Getting hands on with XAML and Xamarin.Forms

Mitch “Rez” Muenster

@MobileRez

mobilerez.tumbler.com

Xamarin Certified Developer

Objectives

- Understanding Xamarin.Forms
- XAML Syntax & Behavior
- Advanced XAML

Understanding Xamarin.Forms

What is Xamarin.Forms?

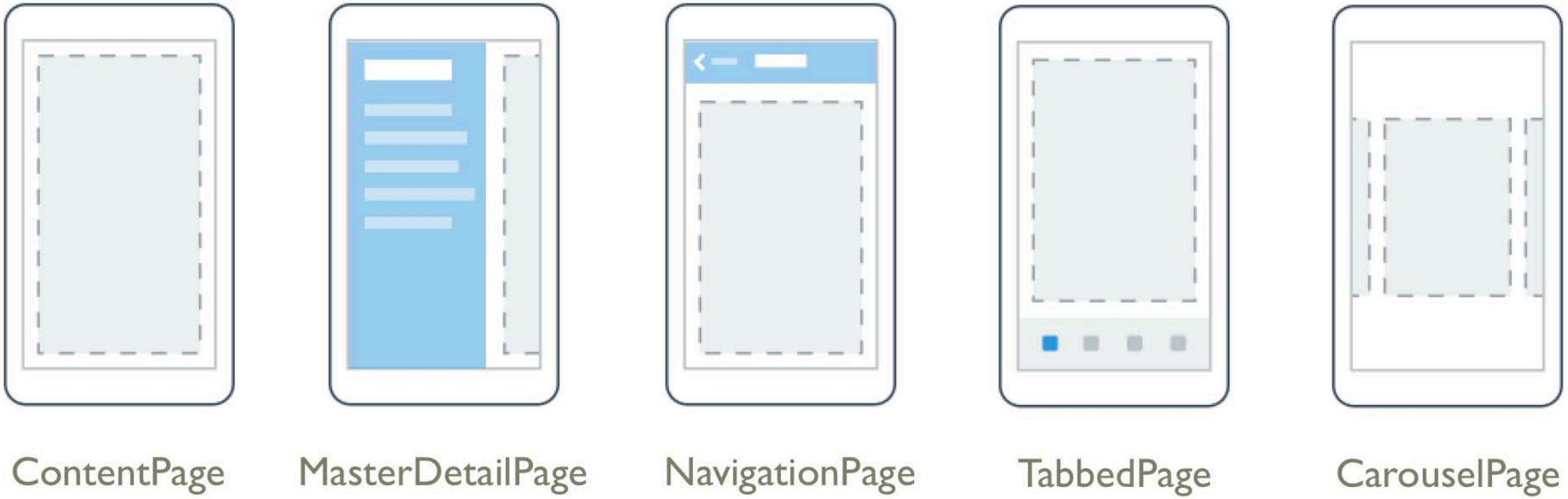
- Xamarin.Forms allows you to rapidly create a cross platform app with a native UI.
- Can be created in either a Shared Class Library or Portable Class Library
- Great for Prototyping or Data-Driven apps.
- Can also use dependency service to access platform specific features.

Understanding Xamarin.Forms UI

- Xamarin.Forms UI is defined in 4 different ways; Pages, Layouts, Cells, and Views.

