XAM 120

Introduction to Xamarin.Forms

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

Xamarin University

Information in this document is subject to change without notice. The example companies, organizations, products, people, and events depicted herein are fictitious. No association with any real company, organization, product, person or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user.

Xamarin may have patents, patent applications, trademarked, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any license agreement from Xamarin, the furnishing of this document does not give you any license to these patents, trademarks, or other intellectual property.

© 2015 Xamarin. All rights reserved.

Xamarin, MonoTouch, MonoDroid, Xamain.iOS, Xamarin.Android, and Xamarin Studio are either registered trademarks or trademarks of Xamarin in the U.S.A. and/or other countries.

Other product and company names herein may be the trademarks of their respective owners.

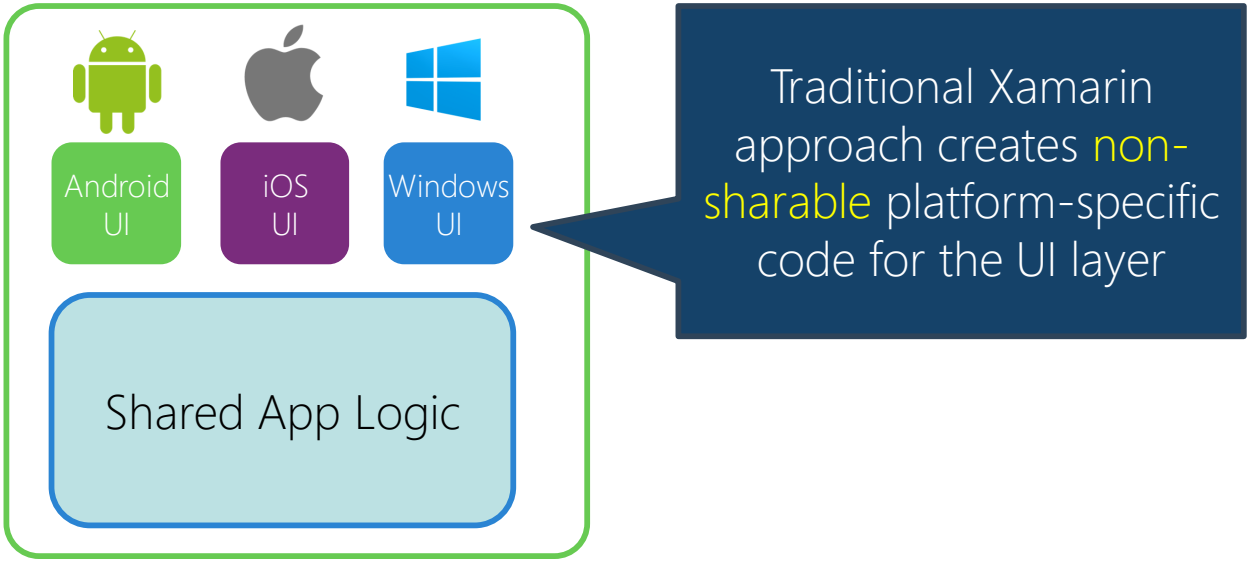
Objectives

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

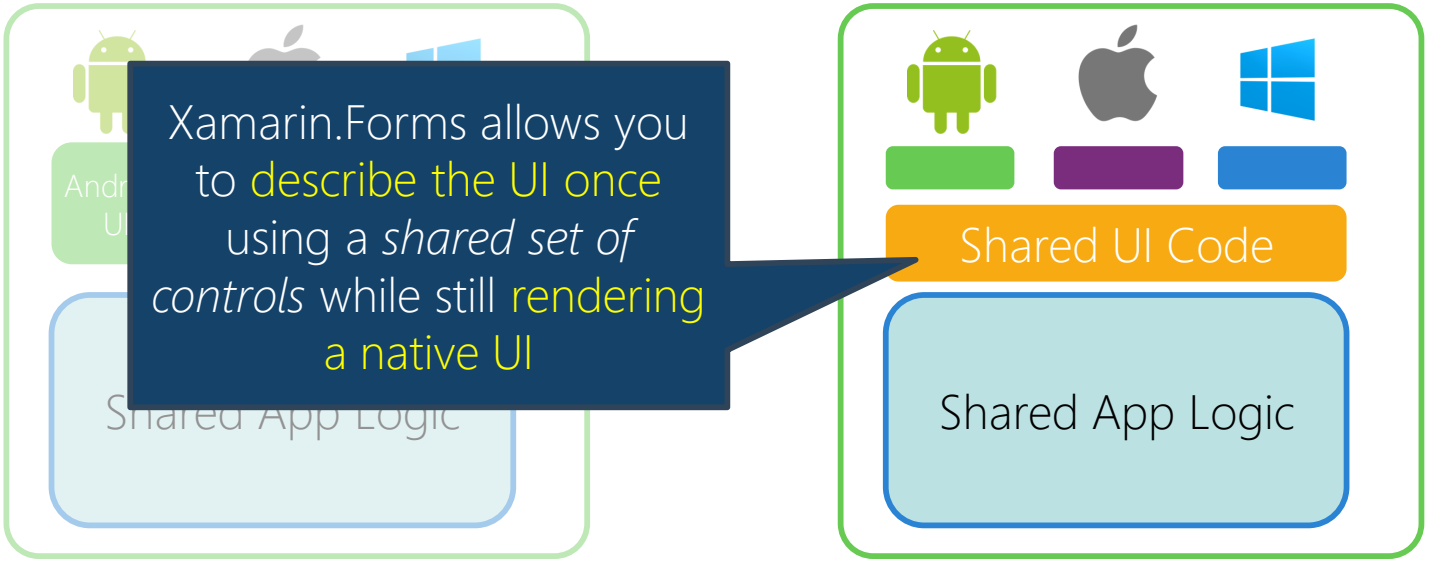
Cross-Platform UI Strategies

Xamarin University

Traditional approach vs. Xamarin.Forms

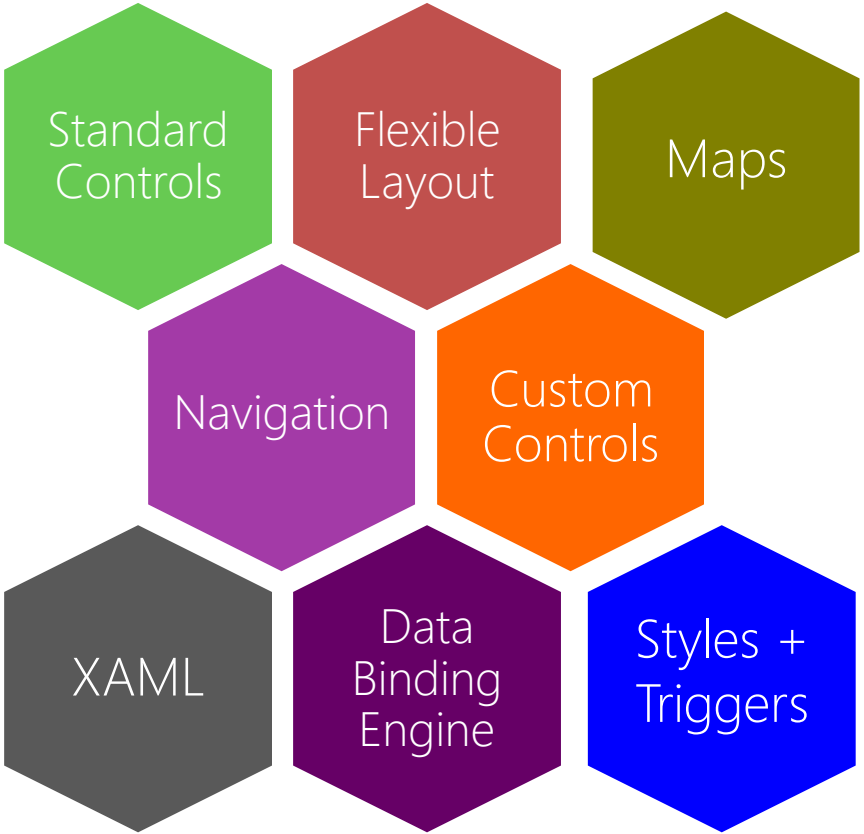


Traditional approach vs. Xamarin.Forms



