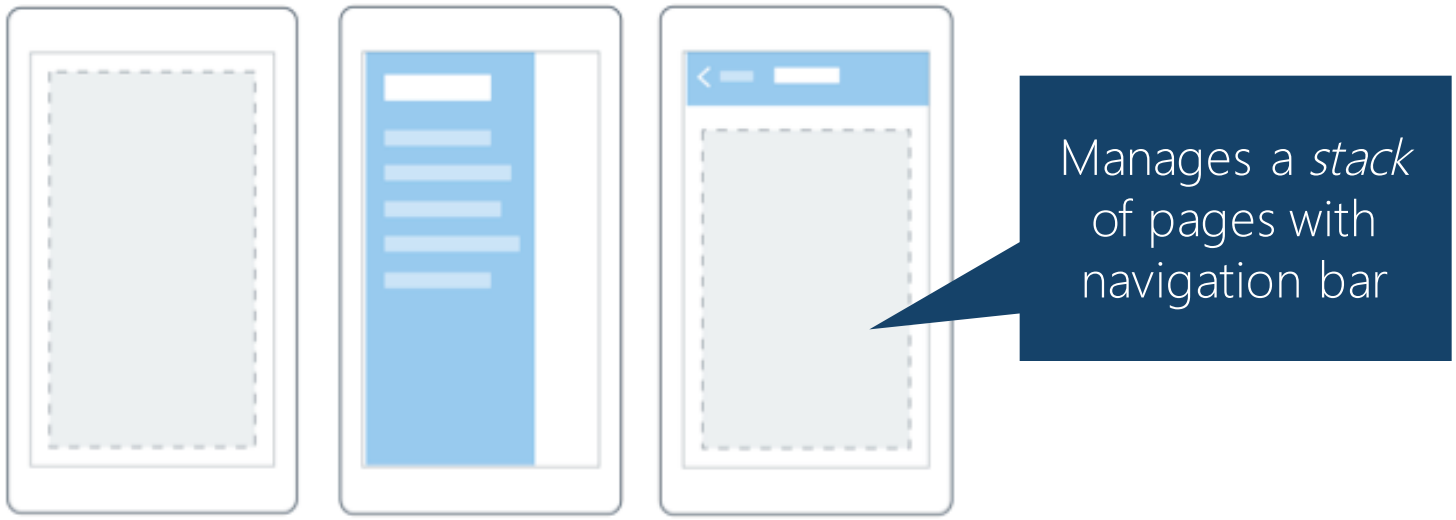
Pages

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

Xamarin University

Pages

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



Xamarin University

Pages

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

Demonstration

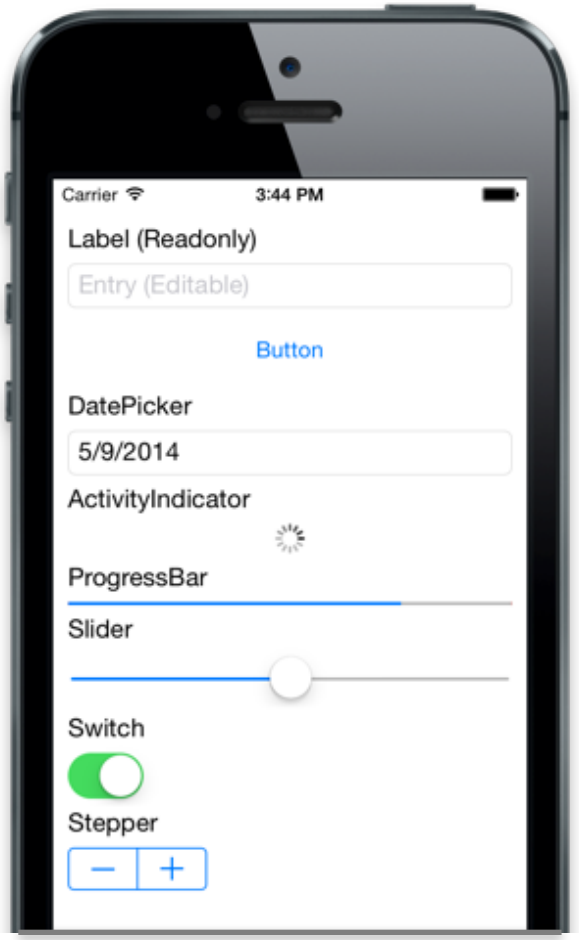
Adding a new ContentPage to a Xamarin.Forms application

Xamarin University

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	



Xamarin University

Views - Button

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


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

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

Xamarin University

Views - Label

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

Hello, Forms!

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

Xamarin University

Views - Entry

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


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

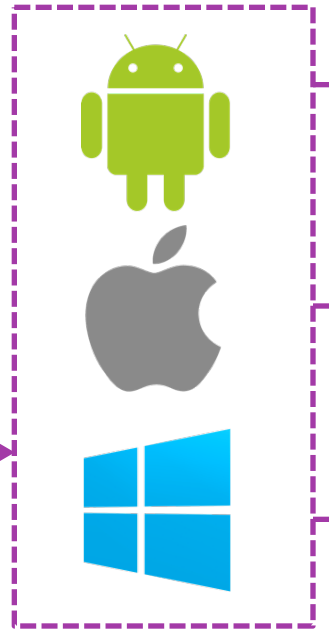
Rendering views

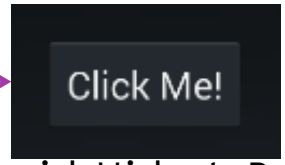
- Platform defines a *renderer* for each view that creates a native representation of the UI

