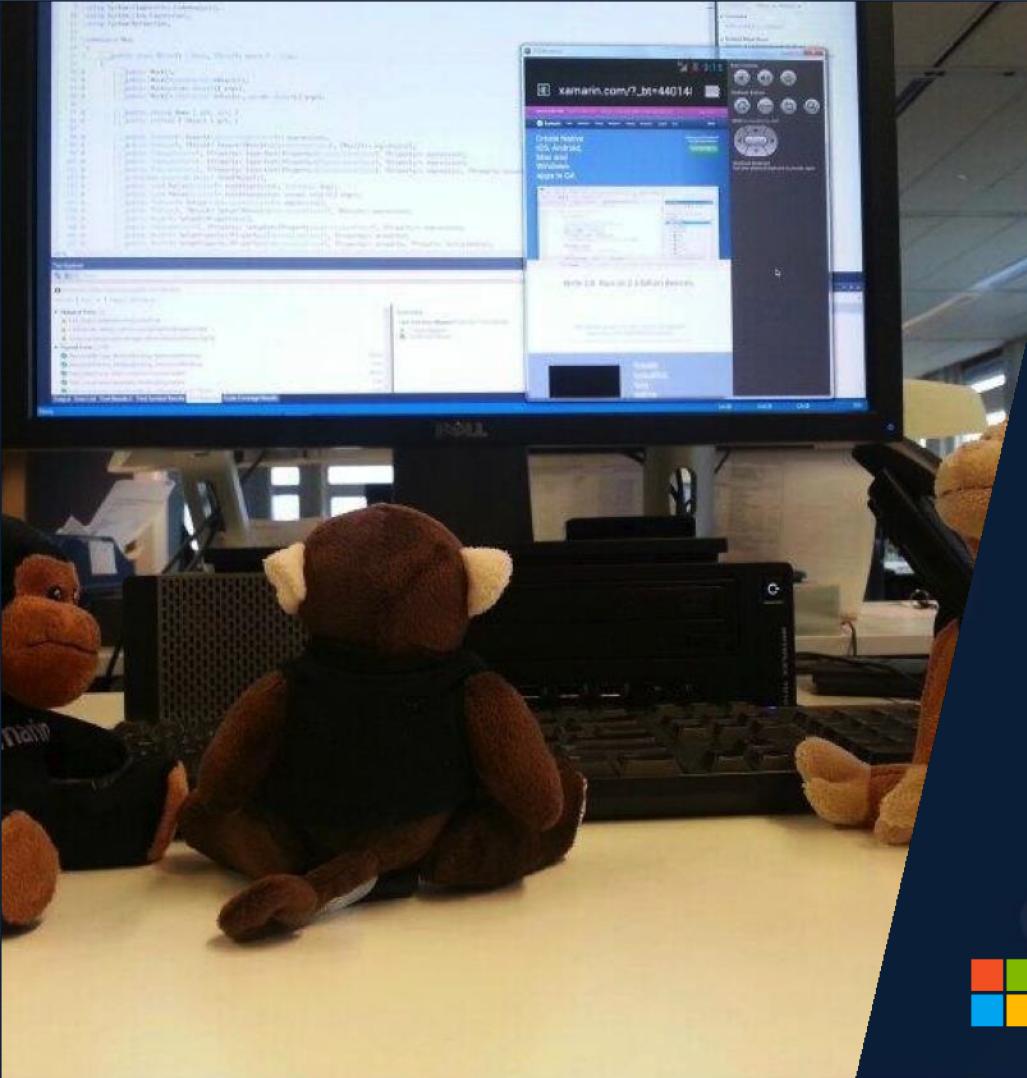


XAM130

# XAML in Xamarin.Forms



Xamarin University

# Objectives

1. Examine XAML syntax
2. Add Behavior to XAML-based pages
3. Explore XAML capabilities





# Examine XAML syntax

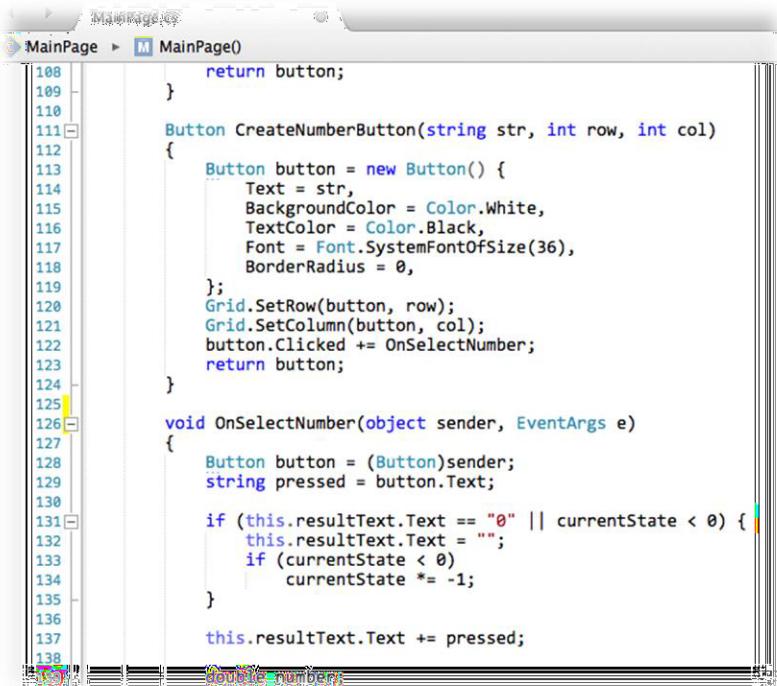
# Tasks

- ❖ Choose between XAML and C# to define your UI
- ❖ Define a UI in Xamarin.Forms using XAML



