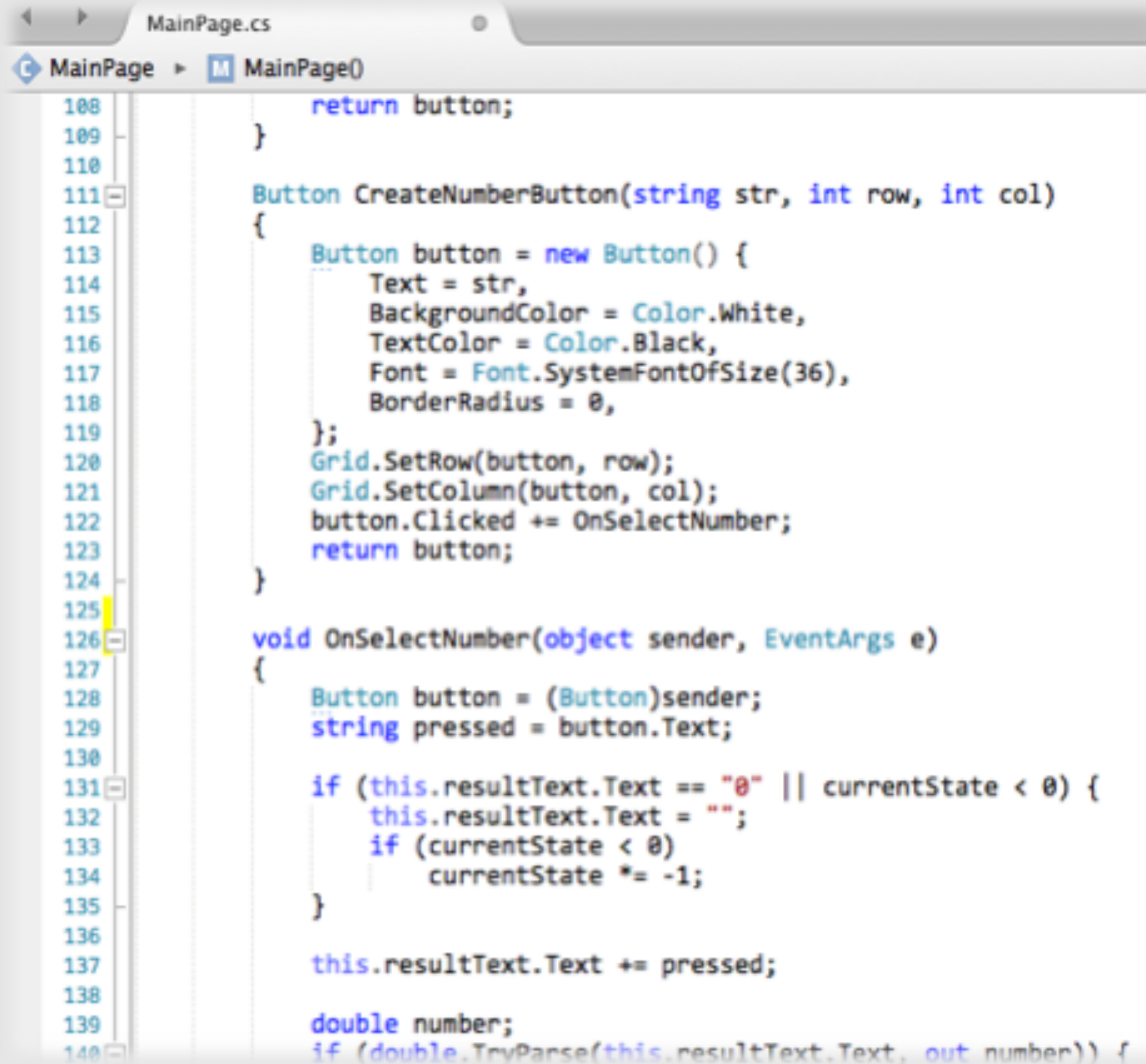
XAML in Xamarin.Forms

Creating Pages in Code

Significant portion of code behind tends to be in **UI creation**: setup and layout

X Mixing of UI and behavior in one file can make **both design and behavior harder to understand / change**