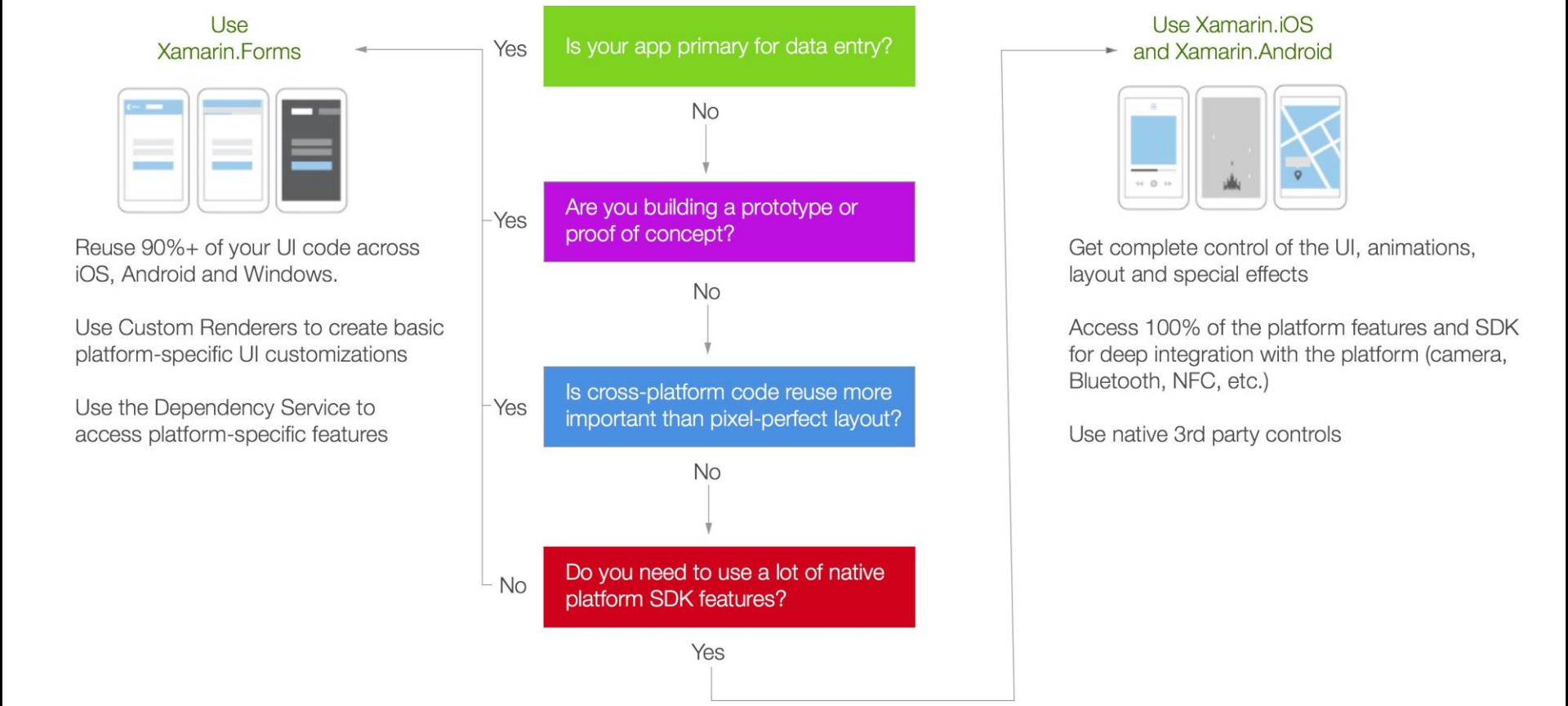
What is Xamarin.Forms?

- 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 (coming soon)



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
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
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
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

Flash Quiz

Xamarin University

Flash Quiz

① Xamarin.Forms uses the native controls on each platform to render a UI

a) True

b) False

Flash Quiz

① Xamarin.Forms uses the native controls on each platform to render a UI

a) True

b) False

Flash Quiz

② Tom wants to build an application that has pixel-perfect layout on both iPhone and iPad devices, Xamarin.Forms would be a perfect choice for this application

a) True

b) False

Flash Quiz

② Tom wants to build an application that has pixel-perfect layout on both iPhone and iPad devices, Xamarin.Forms would be a perfect choice for this application

a) True

b) False

Flash Quiz

③ Xamarin.Forms is perfect for prototyping and quick data-entry type applications which do not require custom UI elements

a) True

b) False

Flash Quiz

③ Xamarin.Forms is perfect for prototyping and quick data-entry type applications which do not require custom UI elements