# Motivation

- ❖ Creating UI in code has some disadvantages
  - Significant portion of code-behind is UI setup and layout
  - Mixing UI and behavior in one file makes design and behavior harder to understand / evolve
  - Prohibits use of a UI designer because a developer is needed for any UI change



```
 MainPage.xaml.cs
108     return button;
109 }
110 }
111 }
112 }
113 Button CreateNumberButton(string str, int row, int col)
114 {
115     Button button = new Button()
116     {
117         Text = str,
118         BackgroundColor = Color.White,
119         TextColor = Color.Black,
120         Font = Font.SystemFontOfSize(36),
121         BorderRadius = 0,
122     };
123     Grid.SetRow(button, row);
124     Grid.SetColumn(button, col);
125     button.Clicked += OnSelectNumber;
126     return button;
127 }
128 void OnSelectNumber(object sender, EventArgs e)
129 {
130     Button button = (Button)sender;
131     string pressed = button.Text;
132
133     if (this.resultText.Text == "0" || currentState < 0) {
134         this.resultText.Text = "";
135         if (currentState < 0)
136             currentState *= -1;
137
138         this.resultText.Text += pressed;
139     }
140 }
```

# Advantages of markup

- ❖ HTML has taught us that markup languages are a great way to define user interfaces because they are:
  - Toolable
  - Human readable
  - Extensible



```
DOCTYPE html PUBLIC
  "xml" xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta name="TITLE" content="My First Web Page" />
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <meta name="keywords" content="HTML, CSS, JavaScript" />
  <meta name="description" content="A simple web page with a title, some text, and a link." />
  <title>My First Web Page</title>
</head>
<body>
  <h1>Hello, World!</h1>
  <p>This is my first web page.</p>
  <a href="http://www.example.com">Example Link</a>
</body>

```

# What is XAML?

- ❖ Extensible Application Markup Language (XAML) is a markup language created by Microsoft specifically to describe UI

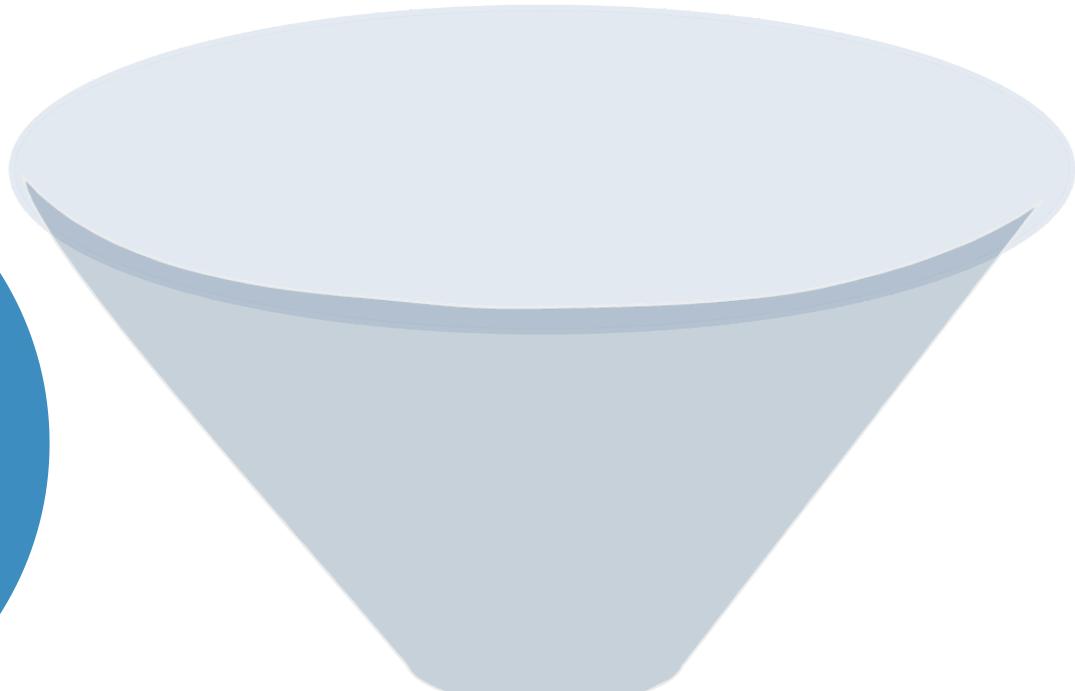
XAML

Xamarin Forms + XAML  
= Sweetness!