X Prohibits designer role involvement

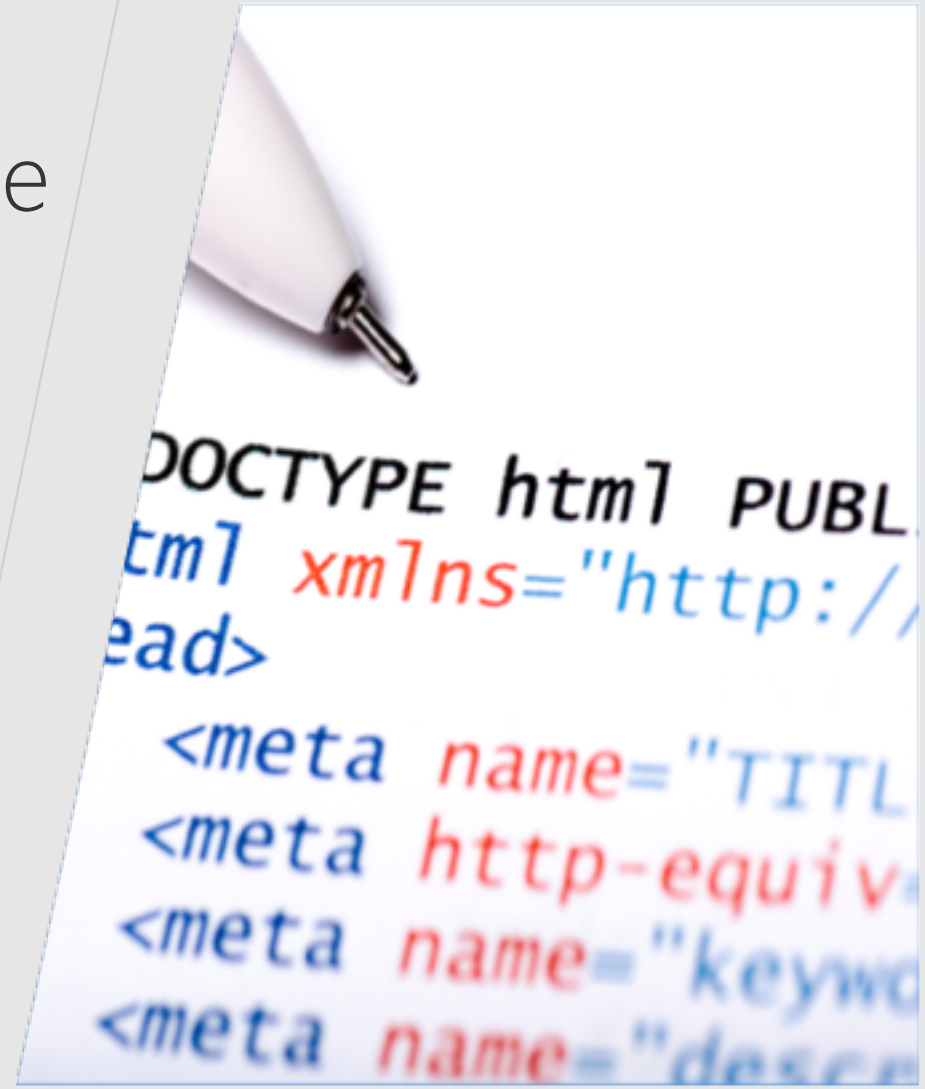


```
        return button;
    }

    Button CreateNumberButton(string str, int row, int col)
    {
        Button button = new Button() {
            Text = str,
            BackgroundColor = Color.White,
            TextColor = Color.Black,
            Font = Font.SystemFontOfSize(36),
            BorderRadius = 0,
        };
        Grid.SetRow(button, row);
        Grid.SetColumn(button, col);
        button.Clicked += OnSelectNumber;
        return button;
    }

    void OnSelectNumber(object sender, EventArgs e)
    {
        Button button = (Button)sender;
        string pressed = button.Text;

        if (this.resultText.Text == "0" || currentState < 0) {
            this.resultText.Text = "";
            if (currentState < 0)
                currentState *= -1;
        }

        this.resultText.Text += pressed;

        double number;
        if (double.TryParse(this.resultText.Text, out number)) {
```

Working in Markup

- HTML has taught us that markup languages are a great way to define user interfaces because they are:

✓Toolable
✓Human readable
✓Extensible



Extensible Application Markup Language

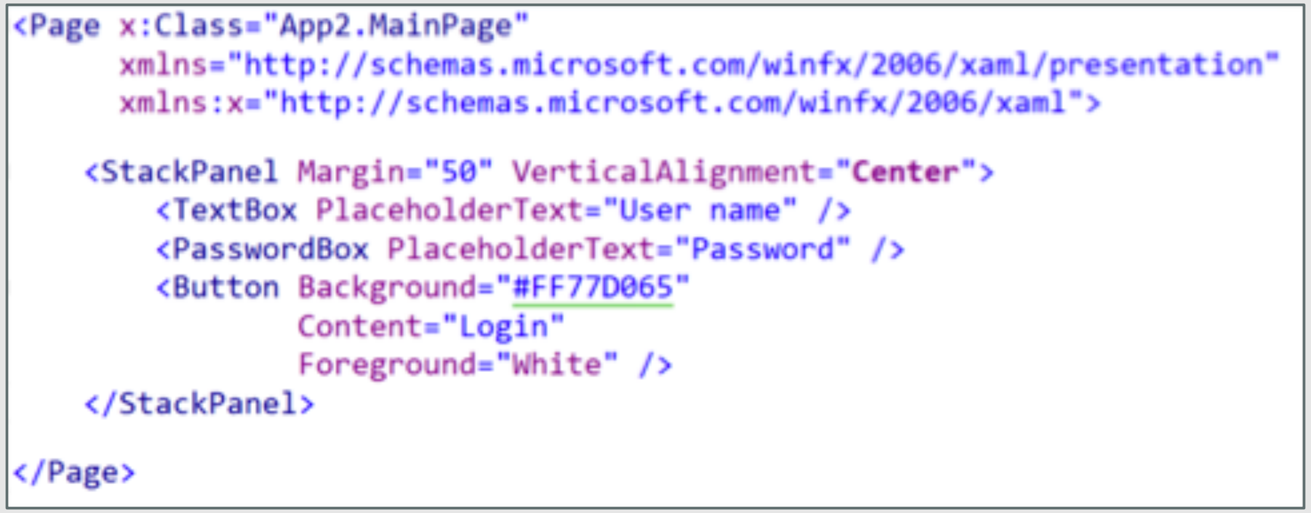
XAML was created by Microsoft specifically to describe UI

XAML

Xamarin Forms + XAML
= Sweetness!

