Wintellect NOW

Building Cross-Platform Mobile Apps with Visual Studio 2015 and Xamarin Forms

Jeff Prosise
jeffpro@wintellect.com

About Wintellect

Custom Application Development

Build applications from scratch or in collaboration with your developers and architects

- Enterprise App Development
- Cross Platform App Development
- Cloud Solutions
- Architecture Design and Planning
- Database Design and Optimization
- Debugging and Performance Tuning

Legacy Application Modernization

Update applications built with legacy technologies to leverage modern platforms and tools

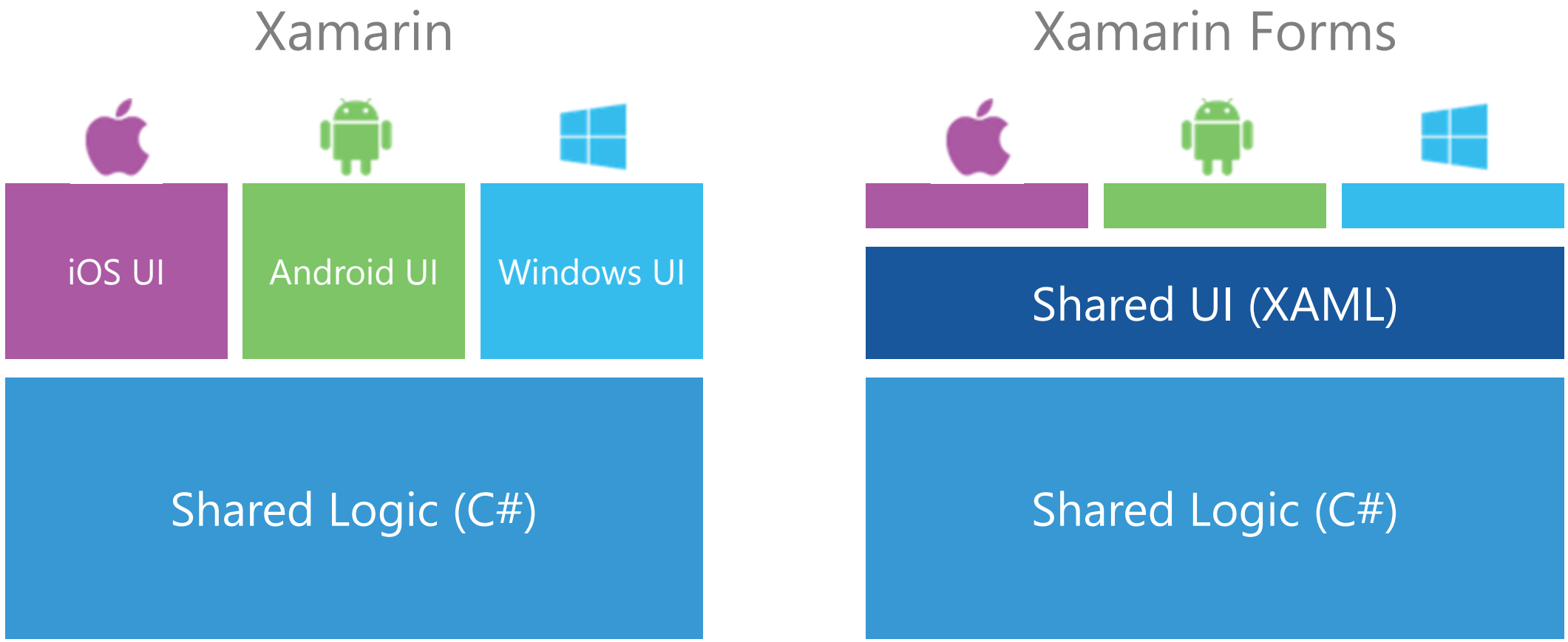
- Silverlight to HTML5
- Web Forms to ASP.NET MVC
- ASP Classic to ASP.NET MVC
- Client-Server to SOA

Multi-Format Developer Training

Feed your developers with classes delivered live, virtually, or on-demand

- Instructor-Led On-Site Training
- On-Demand Developer Training
- Live Virtual Developer Training

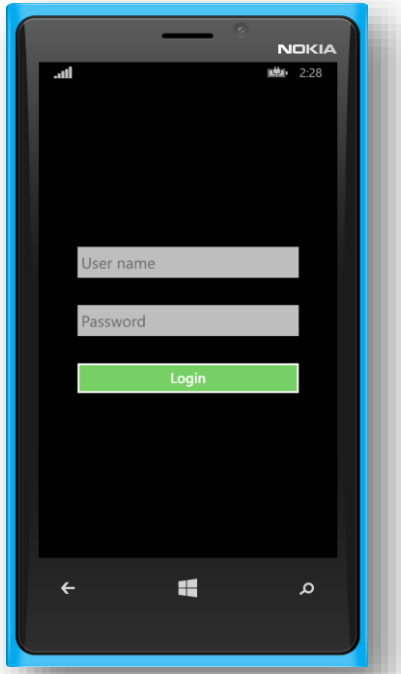
What is Xamarin Forms?