# XAML benefits



Separation of UI  
From Behavior



XAML

# Microsoft XAML vs. Xamarin.Forms

- ❖ Xamarin.Forms conforms to the XAML 2009 specification; it differs from traditional Microsoft XAML mainly in the controls and layout containers

```
<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 (WPF)

```
<?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>

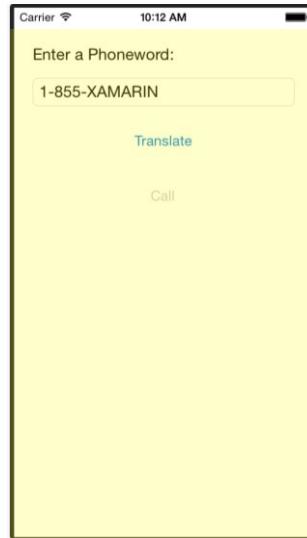
</ContentPage>
```

Xamarin.Forms

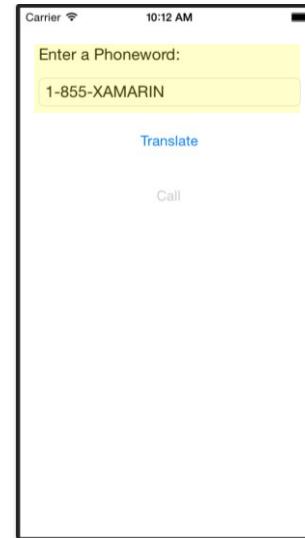
Feature	Supported in Xamarin.Forms
XAML 2009 compliance	✓
Shapes (Rectangle, Ellipse, Path, etc.)	BoxView
Resources, Styles and Triggers	✓
Data binding	✓ *not all features
Data templates	✓
Control templates	Custom renderers
Render Transforms	✓
Animations	Code-only
Custom XAML behaviors	✓
Custom markup extensions	✓
Value converters	✓