a) True

b) False

Xamarin Forms Application Structure

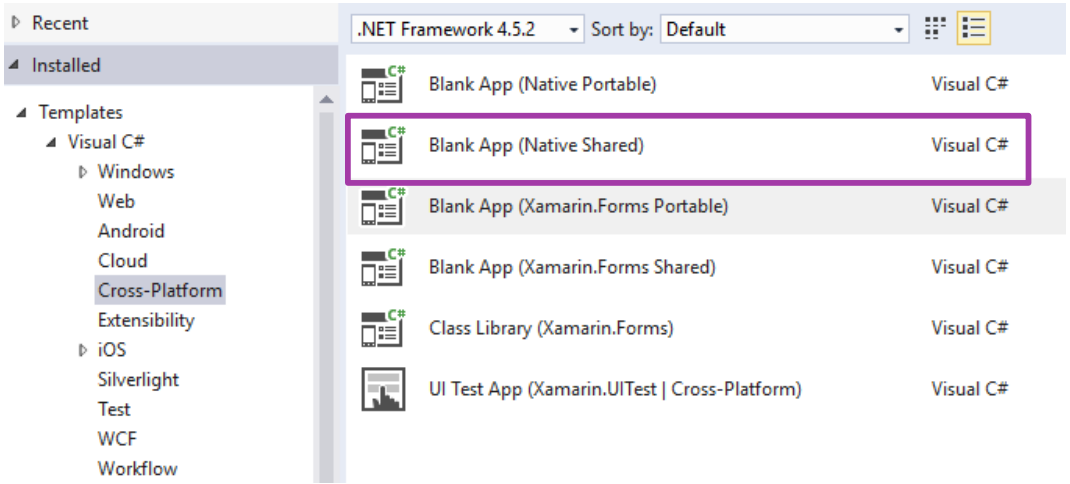
Xamarin University

Tasks

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

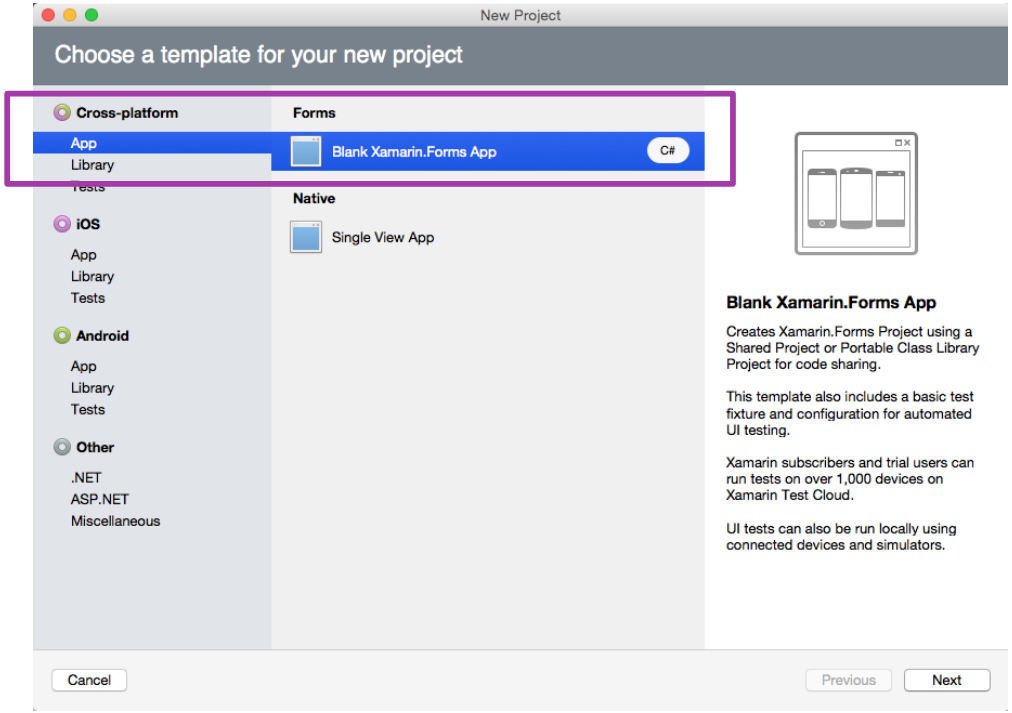
Creating a Xamarin.Forms App

- 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



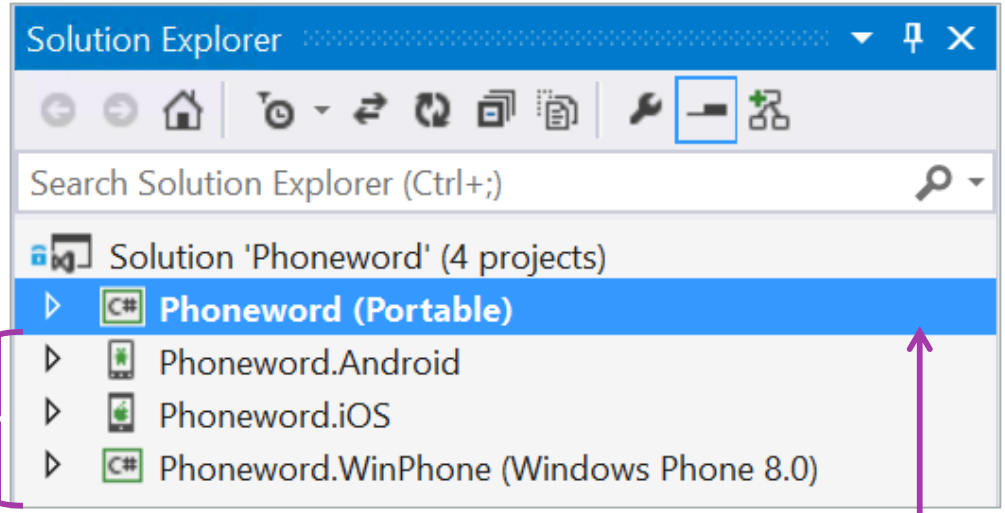
Creating a Xamarin.Forms App

- 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)



Project Structure

- Blank App project template creates several related projects

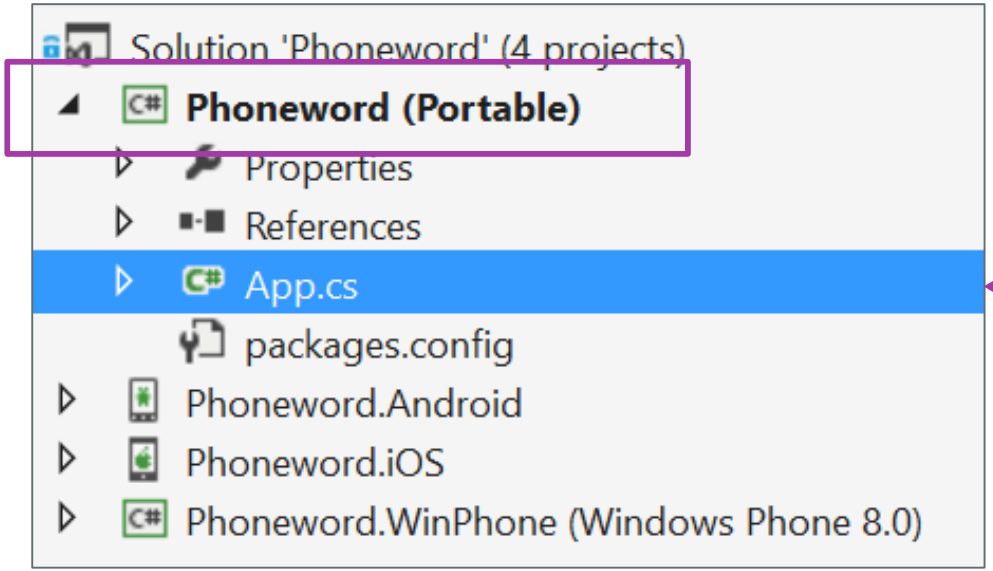


Platform-specific projects act as "host" to create native application

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

Project Structure - PCL

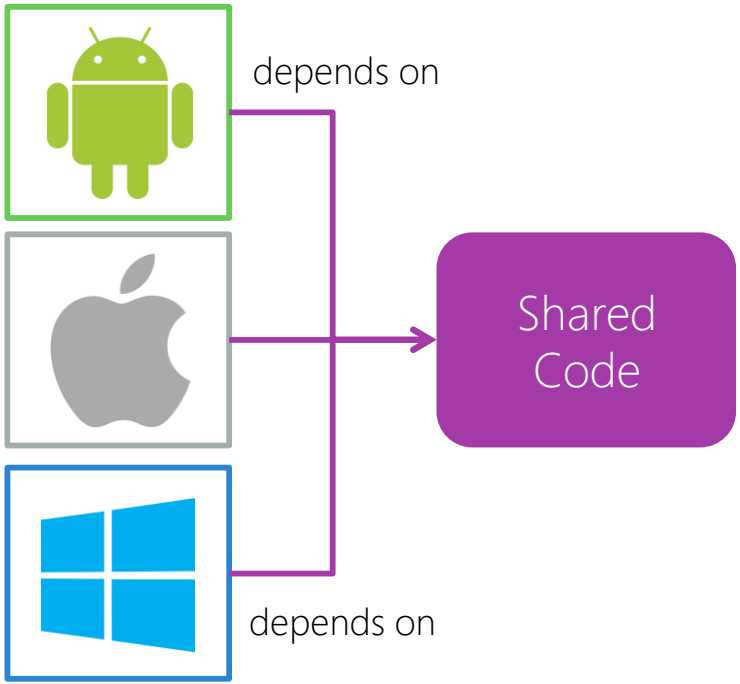
- Most of your code will go into the PCL used for shared logic + UI