UI uses a Xamarin.Forms `Button`

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

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





Android.Widget.Button

UIButton

System.Windows.Button

Xamarin University

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
};
```

Xamarin University

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

Xamarin
University

Flash Quiz

Xamarin University

Flash Quiz

① Xamarin.Forms creates a single binary that can be deployed to Android, iOS or Windows Phone

a) True

b) False

Xamarin University

Flash Quiz

① Xamarin.Forms creates a single binary that can be deployed to Android, iOS or Windows Phone

a) True

b) False

Xamarin University

Flash Quiz

② You must call _____ before using Xamarin.Forms

a) Forms.Initialize
b) Forms.Init
c) Forms.Setup
d) No setup call necessary.

Xamarin University

Flash Quiz

② You must call _____ before using Xamarin.Forms

a) Forms.Initialize

b) Forms.Init

c) Forms.Setup

d) No setup call necessary.

Xamarin University

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

Xamarin University

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

Xamarin University

Summary

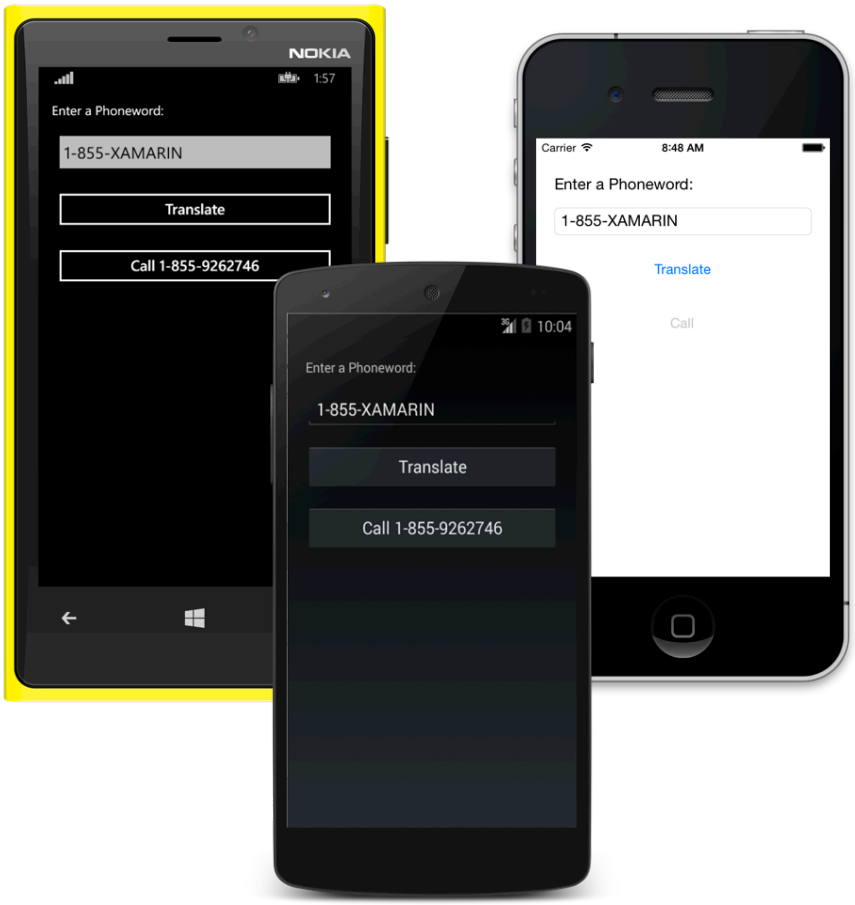
- Xamarin.Forms project structure
- Application Components
- "Hello, Forms!"

Pages, Controls, and Layout

Xamarin University

Tasks

- Layout containers
- Adding views
- Fine-tuning layout

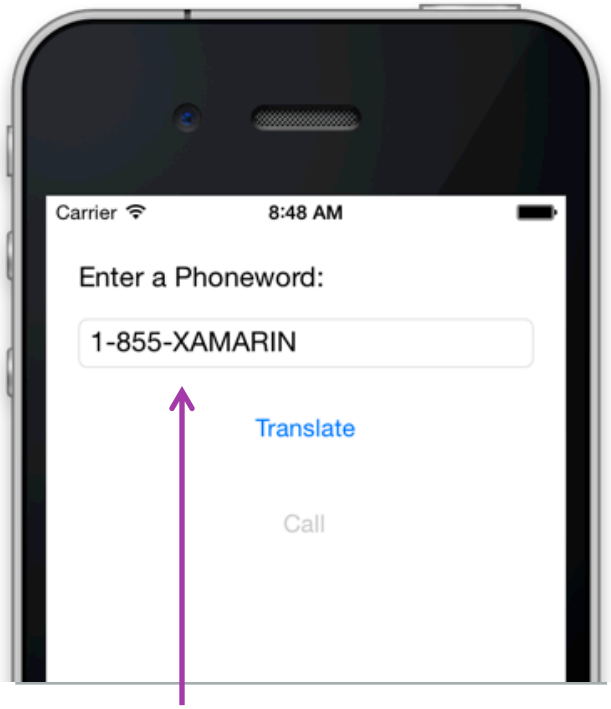


Xamarin University

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 a*daptive* layout which is not as sensitive to dimensions and resolutions

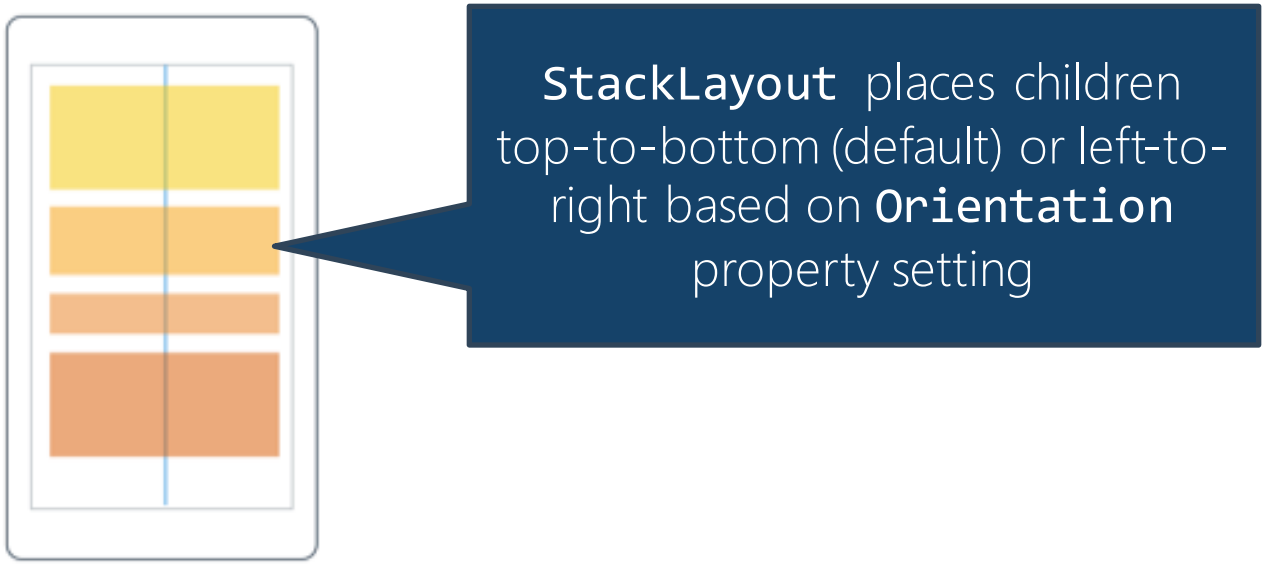


For example, "stacking" views on top of each other with some spacing between them

Xamarin University

Layout containers

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

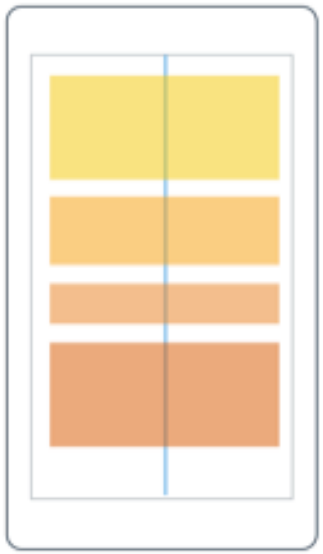


StackLayout

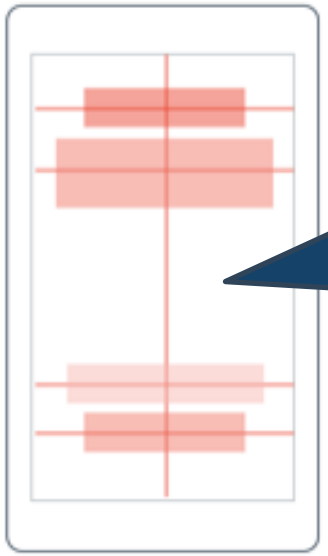
Xamarin University

Layout containers