# Adding a XAML Page

- ❖ There are two Item Templates available to add XAML content



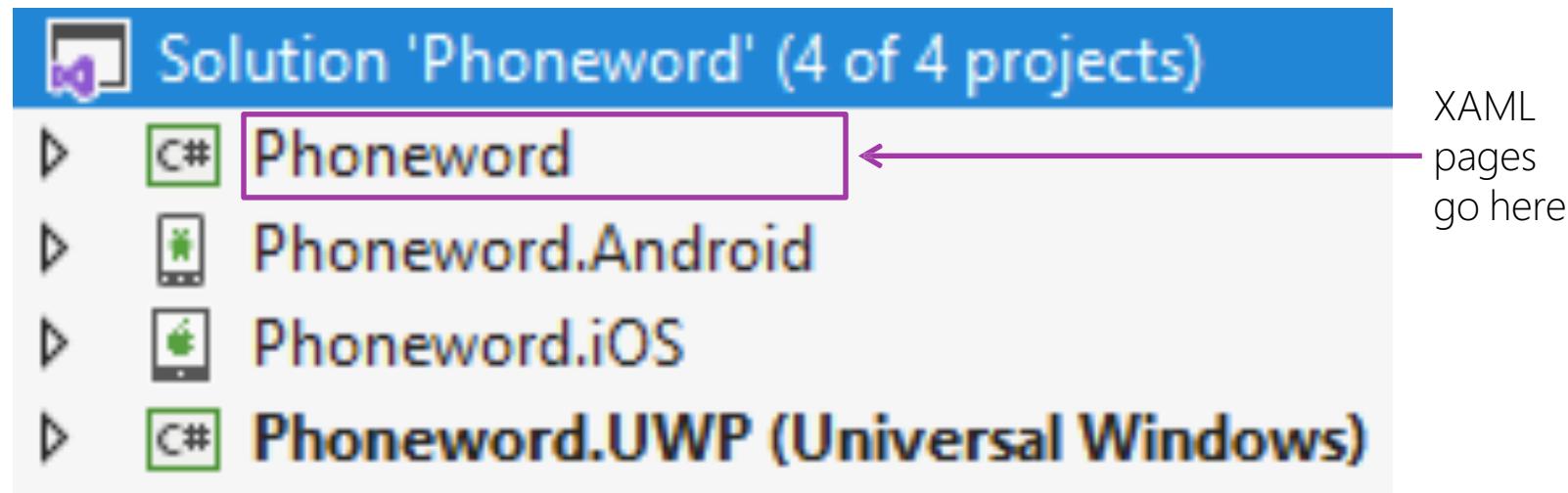
**ContentPage** is an entire screen of content



**ContentView** is a composite control (smaller than a page)

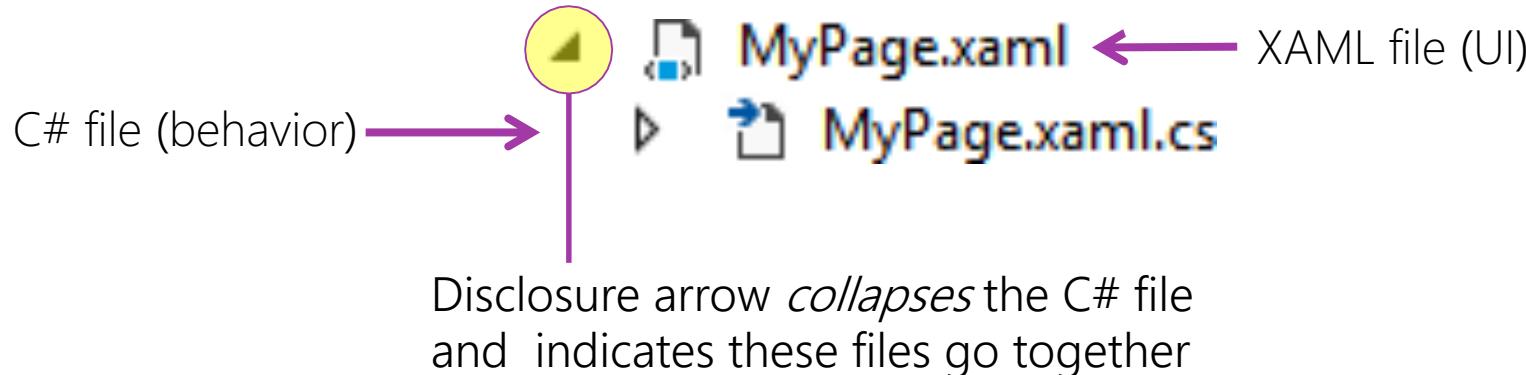
# Where do the XAML pages go?

- ❖ Add XAML content to the platform-independent project in your application – this is shared UI and code for all your target platforms



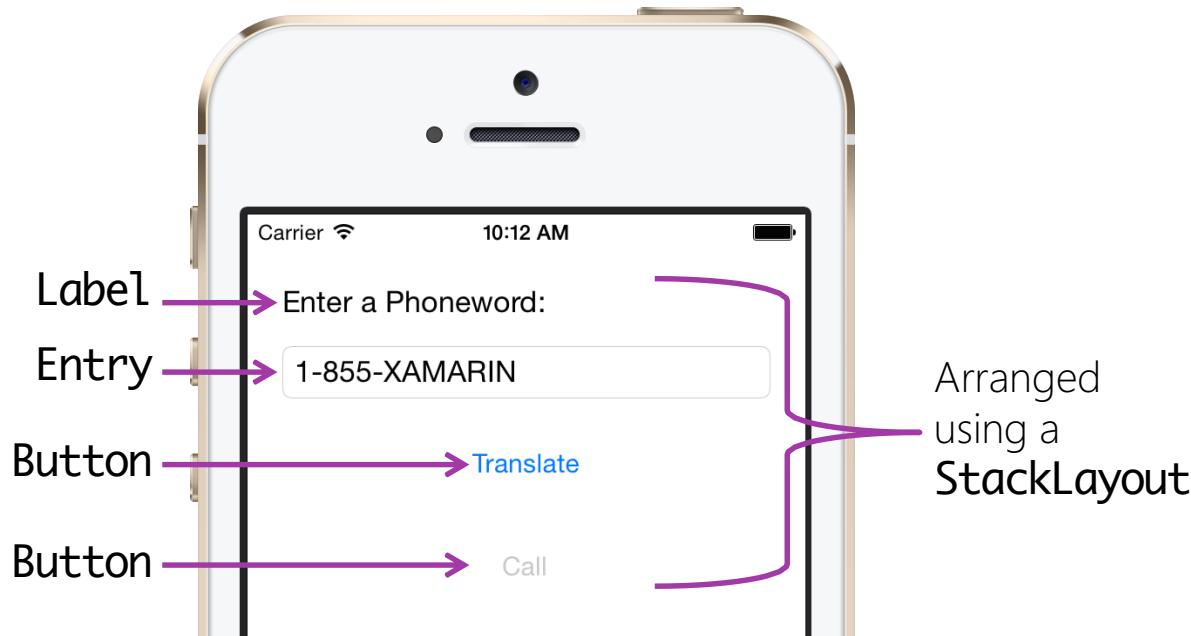
# XAML-page structure

- ❖ XAML pages have two related files that work together to define the class



# Example: creating a XAML UI

- ❖ Our goal is to build the UI for a “Phoneword” app that translates a text phone number to its numeric equivalent



# Describing a screen in XAML

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