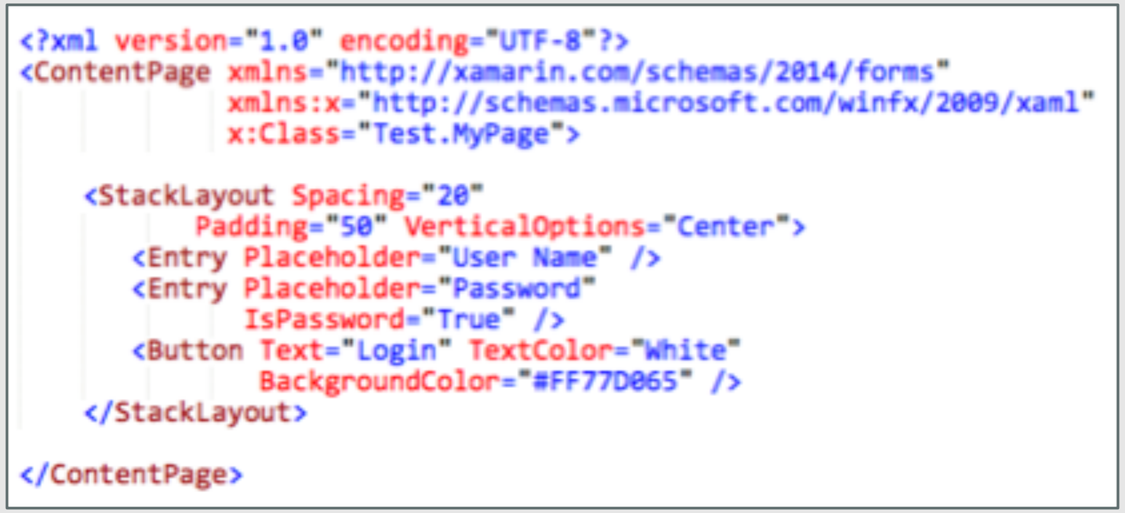
Microsoft XAML vs. Xamarin.Forms

Xamarin.Forms conforms to the **XAML 2009** specification; the differences are really in the controls and layout containers you use



```xml
<Page x:Class="App2.MainPage"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

    <StackPanel Margin="50" VerticalAlignment="Center">
        <TextBox PlaceholderText="User name" />
        <PasswordBox PlaceholderText="Password" />
        <Button Background="#FF77D065"
                Content="Login"
                Foreground="White" />
    </StackPanel>

</Page>
```

Microsoft XAML (WinRT)

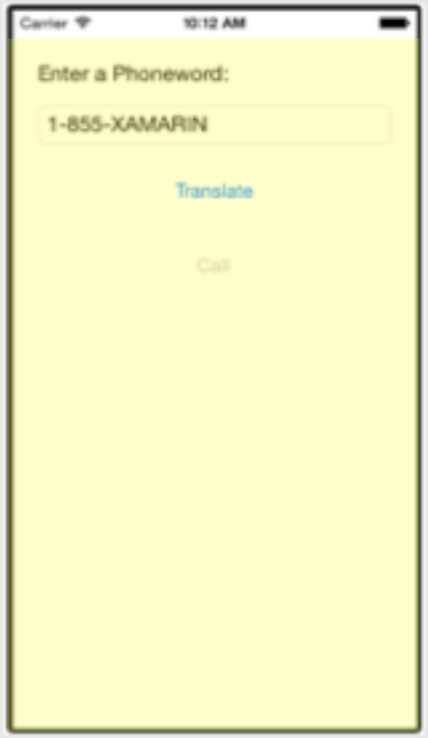


```xml
<?xml version="1.0" encoding="UTF-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Test.MyPage">

    <StackLayout Spacing="20"
          Padding="50" VerticalOptions="Center">
      <Entry Placeholder="User Name" />
      <Entry Placeholder="Password"
             IsPassword="True" />
      <Button Text="Login" TextColor="White"
              BackgroundColor="#FF77D065" />
    </StackLayout>

</ContentPage>
```

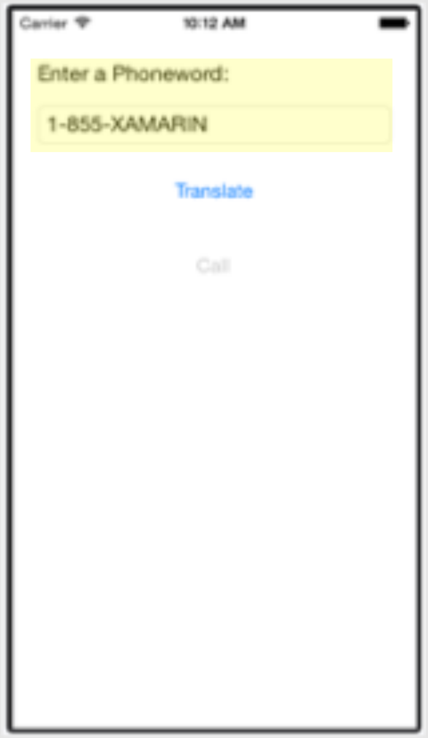
Xamarin.Forms

Adding a XAML Page

There are two Item Templates available to add XAML content



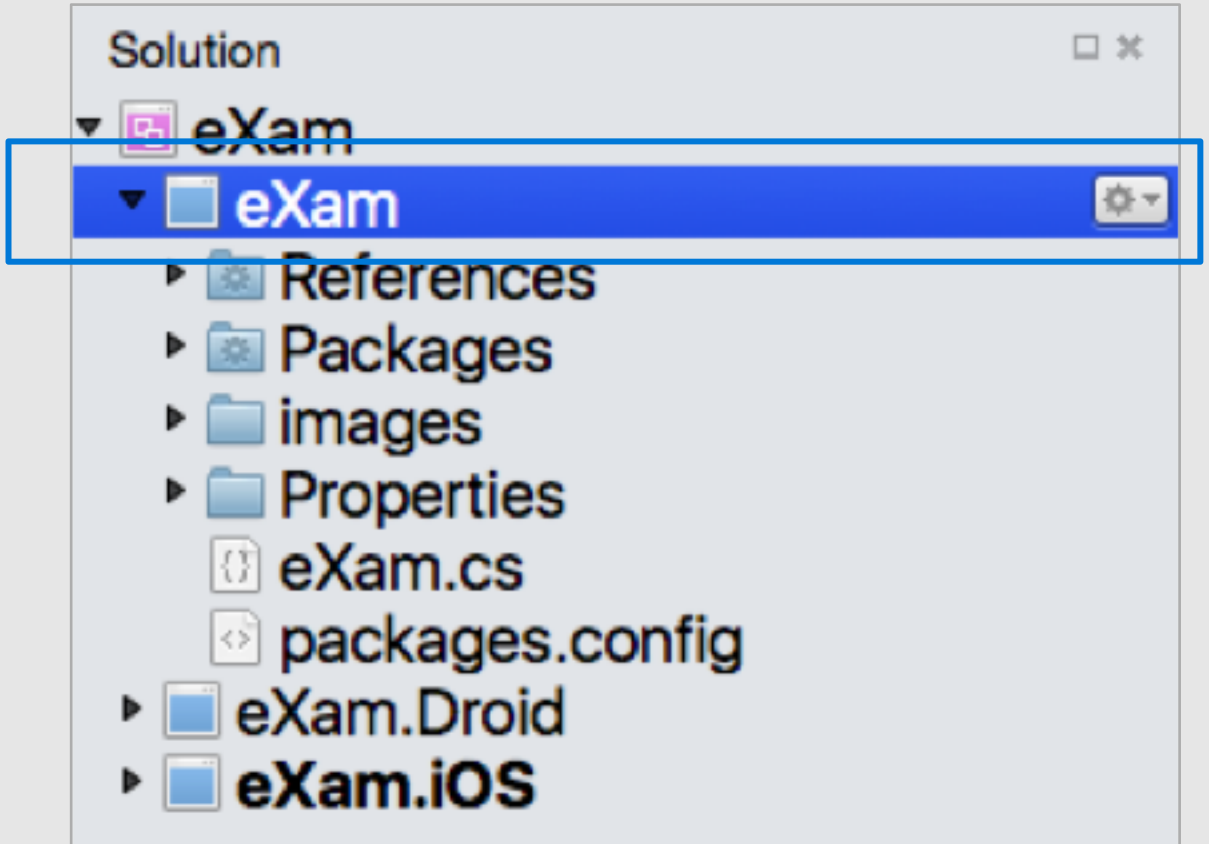
Forms **`ContentPage`** XAML
Entire screen of content