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



StackLayout



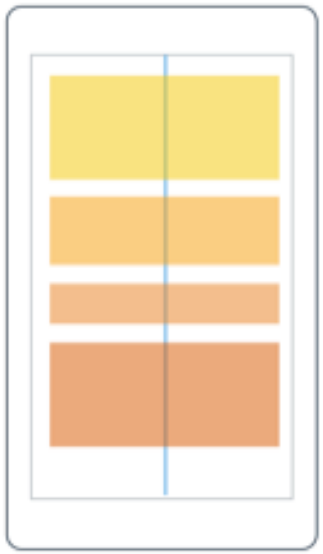
AbsoluteLayout

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

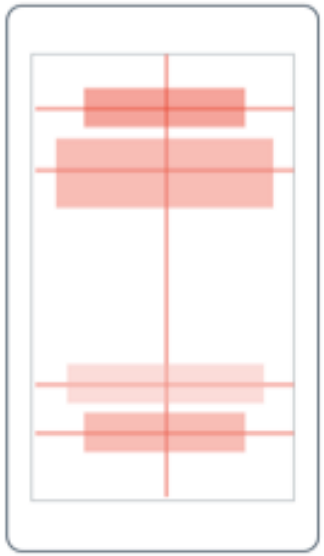
Xamarin University

Layout containers

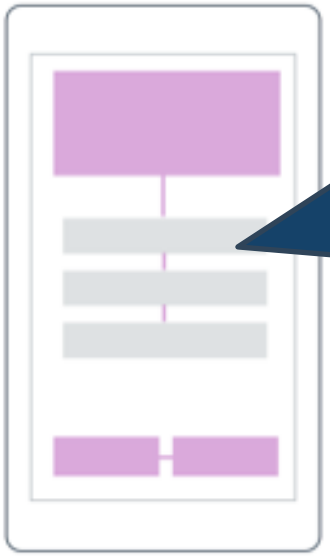
- *Layout Containers* organize child elements based on specific rules



StackLayout



Absolute Layout



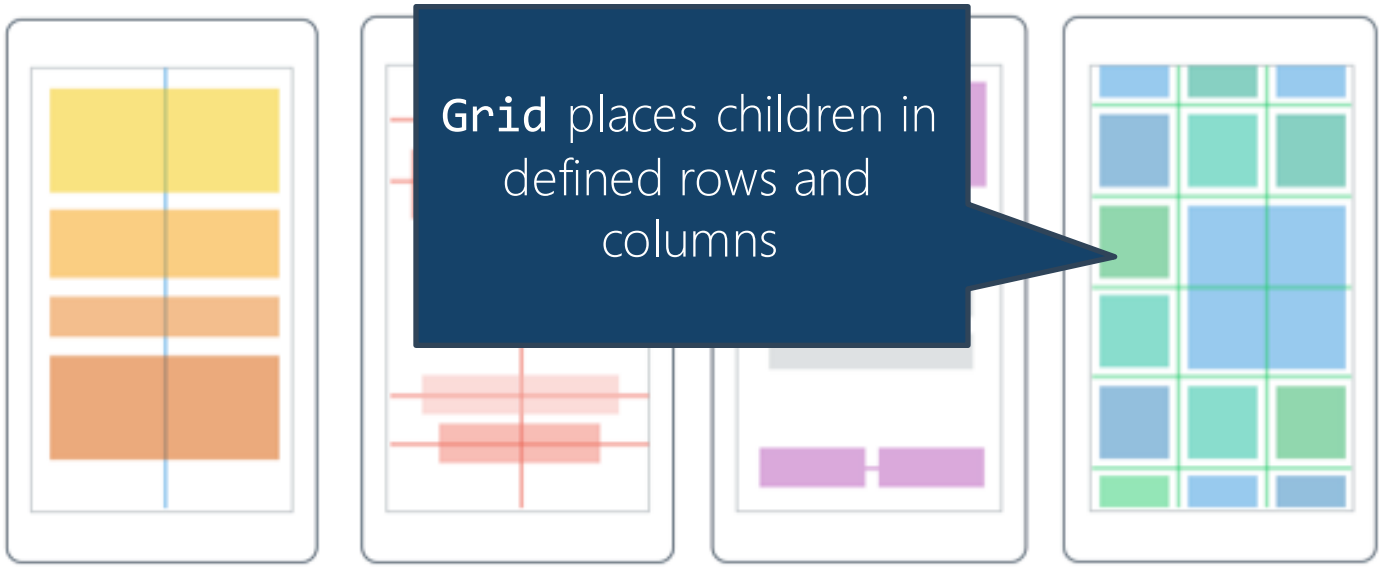
Relative Layout

`RelativeLayout` uses constraints to position the children

Xamarin University

Layout containers

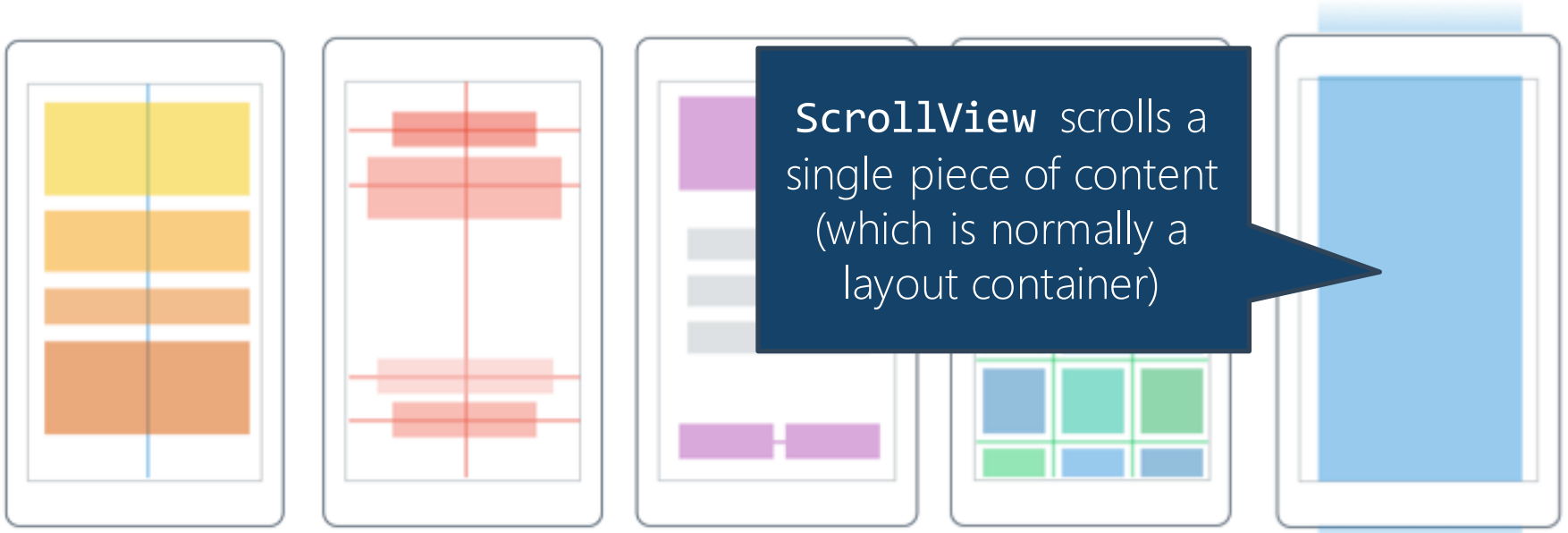
- *Layout Containers* organize child elements based on specific rules



Xamarin University

Layout containers

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



Xamarin University

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

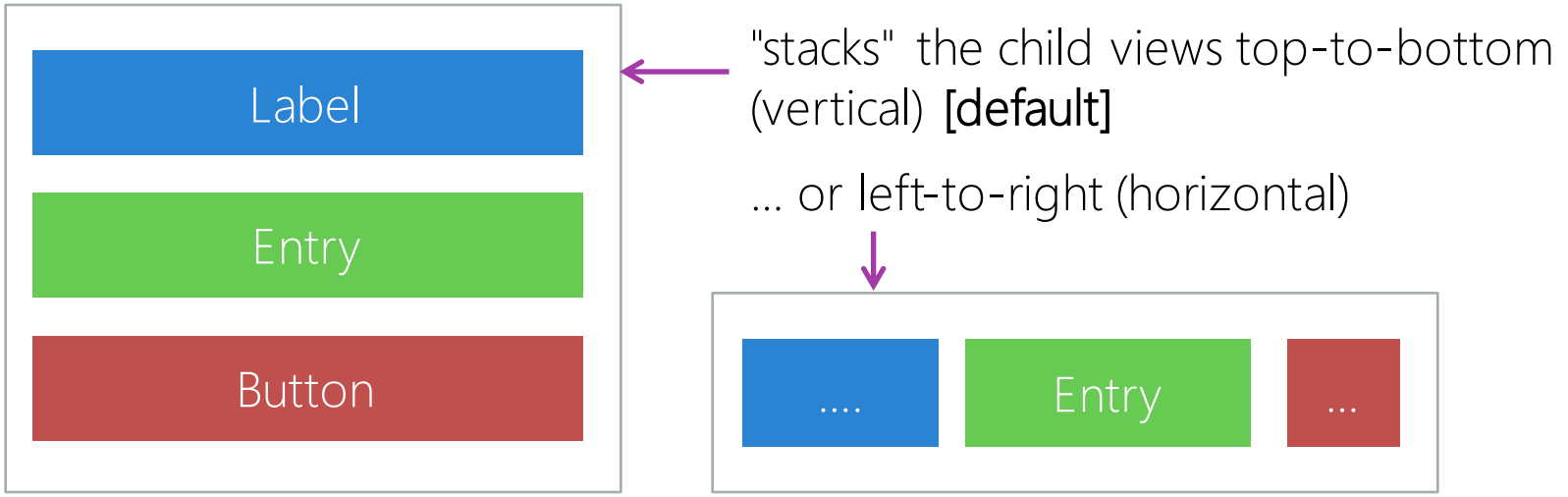
```
Label label = new Label { Text = "Enter Your Name" };
Entry nameEntry = new Entry();