```
<?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>
```

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

# Describing a screen in XAML

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

Element tags  
create objects



```
<?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>
```

# Describing a screen in XAML

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

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

Attributes set  
properties or  
events

# Describing a screen in XAML

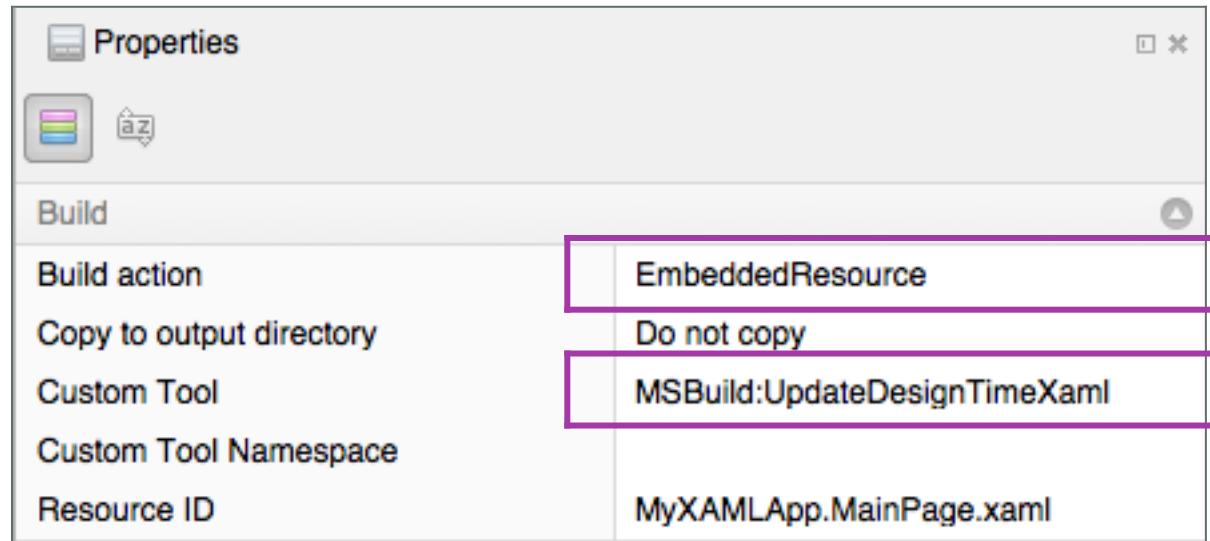
- ❖ XAML is used to construct object graphs, in this case a visual **Page**

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 build type

- ❖ XAML files are stored as *embedded resources* and have a special build type of `MSBuild:UpdateDesignTimeXaml`



# 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

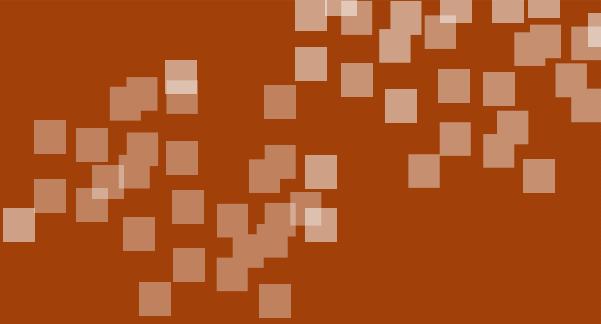
# XAML initialization

- ❖ 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)



# Demonstration

Creating a XAML-based application

# Property Conversions

- ❖ XML attributes only allow for string values – works fine for intrinsic types

```
<Label Text="This is a Label" IsVisible="True" Opacity="0.75"  
      FontAttributes="Bold,Italic" FontSize="Large"  
      Margin="5,20,5,0" TextColor="#ffffc0d34" />
```

**Text** is a **string** which is just set directly

# Property Conversions

- ❖ XML attributes only allow for string values – works fine for intrinsic types

```
<Label Text="This is a Label" IsVisible="True" Opacity="0.75"  
      FontAttributes="Bold,Italic" FontSize="Large"  
      Margin="5,20,5,0" TextColor="#ffffc0d34" />
```

IsVisible is a **bool** which is converted from  
the value using **Boolean.TryParse**

# Property Conversions

- ❖ XML attributes only allow for string values – works fine for intrinsic types

```
<Label Text="This is a Label" IsVisible="True" Opacity="0.75"  
      FontAttributes="Bold,Italic" FontSize="Large"  
      Margin="5,20,5,0" TextColor="#ffffc0d34" />
```



**Opacity** is a **double** which is converted from the value using **Double.TryParse**

# Property Conversions

- ❖ XML attributes only allow for string values – works fine for intrinsic types

```
<Label Text="This is a Label" IsVisible="True" Opacity="0.75"  
      FontAttributes="Bold,Italic" FontSize="Large"  
      Margin="5,20,5,0" TextColor="#ffffc0d34" />
```



Enumerations are parsed with `Enum.TryParse` and support `[Flags]` with comma-separated values

# Property Conversions

- ❖ XML attributes only allow for string values – works fine for intrinsic types

```
<Label Text="This is a Label" IsVisible="True" Opacity="0.75"  
      FontAttributes="Bold,Italic" FontSize="Large"  
      Margin="5,20,5,0" TextColor="#ffffc0d34" />
```

```
[TypeConverter(typeof(ThicknessTypeConverter))]  
public struct Thickness  
{  
    ...  
}
```

# Property Conversions

- ❖ XML attributes only allow for string values – works fine for intrinsic types

```
<Label Text="This is a Label" IsVisible="True" Opacity="0.75"  
      FontAttributes="Bold,Italic" FontSize="Large"  
      Margin="5,20,5,0" TextColor="#ffffc0d34" />
```



Margin is of type Thickness

# Property Conversions

- ❖ XML attributes only allow for string values – works fine for intrinsic types