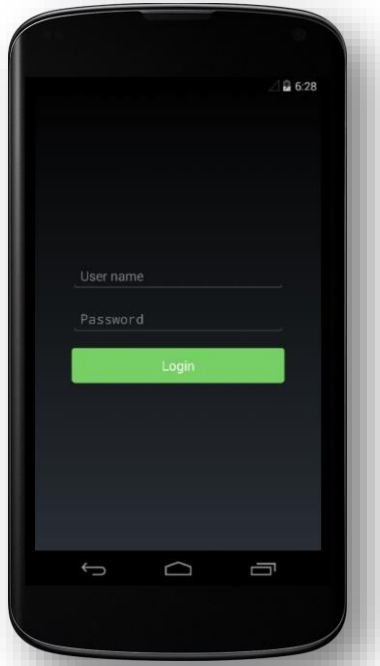
Using XAML to Build Cross-Platform UIs

```
<ContentPage>
  <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>
```

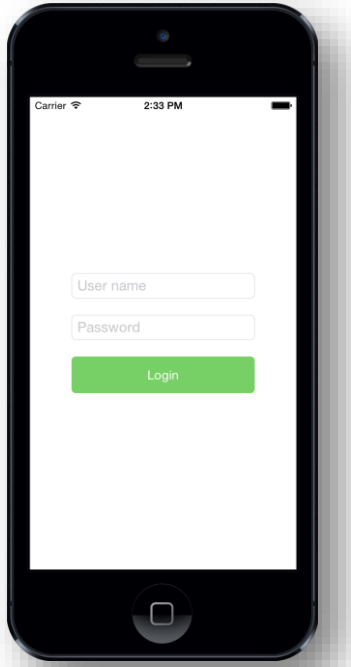
Windows Phone



Android



iOS



Xamarin XAML vs. Microsoft XAML

Xamarin

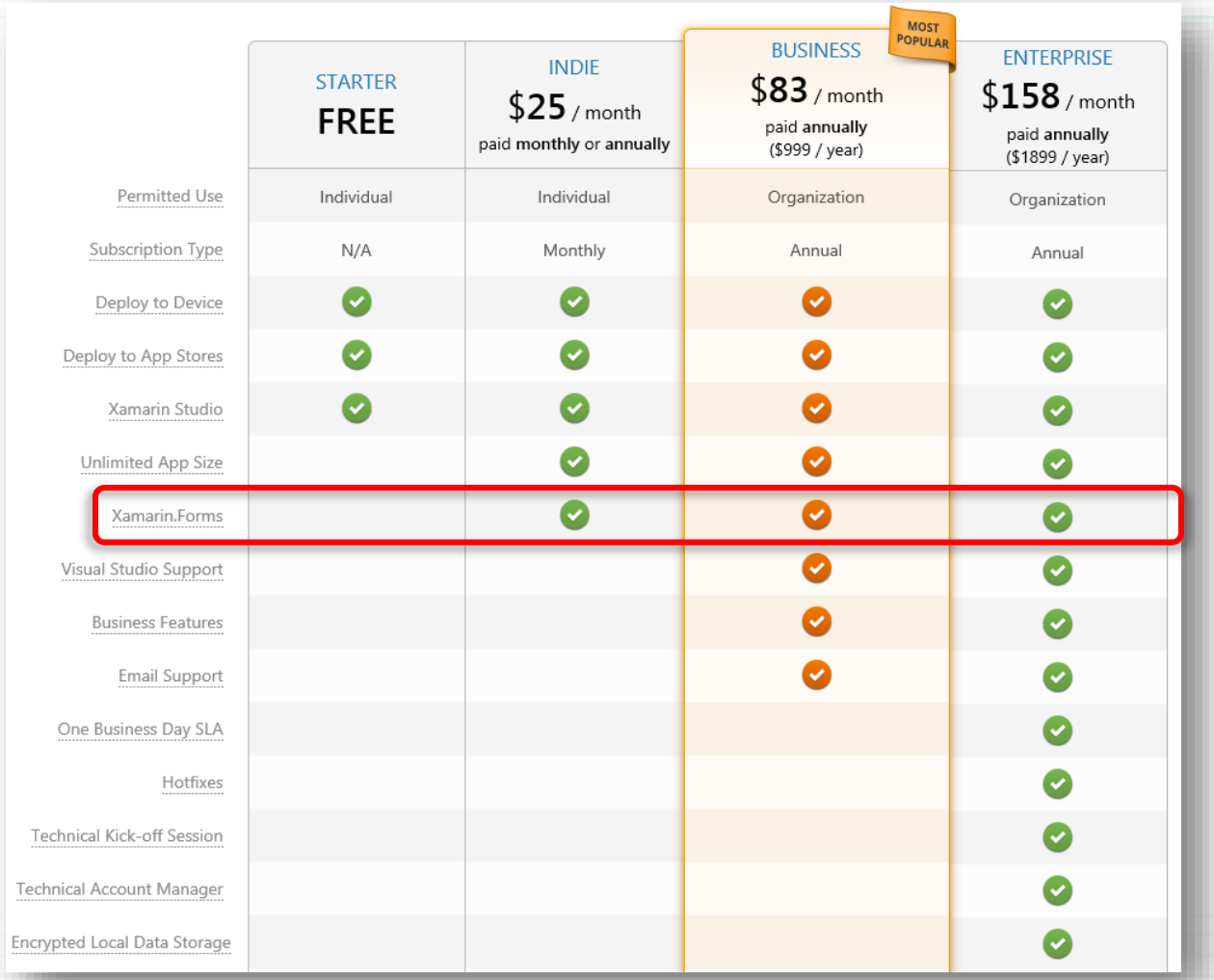
```
<ContentPage>
  <StackLayout Spacing="20" Padding="50"
    VerticalOptions="Center">
    <Entry Placeholder="User name" />
    <Entry Placeholder="Password"
      IsPassword="True" />
    <Button Text="Login" TextColor="White"
      BackgroundColor="##FF77D065" />
  </StackLayout>
</ContentPage>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

Microsoft

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

Licensing Xamarin

- Xamarin is not free*
  - Choose from one of four licenses
  - Xamarin Forms requires at least an Indie license
  - Visual Studio support requires at least a Business license
- MSDN subscribers get a 20% discount on Business and Enterprise licenses (https://xamarin.com/msdn)
- Prices are per developer, per device platform (iOS and Android)



	STARTER FREE	INDIE $25 / month paid monthly or annually	BUSINESS $83 / month paid annually ($999 / year) MOST POPULAR	ENTERPRISE $158 / month paid annually ($1899 / year)
Permitted Use	Individual	Individual	Organization	Organization
Subscription Type	N/A	Monthly	Annual	Annual