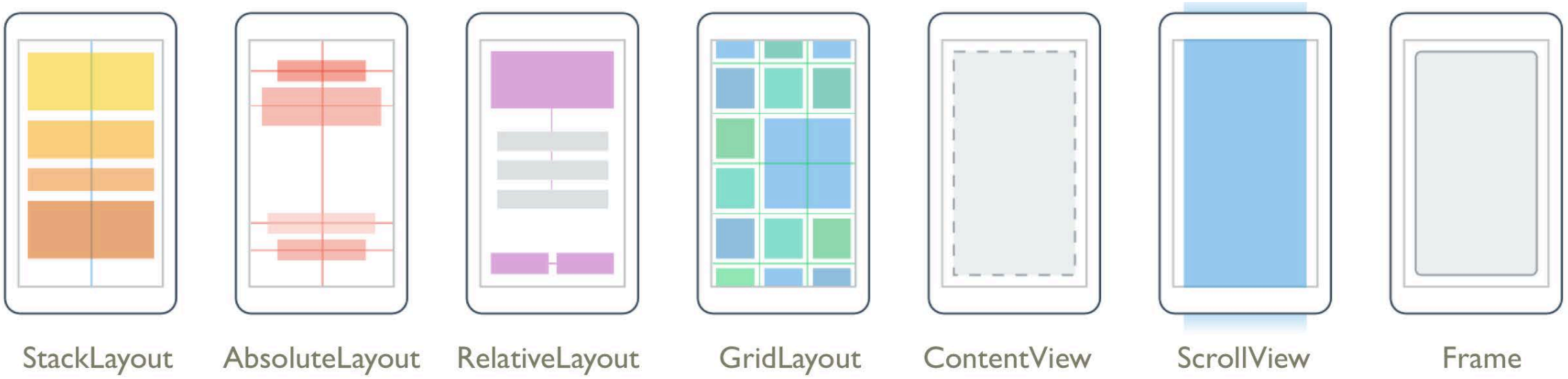
What is a Page?

- A page is used to define a single screen that contains most or all of the screen.



What is a Layout?

- A layout is a special type of view that acts as a container for other views or layouts.



What is a View?

- A View is the term Xamarin.Forms uses for all its basic controls from Buttons to Progress Bars.
- Some of the Views Xamarin.Forms contains are
  - Button
  - Date Picker
  - Entry (*input box*)
  - Label
  - Picker (*The phones form of dropdown list*)
  - Progress Bar

A full list of Views at https://developer.xamarin.com/guides/cross-platform/xamarin-forms/controls/views/

What is a Cell?

- A Cell is a special element that is used inside tables and defines how each item in a list is rendered.
- An example of Cells Xamarin.Forms supports:
  - Entry Cell
  - Switch Cell
  - Text Cell
  - Image Cell

A full list of Cells at https://developer.xamarin.com/guides/cross-platform/xamarin-forms/controls/cells/

Traditional way to build Forms apps

- Xamarin.Forms apps are commonly built using all using C# and not XAML.
- A new Xamarin.Forms app is usually created with a dummy app in a cs file

```
public App()
{
    // The root page of your application
    MainPage = new ContentPage {
        Content = new StackLayout {
            VerticalOptions = LayoutOptions.Center,
            Children = {
                new Label {
                    XAlign = TextAlignment.Center,
                    Text = "Welcome to Xamarin Forms!“
                }
            }
        }
    };
}
```

XAML Syntax & Behavior

What is XAML?

- XAML stands for Extensible Markup Language and was created by Microsoft specifically for working with the UI
- A XAML file is always associated with a C# code file.

Why use XAML over all code in a .cs file?

- Designer can create UI while coder focuses on code in the code file
- XAML allows for features like DataBinding Animations, Custom behaviors, value converters & more.
- Easier to work with for those who like to have a more visual representation of their layouts
- Helps keep a separation between UI and app logic

Building a layout in XAML

```
<?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="Toolbox.View.Page1">
  <StackLayout>
    <StackLayout.Padding>
      <OnPlatform x:TypeArguments="Thickness“
                  iOS="0, 20, 0, 0"/>
    </StackLayout.Padding>
    <Label Text="{Binding MainText}" VerticalOptions="Center" HorizontalOptions="Center" />
    </StackLayout>
</ContentPage>
```