```
<Label Text="This is a Label" IsVisible="True" Opacity="0.75"  
      FontAttributes="Bold,Italic" FontSize="Large"  
      Margin="5,20,5,0" TextColor="#ffffc0d34" />
```



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

# 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

- ❖ Attached Properties provide runtime "attached" data for a visual element
- ❖ Used by layout containers to provide container-specific values on each child

```
<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



# 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>
```

# Identifying Types

- ❖ XAML creates objects when it encounters an element tag, XML namespaces are used to correlate .NET types to tags

Default namespace includes most of the Xamarin.Forms types you use

```
<ContentPage ...  
    xmlns="http://xamarin.com/schemas/2014/forms"  
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">
```

```
    <StackLayout ... />
```

```
    <ContentPage>
```



x: namespace includes XAML types and known CLR types (**Int32**, **String**, etc.)

# Custom Types

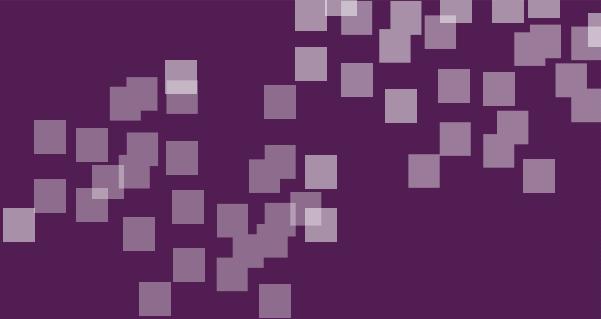
- ❖ XAML can create any public object, including ones with parameterized constructors – you just need to tell it where the type lives

Must supply the namespace, and *possibly* the assembly, the type is defined in

```
<scg>List x:TypeArguments="x:String"  
    xmlns:scg="clr-namespace:System.Collections.Generic;assembly=mscorlib">  
    <x:String>One</x:String>  
    <x:String>Two</x:String>  
    <x:String>Three</x:String>  
</scg>List>
```

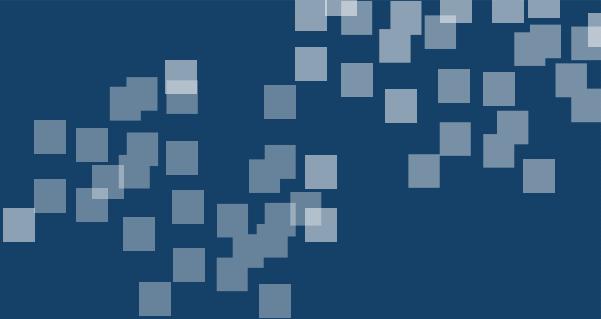


**xmlns** definition can be placed on a single element, or a parent element to use with any children



# Individual Exercise

Create a XAML-based version of Calculator



# Add Behavior to XAML-based pages

# Tasks

- ❖ Access XAML defined elements in the associated code-behind
- ❖ Handle events on XAML defined views



# 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" />
```

MainPage.xaml.g.cs

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

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

# 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) {
        ...
    }
}
```

# Sharing elements

- ❖ Generated field is always private, but **Page** owner can wrap in a public property to allow external access

```
public partial class MainPage : ContentPage
{
    public Entry PhoneNumberEntry
    {
        get { return this.PhoneNumber; }
    }
    ...
}
```

should *not* provide a setter – replacing the field's value will not change the actual element on the screen

# Handling events in XAML

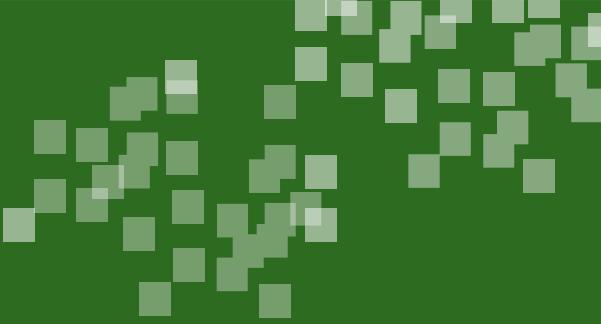
- ❖ Can also wire up events in XAML – event handler *must be defined* in the code behind file and have proper signature or it's a runtime failure

```
<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



# Flash Quiz

# Flash Quiz

- ① Putting an **x:Name** tag onto an element \_\_\_\_\_. (Select all that apply)
- a) Creates a private field in the associated code behind file
  - b) Creates a protected field in the associated code behind file
  - c) Makes the element accessible to other things in XAML
  - d) Makes the element accessible in the code behind after **InitializeComponent** returns

# Flash Quiz

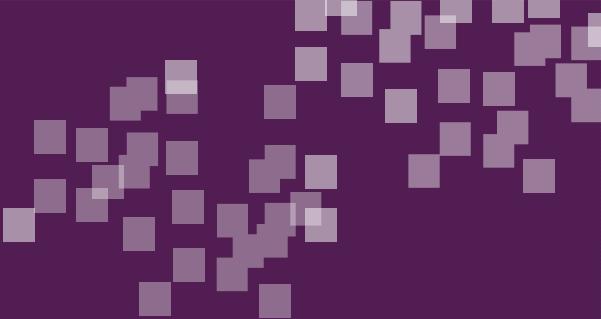
- ① Putting an **x:Name** tag onto an element \_\_\_\_\_. (Select all that apply)
- a) Creates a private field in the associated code behind file
  - b) Creates a protected field in the associated code behind file
  - c) Makes the element accessible to other things in XAML
  - d) Makes the element accessible in the code behind after **InitializeComponent** returns