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

Demonstration

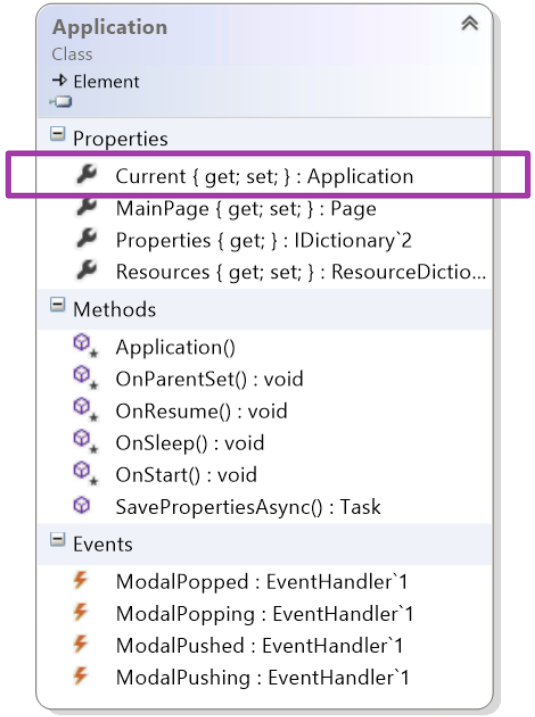
Creating a Xamarin.Forms application

Xamarin University

Xamarin.Forms Application

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

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



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

Xamarin.Forms Application

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

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

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

Xamarin.Forms Application

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

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

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

Xamarin.Forms Application

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

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

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

Persisting information

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

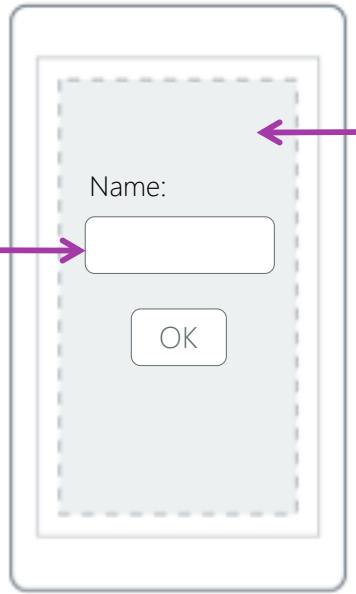
```
// Save off username in global property bag
Application.Current.Properties["username"] = username.Text;
```

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

Creating the application UI

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

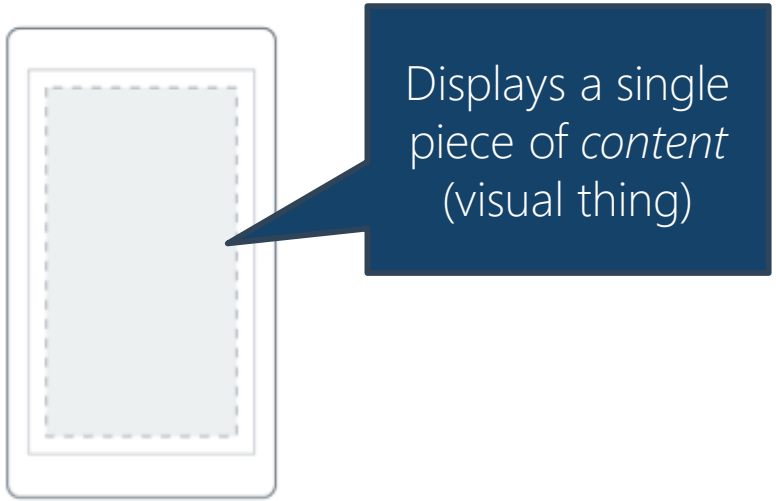
Views are the UI controls the user interacts with