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

Xamarin University

Working with StackLayout

- **`StackLayout`** is used to create typical form style layout

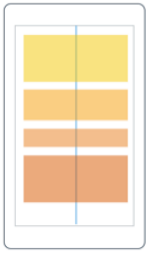


The **`Orientation`** property can be set to either **`Horizontal`** or **`Vertical`** to control which direction the child views are stacked in

Xamarin University

Working with StackLayout

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

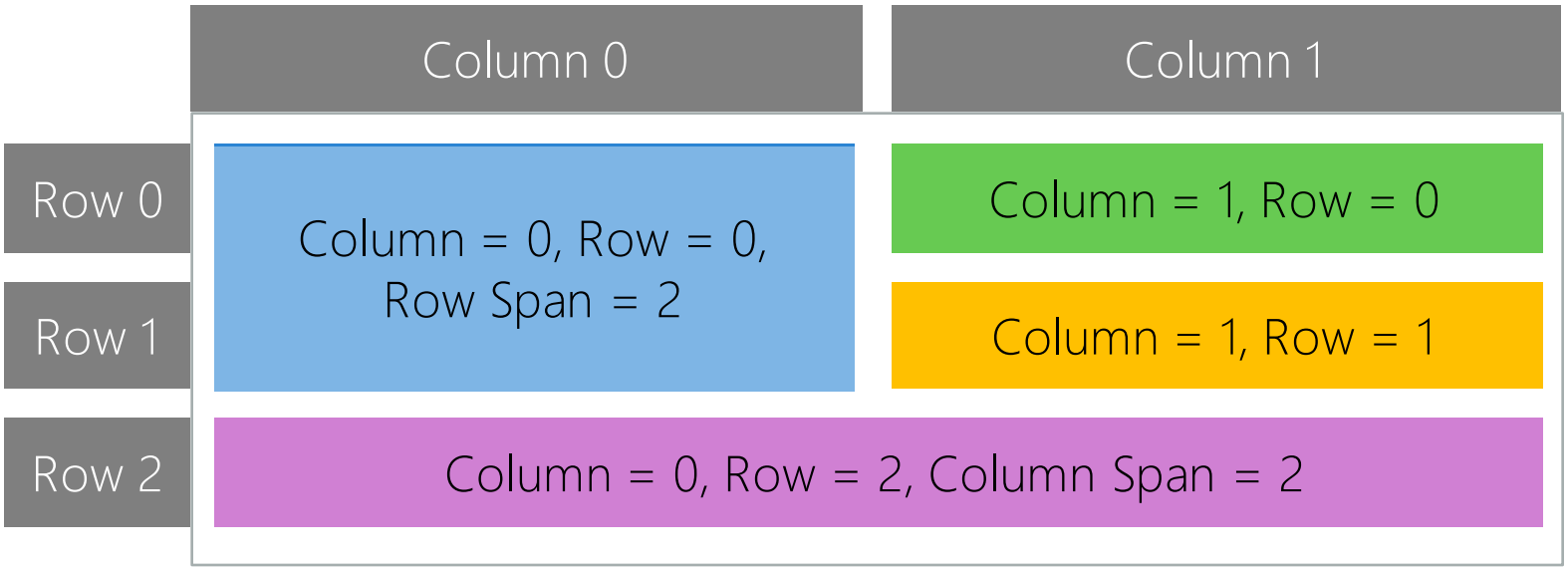


```
var layout = new StackLayout {
    Orientation = StackOrientation.Vertical
};

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

Xamarin University

Working with Grid

- **`Grid`** is used to create rows and columns of information, children identify specific column, row and span



Xamarin University

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

Xamarin University

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, 2);    // L=0, R=2, T=2, B=2
```

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

Xamarin University

Controlling the shape of the grid

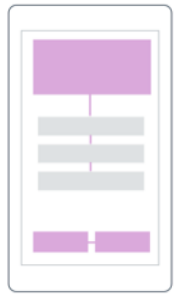
- Can influence the determined shape and size of the columns and rows

```
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
});
```

