

A portrait photograph of Glenn Stephens, a man with short brown hair, wearing a dark t-shirt, standing in front of a brick wall.

Xamarin Evolve 2014

Extending Xamarin.Forms

Glenn Stephens
glen.stephens@xamarin.com

 **Xamarin**
University

Extending Xamarin.Forms lets you put into your apps new and additional device-specific functionality or controls which make apps that look and feel better.

Xamarin University

History

Xamarin.Forms was designed as an abstraction framework to build cross platform data-centric apps quickly

Xamarin University

Agenda

1. Xamarin.Forms concepts
2. Places for Extension
3. Markup Extensions
4. Elements and Renderers

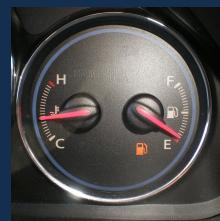
Xamarin University

Concepts

Xamarin University

Reasons for Extension

- Change display aspects
- Take advantage of device features
- Newly created control



Xamarin University

Agenda

1. ~~Xamarin.Forms concepts~~
2. Places for Extension
3. Markup Extensions
4. Elements and Renderers



Xamarin University

Places for Extension

- Services with DependencyService
- Markup Extensions
- Composite Controls
- Renderers

Interfaces with DependencyService (non-visual)

Markup Extensions (XAML files)

Composite Controls (Xamarin.Forms control)

Platform Specific Renderers (Native controls)

Xamarin University

Xamarin University

Custom Services

- You can use the Dependency Service to provide platform services

Define an interface for the service and then provide platform implementations

The diagram illustrates the implementation of the `ISpeech` interface. On the left, a green rounded rectangle contains the `ISpeech` interface definition, which includes the `Speak()` method and an ellipsis. A large blue arrow points from this interface to three separate platform implementations. The first implementation, for iOS, shows the Apple logo next to a box labeled `ISpeech` which maps to `AVSpeechSynthesizer`. The second implementation, for Windows, shows the Windows logo next to a box labeled `ISpeech` which maps to `Synthesis`. The third implementation, for Android, shows the Android logo next to a box labeled `ISpeech` which maps to `TextToSpeech`. The entire diagram is framed by a dark blue border with the text "Xamarin University" at the bottom.

Xamarin University

Provide an implementation for each platform

ISpeech → AVSpeechSynthesizer

ISpeech → Synthesis

ISpeech → TextToSpeech

Xamarin University

Example of Markup Extensions

- Markup Extensions can create run-time calculated values from XAML

```
<Label Text="Name" Font="Bold" />
<Entry Text="{Binding Name}" />

<Label Text="Email" Font="Bold" />
<Entry Text="{Binding Email}" />
```

Xamarin University

Creating Composite Controls

- We could also create custom controls by using existing controls

October 2014						
S	M	T	W	T	F	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

Xamarin University

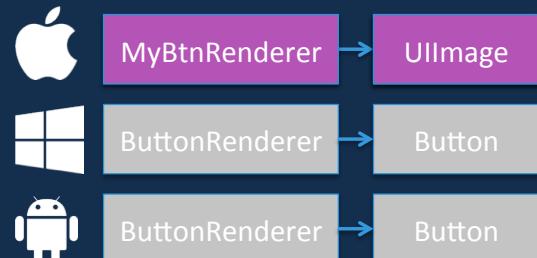
Extending a control on a platform

- If we don't like the way a control is rendered on the platform, we can change it

Element describes the visual element



Renderer creates platform visualization based on the model



Xamarin University

Xamarin University

Writing new extensions

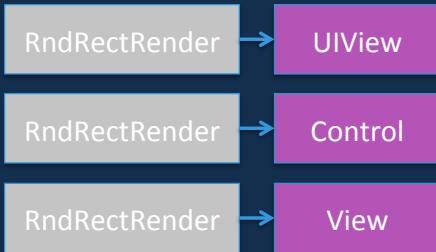
- The same element/renderer model is used for new components

Element describes the visual element

RoundedRect
PenWidth
Radius
...



