Introduction to Xamarin.Forms

Enterprise & Mobile .Net

DE HOGESCHOOL
MET HET NETWERK

Hogeschool PXL – Elfde-Liniestraat 24 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

Agenda

- Introduction to Xamarin Forms
  - Project structure
- Pages
- Views and layouts
- Lists
- Navigation
- Platform features

PXL IT

Introduction to Xamarin Forms

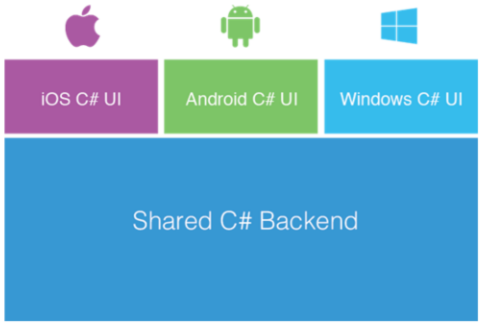
DE HOGESCHOOL
MET HET NETWERK

Hogeschool PXL – Elfde-Liniestraat 24 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

Xamarin Forms is a new set of APIs allowing you to quickly and easily write shared User Interface code that is still rendered natively on each platform, while still providing direct access to the underlying SDKs if you need it.

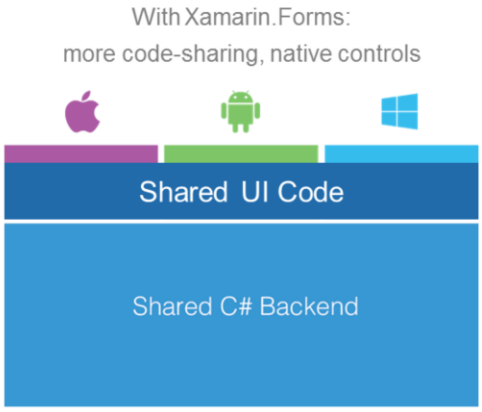
So far with regular Xamarin

- UI is platform specific, tuned to the platform
- App logic is shared
  - PCL
  - Shared Projects
- 70-90% shared code on average



With Xamarin.Forms

- With Xamarin.Forms, we get an API which allows to build sharable user interface code
- Still renders native UI on IOS, Android and Windows Phone
- UI + logic written in C#, allowing 90-99% code sharing



So what is Xamarin.Forms really?

- *Xamarin.Forms is a cross-platform natively backed UI toolkit abstraction that allows developers to easily create user interfaces that can be shared across Android, iOS, and Windows Phone*
  - Allows rapid development of cross-platform Uis
  - Abstraction on top of UI elements
  - More code sharing than ever before
  - Still native apps
  - Still native performance and look and feel

PXL IT

Xamarin.Forms is a cross-platform natively backed UI toolkit abstraction that allows developers to easily create user interfaces that can be shared across Android, iOS, and Windows Phone. The user interfaces are rendered using the native controls of the target platform, allowing Xamarin.Forms applications to retain the appropriate look and feel for each platform. This guide will provide a quick introduction to Xamarin.Forms and how to get started writing applications with it.

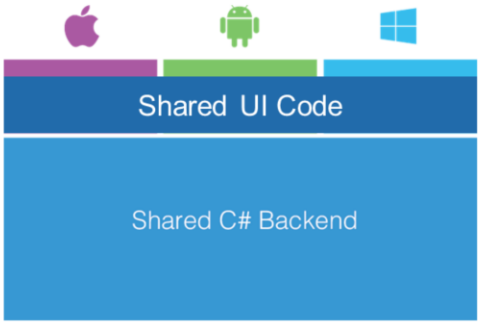
Xamarin.Forms is a framework that allows developers to rapidly create cross platform user interfaces. It provides it's own abstraction for the user interface that will be rendered using native controls on iOS, Android, or Windows Phone. This means that applications can share a large portion of their user interface code and still retain the native look and feel of the target platform.

