Visual Studio LIVE! | LasVegas
EXPERT SOLUTIONS FOR .NET DEVELOPERS

HOL02 – HOL Xamarin
Breaking through abstractions

Roy Cornelissen &
Marcel de Vries

Xpirit

Respect the Past.
Code the Future.

Level: Introductory - Intermediate

Visual Studio LIVE! 25 YEARS OF CODING INNOVATION

Xamarin.Forms elements

- Xamarin.Forms allows you to define your UI using a set of elements that are common across all platforms

Button is available everywhere →

```
public class Button : Element
{
    public Color  BorderColor  { get; set; }
    public int    BorderRadius { get; set; }
    public double BorderWidth  { get; set; }
    public string Text         { get; set; }
    public Color  TextColor    { get; set; }
    ...
}
```

Xamarin.Forms elements are models

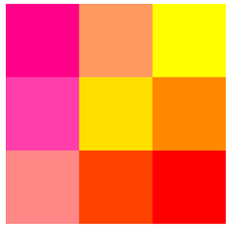
- Elements provide a *representation* of the UI we want to create and display

Properties let you customize runtime visuals and behavior


```
public class Button : Element
{
    public Color  BorderColor  { get; set; }
    public int    BorderRadius { get; set; }
    public double BorderWidth  { get; set; }
    public string Text         { get; set; }
    public Color  TextColor    { get; set; }
    ...
}
```

Customizing elements

- Changing the properties of Xamarin.Forms elements allows for limited customization – which may or may not be sufficient for your needs



Can change most colors

Can adjust position + width/height

Can add background images into views

Can control fonts

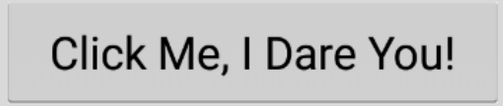
From Element to Visual

- At runtime, a platform-specific control is created to visualize each Xamarin.Forms element

```
public class Button : Element
{
    public Color  BorderColor  { get; set; }
    public int    BorderRadius { get; set; }
    public double BorderWidth  { get; set; }
    public string Text         { get; set; }
    public Color  TextColor    { get; set; }
    ...
}
```