XAML Syntax: Attached properties & Property Elements

```
<?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="Toolbox.View.MunitConverter“>
  <StackLayout VerticalOptions="CenterAndExpand" Padding="20" >
    <Grid HorizontalOptions="Center" >
      <Grid.RowDefinitions>
        <RowDefinition Height="*"/>
        <RowDefinition Height="*"/>
        <RowDefinition Height="*"/>
        <RowDefinition Height="*"/>
        <RowDefinition Height="*"/>
        <RowDefinition Height="*"/>
        <RowDefinition Height="*"/>
        <RowDefinition Height="*"/>
      </Grid.RowDefinitions>
      ...
      <Label x:Name="endLabel" Grid.Row="7" StyleId="EndLabel" XAlign="Center"/>
    </Grid>
  </StackLayout>
</ContentPage>
```

XAML Syntax: OnPlatform & OnIdiom

```
<?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="Toolbox.View.Page1">
  <StackLayout>
    <StackLayout.Padding>
      <OnPlatform x:TypeArguments="Thickness“
                  Android="0, 0, 0, 0“
                  iOS="0, 20, 0, 0“
                  WinPhone="0, 0, 0, 0"/>
    </StackLayout.Padding>
    <StackLayout.BackgroundColor>
      <OnIdiom x:TypeArguments="Color“
               Phone="Teal“
               Tablet="Green"/>
    </StackLayout.BackgroundColor>
    <Label Text="{Binding MainText}" VerticalOptions="Center" HorizontalOptions="Center" />
   </StackLayout>
</ContentPage>
```

XAML Syntax: OnPlatform & OnIdiom

- OnPlatform allows us to define code for a specific platform
- OnIdiom allows us to define code for either a tablet or phone
- Both are useful for providing those quick tweaks to get the UI to look the same on all devices or only changing one platform to fit the rest of their needs

Advanced XAML

Using Resource Dictionary

- A Resource Dictionary is a dictionary that is specifically for use within the UI
- Can be defined in the XAML or Code behind
- Use the x:Key to define the ID of the dictionary entry so you can reference it later

Using Resource Dictionary

```
<?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="Toolbox.View.Page1“ BackgroundColor="White">
  <ContentPage.Resources>
    <ResourceDictionary>
      <Color x:Key="TxtRed">Red</Color>
      <OnPlatform x:Key="TxtColor“ x:TypeArguments="Color“ Android="Green“ iOS="Teal“
                  WinPhone="Purple"/>
      <LayoutOptions x:Key="HorzCenter">Center</LayoutOptions>
    </ResourceDictionary>
  </ContentPage.Resources>
  <StackLayout>
    <Label Text="Im a label" HorizontalOptions="{StaticResource HorzCenter}“
           TextColor="{StaticResource TxtColor}" />
    <Label Text="Im a label too, but different" HorizontalOptions="{StaticResource HorzCenter}“
           TextColor="{StaticResource TxtRed}"/>
    <Label Text="Im also a different label but look like the first label"
           HorizontalOptions="{StaticResource HorzCenter}“
           TextColor="{StaticResource TxtColor}"/>
  </StackLayout>
</ContentPage>
```

Resource Dictionary hierarchy

- Resource files can inherit from global resource files
- Resource files prioritize definitions closer to where they started in the hierarchy.
- Order of priority is View, Layout, Page, Application

Resource Dictionary Hierarchy

```
<ContentPage.Resources>
   <ResourceDictionary>
     <x:String x:Key="TxtL1">I’m different</x:String>
   </ResourceDictionary>
 </ContentPage.Resources>
 <StackLayout>
   <Label Text="{StaticResource TxtL1}"></Label>
 </StackLayout>
```

```
<ContentPage.Resources>
   <ResourceDictionary>
     <x:String x:Key="TxtL1">Im a label</x:String>
   </ResourceDictionary>
 </ContentPage.Resources>
 <StackLayout>
   <Label Text="{StaticResource TxtL1}"></Label>
 </StackLayout>
```

Data Binding + XAML + Forms

Data Binding + XAML + Forms

Hands on Lab

What's Next?

- Data Binding with Xamarin.Forms & XAML
- List views and collections with Data Binding, XAML, & Xamarin.Forms

Questions?

mitchmuenster@gmail.com

@MobileRez

mobilerez.tumbler.com