So what is Xamarin.Forms really?

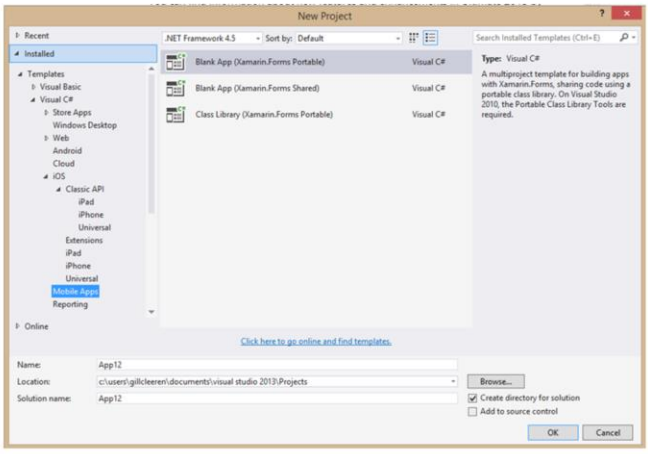
- All Forms app have to be written in C#
- RAD but can evolve to more complex apps
  - API they can access is the same as “plain Xamarin”
- Commonly built on top of PCL or shared project layers
  - Consumed by Xamarin.Forms code
- UIs can be built with XAML (with code behind) or fully in code

Main features

- RAD, forms-based app creation
- 40+ Pages, Layouts, and Controls
  - Build from code behind or XAML
- Two-way Data Binding
- Navigation
- Animation API
- Dependency Service
- Messaging Center
- Support for
  - Android 4.0+
  - iOS 6.1+
  - Windows Phone 8.0+ (Silverlight only) and UWP



- XS and VS have Xamarin.Forms templates
  - Based on Shared project or PCL
  - Each contains WP8 (VS only), iOS and Android project



As discussed above, Xamarin.Forms is implemented as a .NET Portable Class Library (*PCL*), which makes it very easy to share the Xamarin.Forms API's across a variety of platforms. The first step to getting started is to create a solution for the various projects that will make up the application.

A Xamarin.Forms solution can be created in Xamarin Studio or Visual Studio and will typically contain the following projects:

Portable Library - This project is the cross platform application library that holds all of the shared code and share UI.

Xamarin.Android Application - This project holds Android specific code and is the entry point for Android applications.

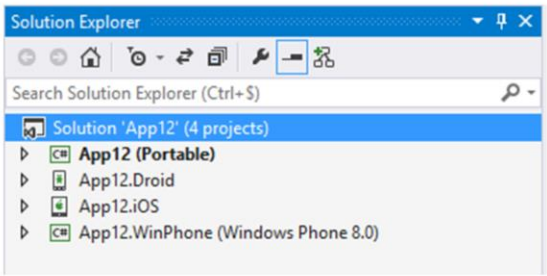
Xamarin.iOS Application - This project holds iOS specific code and is the entry point for iOS applications.

Windows Phone Application - This project holds the Windows Phone specific code and is the entry point for Windows Phone applications.

Xamarin 3.0 provides templates that will create a complete solution with all of the necessary projects for a Xamarin.Forms application.

Getting started

- Resulting solution:



  - Note that XS doesn't support the WP8 currently
    - Will open but not load/compile
- Shared Project or PCL holds shared code, used by all other projects
- iOS, Andriod and WP8 projects contain platform-specific UI code

PXL IT

Solutions created with Xamarin Studio do *not* include a Windows Phone project. Use Visual Studio to create new Xamarin.Forms solutions that support iOS, Android as well as Windows Phone. Note that while Xamarin Studio does not support the creation of Windows Phone applications they can still be loaded as a part of an existing solution (ie. one that was created with Visual Studio). This allows you to browse your Windows Phone code in Xamarin Studio, even though it cannot be built or deployed.