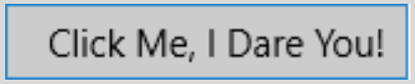
Shared

Android.Widget.Button



UIKit.UIButton

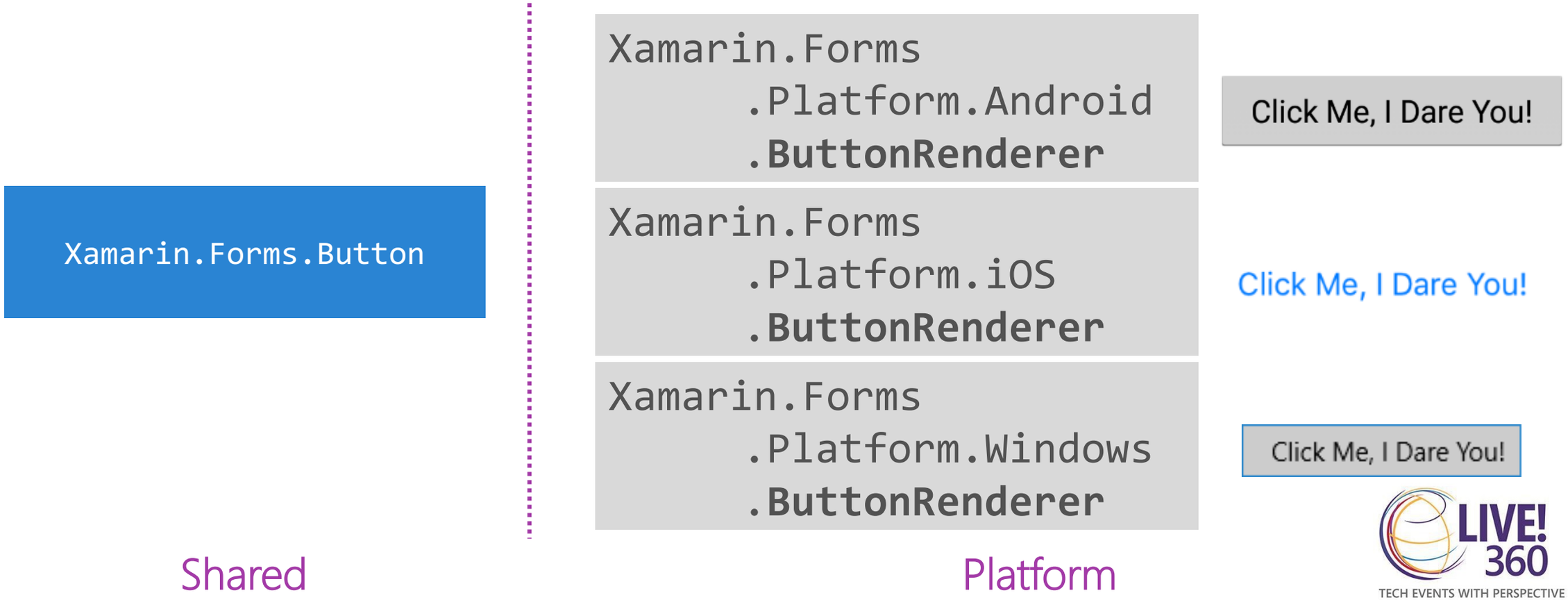
Windows.UI.Xaml.Controls.Button



Platform

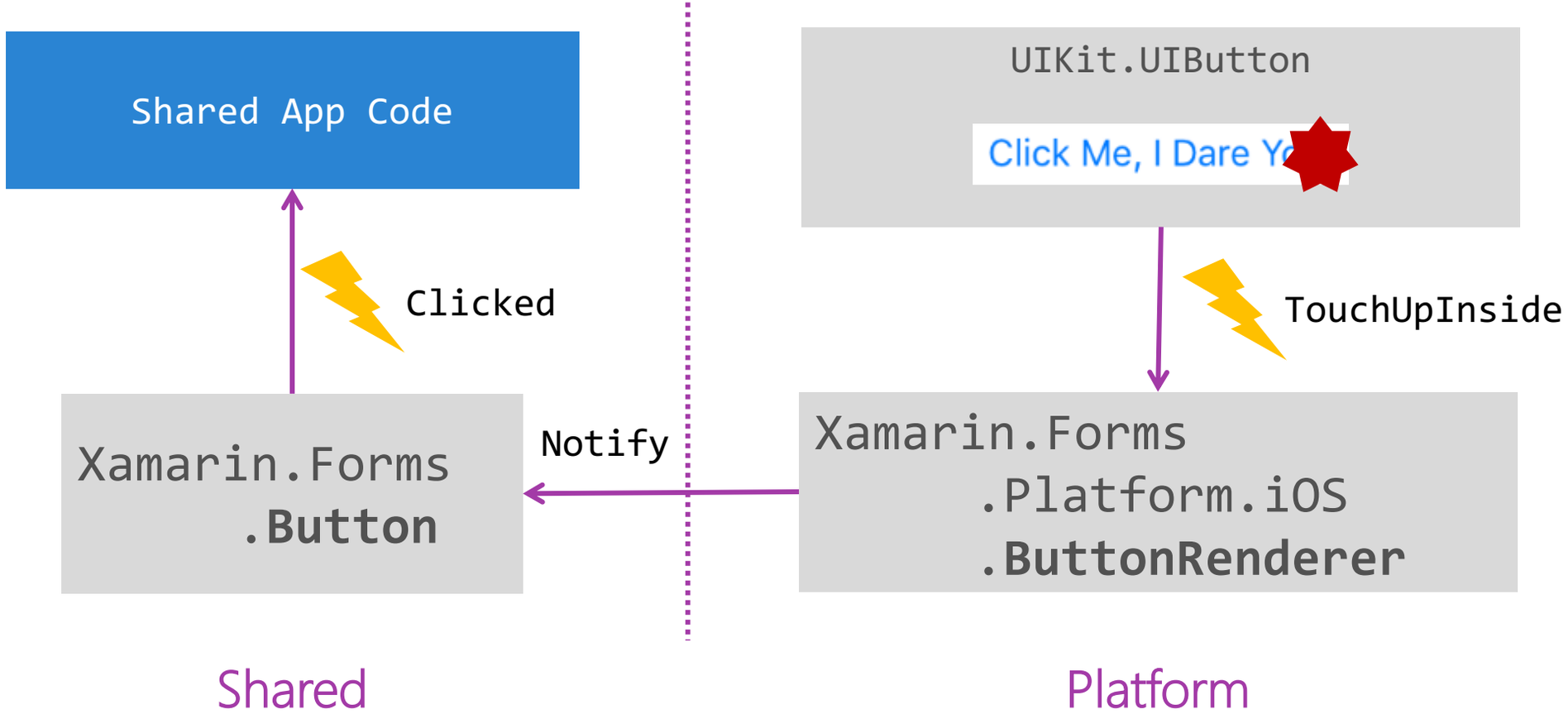
Platform renderers

- The *platform renderer* is the code that translates Xamarin.Forms elements to a platform native control



From Visual to Element

- The renderer is responsible for **watching** the native control notifications and **forwarding** them to the Xamarin.Forms element