Forms **`ContentView`** XAML
Composite control (smaller than a page)

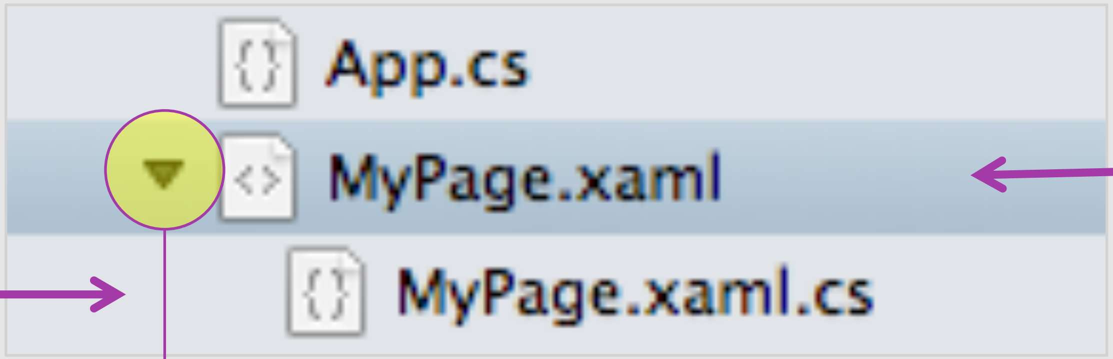
Where do the XAML pages go?

You should always add the XAML content to the *platform-independent* part of your application – this is **shared UI and code** for all your target platforms



What gets created?

XAML pages have two related files which work together to define the class



XAML file (UI)

C# file (behavior)

Disclosure arrow *collapses* the C# file and indicates these files go together

Creating a page

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

XML based: case sensitive, open tags must be closed, etc.

Attributes set properties or events

Element tags create objects

Child nodes used to establish relationship

```
<?xml version="1.0" encoding="UTF-8" ?>
<ContentPage ...>
   <StackLayout Padding="20" Spacing="10">
      <Label Text="Enter a Phoneword:" />
      <Entry Placeholder="Number" />
      <Button Text="Translate" />
      <Button Text="Call" IsEnabled="False" />
   </StackLayout>
</ContentPage>
```

XAML + Code Behind

XAML and code behind files are tied together

```
<?xml version="1.0" encoding="UTF-8" ?>
<ContentPage x:Class="Phoneword.MainPage" ...>
```

```
namespace Phoneword
{
   public partial class MainPage : ContentPage
   {
      ...
   }
}
```

`x:Class` Identifies the full name of the class defined in the code behind file

XAML initialization

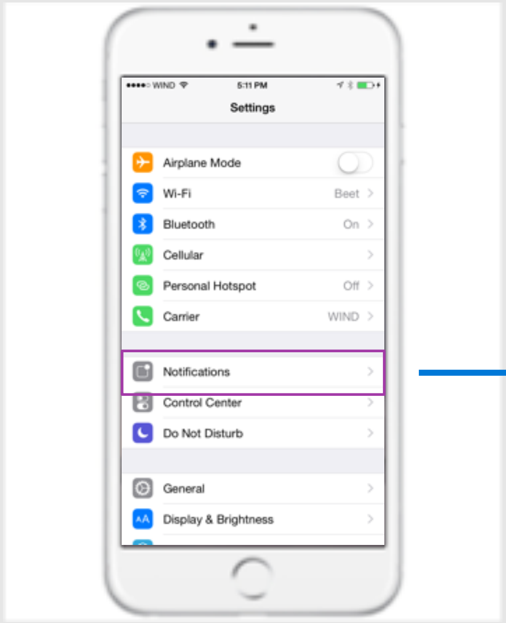
Code behind constructor has call to **`InitializeComponent`** which is responsible for loading the XAML and creating the objects

```
public partial class MainPage : ContentPage
{
    public MainPage ()
    {
        InitializeComponent ();
    }
}
```

implementation of method generated by XAML compiler as a result of the **`x:Class`** tag – added to hidden file (same partial class)