Portable Class Library (PCL)

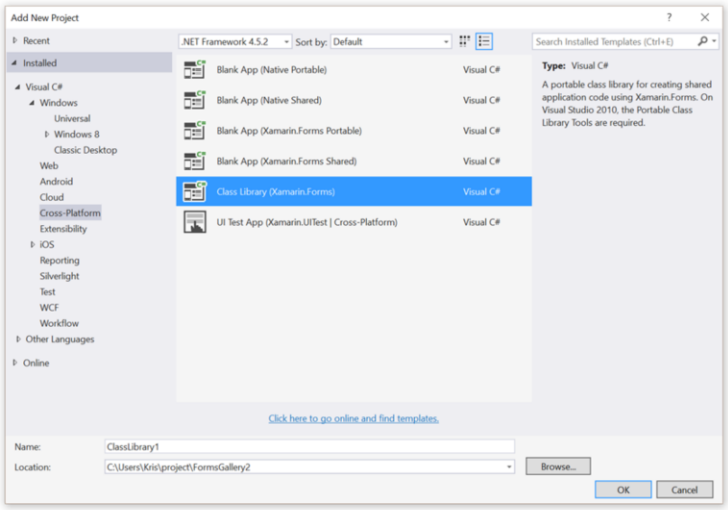
- Targets a special kind of library (dll)
- You can use a subset of the .NET framework and other PCL
- Supports crossplatform development
- Several targets exists (Xamarin, Windows 8, …)
- Platform specific features are realized by means of Dependency Injection

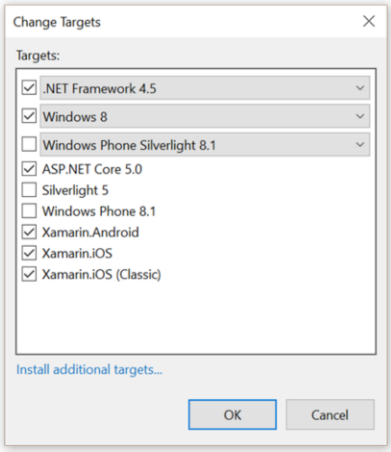
PXL IT

Dependency Injection:

Suppose you want a TextToSpeech functionality, but this is implemented differently in iOS and Android. You define a common interface to which you implement in the PCL. This interface is implemented in the platform specific projects (twice: iOS and Android).

Portable Class Library





PXL IT

If you add a PCL to an existing Xamarin.Forms project, choose Cross-Platform > Class Library (Xamarin.Forms)

In the properties of the project, you could change the targets

Shared Project

- Another way to share code between projects
- Files in this type of project get copied into the main project and are compiled/linked into the binary
- Conditional compilation makes is possible to differentiate between platforms
  - Ugly code
  - Not recommended