Customization

Platform Themes

Effects

Native Embedding

Custom Renderers

Platform Themes

- Each platform has an API you can use to control the native visual appearance of your app

`UIAppearance API`

`android:theme`

`Style + ControlTemplate`

Style + ControlTemplate

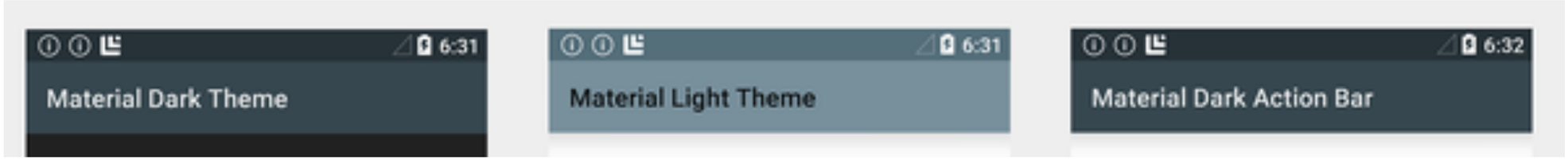
- Each Windows XAML control has a default style and control template – these can be modified to customize appearance and behavior

```
<?xml version="1.0" encoding="UTF-8" ?>
<Application.Resources>
   <Style TargetType="TextBlock">
      <Setter Property="Foreground" Value="Yellow" />
      <Setter Property="FontFamily" Value="Verdana" />
      <Setter Property="FontSize" Value="96" />
   </Style>
</Application.Resources>
```

Native Windows **Style**s will affect controls created by the Xamarin.Forms renderer

Themes

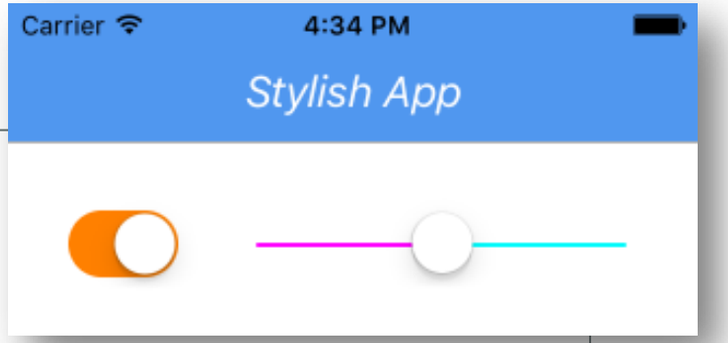
- Android Themes determine the look and feel of views and activities; there are built in themes and you can create custom themes




```
[Activity (Label = "DroidThemes",
           Theme = "@android:style/Theme.Material.Light",
           MainLauncher = true, Icon = "@mipmap/icon")]
public class MainActivity : Activity
{
    ...
}
```

Appearance API

- The iOS Appearance API lets you define visual settings at a class level that apply to all instances of that type




```
public override bool FinishedLaunching(...)
{
      UISwitch.Appearance.OnTintColor = UIColor.Orange;
      UISlider.Appearance.MinimumTrackTintColor = UIColor.Magenta;
      UISlider.Appearance.MaximumTrackTintColor = UIColor.Cyan;

      UINavigationBar.Appearance.BarTintColor = UIColor.FromRGB(51, 134, 238);
      UINavigationBar.Appearance.SetTitleTextAttributes(new UITextAttributes()
      { TextColor = UIColor.White, Font = UIFont.ItalicSystemFontOfSize(20)});
}
```

Effects

❖ The **Effects API** lets your code *tweak* the visual appearance and behavior of the native controls generated by the renderer