Page represents a single screen displayed in the app

Pages

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

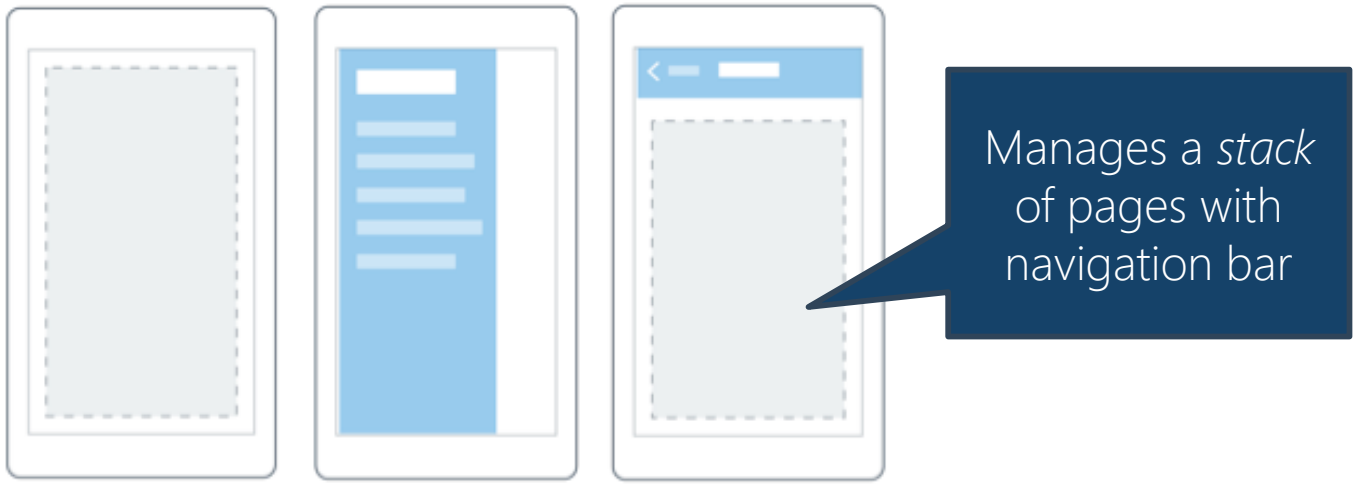


Pages

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

Pages

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

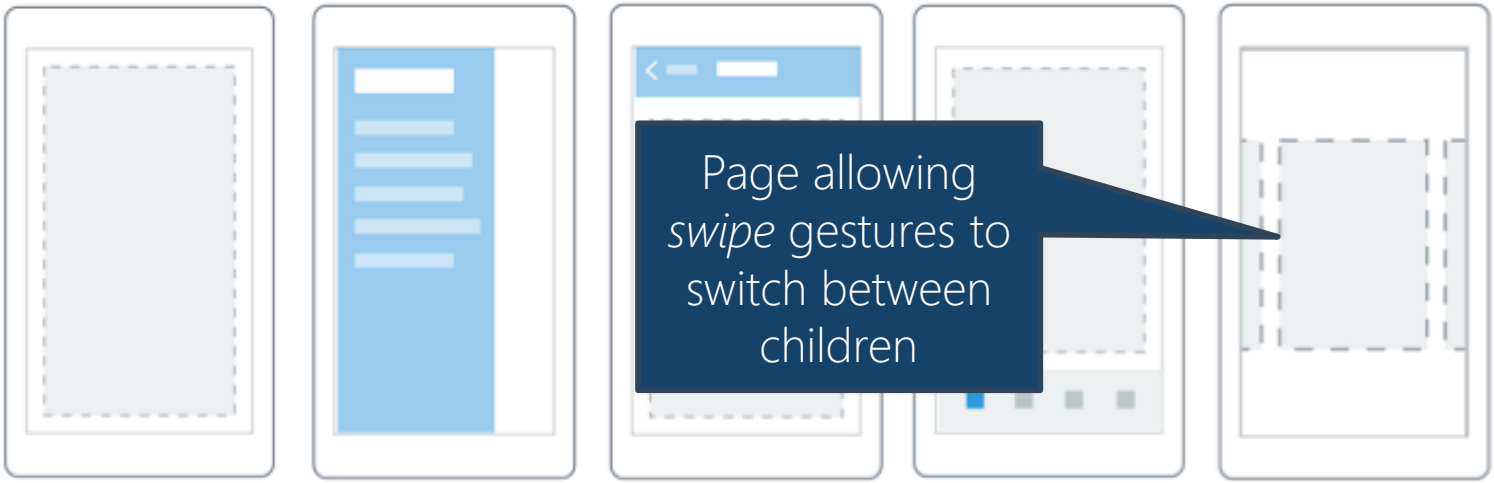


Pages

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

Pages

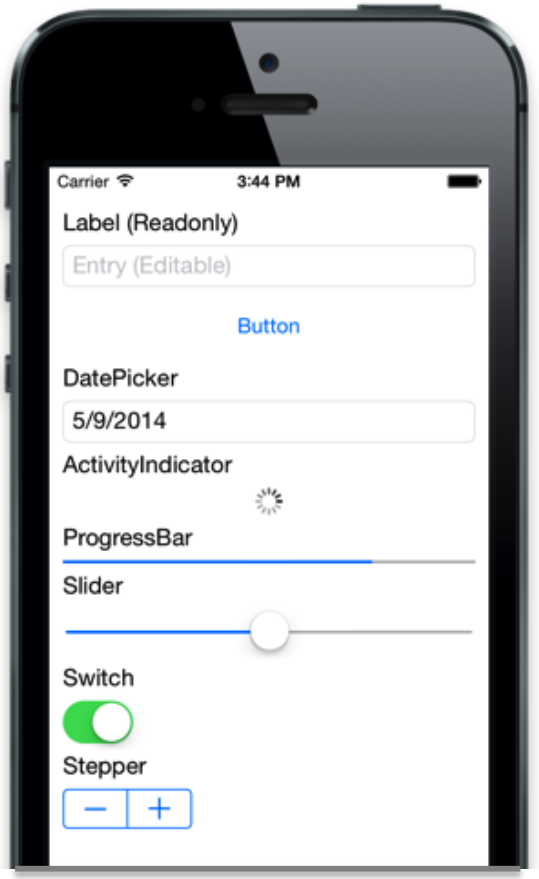
- **Page** is an abstract class used to define a single screen of content
  - derived types provide specific visualization / behavior



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	



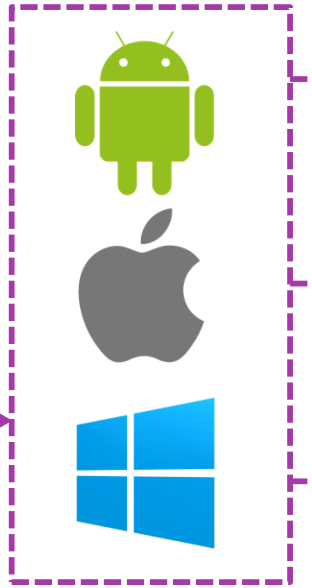
Rendering views

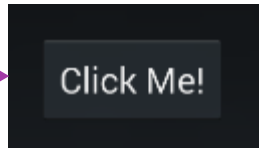
- Platform defines a *renderer* for each view that creates a native representation of the UI

UI uses a Xamarin.Forms `Button`

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

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





Android.Widget.Button

UIButton

System.Windows.Button

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

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

Flash Quiz

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

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

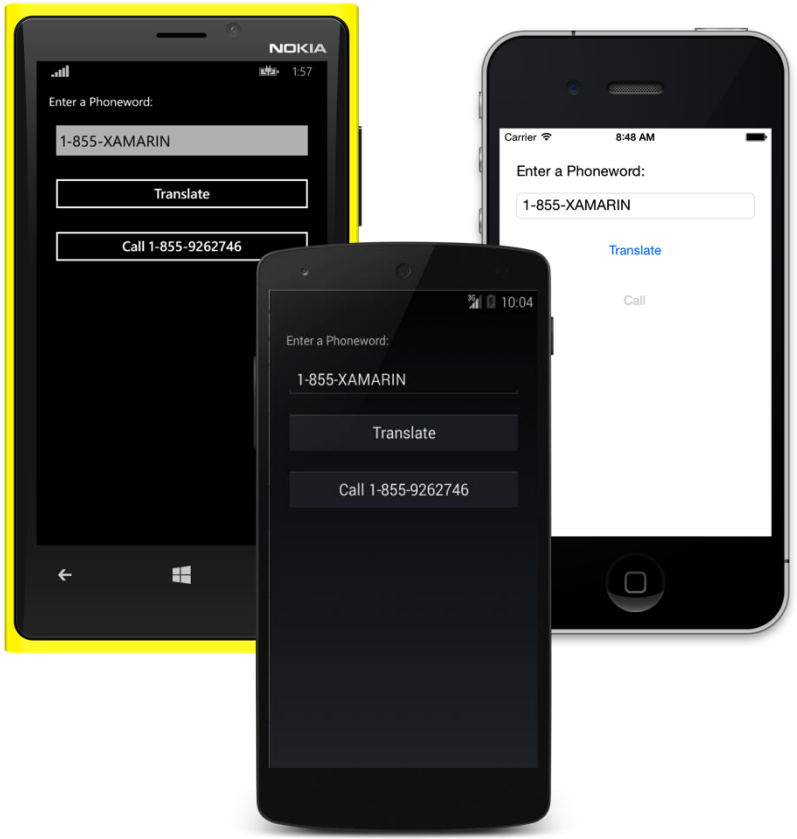
Summary

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

Creating Phoneword in Xamarin.Forms

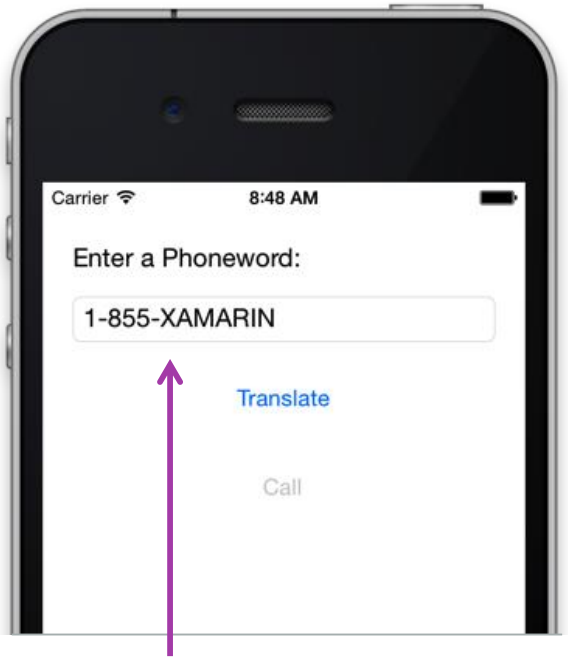
Tasks

- Layout containers
- Adding views
- Fine-tuning layout



Organizing content

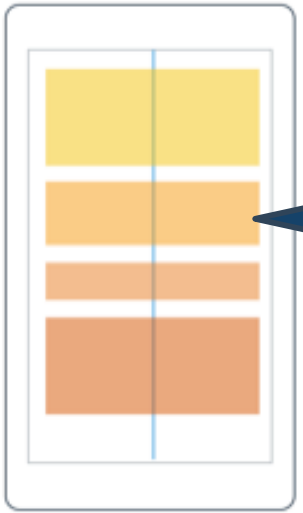
- Rather than specifying positions with coordinates (pixels, dips, etc.), you use layout containers to control how views are positioned relative to each other; this provides for a more a*daptive* layout which is not as sensitive to dimensions and resolutions