Creating our first Xamarin.Forms app

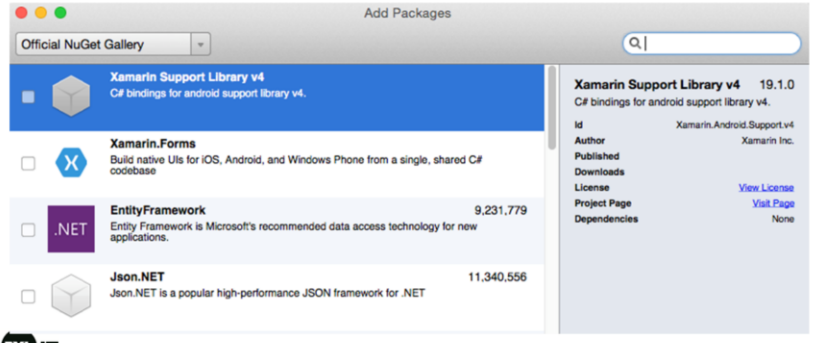
DEMO

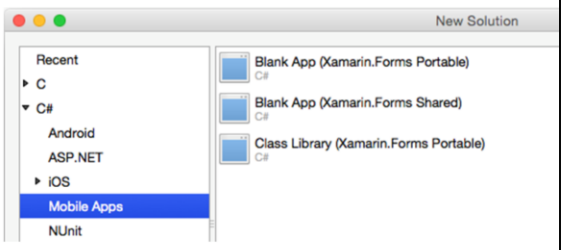
Create new app, and run => HelloXamarin

Depending on the IDE (Xamarin Studio or Visual Studio) and the development OS (Windows or Mac) there will be the possibility to create Android or iOS projects.

How does Xamarin.Forms work then?

- Based on PCL or shared project
- NuGet package

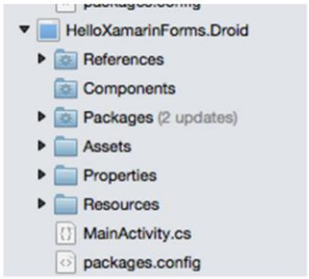




App plays an important role

```
public class App : Application
{
    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!"
                    }
                }
            }
        };
    }
```

In Android

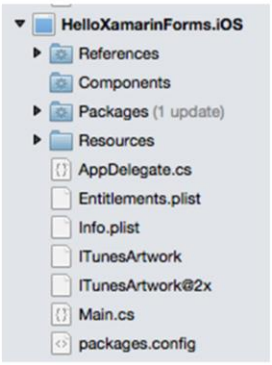

HelloXamarinForms.Droid
References
Components
Packages (2 updates)
Assets
Properties
Resources
MainActivity.cs
packages.config

Zooming in on MainActivity

- Create an activity with MainLauncher to true
- Activity must inherit from Xamarin.Forms.Platform.Android.FormsApplicationActivity
- Must initialize Xamarin.Forms
- Display Page in OnCreate

```
[Activity (Label = "XamFormsTest.Droid", Icon = "@drawable/icon", MainLauncher = true, ConfigurationChanges = ConfigChanges.ScreenSize | ConfigChanges.Orientation)]
public class MainActivity : global::Xamarin.Forms.Platform.Android.FormsApplicationActivity
{
   protected override void OnCreate (Bundle bundle)
   {
       base.OnCreate (bundle);
       global::Xamarin.Forms.Forms.Init (this, bundle);
       LoadApplication (new App ());
   }
}
```

In iOS


HelloXamarinForms.iOS
References
Components
Packages (1 update)
Resources
AppDelegate.cs
Entitlements.plist
Info.plist
ITunesArtwork
ITunesArtwork@2x
Main.cs
packages.config

PXL IT

Launching iOS

- AppDelegate must initialize Xamarin.Forms
  - Call LoadApplication, passing in App instance
  - All in FinishedLaunching

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    Forms.Init();

    window = new UIWindow(UIScreen.MainScreen.Bounds);

    LoadApplication (new App ());