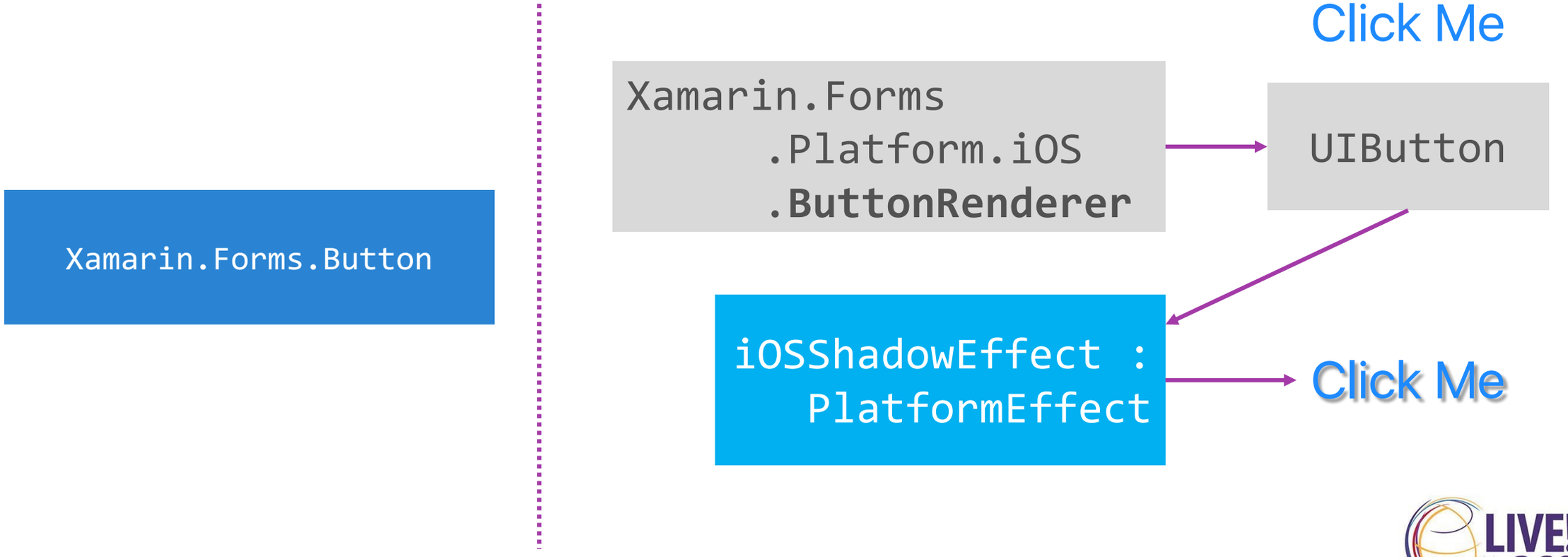
- Change properties not exposed by X.F.
- Access platform features (e.g. shadows)
- Handle native control notifications
- Add or remove visual children

What is an Effect?

- An *effect* is a platform-specific class that uses the native APIs to change the appearance and behavior of the native control that underlies a Xamarin.Forms Element

Xamarin.Forms Effects API

- The Effects API allows you to interact with and change properties on the controls created by the native renderers



One effect per platform

- The author of an effect implements one class for each platform they choose to support

```
public class ShadowEffect : RoutingEffect
{
    ...
}
```

```
public class AndroidShadowEffect :  ...
{
    ...
}
```

```
public class iOSShadowEffect :  ...
{
    ...
}
```

```
public class UWPShadowEffect :  ...
{
    ...
}
```

Shared

Platform

Effect implementation

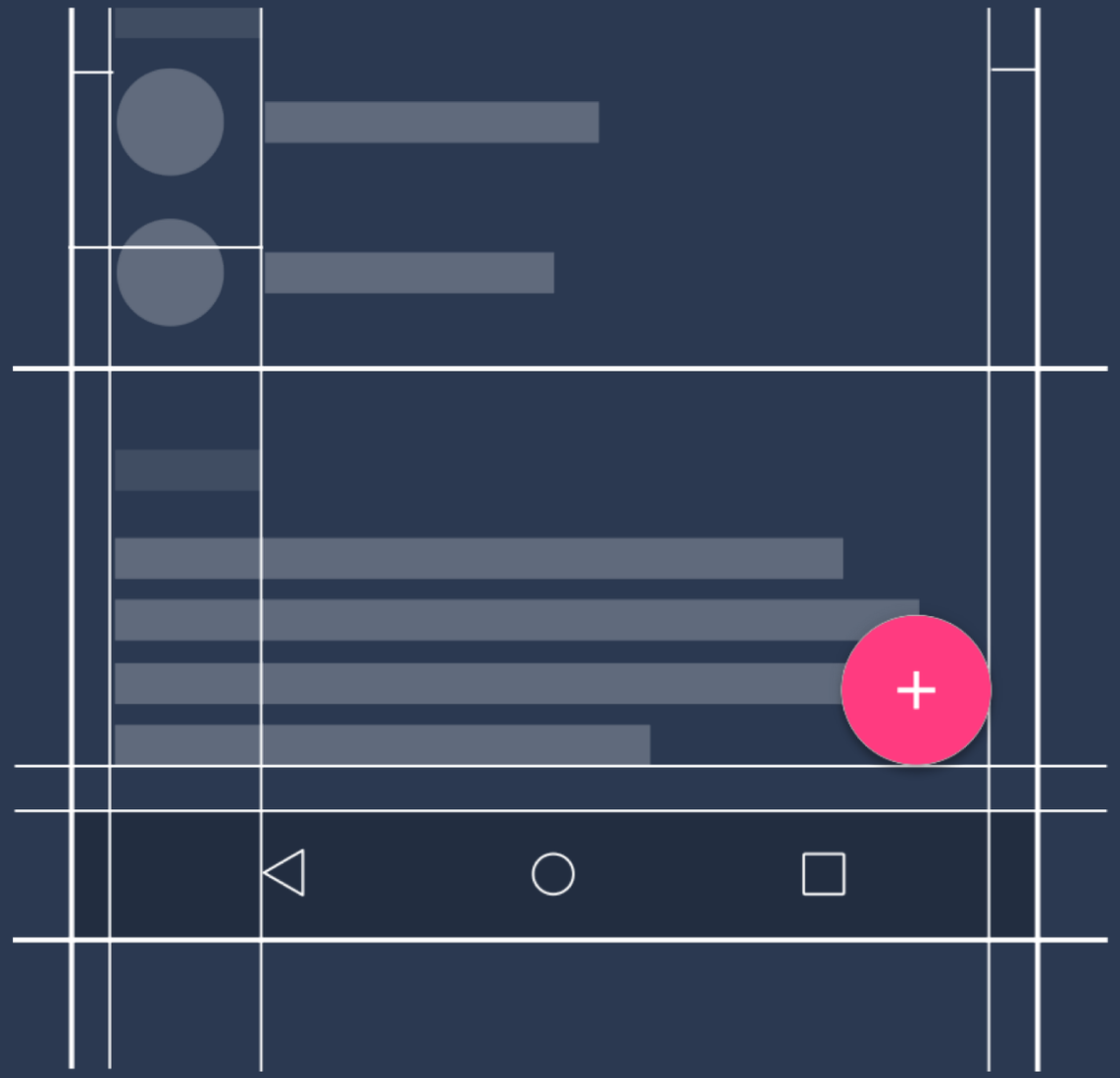
```
using System.ComponentModel;
using EffectsTest.iOS;
using Xamarin.Forms;
using Xamarin.Forms.Platform.iOS;

[assembly: ResolutionGroupName("MyCompany")] // prevent naming collisions
[assembly: ExportEffect(typeof(MyEffect), "MyEffect")]
namespace EffectsTest.iOS
{
    public class MyEffect : PlatformEffect
    {
        protected override void OnAttached()
        {
        }

        protected override void OnDetached()
        {
        }

        protected override void OnElementPropertyChanged(PropertyChangedEventArgs args)
        {
        }
    }
}
```