Renderer creates
platform
visualization based
on the model



Xamarin University

Xamarin University

Agenda

1. `Xamarin.Forms` concepts
2. Places for Extension
3. Mark up Extensions
4. Elements and Renderers



Xamarin University

Markup Extensions

Xamarin University

Existing Markup Extensions

```
<Label Text="Name" Font="Bold" />
<Entry Text="{Binding Name, Mode=TwoWay}" />

<Label Text="Email" Font="Bold" />
<Entry Text="{Binding Email, Mode=TwoWay}" />
```

Xamarin University

Localization Markup Example

```
<Label Text="{i18n:Translate Name}" />  
  
<Entry Text="" x:Name="nameEntry" Placeholder="task name" />  
  
<Label Text="{i18n:Translate Notes}" />  
  
<Entry Text="{Binding Path=Notes}" x:Name="notesEntry" />  
  
<Button Text="{i18n:Translate Save}" Clicked="OnSave" />
```

Xamarin University

Markup Extensions

Add a content property attribute to capture the Markup Extension Value

```
[ContentProperty("Name")] // {DateDisplay Name}  
  
public class DateDisplayExtension : IMarkupExtension  
{  
    public string Name {  
        get;  
        set;  
    }  
}
```

Xamarin University

Markup Extensions

Implement the `ProvideValue` method to calculate the value

```
public class DateDisplayExtension : IMarkupExtension
{
    public object ProvideValue
        (IServiceProvider serviceProvider)
    {
        if (serviceProvider == null)
            throw new ArgumentNullException ("No Provider");

        if (Name == "Date")
            return DateTime.Now.ToString ("D");
        else
            return "The value could not be determined";
    }
}
```

Xamarin University

Markup Extensions

After the namespace is registered, use the Markup extension

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:dd="clr-namespace:ExtendingXamarinForms;assembly=ExtendingXamarinForms"
    x:Class="ExtendingXamarinForms.MarkupExtensionExamplePage">
    <Label Text="{dd:DateDisplay Date}"></Label>
</ContentPage>
```

This is your class name
without the Extension
suffix

Xamarin University

Demonstration

Xamarin University

Agenda

1. ~~Xamarin.Forms~~ concepts
2. ~~Places for Extension~~
3. ~~Mark up Extensions~~
4. Elements and Renderers



Xamarin University

Creating abstractions

Xamarin.Forms uses abstractions to define what it wants done and then provides platform implementations for those mechanisms



Xamarin University

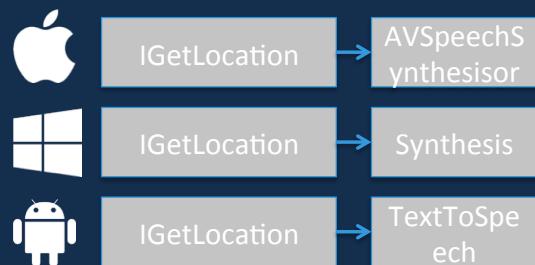
Creating Services

- You may be familiar with DependencyService that creates abstractions

Define an interface for the service and then provide platform implementations

ISpeech
Speak
...

Provide an implementation for each platform



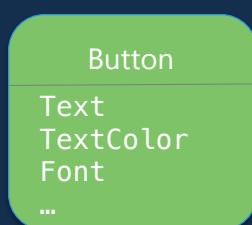
Xamarin University

Xamarin University

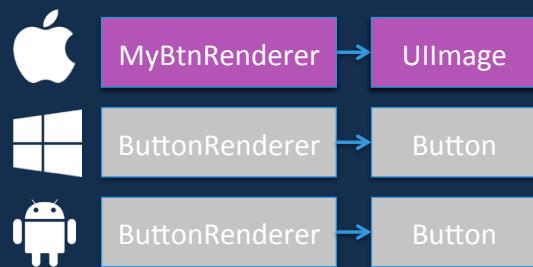
Extending a control on a platform

- If we don't like the way a control is rendered on the platform, we can change it

Element describes the visual element



Renderer creates platform visualization based on the model



Xamarin University

Xamarin University

Registering a Custom Renderer

Find the right renderer to Subclass

```
namespace ExtendingXamarinForms.iOS
{
    public class iOS8NavigationPageRenderer : NavigationRenderer
    {
        public iOS8NavigationPageRenderer () : base()
        {
            if (UIDevice.CurrentDevice.CheckSystemVersion (8, 0))
                this.HidesBarsOnSwipe = true;
        }
    }
}
```

Xamarin University

Registering a Custom Renderer

The renderers are subclasses of the actual device control