Xamarin University

Working with RelativeLayout

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

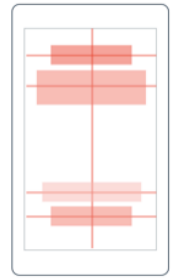


```
var layout = new RelativeLayout();
...
layout.Children.Add(label,
      Constraint.RelativeToParent(
            parent => (0.5 * parent.Width) - 25),    // X
     Constraint.RelativeToView(button,
            (parent, sibling) => sibling.Y + 5),     // Y
     Constraint.Constant(50),           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

Xamarin University

Working with AbsoluteLayout

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

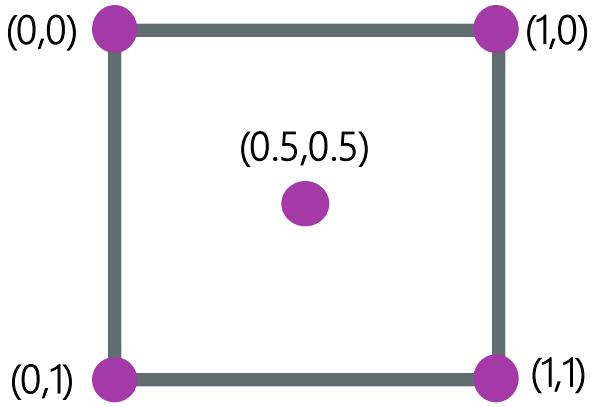


```
var layout = new AbsoluteLayout();
...
// Can do absolute positions by coordinate point
layout.Children.Add(label1, new Point(100, 100));

// Or use a specific bounding box
layout.Children.Add(label2, new Rectangle(20, 20, 100, 25));
```

Xamarin University

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



Xamarin University

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)

Xamarin University

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]

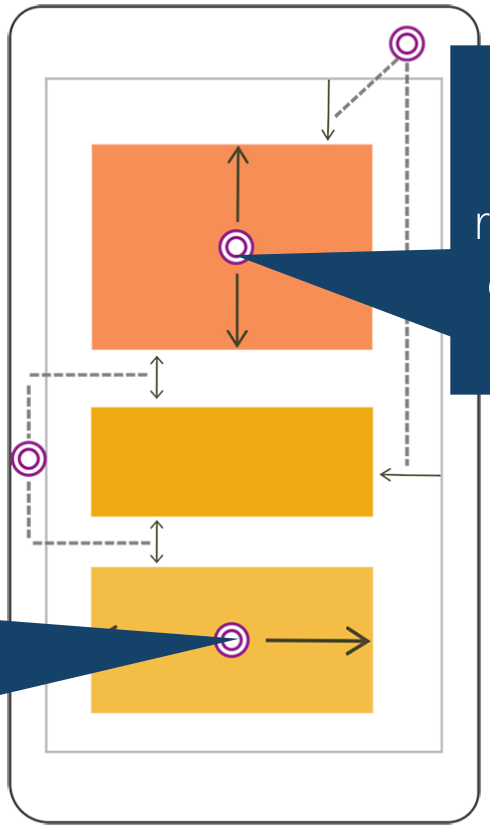
Xamarin University

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);
}
```

Adding spacing and padding

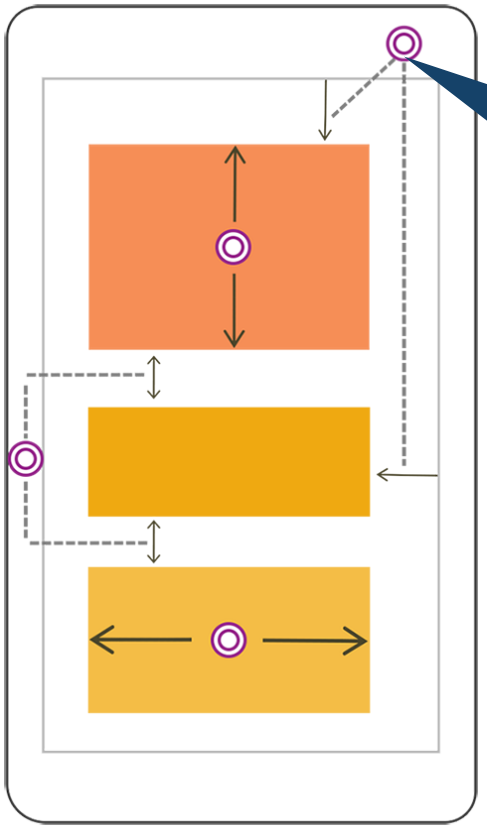


`HeightRequest` property used to request a specific height on a view in the parent container

`WidthRequest` property used to request a specific width on a view in the parent container

```
Button okButton = ...;
okButton.WidthRequest  = 100;
okButton.HeightRequest  = 75;
```

Adding spacing and padding



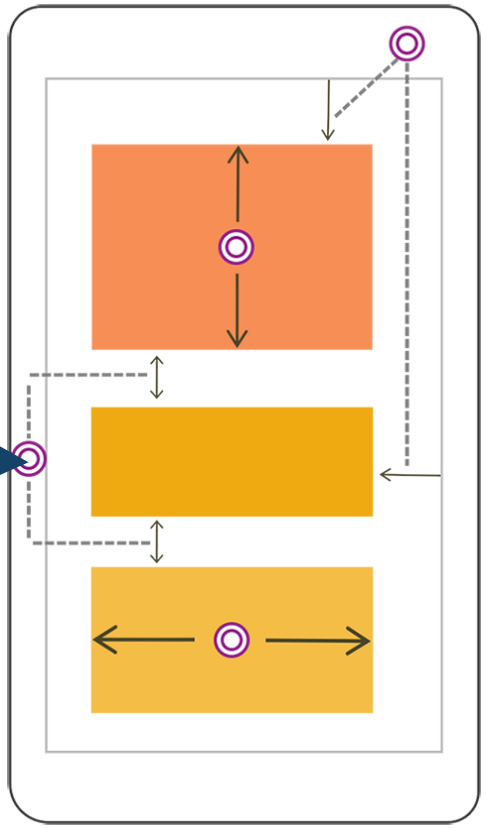
Padding property on parent containers is used to add padding *around* the children

```
ContentPage mainPage = ...;
mainPage.Padding =
    new Thickness(5,20,5,5);
```