# Flash Quiz

- ② Event Handlers in code behind that are wired up in XAML must be public
  - a) True
  - b) False

# Flash Quiz

- ② Event Handlers in code behind that are wired up in XAML must be public
  - a) True
  - b) False



# Individual Exercise

Adding Behavior to XAML Calculator



# Explore XAML capabilities



# Tasks

- ❖ Using device-specific values to define your app's UI
- ❖ Use Markup Extensions in XAML
- ❖ Using **ContentView** to share XAML across multiple Pages
- ❖ Compile XAML to improve performance



# Using device-specific values

- ❖ XAML is a static (compile-time) definition of the UI; can provide different values for each platform just like we do in code with `Device.OnPlatform`

`x:TypeArguments` used for generic instantiation

```
<StackLayout Spacing="10">
    <StackLayout.Padding>
        <OnPlatform x:TypeArguments="Thickness"
                    iOS="0,20,0,0" Android="0" WinPhone="0" />
    </StackLayout.Padding>
    ...
</StackLayout>
```

can then supply different platform-specific value for property

# Using runtime values

- ❖ XAML defines a way to set properties to values known at runtime called *markup extensions*, these conform to the **IMarkupExtension** interface

```
public interface IMarkupExtension  
{  
    object ProvideValue(IServiceProvider serviceProvider);  
}
```



method is called during the XAML load process to retrieve a runtime value and apply it to the property

# Using Markup Extensions

- ❖ Markup Extensions are identified by "{extension\_here}" curly braces

parser expects to find a class named **BindingExtension** that implements **IMarkupExtension** when it encounters the curly brace as the first character

```
<StackLayout BindingContext="{Binding Details}">
    <Label Text="{}{Want a Curly Brace Here!}" />
    ...
</StackLayout>
```

literal curly braces need to be escaped properly to avoid a parser error

# Reading static properties

- ❖ A very useful markup extension is **x:Static** which lets you get the value of public static fields or properties

```
public static class Constants
{
    public static string Title = "Hello, Forms";
    public static Thickness Padding = new Thickness(5, Device.OnPlatform(20, 0, 0), 5, 0);
    public static Color TextColor = Color.Yellow;
}
```

```
<ContentPage ... Padding="{x:Static me:Constants.Padding}">
    <Label Text="{x:Static me:Constants.Title}"
        TextColor="{x:Static me:Constants.TextColor}" />
</ContentPage>
```

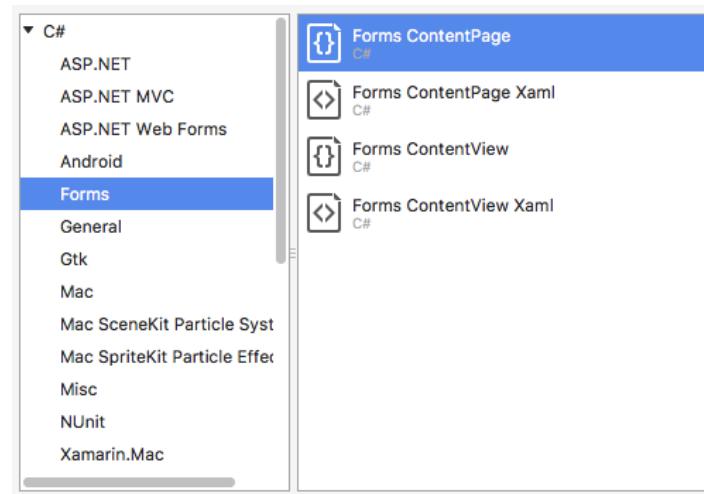
# Other built-in Markup Extensions

- ❖ Use resource values with `{StaticResource}` and `{DynamicResource}`
- ❖ Supply a `null` value with `{x:Null}`
- ❖ Lookup a `Type` with `{x:Type}`
- ❖ Create an array with `{x:Array}`
- ❖ Create data bindings with `{Binding}`

```
<ListView SelectedItem="{x:Null}">
<ListView.ItemsSource>
  <x:Array Type="{x:Type x:Int32}">
    <x:Int32>10</x:Int32>
    <x:Int32>20</x:Int32>
    <x:Int32>30</x:Int32>
  </x:Array>
</ListView.ItemsSource>
</ListView>
```

# Sharing XAML fragments

- ❖ Can be useful to split XAML into different files
  - Reuse useful UI pieces
  - Refactor large pages
  
- ❖ **ContentView** allows for this
  - Similar to Android Fragments
  - ... or User Controls in Windows



# ContentView structure

- ❖ ContentView combines a piece of XAML with code behind behavior - just like **ContentPage**, can name elements, wire up events, etc.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <ContentView xmlns="http://xamarin.com/schemas/2014/forms"
3    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
4    x:Class="Phoneword.PhoneView">
5
6    <!-- Content goes here -->
7
8  </ContentView>
```