Working with multiple pages

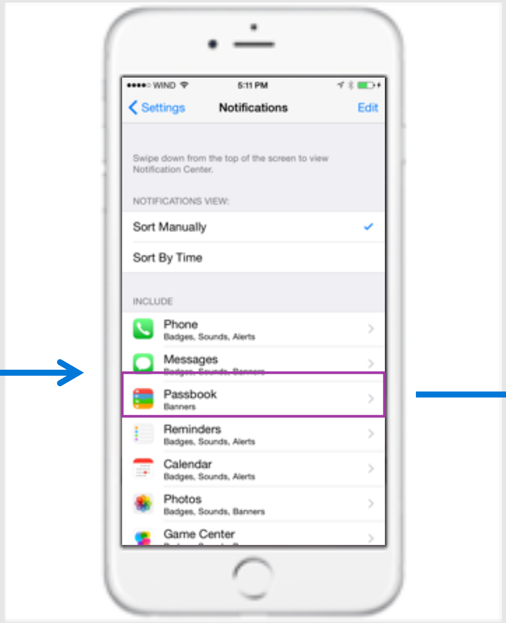
Stack navigation is a common paradigm used in mobile apps to display hierarchies of related information



Settings

- Airplane Mode
- Wi-Fi Beet
- Bluetooth On
- Cellular
- Personal Hotspot Off
- Carrier WIND
- Notifications
- Control Center
- Do Not Disturb
- General
- Display & Brightness

Settings



Settings Notifications Edit

Swipe down from the top of the screen to view Notification Center.

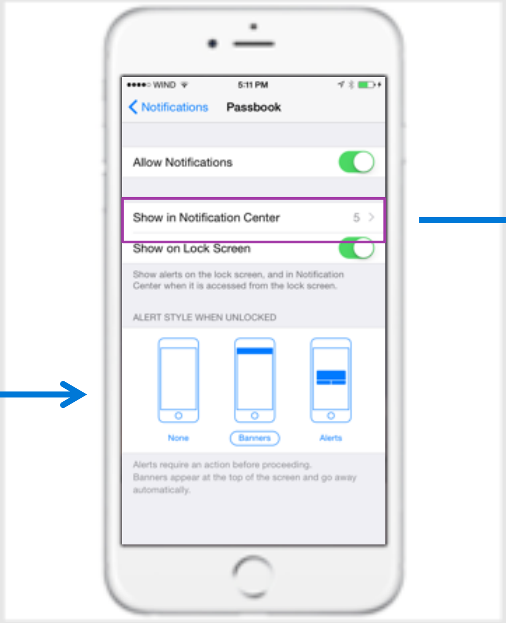
NOTIFICATIONS VIEW:

- Sort Manually ✓
- Sort By Time

INCLUDE

- Phone — Badges, Sounds, Alerts
- Messages — Badges, Sounds, Banners
- Passbook — Banners
- Reminders — Badges, Sounds, Alerts
- Calendar — Badges, Sounds, Alerts
- Photos — Badges, Sounds, Banners
- Game Center

Notifications



Notifications Passbook

- Allow Notifications
- Show in Notification Center 5
- Show on Lock Screen

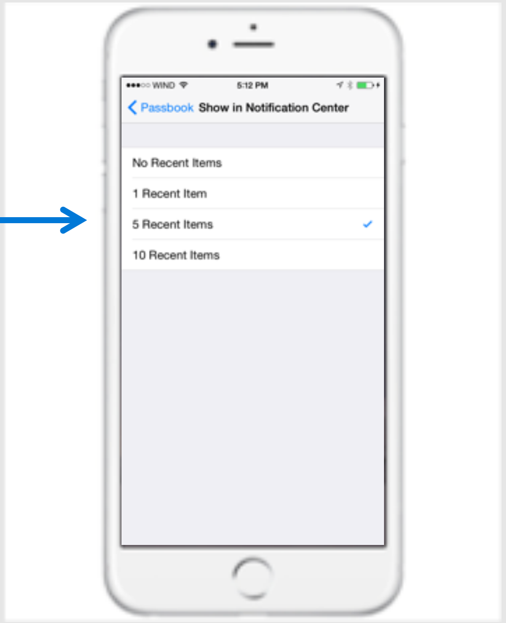
Show alerts on the lock screen, and in Notification Center when it is accessed from the lock screen.

ALERT STYLE WHEN UNLOCKED

None Banners Alerts

Alerts require an action before proceeding. Banners appear at the top of the screen and go away automatically.

Passbook



Passbook Show in Notification Center

- No Recent Items
- 1 Recent Item
- 5 Recent Items ✓
- 10 Recent Items

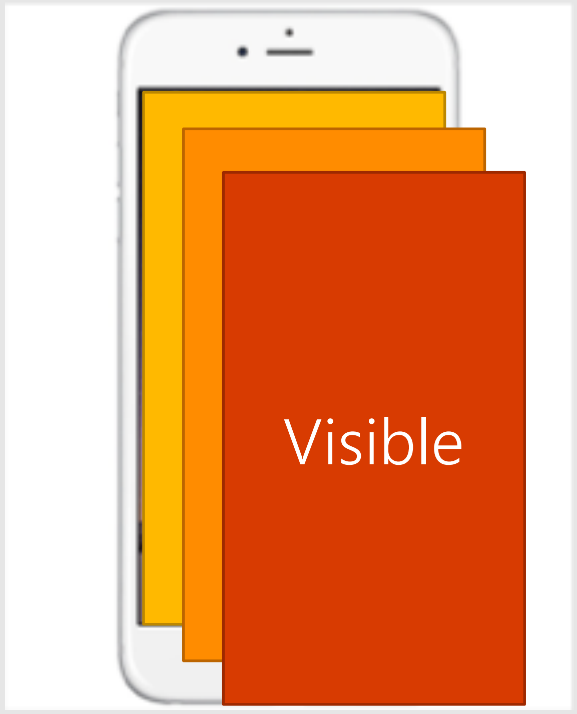
Notification Center

Stack Navigation

When a new page is *pushed* onto the stack, it becomes visible and hides the previous page