    return base.FinishedLaunching (app,options);
}
```

PXL IT

As with the Android application, the first step in the FinishedLaunching event is to initialize the Xamarin.Forms framework with a call to Xamarin.Forms.Forms.Init(). This step causes the iOS specific implementation of Xamarin.Forms to be globally loaded in the application. The next step is set the root view controller of the application. This is done by invoking the CreateViewController() method on an instance of the HelloWordPage class that we created in the cross platform application library.

And let's see that again…

DEMO

Create new app, and run

This time: Phoneword_XamarinForms

PXL IT

Pages, Controls and some other things ☺

DE HOGESCHOOL
MET HET NETWERK

Hogeschool PXL – Elfde-Liniestraat 24 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

Important building blocks in Xamarin.Forms

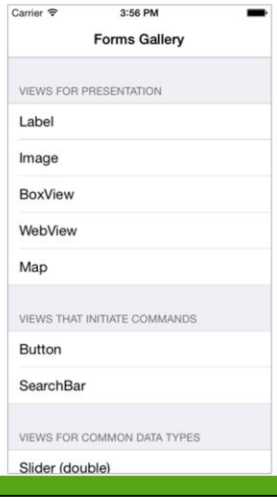
- 4 main classes are used for composing the Xamarin.Forms UI
  - **Page**
    - Xamarin.Forms represents a single screen in your application
      - Analagous to an Android Activity, a Windows Phone Page, or an iOS View Controller
  - **Layout**
    - Specialized View subtype
      - Meant to act as a container for other Layout or Views
      - Layout subtypes typically contain logic that is specific to organizing the child views in a certain way
  - **View**
    - controls or widgets in other platforms
      - UI elements such as labels, buttons, text fields, etc
  - **Cell**
    - Specialized element that is used for items in a list or a table