Deploy to Device	✓	✓	✓	✓
Deploy to App Stores	✓	✓	✓	✓
Xamarin Studio	✓	✓	✓	✓
Unlimited App Size		✓	✓	✓
Xamarin.Forms		✓	✓	✓
Visual Studio Support			✓	✓
Business Features			✓	✓
Email Support			✓	✓
One Business Day SLA				✓
Hotfixes				✓
Technical Kick-off Session				✓
Technical Account Manager				✓
Encrypted Local Data Storage				✓

Xamarin Studio vs. Visual Studio

- Xamarin Studio runs on Windows and Mac OS
  - Mac version lets you build apps for iOS and Android
  - Windows version lets you build Android apps
- Visual Studio runs only on Windows
  - Can build apps for Windows Phone and Android
  - Can build iOS apps when paired with a Mac configured as a build server
    - http://developer.xamarin.com/guides/ios/getting_started/installation/windows/introduction_to_xamarin_ios_for_visual_studio/
- To build Xamarin Forms apps for iOS, Android, and Windows Phone, you need Visual Studio on Windows, licenses for Xamarin.Android and Xamarin.iOS, and a Mac to act as an iOS build server

DEMO

Your First Xamarin Forms App

Views and Cells (Controls)

- Buttons, Labels, WebViews, and other control elements

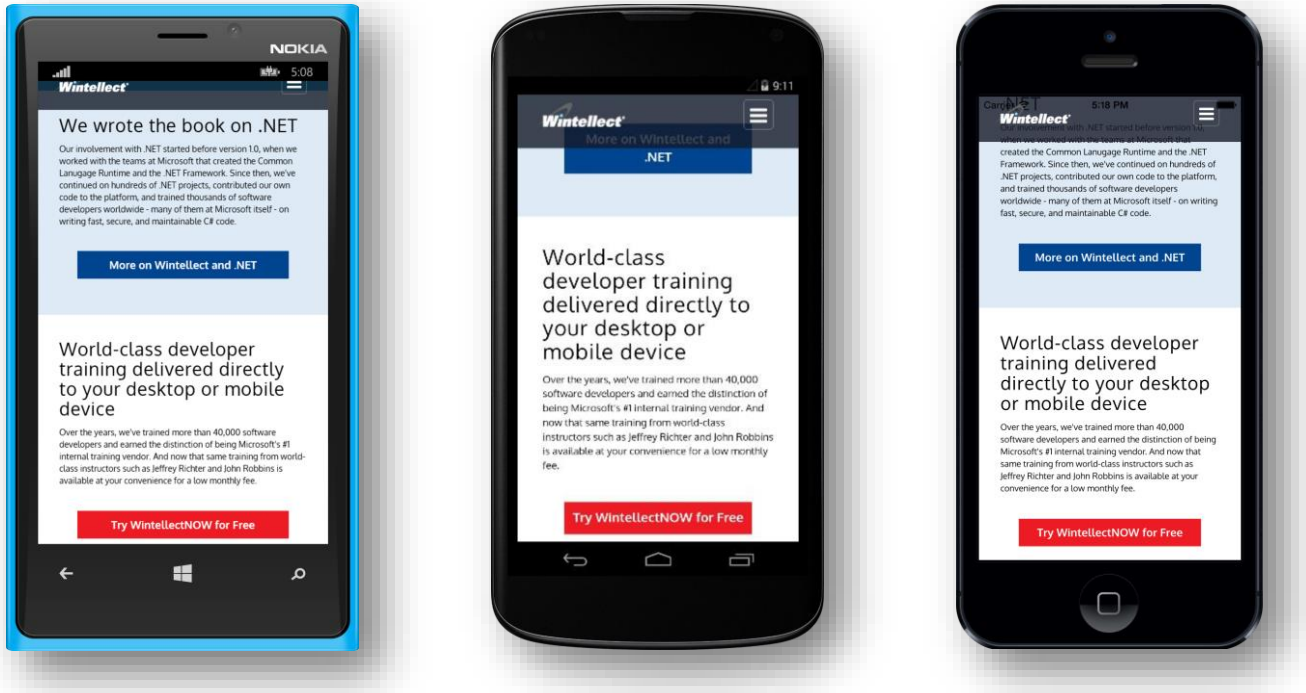
ActivityIndicator	BoxView	Button	DatePicker	Editor
Entry	Image	Label	ListView	Map
OpenGLView	Picker	ProgressBar	SearchBar	Slider
Stepper	Switch	TableView	TimePicker	WebView
EntryCell	ImageCell	SwitchCell	TextCell	ViewCell

DatePicker