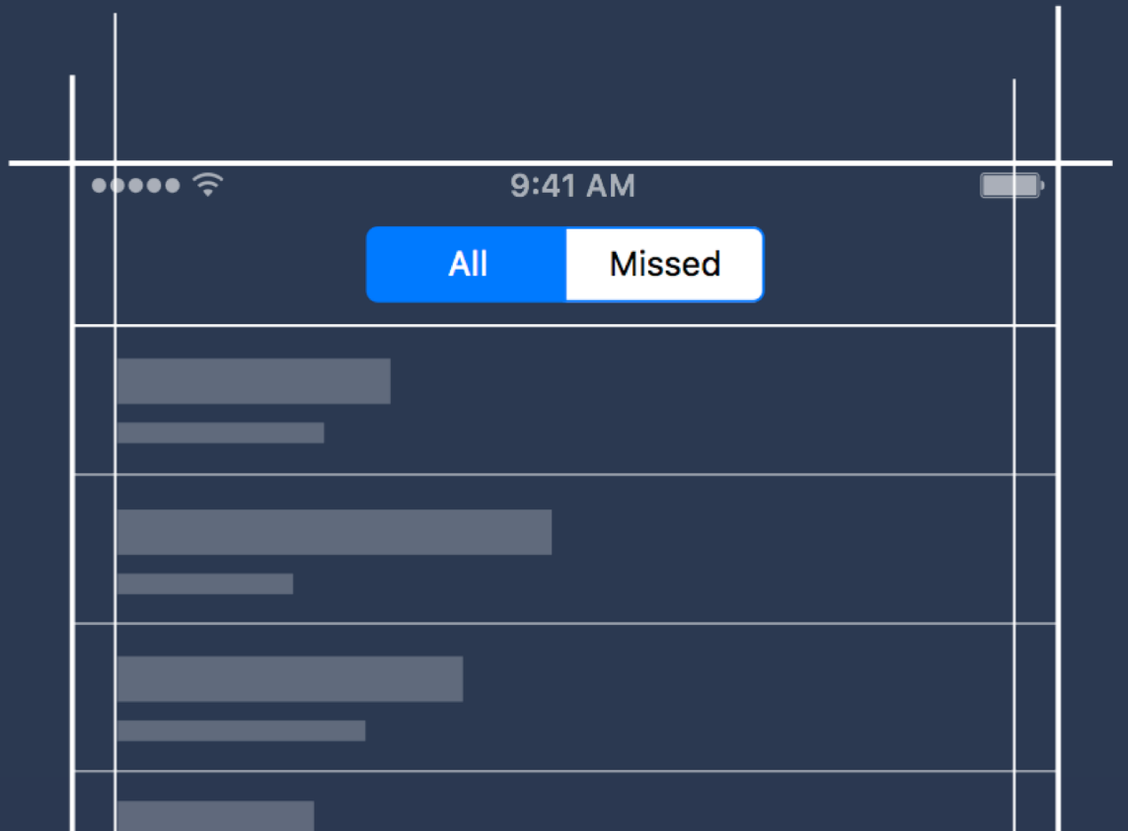
Using an effect and platform themes

Lab - app-quotes

Lab04 – exercises 1 & 2

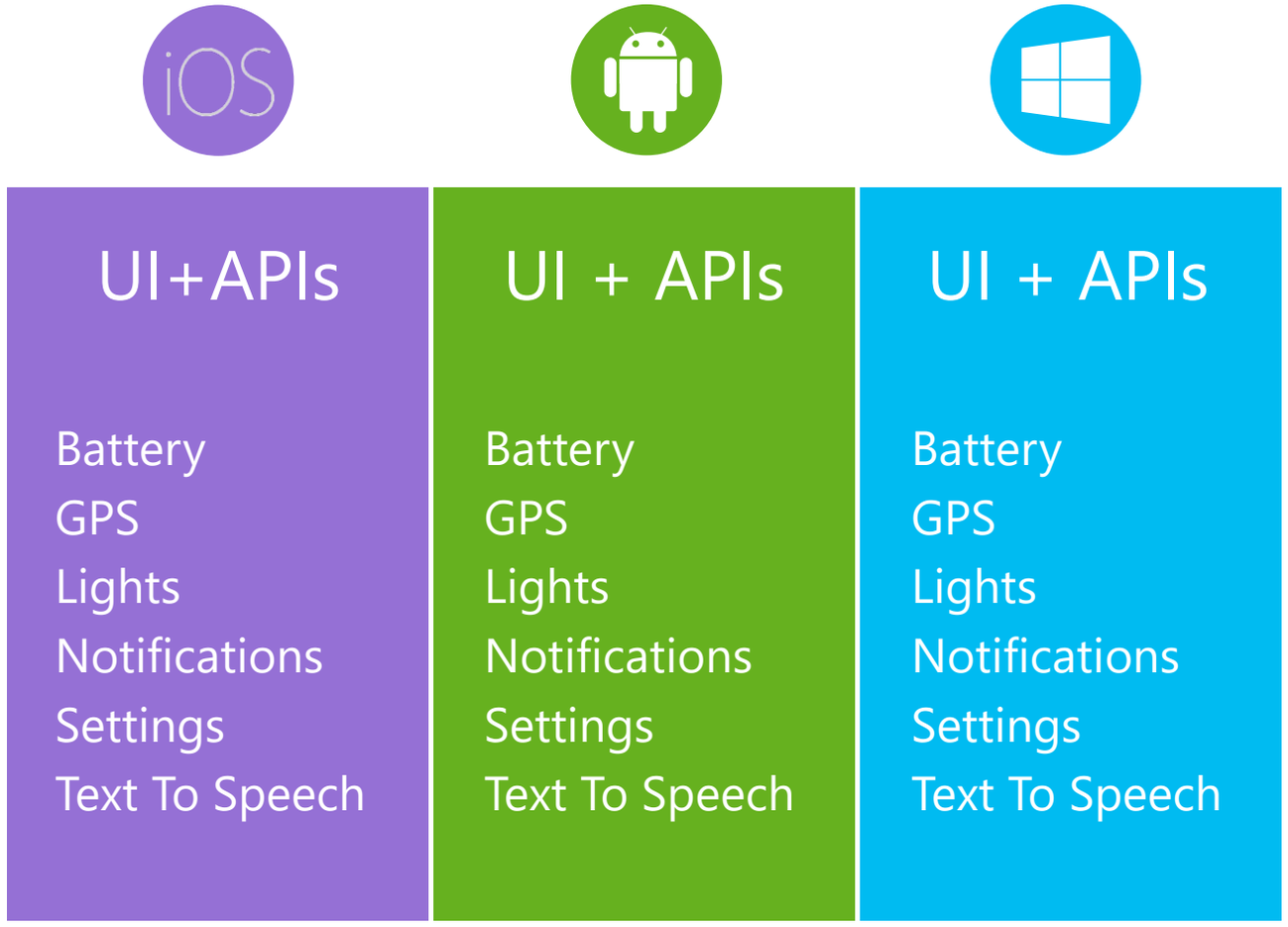
Native Embedding





Native Embedding

```
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:ios="clr-namespace:UIKit;assembly=Xamarin.iOS;targetPlatform=iOS"
5               xmlns:androidWidget="clr-namespace:Android.Widget;assembly=Mono.Android;targetPlatform=A