```
1  using Xamarin.Forms;
2
3  namespace Phoneword
4  {
5      public partial class PhoneView : ContentView
6      {
7          public PhoneView()
8          {
9              InitializeComponent();
10         }
11     }
12 }
```

Can be placed into a separate class library if desired

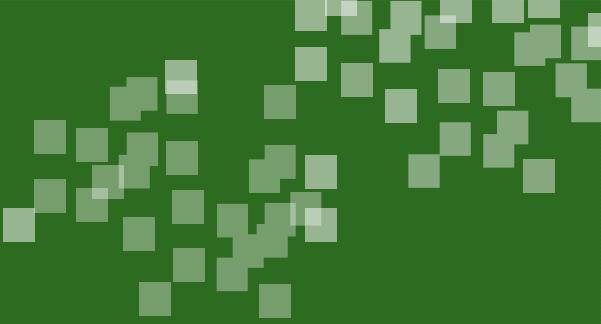
# Using a ContentView

- ❖ **ContentView** is not displayed on it's own - must be added to a **Page**

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
3      xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
4      xmlns:local="clr-namespace:Phoneword;assembly=Phoneword"
5      x:Class="TestApp.MainPage">
6
7      <local:PhoneView PhoneNumber="1-800-XAMARIN"
8          PhoneNumberChanged="OnPhoneNumberChanged" />
9
10 </ContentPage>
```



**ContentView** can expose it's own properties and events to provide customization or "hooks" into the logic



# Flash Quiz

# Flash Quiz

- ① To specify a platform-specific value in XAML you use \_\_\_\_\_.  
a) Device<T>  
b) OnPlatform<T>  
c) Platform<T>  
d) x:Platform<T>

# Flash Quiz

- ① To specify a platform-specific value in XAML you use \_\_\_\_\_.
- a) Device<T>
  - b) OnPlatform<T>
  - c) Platform<T>
  - d) x:Platform<T>

# Flash Quiz

- ② To share a value you can use \_\_\_\_\_ (select all that apply).
- a) Resource Dictionary with {StaticResource}
  - b) Resource Dictionary with {x:Static}
  - c) Static properties in code and {x:Static}
  - d) Static properties in code and {StaticResource}

# Flash Quiz

- ② To share a value you can use \_\_\_\_\_ (select all that apply).
- a) Resource Dictionary with {StaticResource}
  - b) Resource Dictionary with {x:Static}
  - c) Static properties in code and {x:Static}
  - d) Static properties in code and {StaticResource}

# Flash Quiz

- ③ Which one of these is not a system-provided markup extension?
- a) {StaticResource}
  - b) {x:Null}
  - c) {ImageResource}
  - d) {x:Type}

# Flash Quiz

- ③ Which one of these is not a system-provided markup extension?
- a) {StaticResource}
  - b) {x:Null}
  - c) {ImageResource}
  - d) {x:Type}

# Flash Quiz

- ④ To have a property value be set to "{Text" you would type: \_\_\_\_\_.
- a) "\{Text"
  - b) "{{Text"
  - c) "{Text"
  - d) "{}{Text"

# Flash Quiz

- ④ To have a property value be set to "{Text" you would type: \_\_\_\_\_.
- a) "\{Text"
  - b) "{{Text"
  - c) "{Text"
  - d) "{}{Text"

# 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 a .NET attribute

```
using Xamarin.Forms.Xaml;  
[XamlCompilationAttribute(XamlCompilationOptions.Compile)]  
public partial class MainPage : ContentPage {
```



... or on a specific XAML-based class

# What does XAMLC do?

- ❖ Attribute presence causes MSBuild command to be run which parses the XAML and generates **InitializeComponent** to create the page in code

```
private void InitializeComponent()
{
    Label label = new Label();
    StackLayout stackLayout = new StackLayout();
    stackLayout.SetValue(VisualElement.BackgroundColorProperty,
        new ColorTypeConverter().ConvertFrom("Red"));
    stackLayout.SetValue(Layout.PaddingProperty,
        new ThicknessTypeConverter().ConvertFrom("10"));
    stackLayout.SetValue(StackLayout.SpacingProperty, 5);
    label.SetValue(Label.TextProperty, "Hello, Forms");
    stackLayout.Children.Add(label);
    ...
    this.Content = stackLayout;
}
```

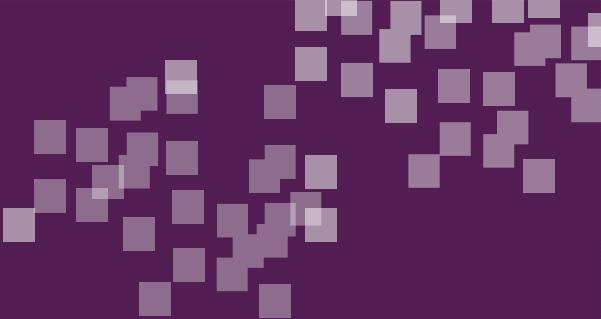
# Disabling XAMLC

- ❖ 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`



# Individual Exercise

Cleanup the XAML code and tailor the UI to the platform

# Thank You!