    - It describes how each item in a list should be drawn

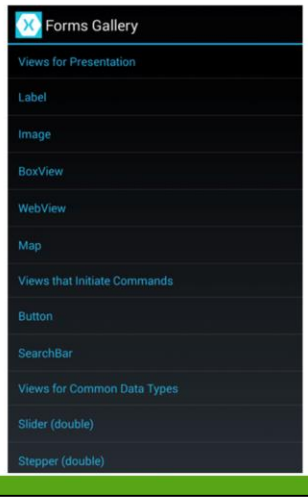
It all starts with a Page

- Page is a visual element that occupies most or all of the screen and contains a single child
  - Activity in Android
  - View Controller in iOS
  - Page in WP
- Available pages
  - ContentPage
  - MasterDetailPage
  - NavigationPage
  - TabbedPage
  - CarouselPage

Pages: ContentPage

- A Content Page displays a single View
  - Commonly a StackLayout or a ScrollView





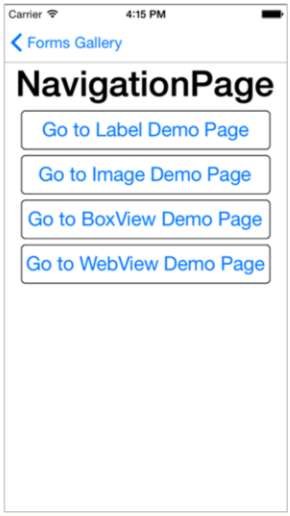
PXL IT

Pages: Master Detail page

- A Page that manages two panes of information

Pages: NavigationPage

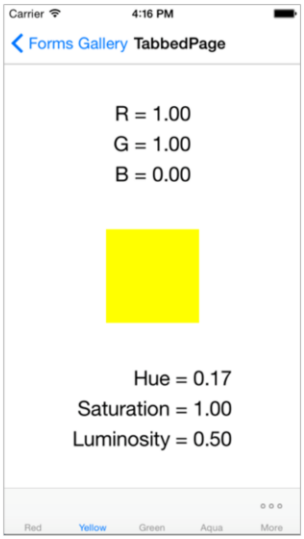
- A Page that manages the navigation and user-experience of a stack of other pages



PXL IT

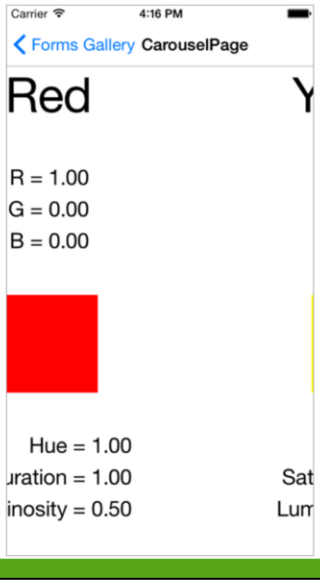
Pages: TabbedPage

- A Page that allows navigation between children pages, using tabs



Pages: CarouselPage

- A Page allowing swipe gestures between subpages, like a gallery



Pages

DEMO

Demo: Forms Gallery contains demos for all possible controls

Start the App and go to the Pages section

Note: this demo uses NO XAML, everything is coded in C#

Hosted on github: https://developer.xamarin.com/samples/xamarin-forms/FormsGallery/

PXL IT

Views and layouts

DE HOGESCHOOL
MET HET NETWERK

Hogeschool PXL – Elfde-Liniestraat 24 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

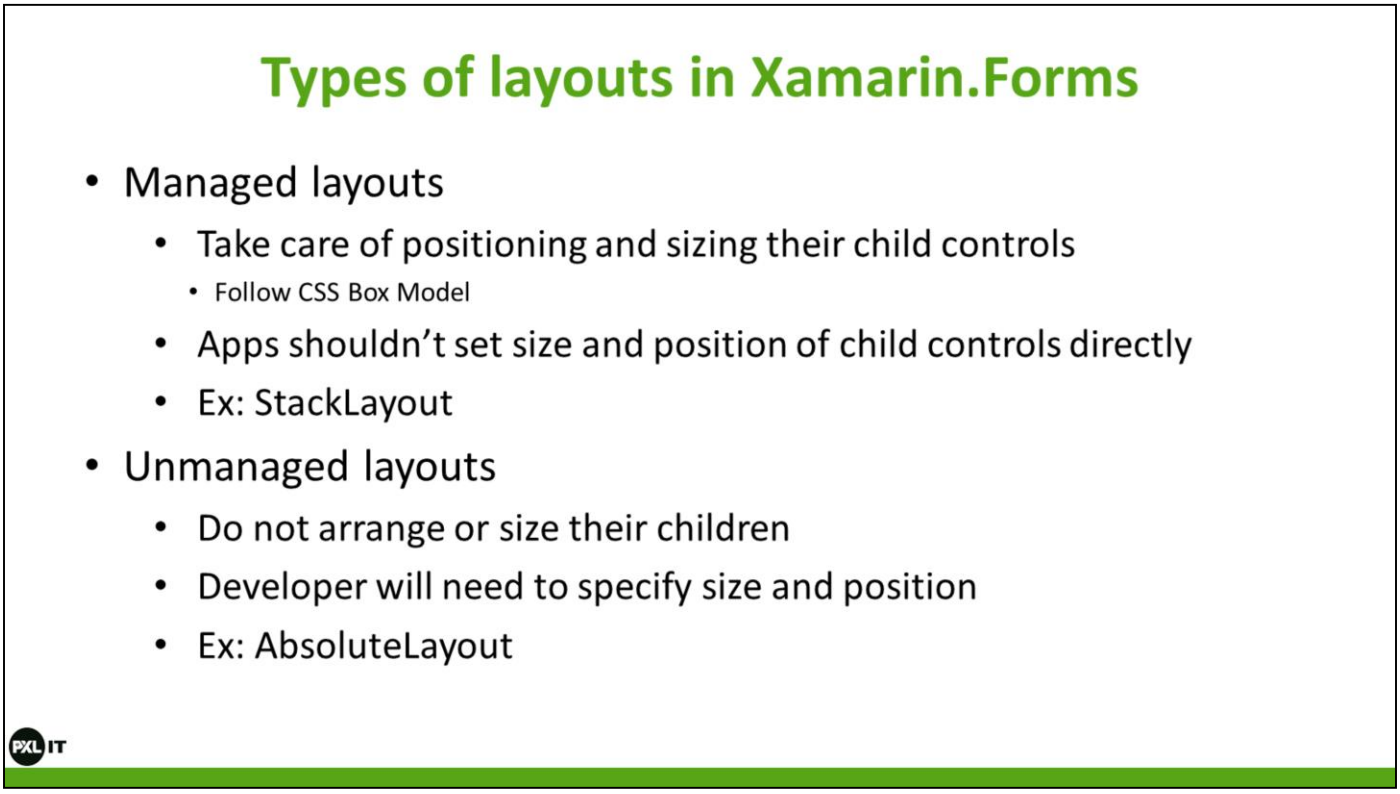
Reminder: Views are Controls

Typical views (controls)

Xamarin.Forms Control	Description
Label	The Label is a read-only text display control