```
<DatePicker />
```

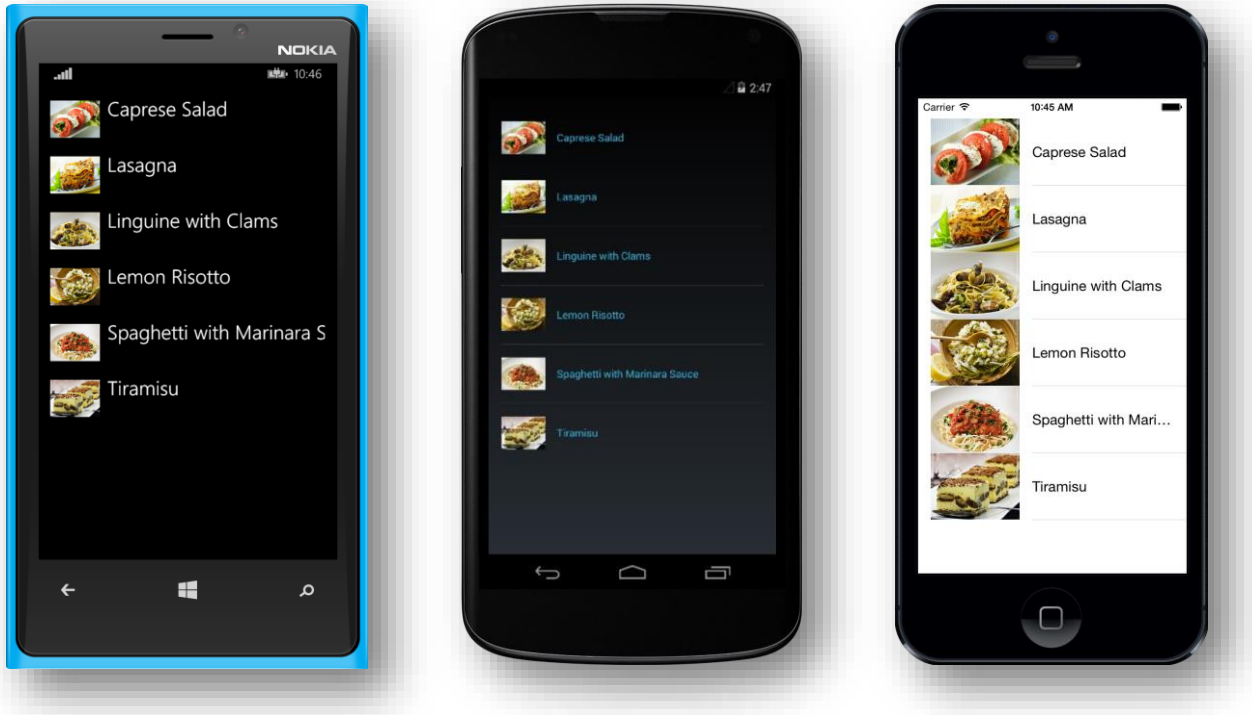
WebView

```
<WebView Source="http://www.wintellect.com" />
```



ListView and ImageCell

```
<ListView RowHeight="80"
  ItemsSource="{Binding Recipes}">
  <ListView.ItemTemplate>
    <DataTemplate>
      <ImageCell
        ImageSource="{Binding Image}"
        Text="{Binding Title}" />
    </DataTemplate>
  </ListView.ItemTemplate>
</ListView>
```



ListView and ViewCell

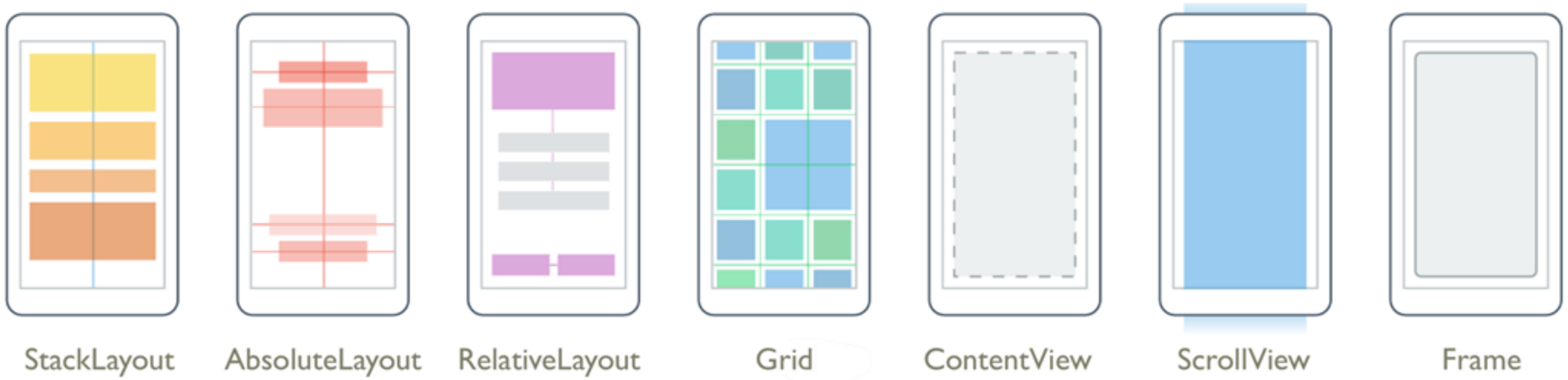
```
<ListView RowHeight="80"
  ItemsSource="{Binding Recipes}">
  <ListView.ItemTemplate>
    <DataTemplate>
      <ViewCell>
        <Grid Padding="8">
          ...
          <Image Source="{Binding Image}" />
          <Grid Grid.Column="1" Padding="8">
            ...
            <Label Text="{Binding Title}"
              FontSize="Large"
              LineBreakMode="WordWrap" />
          </Grid>
        </Grid>
      </ViewCell>
    </DataTemplate>
  </ListView.ItemTemplate>
</ListView>
```

TableView and SwitchCell