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

Layout containers

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

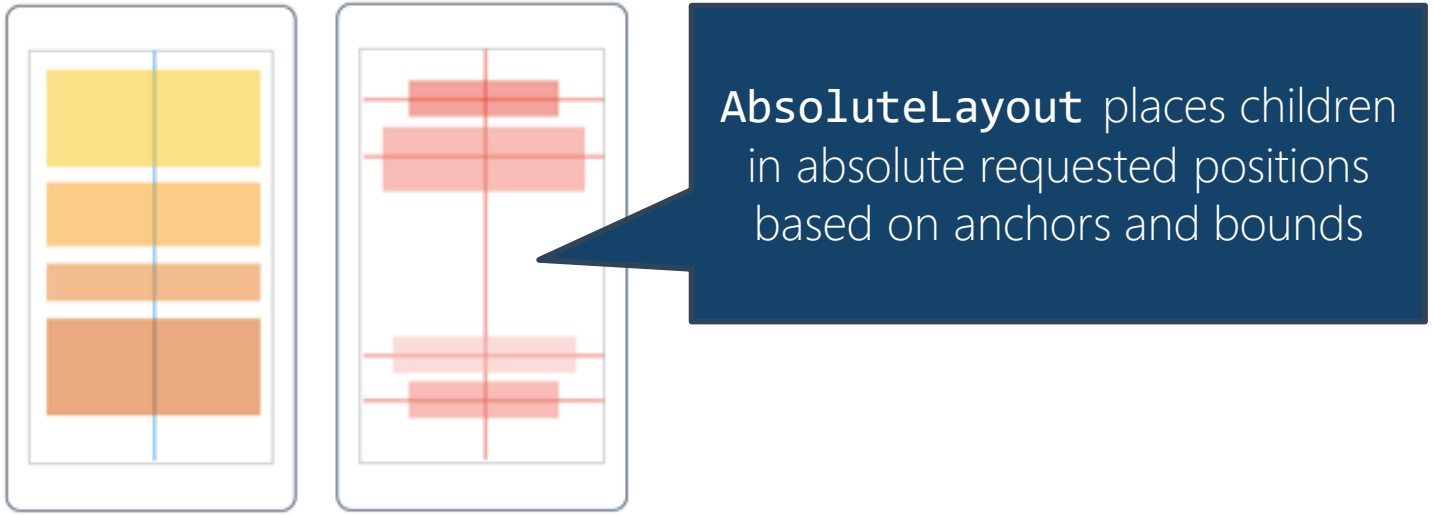


StackLayout

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

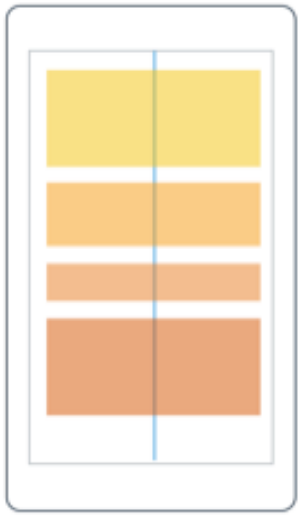
Layout containers

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

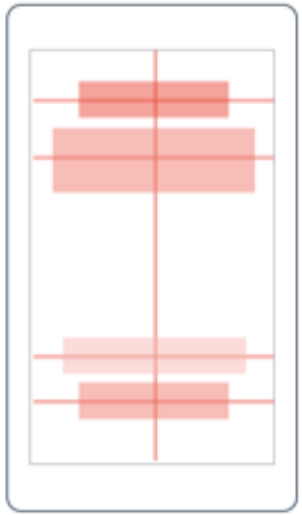


Layout containers

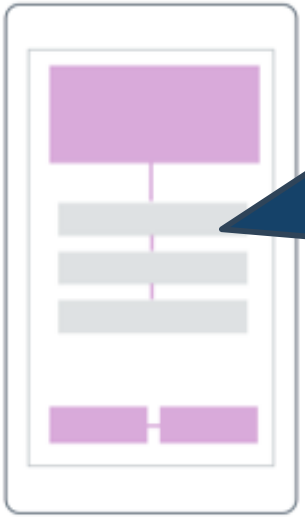
- *Layout Containers* organize child elements based on specific rules