Xamarin University

Adding spacing and padding

`Spacing` property on **`StackLayout`** and **`Grid`** allows you to control spacing *in-between* children



```
StackLayout layout = ...;
layout.Spacing = 20;
```

```
Grid layout = ...;
layout.RowSpacing = 10;
layout.ColumnSpacing = 20;
```

Individual Exercise

Creating Xamarin.Forms Phoneword

Flash Quiz

Xamarin University

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

Xamarin University

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

Xamarin University

Flash Quiz

② Which of these controls is not available in Xamarin.Forms?

a) Button
b) DatePicker
c) ListBox
d) ListView

Xamarin University

Flash Quiz

② Which of these controls is not available in Xamarin.Forms?

a) Button
b) DatePicker
c) ListBox
d) ListView

Xamarin University

Flash Quiz

③ To adjust spacing between children when using the `StackLayout` container we can change the _____ property.

a) Margin

b) Padding

c) Spacing

Xamarin University

Flash Quiz

③ To adjust spacing between children when using the `StackLayout` container we can change the _____ property.

a) Margin

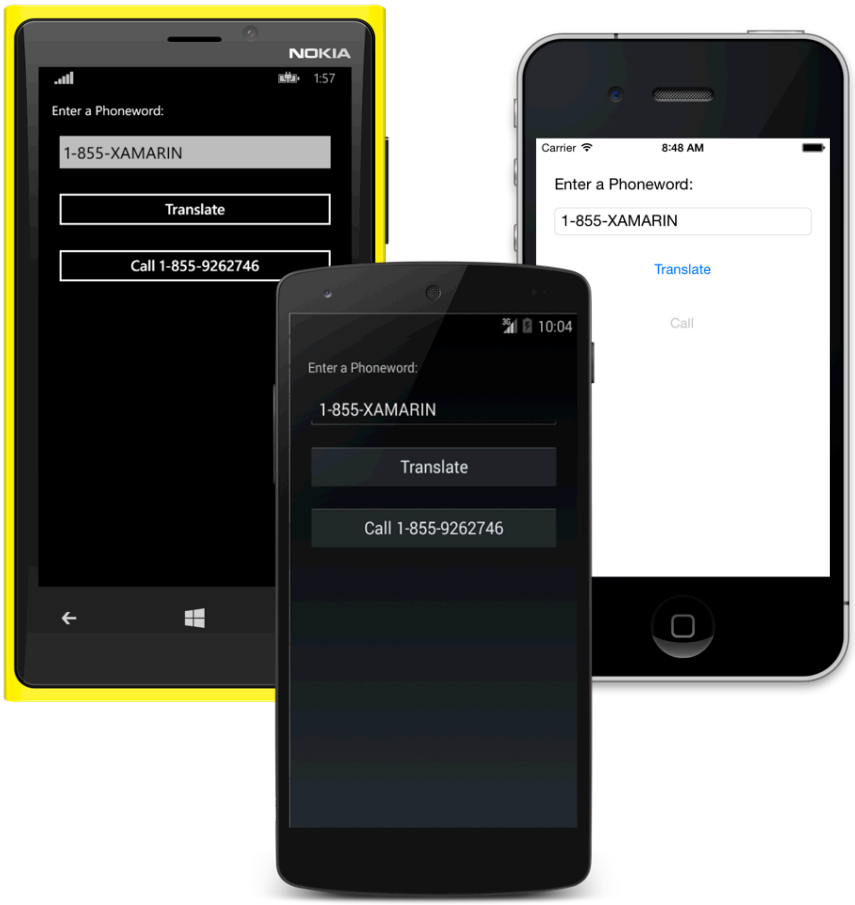
b) Padding

c) Spacing

Xamarin University

Summary

- Layout containers
- Adding views
- Fine-tuning layout

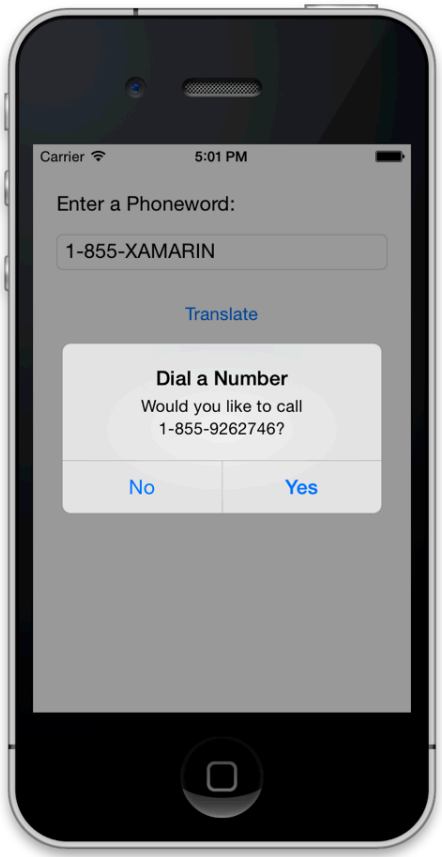


Using Platform-Specific Features

Xamarin University

Tasks

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



Xamarin University

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

Xamarin University

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

Xamarin University