```
<TableView>
  <TableView.Root>
    <TableSection Title="Privacy">
      <SwitchCell Text="Allow Spamming" />
      <SwitchCell Text="Track Location"
        On="True" />
    </TableSection>
    <TableSection Title="Performance">
      <SwitchCell Text="Run Super-Fast"
        On="True" />
      <SwitchCell Text="Cache Data Locally" />
      <SwitchCell Text="Steal Clock Cycles" />
    </TableSection>
  </TableView.Root>
</TableView>
```

Layouts

- Controls that contain other controls and provide layout and positioning

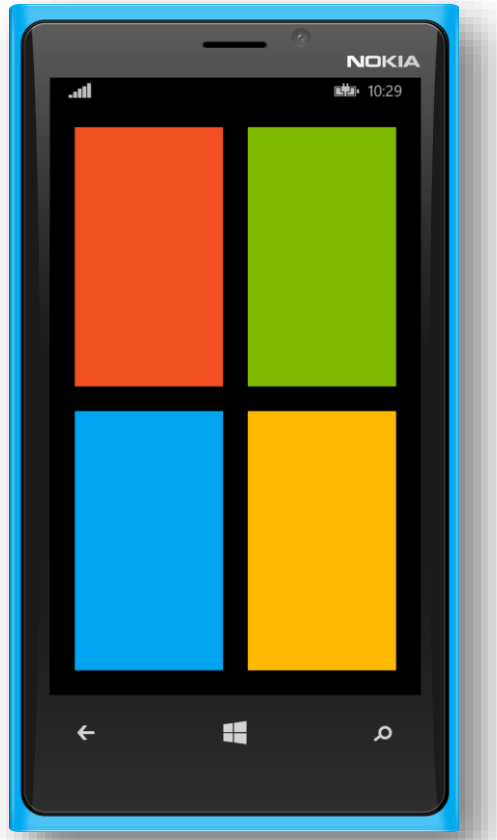


Using StackLayout

```
<StackLayout Padding="32" Spacing="32">
  <BoxView Color="#FFF25022" HeightRequest="128" />
  <BoxView Color="#FF7FBA00" HeightRequest="128" />
  <BoxView Color="#FF01A4EF" HeightRequest="128" />
  <BoxView Color="#FFFFB901" HeightRequest="128" />
</StackLayout>
```

Using Grid

```
<Grid Padding="32" RowSpacing="32" ColumnSpacing="32">
  <Grid.RowDefinitions>
    <RowDefinition Height="*" />
    <RowDefinition Height="*" />
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="*" />
    <ColumnDefinition Width="*" />
  </Grid.ColumnDefinitions>
  <BoxView Grid.Row="0" Grid.Column="0" Color="#FFF25022" />
  <BoxView Grid.Row="0" Grid.Column="1" Color="#FF7FBA00" />
  <BoxView Grid.Row="1" Grid.Column="0" Color="#FF01A4EF" />
  <BoxView Grid.Row="1" Grid.Column="1" Color="#FFFFB901" />
</Grid>
```

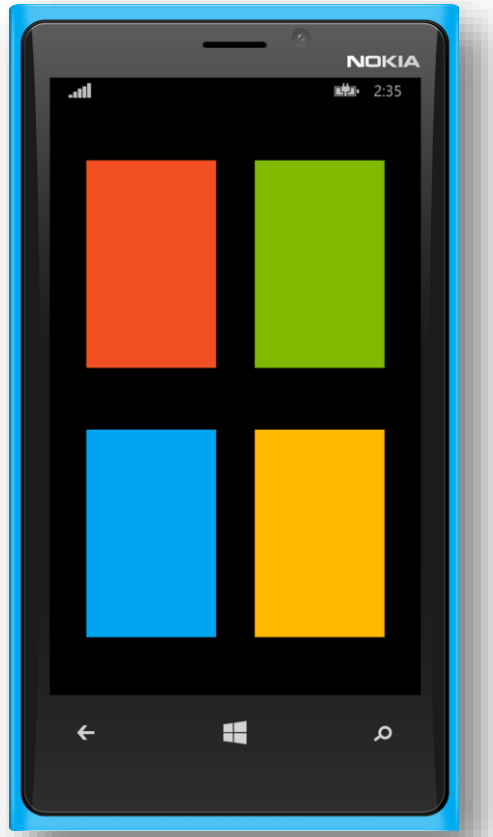


Using AbsoluteLayout with Device-Independent Units

```
<AbsoluteLayout>
  <BoxView Color="#FFF25022"
    AbsoluteLayout.LayoutBounds="32, 32, 150, 300" />
  <BoxView Color="#FF7FBA00"
    AbsoluteLayout.LayoutBounds="214, 32, 150, 300" />
  <BoxView Color="#FF01A4EF"
    AbsoluteLayout.LayoutBounds="32, 364, 150, 300" />
  <BoxView Color="#FFFFB901"
    AbsoluteLayout.LayoutBounds="214, 364, 150, 300" />
</AbsoluteLayout>
```

Using AbsoluteLayout with Proportional Units