6               xmlns:formsandroid="clr-namespace:Xamarin.Forms;assembly=Xamarin.Forms.Platform.Android;
7               xmlns:win="clr-namespace:Windows.UI.Xaml.Controls;assembly=Windows, Version=255.255.255.
                ContentType=WindowsRuntime;targetPlatform=Windows"
8               x:Class="NativeViewDeclaration.NativeViewDeclarationPage">
9      <ContentPage.Content>
10         <ios:UILabel Text="Native Text" View.HorizontalOptions="Start"/>
11         <androidWidget:TextView Text="Native Text" x:Arguments="{x:Static formsandroid:Forms.Context}
12         <win:TextBlock Text="Native Text"/>
13     </ContentPage.Content>
14 </ContentPage>
```



Device Capabilities

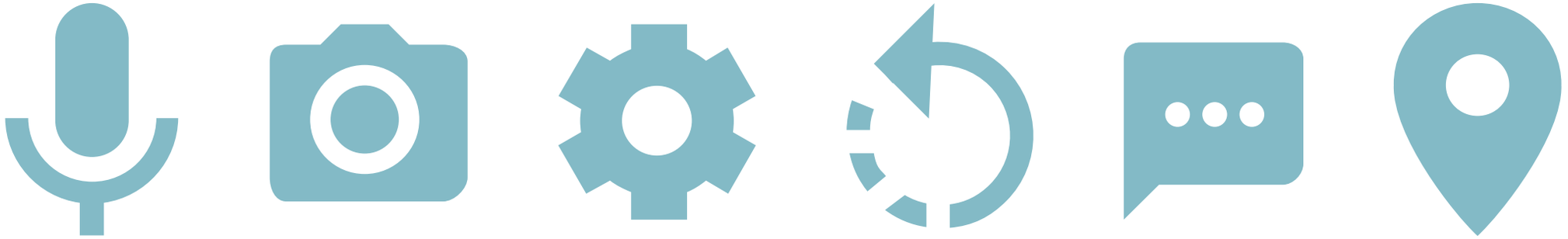
What if we didn't have to write this code?

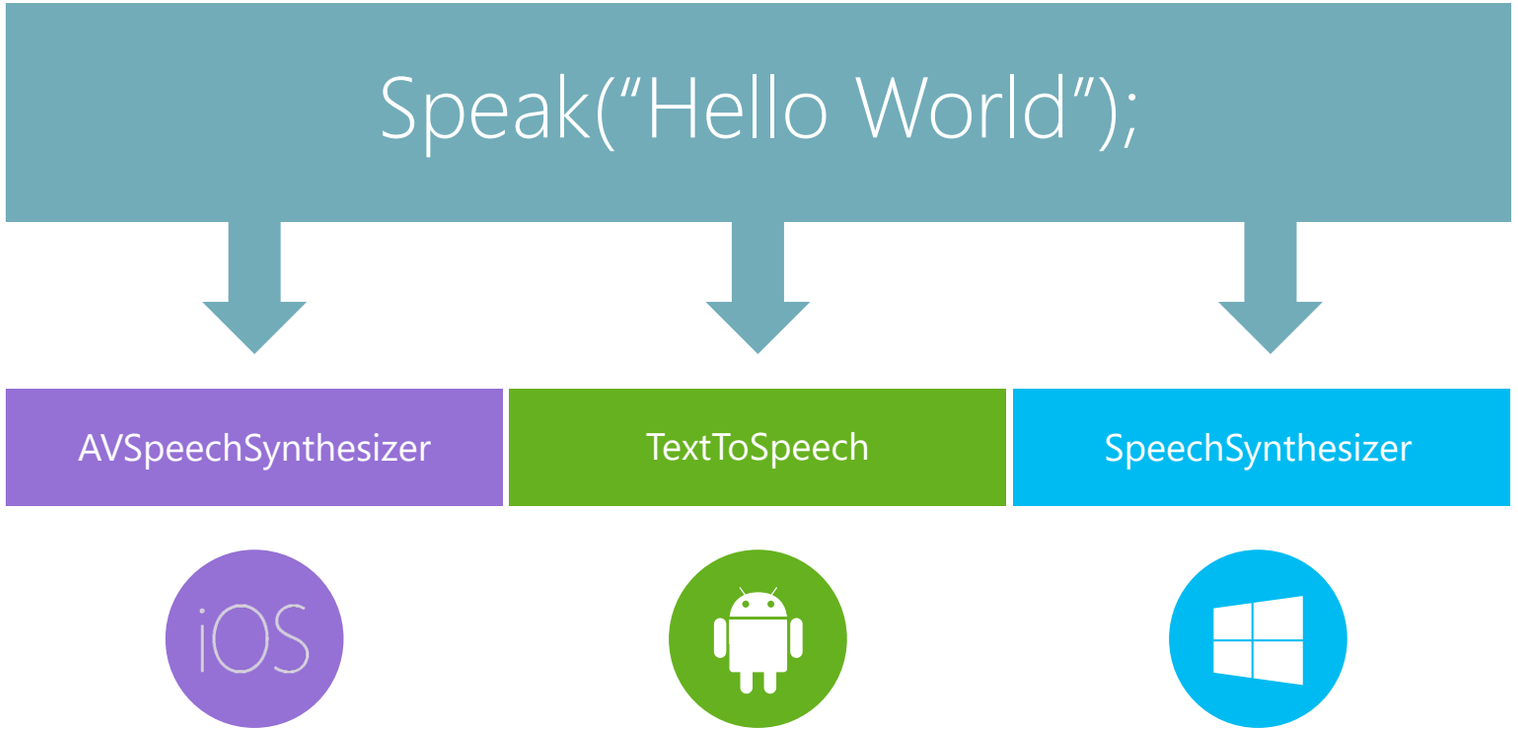
What if we could access it from shared code?