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

Xamarin University

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`

Xamarin University

Using Platform Features

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

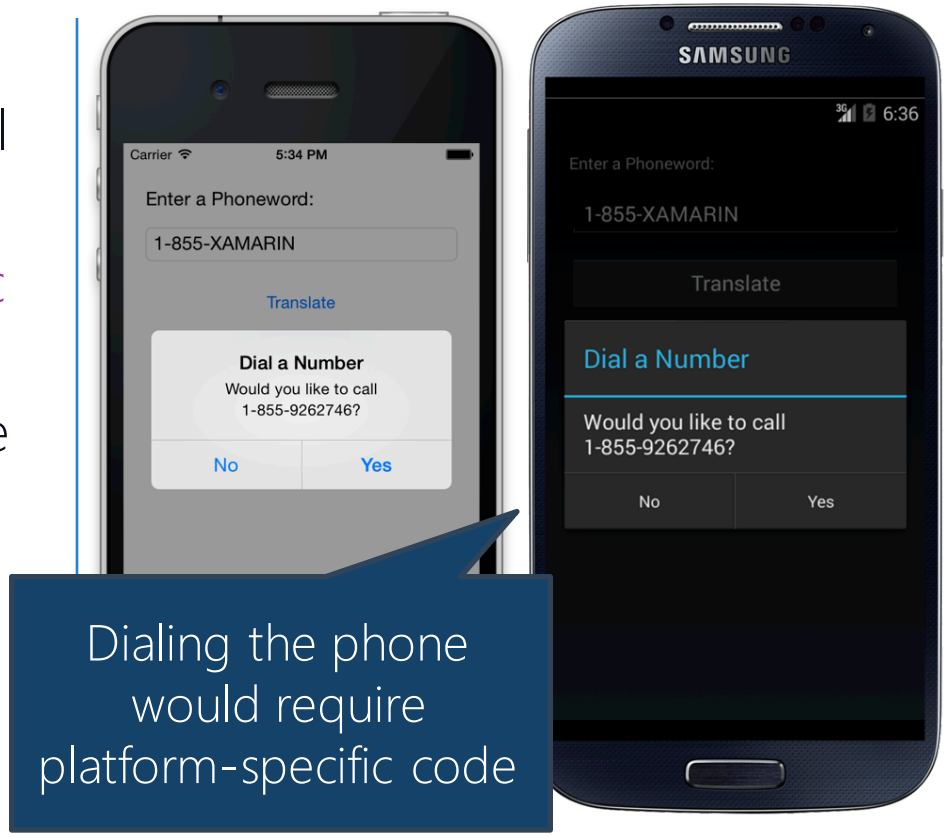
UI Thread marshaling with Device.BeginInvoke OnMainThread

Mapping and Location through `Xamarin.Forms.Maps`

Xamarin University

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
- Attend **XAM110** and **XAM300** for more details



Xamarin University

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

Xamarin University

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);
}
```

Xamarin University

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
    }
}
```

Xamarin University

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

Xamarin University

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

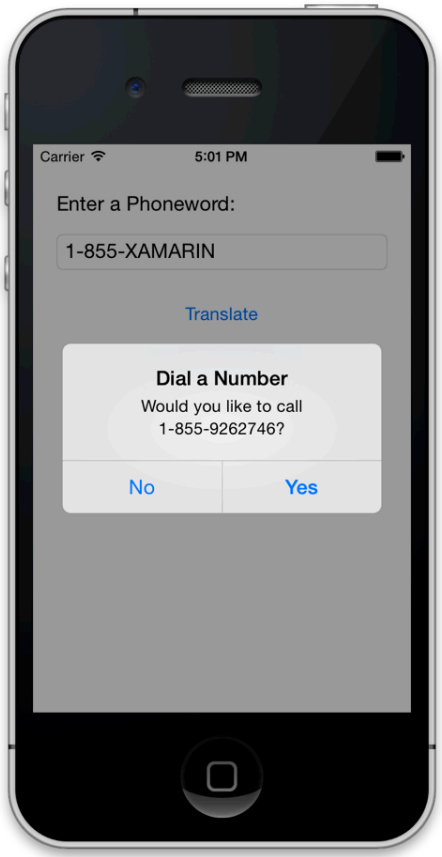
Adding support for dialing the phone

Xamarin University

Xamarin University

Summary

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

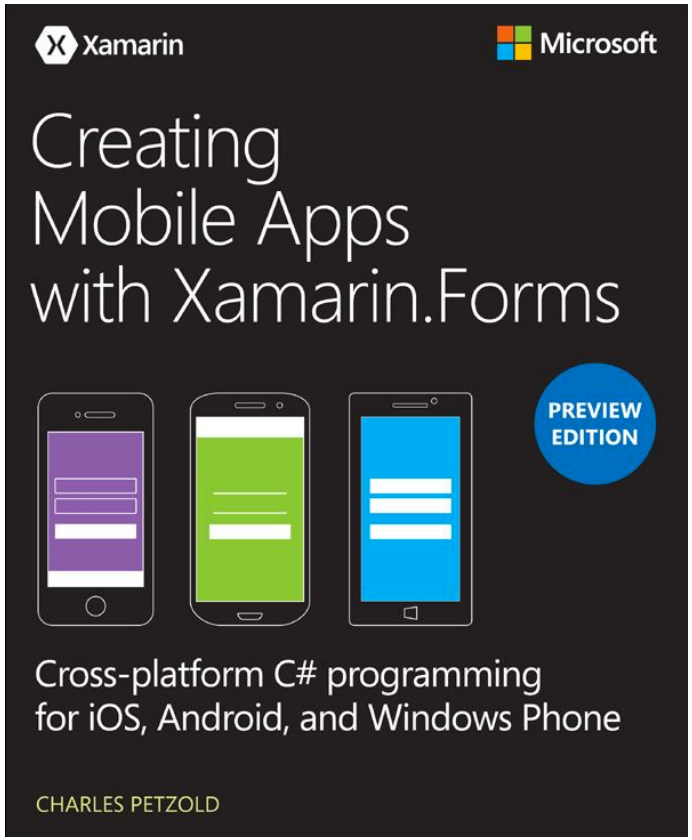


Xamarin University

What's Next?

- ✓ **XAM130** continues your exploration of Xamarin.Forms by diving into XAML
- ✓ **XAM140** looks at Styles and Triggers
- ✓ **XAM310** covers Data Binding
- ✓ **XAM311/312** explores the **`ListView`**
- ✓ **XAM320** caps it off with MVVM coverage

Also, make sure to download Charles Petzold's book online: bit.ly/xforms-book



Thank You!

Please complete the class survey in your profile:
university.xamarin.com/profile