```
<AbsoluteLayout>
  <BoxView Color="#FFF25022"
    AbsoluteLayout.LayoutFlags="All"
    AbsoluteLayout.LayoutBounds="0.15, 0.15, 0.35, 0.35" />
  <BoxView Color="#FF7FBA00"
    AbsoluteLayout.LayoutFlags="All"
    AbsoluteLayout.LayoutBounds="0.85, 0.15, 0.35, 0.35" />
  <BoxView Color="#FF01A4EF"
    AbsoluteLayout.LayoutFlags="All"
    AbsoluteLayout.LayoutBounds="0.15, 0.85, 0.35, 0.35" />
  <BoxView Color="#FFFFB901"
    AbsoluteLayout.LayoutFlags="All"
    AbsoluteLayout.LayoutBounds="0.85, 0.85, 0.35, 0.35" />
</AbsoluteLayout>
```



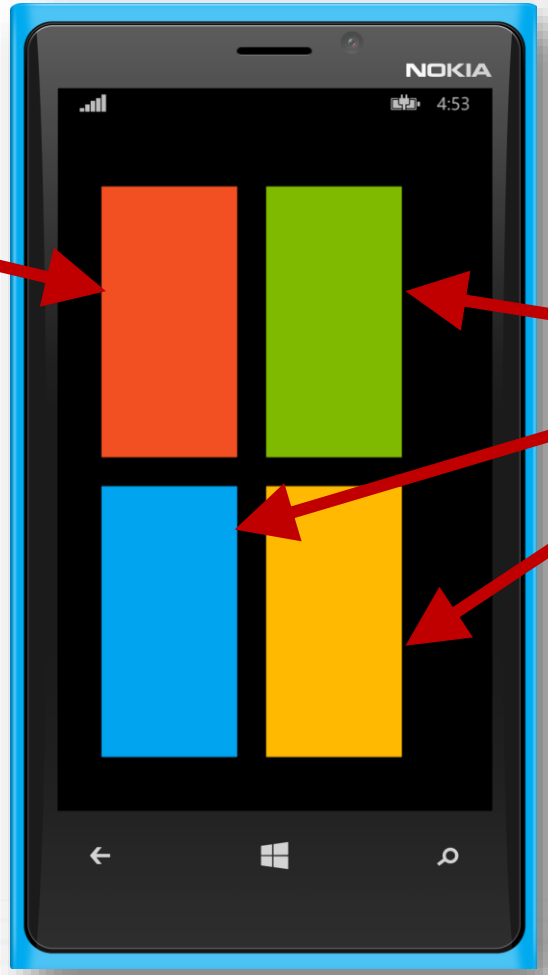
$$layoutBounds.X = \frac{fractionalChildCoordinate.X}{(1 - layoutBounds.Width)}$$

Using RelativeLayout

```
<RelativeLayout>

  <BoxView Color="#FFF25022" WidthRequest="150" HeightRequest="300" x:Name="RedBox"
    RelativeLayout.XConstraint="{ConstraintExpression Type=RelativeToParent, Property=Width, Factor=0.1}"
    RelativeLayout.YConstraint="{ConstraintExpression Type=RelativeToParent, Property=Height, Factor=0.1}" />

  <BoxView Color="#FF7FBA00" WidthRequest="150" HeightRequest="300"
    RelativeLayout.XConstraint="{ConstraintExpression Type=RelativeToView, ElementName=RedBox, Property=X, Constant=182}"
    RelativeLayout.YConstraint="{ConstraintExpression Type=RelativeToView, ElementName=RedBox, Property=Y, Constant=0}" />

  <BoxView Color="#FF01A4EF" WidthRequest="150" HeightRequest="300"
    RelativeLayout.XConstraint="{ConstraintExpression Type=RelativeToView, ElementName=RedBox, Property=X, Constant=0}"
    RelativeLayout.YConstraint="{ConstraintExpression Type=RelativeToView, ElementName=RedBox, Property=Y, Constant=332}" />

  <BoxView Color="#FFFFB901" WidthRequest="150" HeightRequest="300"
    RelativeLayout.XConstraint="{ConstraintExpression Type=RelativeToView, ElementName=RedBox, Property=X, Constant=182}"
    RelativeLayout.YConstraint="{ConstraintExpression Type=RelativeToView, ElementName=RedBox, Property=Y, Constant=332}" />

</RelativeLayout>
```

RelativeLayout, Continued

Upper-left corner of red box positioned 1/10th of the way across the screen and 1/10th of the way down



Other boxes "anchored" to red box so they move if it moves

Grids, Buttons, and Labels…Oh My!

OnPlatform

- Easy-to-use mechanism for specifying property values and executing code on a per-platform basis in shared code
  - Generic class usable in XAML (<OnPlatform>)
  - Static method accessible from code (Device.OnPlatform)
- Essential for tweaking UIs to get just the right look on every platform

Using OnPlatform in XAML

```
<BoxView HorizontalOptions="Center">
  <BoxView.Color>
    <OnPlatform x:TypeArguments="Color"
      iOS="Green"
      Android="#738182"
      WinPhone="Accent" />
  </BoxView.Color>
  <BoxView.WidthRequest>
    <OnPlatform x:TypeArguments="x:Double"
      iOS="30"
      Android="40"
      WinPhone="50" />
  </BoxView.WidthRequest>
</BoxView>
```

Using OnPlatform in Code

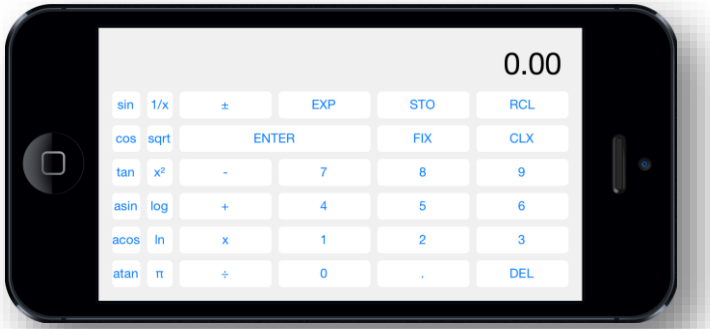
```
// Assign platform-specific values to cx and cy
double cx = Device.OnPlatform(iOS: 24, Android: 30, WinPhone: 36);
double cy = Device.OnPlatform(iOS: 32, Android: 40, WinPhone: 48);

// Execute platform-specific code on iOS and Android
Device.OnPlatform(iOS: () =>
{
    this.BackgroundColor = Color.Red; // Set page background to red
},
Android: () =>
{
    this.BackgroundColor = Color.Blue; // Set page background to blue
});
```

DEMO

Tweaking the UI for Each Platform

Orientation Changes

- Xamarin Forms don't fire events reporting device-orientation changes
- Use Page.SizeChanged events or override Page.OnSizeAllocated instead
  - Latter can be called multiple times each time device is rotated



Using OnSizeAllocated

```
public partial class MainPage : ContentPage
{
    private double _width = 0.0;
    private double _height = 0.0;

    protected override void OnSizeAllocated(double width, double height)
    {
        base.OnSizeAllocated(width, height); // Important!