```
namespace ExtendingXamarinForms.iOS
{
    public class iOS8NavigationPageRenderer : NavigationRenderer
    {
        public iOS8NavigationPageRenderer () : base()
        {
            if (UIDevice.CurrentDevice.CheckSystemVersion (8, 0))
                this.HidesBarsOnSwipe = true;
        }
    }
}
```

↑
This is a property of
UINavigationController

Xamarin University

Registering a Custom Renderer

Each custom renderer needs to be registered

```
[assembly: ExportRenderer(typeof(NavigationPage),
    typeof(iOS8NavigationPageRenderer))]
```

↑
This is your
custom renderer

↓ This is the Xamarin.Forms
component you want to render

Xamarin University

Demonstration

Xamarin University

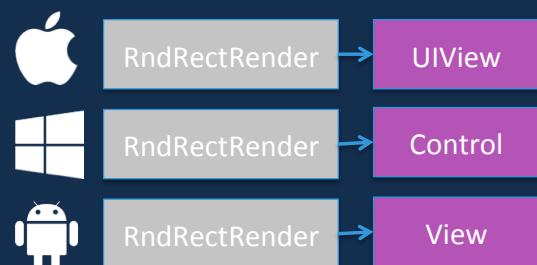
Xamarin.Forms rendering model

- Two pieces to each control: The element and the renderer

Element describes the visual element



Renderer creates
platform
visualization based
on the model



Xamarin University

Xamarin University

Creating a new control & renderers

- Step 1 - Create the element definition in your PCL/Shared Project

```
public class RoundedBoxView : BoxView  
{  
}
```

↑
This is an existing view if you are extending the view,
or View if you are creating a new one

Xamarin University

Creating a new control & renderers

- Step 2 – Add properties to your View definition

```
public static readonly BindableProperty CornerRadiusProperty =  
    BindableProperty.Create<RoundedBoxView, double>(  
        p => p.CornerRadius, defaultValue: 0);  
  
public double CornerRadius  
{  
    get { return (double)base.GetValue(CornerRadiusProperty); }  
    set { base.SetValue(CornerRadiusProperty, value); }  
}
```

Xamarin University

Creating a new control & renderers

- Step 3 – Implement a renderer for each platform

```
public class RoundedBoxViewRenderer :  
    ViewRenderer<RoundedBoxView, UIView>  
{  
}  
  
Define the control that  
you are rendering against  
↑  
  
↑  
Define the native control  
that your renderer will  
be based on
```

Xamarin University

Implementing a composite renderer

```
protected override void OnElementChanged(  
    ElementChangedEventArgs<RoundedBoxView> e)  
{  
    base.OnElementChanged(e);  
    var rbv = e.NewElement;  
    if (rbv != null) {  
        childView = new UIView() {  
            BackgroundColor = rbv.Color.ToUIColor(),  
            AutoresizingMask = UIViewAutoresizing.FlexibleWidth |  
                UIViewAutoresizing.FlexibleHeight  
        };  
        SetNativeControl(childView);  
    }  
}
```

Xamarin University

Implementing a drawn renderer

```
public override void Draw(System.Drawing.RectangleF rect)
{
    RoundedBoxView rbv = (RoundedBoxView)this.Element;

    using (var context = UIGraphics.GetCurrentContext()) {

        context.SetFillColor(rbv.Color.ToCGColor());
        ...
        var path = CGPath.FromRoundedRect(rc, radius, radius);
        context.AddPath(path);
        context.DrawPath(CGPathDrawingMode.FillStroke);
    }
}
```

Xamarin University

Register the renderers

- Step 4 – provide assembly registration per platform

The diagram illustrates the registration of a custom renderer. It shows a code snippet with annotations:

- An annotation "This is the Xamarin.Forms element" points to the line `[assembly: ExportRendererAttribute(typeof(RoundedBoxView),`.
- An annotation "This is your custom renderer" points to the line `typeof(RoundedBoxViewRenderer))]`.
- A blue arrow points from the "This is the Xamarin.Forms element" annotation to the first line of the code.
- A blue arrow points from the "This is your custom renderer" annotation to the closing bracket of the code.

```
[assembly: ExportRendererAttribute(typeof(RoundedBoxView),
    typeof(RoundedBoxViewRenderer))]
```

Xamarin University

Demonstration

Xamarin University

Summary

- Concepts
- Places for Extensions
- Creating Markup Extensions
- Working with Elements and Renderers

Xamarin University

Xamarin Evolve 2014

Extending Xamarin.Forms

Extending Xamarin.Forms

Glenn Stephens
glen.stephens@xamarin.com

Xamarin University