StackLayout



Absolute Layout

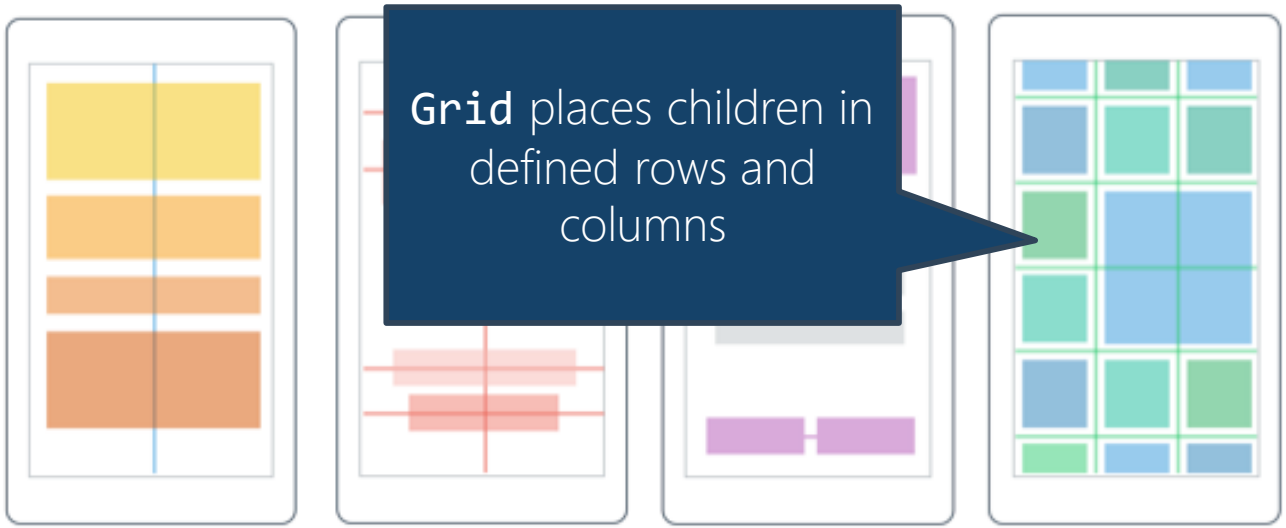


Relative Layout

`RelativeLayout` uses constraints to position the children

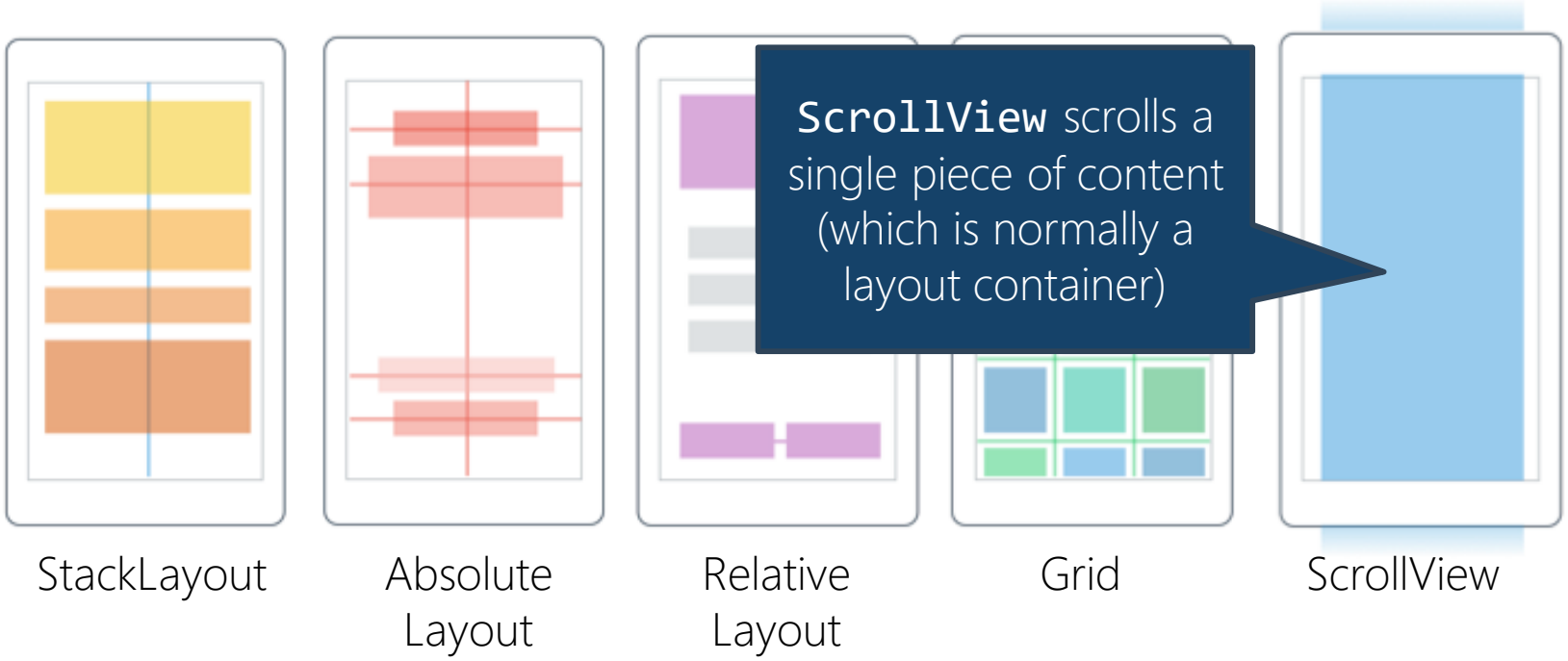
Layout containers

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



Layout containers

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



Adding views to layout containers

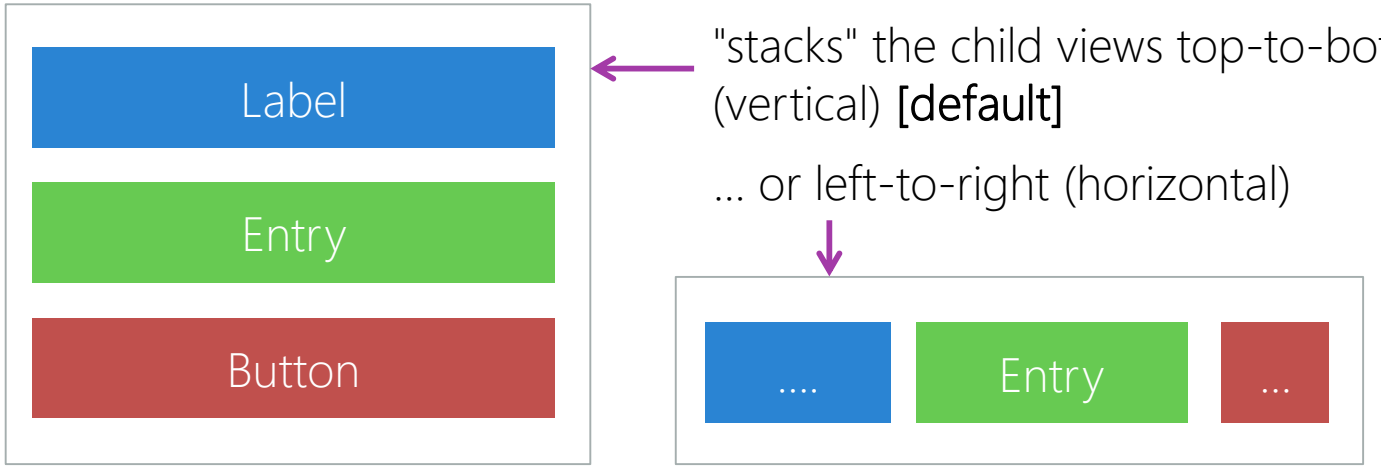
- Layout containers have a **`Children`** collection property which is used to hold the views that will be organized by the container

```
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

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

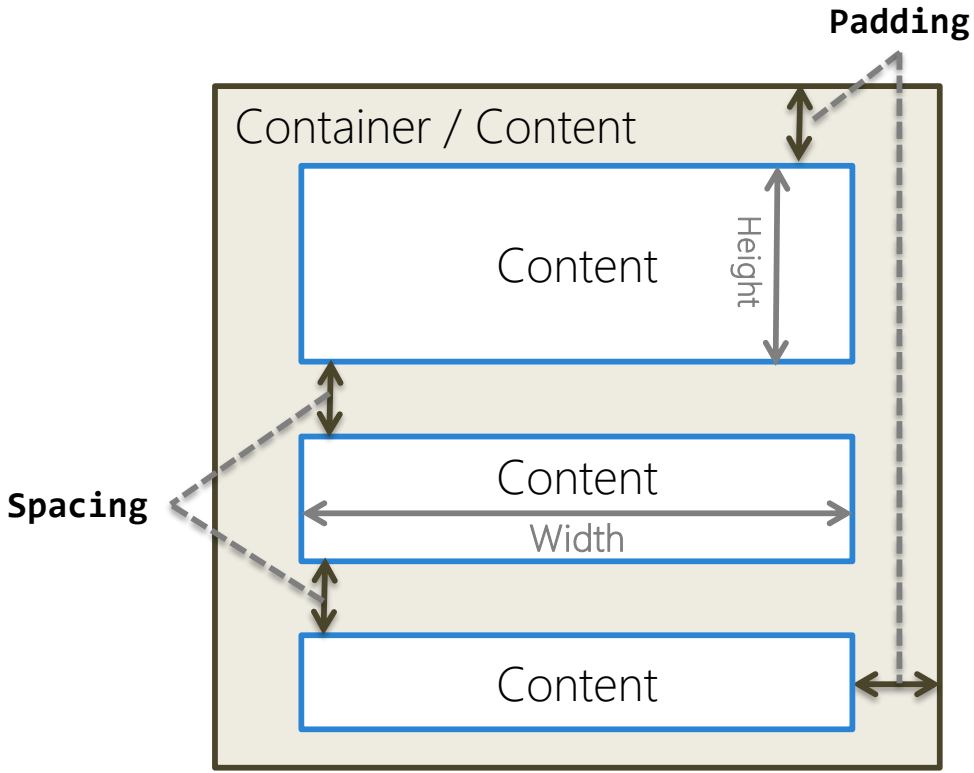
Controlling Width and Height

- 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

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

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.

a) Margin

b) Padding

c) Spacing

Flash Quiz

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

a) Margin

b) Padding