        if (width != _width || height != _height)
        {
            _width = width;
            _height = height;
            // TODO: Respond to orientation change
        }
    }
}
```

Using SizeChanged

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

        this.SizeChanged += (s, e) =>
        {
            if (Width != Height) // On Windows Phone, first call has both set to 0.0
            {
                // TODO: Respond to orientation change
            }
        };
    }
}
```

DEMO

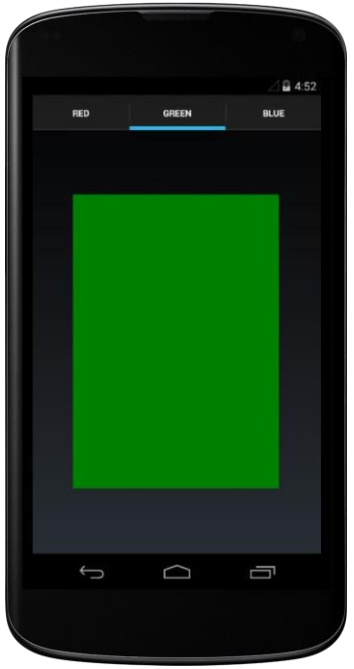
Responding to Orientation Changes

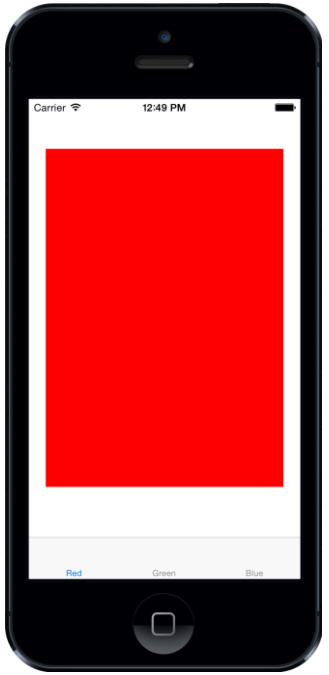
Pages

- Controls that represent pages

Creating a Tabbed Page

```
<TabbedPage ... Title="Tabbed Page">
  <TabbedPage.Children>
    <ContentPage Title="Red">
      <BoxView Color="Red" WidthRequest="280"
        HeightRequest="400" HorizontalOptions="Center"
        VerticalOptions="Center" />
    </ContentPage>
    <ContentPage Title="Green">
      <BoxView Color="Green" WidthRequest="280"
        HeightRequest="400" HorizontalOptions="Center"
        VerticalOptions="Center" />
    </ContentPage>
    <ContentPage Title="Blue">
      <BoxView Color="Blue" WidthRequest="280"
        HeightRequest="400" HorizontalOptions="Center"
        VerticalOptions="Center" />
    </ContentPage>
  </TabbedPage.Children>
</TabbedPage>
```





Creating a Navigation Page

```
// In App.cs
this.MainPage = new NavigationPage(new MasterPage());
```

Navigating to Another Page

```
// In the code-behind for the current page
this.Navigation.PushAsync(new DetailPage());
```

Adding a Toolbar to a Page

```
<ContentPage.ToolbarItems>
  <ToolbarItem Text="Start" Command="{Binding StartCommand}">
    <ToolbarItem.Icon>
      <OnPlatform x:TypeArguments="FileImageSource" WinPhone="Toolkit.Content/Play.png" />
    </ToolbarItem.Icon>
  </ToolbarItem>
  <ToolbarItem Text="Stop" Command="{Binding StopCommand}">
    <ToolbarItem.Icon>
      <OnPlatform x:TypeArguments="FileImageSource" WinPhone="Toolkit.Content/Pause.png" />
    </ToolbarItem.Icon>
  </ToolbarItem>
</ContentPage.ToolbarItems>
```

Indicating that a Page is Busy

```
this.IsBusy = true; // "this" refers to page
```

Multipage Apps

Application Lifecycle

- Application class has virtual methods for managing app lifecycle

Method	Description
OnStart	Called when app starts (or when restarted following forced or voluntary termination)
OnResume	Called when app resumes after being suspended (but not if app had to be restarted)
OnSleep	Called when app is deactivated (switched away from)

- Application class also has a property named Properties (Dictionary<string, object>) for storing app state across runs
- Xamarin Forms 1.4 added Application.SavePropertiesAsync method

Saving State When Deactivated

```
// In App.cs
protected override void OnSleep()
{
    Application.Current.Properties["foo"] = 1;
    Application.Current.Properties["bar"] = 2;
}
```