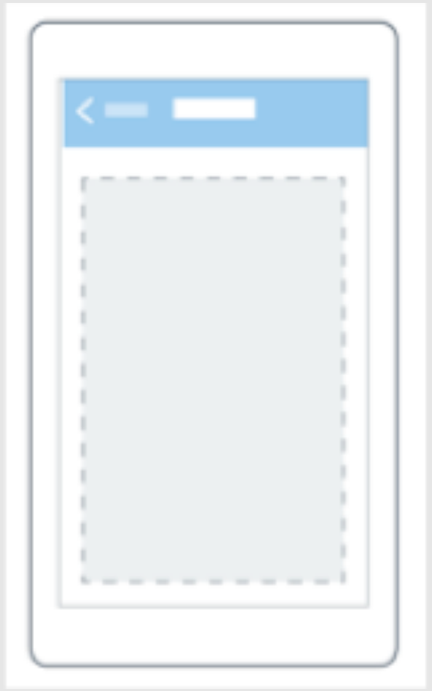
Only one page is ever visible at a time (the last one added)

Great for displaying multi-level relationships because it allows "drilling" into details



NavigationPage

Xamarin.Forms implements this through a special decorator page – **`NavigationPage`** which implements a navigation API to manipulate the current page



Using NavigationPage

NavigationPage should be the **MainPage** in your app class

```csharp
public class App : Application
{
    public App()
    {
        // The root page of your application
        MainPage = new NavigationPage(new LemonPage()) {
             BarBackgroundColor = Color.Yellow,
             BarTextColor = Color.Black
        };
    }
}
```

Using NavigationPage

`NavigationPage` should always be the **`MainPage`** in your app class

```
public class App : Application
{
    public App()
    {
        // The root page of your application
        MainPage = new NavigationPage(new LemonPage()) {
             BarBackgroundColor = Color.Yellow,
             BarTextColor = Color.Black
        };
    }
}
```

Identify the root content page in your hierarchy through the constructor

Using NavigationPage

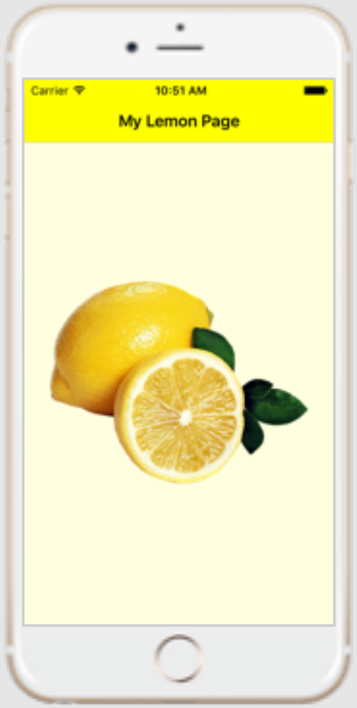
`NavigationPage` should always be the **`MainPage`** in your app class

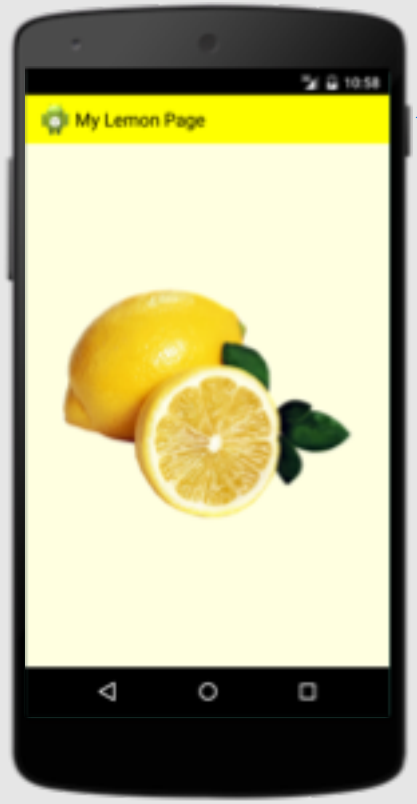
```
public class App : Application
{
    public App()
    {
        // The root page of your application
        MainPage = new NavigationPage(new LemonPage()) {
              BarBackgroundColor = Color.Yellow,
              BarTextColor = Color.Black
        };
    }
}
```

Can customize the colors used in the UI through properties

What does NavigationPage do?

`NavigationPage` activates a *navigation bar* in the UI – this is what will allow the user to interactively move backward in the stack





Bar is displayed using native platform's paradigm, and allows customization such as page title to be displayed

Pushing pages onto the navigation stack

Each **`Page`** has a **`Navigation`** property which exposes the navigation API, can use this to manipulate the navigation stack

```
public class LemonPage : ContentPage
{
    async void OnShowAboutPage(object sender, EventArgs e)
    {
        await this.Navigation.PushAsync(new AboutPage(), true);
    }
}
```

Can indicate whether you want platform-specific animations to be used as the transition between the pages (**default = true**)

Returning to the previous page

Back button allows user to return to the previous screen, but can also perform this operation through the navigation API (with optional animation)

```
async void GoBackOnePage()
{
    await this.Navigation.PopAsync();
}

async void BackToMainMenu()
{
    await this.Navigation.PopToRootAsync();
}
```

Page notifications

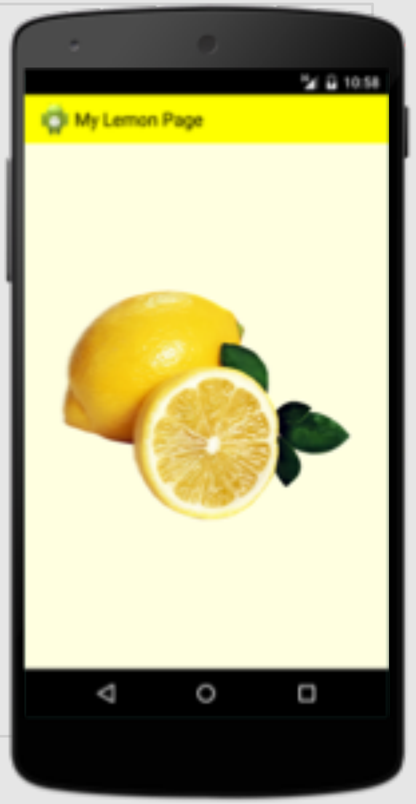
Can override virtual **Page** methods to be notified about transitions and provide additional logic (e.g. refresh page information, persist state, etc.)

```
protected override void OnAppearing() {
    base.OnAppearing();
    // Logic to refresh page
}

protected override void OnDisappearing() {
    base.OnDisappearing();
    // Logic to persist state
}
```