Plugins for Xamarin & Windows

Xamarin.com/plugins

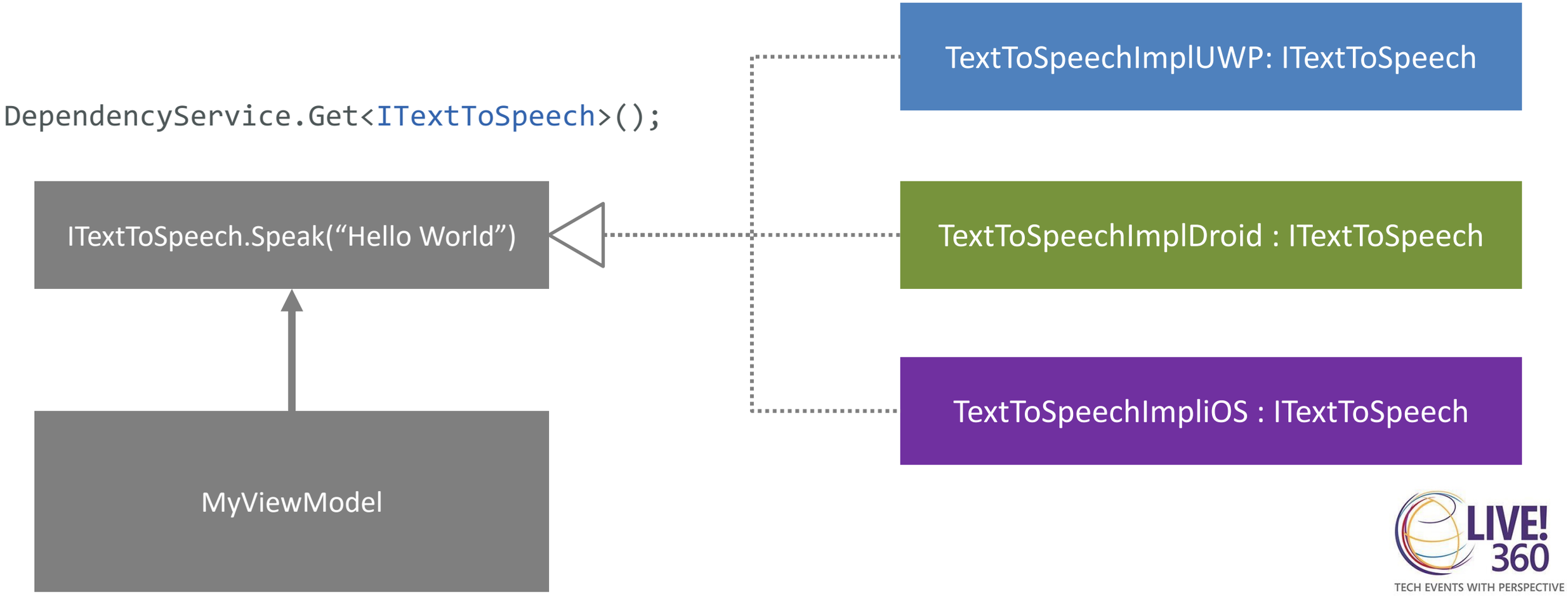
Common API





Device services with IoC

```
[assembly: Dependency (typeof (TextToSpeechImpl))]
```




Nuget
Search for:
“Plugin for Xamarin”

Adding Text-To-Speech

Lab – app-quotes

Lab04 – exercises 3 & 4