Entry	An Entry is a simple single-line text-input control
Button	Buttons are used to initiate commands
Image	This control is used to display a bitmap
ListView	The ListView presents a scrolling list of items. The items inside a list are known as cells

Layouts in Xamarin.Forms

- Layout Containers organize their child elements based on specific rules
  - StackLayout
  - Grid
  - ScrollView
  - RelativeLayout
  - AbsoluteLayout



Controls themselves are hosted inside of a layout. Xamarin.Forms has two different categories of layouts that arrange the controls in very different ways:

Managed Layouts - these are layouts that will take care of positioning and sizing child controls on the screen and follow the CSS Box Model. Applications should not attempt to directly set the size or position of child controls. One common example of a managed Xamarin.Forms layout is the StackLayout.

Reference: http://www.w3schools.com/css/css_boxmodel.asp

Unmanaged Layouts - as opposed to managed layouts, unmanaged layouts will not arrange or position their children on the screen. Typically, the user will specify the size and location of the child control as it is being added to the layout. The AbsoluteLayout is an example of an unmanaged layout control.

StackLayout

- Very common managed layout
  - Arranges controls automatically, no matter what the screen size
  - Positioned horizontally or vertically next to each other in order they were added
  - Spacing is determined by HorizontalOptions and VerticalOptions
    - Default is using entire screen

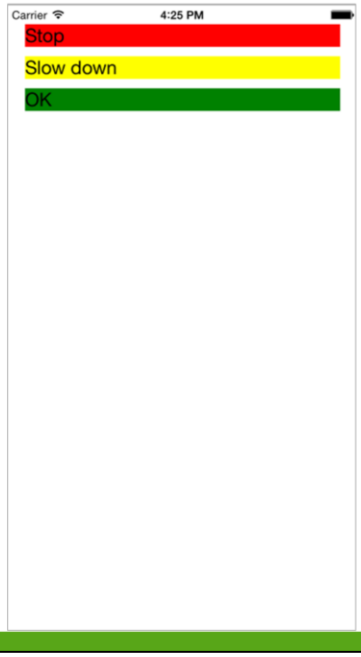
In code