Customizing a page's view

Can use properties on both the navigation page and the current page to customize the UI displayed for the active screen

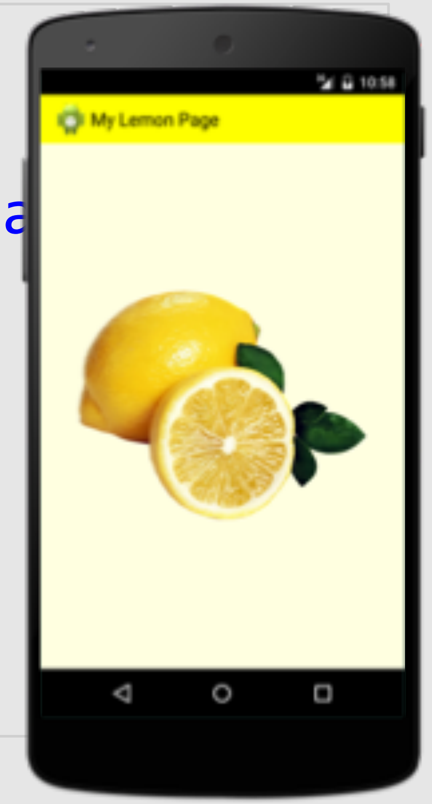
```
public LemonPage() {
   ...
   Title = "My Lemon Page";
   Icon = "DroidIcon.png";

   NavigationPage.SetBackButtonTitle(this, "Go Back");
   NavigationPage.SetHasBackButton(this, true);
   NavigationPage.SetHasNavigationBar(this, true);
}
```



Customizing a page's view

Can use properties on both the navigation page and the current page to customize the UI displayed for the active screen

```
<ContentPage x:Class="FruityApp.LemonPage"
    xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xa
    Title="My Lemon Page"
    Icon="LemonIcon.png"
    NavigationPage.BackButtonTitle="Go Back"
    NavigationPage.HasBackButton="True"
    NavigationPage.HasNavigationBar="True">
...
```



Exercise #4

Set up stack navigation

Property Conversions

```
<Label
   Text="Hello Forms!"
   Rotation="45.75"
   VerticalOptions="Center"
   FontAttributes="Bold"
   FontSize="36"
   TextColor="Red" />
```

- XML attributes only allow for **string values** – works fine for intrinsic types
- **Enum**s are matched by name, use comma separators to combine flags
- XAML invokes *type converters* to convert string to proper type

Built-in Type Converters

`FontAttributes`, **`ImageSource`**, **`Color`** and **`Thickness`** all have built-in type converters to make them easy to set in XAML

```
<Label …
   FontAttributes="Bold,Italic"
   FontSize="Large"
   TextColor="#fffc0d34" />
```

```
<StackLayout …
     Padding="5,20,5,0" />
```

Colors can be specified as a known value (e.g. **`"Red"`**, **`"Green"`**, …) or as a hex value (RGB or aRGB)

`Thickness` is specified as a single number, two numbers, or four numbers (L,T,R,B)

Setting Complex Properties

When a more complex object needs to be created and assigned, you can use the *Property Element* syntax

This changes the style to use an element tag (create-an-object) as part of the assignment

```
<BoxView Color="Transparent">
   <BoxView.GestureRecognizers>
      <TapGestureRecognizer
        NumberOfTapsRequired="2"
        ... />
   </BoxView.GestureRecognizers>
</BoxView>
```

Property value is set as a child tag of the `<Type.PropertyName>` element

Setting Attached Properties

```
<Grid>
 <Label Text="Position" />
 <Entry Grid.Column="1" />
</Grid>
```

Set in XAML with **`OwnerType.Property="Value"`** form, can also use property-element syntax for more complex values

Attached Properties provide runtime "attached" data for a visual element, this is primarily used by layout containers to provide container-specific values on each child

Recall that in code behind we used static methods on the layout containers to associate these values

Content Properties

Some types have a *default* property which is set when child content is added to the element

This is the *Content Property* and is identified through a `[ContentAttribute]` applied to the class

```
<ContentPage ...>
   <Label>
      This is the Text
   </Label>
</ContentPage>
```

These create the same UI

```
<ContentPage ...>
 <ContentPage.Content>
   <Label>
     <Label.Text>
       This is the Text
     </Label.Text>
   </Label>
 </ContentPage.Content>
</ContentPage>
```