Restoring State When Restarted

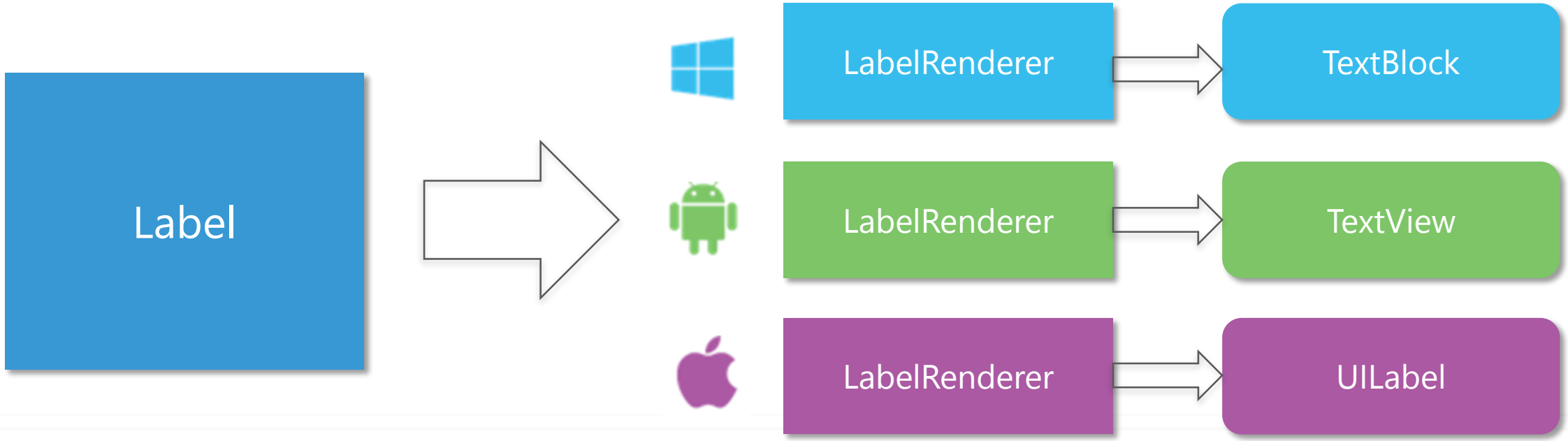
```
// In App.cs
protected override void OnStart()
{
    int foo, bar;

    if (Application.Current.Properties.ContainsKey("foo"))
        foo = (int)Application.Current.Properties["foo"];

    if (Application.Current.Properties.ContainsKey("bar"))
        bar = (int)Application.Current.Properties["bar"];
}
```

DEMO

Application Lifecycle

Custom Renderers

- Renderers are platform-specific classes that render elements into native controls
- Allow existing elements to be modified and new elements to be created



Implementing WrappedTruncatedLabel

```
public class WrappedTruncatedLabel : Label
{
}
```

Using WrappedTruncatedLabel

```
<?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:local="clr-namespace:CustomRendererDemo;assembly=CustomRendererDemo"
             x:Class="CustomRendererDemo.MainPage">

    <ContentView Padding="100">
        <local:WrappedTruncatedLabel Text="{Binding Description}" />
    </ContentView>
</ContentPage>
```

Implementing a Custom Renderer (iOS)

```
[assembly: ExportRenderer(typeof(WrappedTruncatedLabel), typeof(WrappedTruncatedLabelRenderer))]
namespace CustomRendererDemo.iOS
{
    public class WrappedTruncatedLabelRenderer : LabelRenderer
    {
        protected override void OnElementChanged(ElementChangedEventArgs<Label> e)
        {
            base.OnElementChanged(e);

            if (Control != null)
            {
                Control.LineBreakMode = UILineBreakMode.TailTruncation;
                Control.Lines = 0;
            }
        }
    }
}
```

UILabel

DEMO

Custom Renderers

Gesture Recognizers

- TapGestureRecognizer can be attached to XAML elements to respond to taps
  - Attach via GestureRecognizers property
  - Fires Tapped event when tap occurs
  - Or executes command bound to Command property
- NumberOfTapsRequired property (default == 1) specifies number of taps
- Custom gesture recognizers are currently not supported
- To build rich touch interfaces, use custom renderers

Using TapGestureRecognizer

```
// XAML
<Image x:Name="MyImage" Source="Logo.jpg">
  <Image.GestureRecognizers>
    <TapGestureRecognizer Tapped="OnTapped"  />
  </Image.GestureRecognizers>
</Image>

// C#
var recognizer = new TapGestureRecognizer();
recognizer.Tapped += (s, e) =>
{
    // TODO: Respond to single tap
};
MyImage.GestureRecognizers.Add(recognizer);
```

DEMO

Using Custom Renderers to Build Rich Touch Interfaces

Calling Native APIs (DependencyService)

- Xamarin Forms Dependency Service allows platform-specific APIs to be called from shared code
  - Define an interface in shared code
  - Implement the interface in platform-specific projects
  - Register the implementations with Xamarin.Forms.Dependency attribute
  - In shared code, use DependencyService.Get to retrieve interface
- Examples
  - Location APIs
  - Text-to-speech and speech-to-text APIs
  - Any native API that you need to access from shared code

Defining a Platform-Neutral Interface in Shared Code

```
public interface ISimpleLocation
{
    Task<Location> GetCurrentLocationAsync();
}

public class Location
{
    public double Latitude { get; set; }
    public double Longitude { get; set; }
}
```

Implementing a Platform Service (Windows Phone)

```
[assembly: Xamarin.Forms.Dependency(typeof(SimpleLocationProvider))]
namespace LocalWeatherDemo.WinPhone
{
    public class SimpleLocationProvider : ISimpleLocation
    {
        public async Task<Location> GetCurrentLocationAsync()
        {
            var locator = new Geolocator();
            locator.DesiredAccuracy = PositionAccuracy.High;

            try
            {
                var position = await locator.GetGeopositionAsync();
                return new Location() { Latitude = position.Coordinate.Latitude,
                    Longitude = position.Coordinate.Longitude };
            }
            catch (Exception) { return null; }
        }
    }
}
```

Using ISimpleLocation in Shared Code

```
ISimpleLocation locator = DependencyService.Get<ISimpleLocation>();
Location location = await locator.GetCurrentLocationAsync();

if (location != null)
{
    var latitude = location.Latitude;
    var longitude = location.Longitude;
}
```

Download the Code

RPN Calculator
http://1drv.ms/1EYM0tl

RPN Calculator with rounded buttons
http://1drv.ms/1b6wBxu

Contoso Cookbook
http://1drv.ms/1GcuFNT

Contoso Cookbook (Azure version)
http://1drv.ms/1xkKai2

Contoso Cookbook with wrapped, truncated text
http://1drv.ms/1b6wWjz

MonoLife
http://1drv.ms/1b6wXUS