c) Spacing

Summary

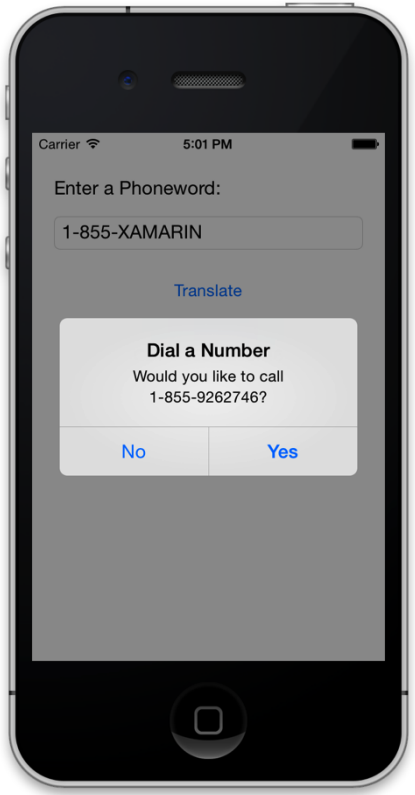
- Layout containers
- Adding views
- Fine-tuning layout

Using Platform-Specific Code

Xamarin University

Tasks

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

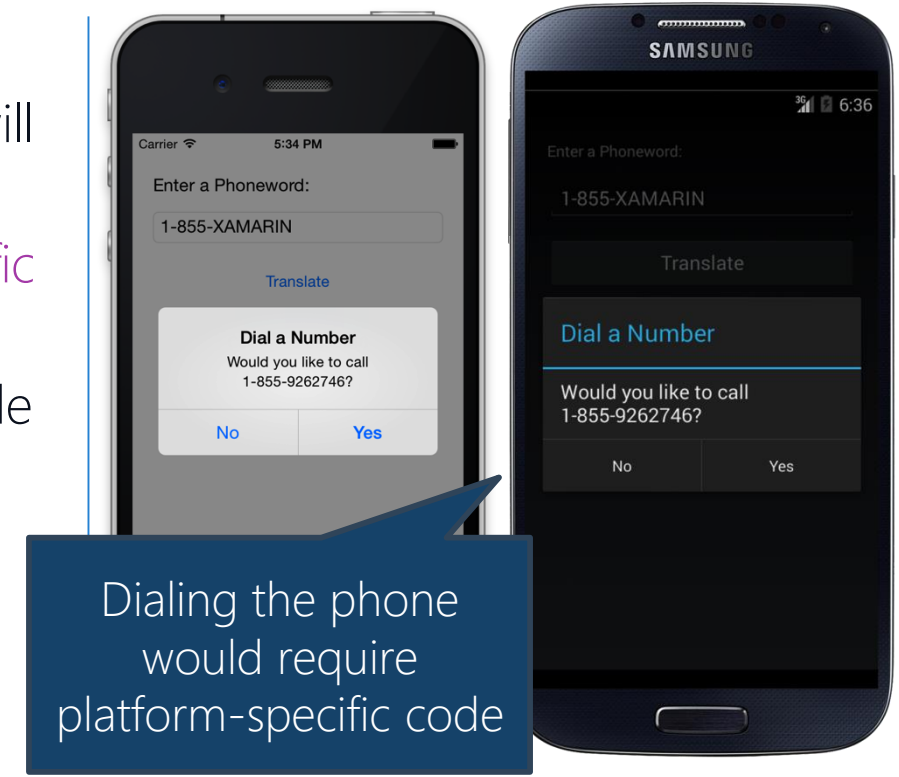
- Xamarin.Forms has support for dealing with a few, very common platform-specific features

UI Thread marshaling with Device.BeginInvoke OnMainThread

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

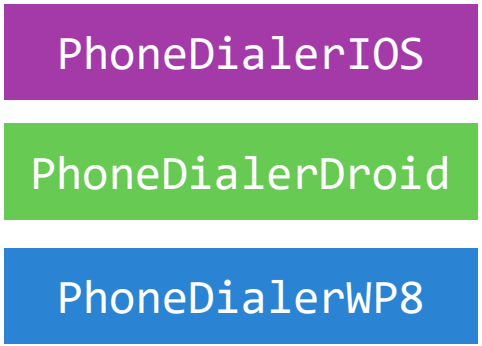


Creating abstractions

- 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



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

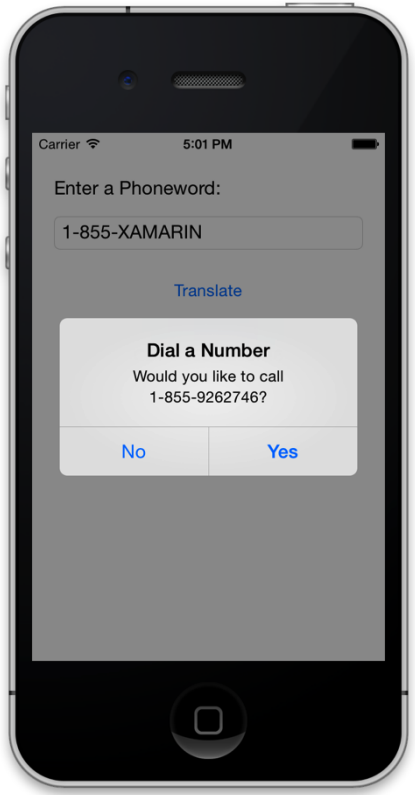
Individual Exercise

Adding support for dialing the phone

Xamarin University

Summary

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

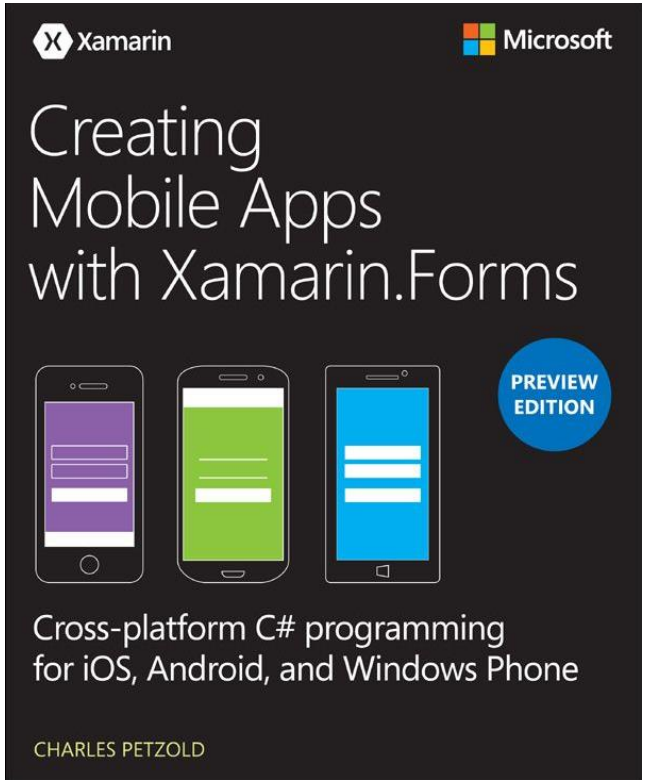


What's Next?

- **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



Thank You!

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