Exercise #5

Add a XAML page

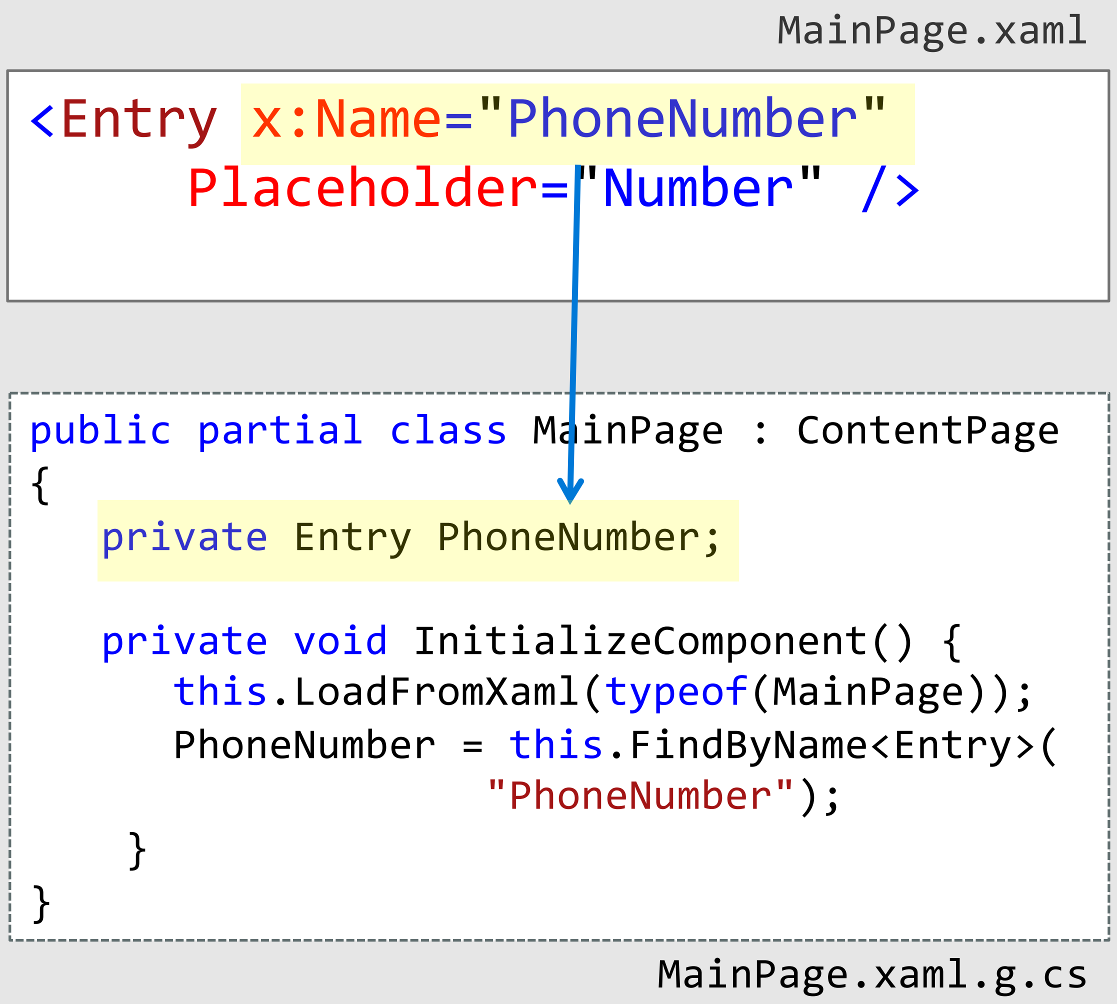
Naming Elements in XAML

Use `x:Name` to assign field name

- allows you to reference element in XAML and code behind

Adds a private field to the XAML-generated partial class (.g.cs)

Name must conform to C# naming conventions and be unique in the file



MainPage.xaml

```
<Entry x:Name="PhoneNumber"
      Placeholder="Number" />
```

```
public partial class MainPage : ContentPage
{
   private Entry PhoneNumber;

   private void InitializeComponent() {
      this.LoadFromXaml(typeof(MainPage));
      PhoneNumber = this.FindByName<Entry>(
                        "PhoneNumber");
    }
}
```

MainPage.xaml.g.cs

Working with named elements

Can work with named elements as if you defined them in code, but keep in mind the field is not set until *after* **`InitializeComponent`** is called

Can wire up events, set properties, even add new elements to layout

```
public partial class MainPage : ContentPage
{
    public MainPage () {
        InitializeComponent ();
        PhoneNumber.TextChanged += OnTextChanged;
    }

    void OnTextChanged(object sender, TextChangedEventArgs e) {
       ...
    }
}
```

XAML resources

By default, your XAML files are included as a plain-text resource in the generated assembly which is **parsed at runtime** to generate the page

```
private void InitializeComponent()
{
    this.LoadFromXaml(typeof(MainPage));
}
```

This **Page** method looks up the embedded resource by name, parses it, and creates each object found; it returns the **root created object**

Compiling XAML

XAML can be optionally compiled to intermediate language (IL)

- Provides compile-time validation of your XAML files
- Reduces the load time for pages
- Reduces the assembly size by removing text-based **.xaml** files

Enabling XAMLC

XAMLC (the XAML compiler) is disabled by default to ensure backwards compatibility; can be enabled through a .NET attribute

```
using Xamarin.Forms.Xaml;

[assembly: XamlCompilationAttribute(
                  XamlCompilationOptions.Compile)]
```

Can enable the compiler for all XAML files in the assembly

Enabling XAMLC

XAMLC (the XAML compiler) is disabled by default to ensure backwards compatibility; can be enabled through .NET attribute

```
using Xamarin.Forms.Xaml;

[XamlCompilationAttribute(XamlCompilationOptions.Compile)]
public partial class MainPage : ContentPage {
```

... or on a specific XAML-based class

Disabling XAMLC

Attribute also lets you disable XAMLC for a specific class

```
using Xamarin.Forms.Xaml;

[XamlCompilationAttribute(XamlCompilationOptions.Skip)]
public partial class DetailsPage : ContentPage {
```

Specify **Skip** to turn off compiler for this specific page; goes back to using `LoadFromXaml`

Exercise #6

Turn on XAMLC

Handling events in XAML

Can also wire up events in XAML – event handler *must be defined* in the code behind file and have **proper signature** or it's a runtime failure

```
<Entry Placeholder="Number" TextChanged="OnTextChanged" />
```

```
public partial class MainPage : ContentPage
{
    ...
    void OnTextChanged(object sender, TextChangedEventArgs e) {
        ...
    }
}
```

Handling events in code behind

Many developers prefer to wire up all events in code behind by naming the XAML elements and adding event handlers in code

- Keeps the UI layer "pure" by pushing all behavior + management into the code behind
- Names are validated at compile time, but event handlers are not
- Easier to see how logic is wired up

Pick the approach that works for your team / preference

Exercise #7

Navigate on button tap

Summary

- XAML is a markup language used to describe pages in Xamarin.Forms
- Based on Microsoft's XAML 2009 specification – syntax is identical
- Can name elements to provide access to markup elements in code
- Can wire up events to provide runtime behavior