```
public class StackLayoutExample: ContentPage
{
    public StackLayoutExample()
    {
        Padding = new Thickness(20);
        var red = new Label
        {
            Text = "Stop",
            BackgroundColor = Color.Red,
            Font = Font.SystemFontOfSize (20)
        };
        var yellow = new Label
        {
            Text = "Slow down",
            BackgroundColor = Color.Yellow,
            Font = Font.SystemFontOfSize (20)
        };

        Content = new StackLayout
        {
            Spacing = 10,
            Children = { red, yellow}
        };
    }
}
```

In XAML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             x:Class="HelloXamarinFormsWorldXaml.StackLayoutExample1"
             Padding="20">

  <StackLayout Spacing="10">

    <Label Text="Stop"
           BackgroundColor="Red"
           Font="20" />

    <Label Text="Slow down"
           BackgroundColor="Yellow"
           Font="20" />

    <Label Text="Go"
           BackgroundColor="Green"
           Font="20" />

  </StackLayout>
</ContentPage>
```

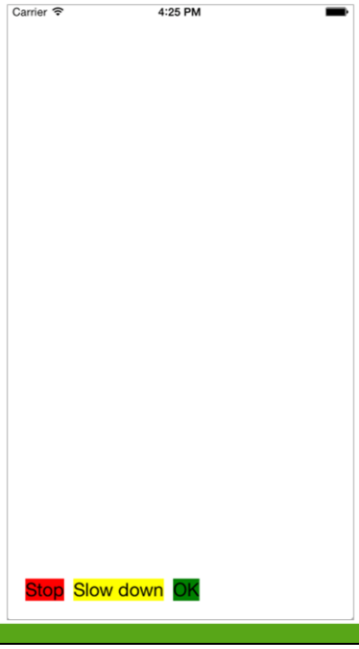
The result



Different orientation

```
public class StackLayoutExample: ContentPage
{
    public StackLayoutExample()
    {
        // Code that creates labels removed for clarity

        Content = new StackLayout
        {
            Spacing = 10,
            VerticalOptions = LayoutOptions.End,
            Orientation = StackOrientation.Horizontal,
            HorizontalOptions = LayoutOptions.Start,
            Children = { red, yellow, green }
        };
    }
}
```

The result

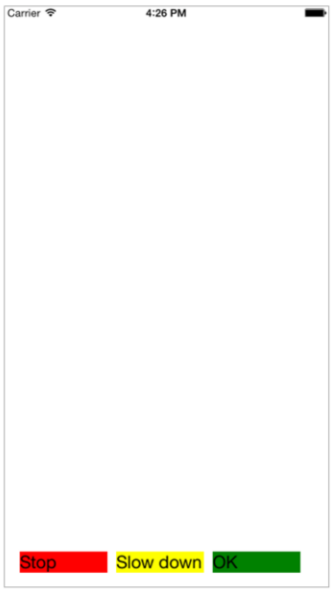


Setting width and height

- Setting size isn't possible, however, we can specify WidthRequest and HeightRequest

```
var red = new Label
{
    Text = "Stop",
    BackgroundColor = Color.Red,
    Font = Font.SystemFontOfSize (20),
    WidthRequest = 100
};
var yellow = new Label
{
    Text = "Slow down",
    BackgroundColor = Color.Yellow,
    Font = Font.SystemFontOfSize (20),
    WidthRequest = 100
};
var green = new Label
{
    Text = "Go",
    BackgroundColor = Color.Green,
    Font = Font.SystemFontOfSize (20),
    WidthRequest = 200
};
```

The result



Managing spacing

- VerticalOptions/HorizontalOptions
  - Sets how the child content is stretched or positioned
  - Used mostly on containers
- Spacing
  - Spacing added between child elements
  - Used on StackLayout
- Padding
  - Padding around the element

Managing width and height

- WidthRequest/HeightRequest
  - Request a specific width/height
  - Overrides the measured width and height
- MinimumWidthRequest/MinimumHeightRequest
  - Absolute minimum for the control
- Width/Height
  - Read-only
  - Final calculated width and height
- Bounds
  - Read-only
  - Position and size of the frame (of the control) relative to its parent

StackLayout

DEMO

AbsoluteLayout

- Unmanaged container: all controls must be explicitly get a position within the layout
  - Similar to iOS AbsoluteLayout or WinForms
- Very precise, more difficult on multiple resolutions

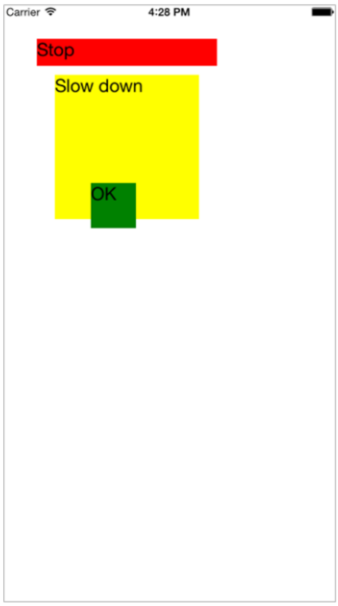
```
public class MyAbsoluteLayoutPage : ContentPage
{
    public MyAbsoluteLayoutPage()
    {
        var red = new Label
        {
            Text = "Stop",
            BackgroundColor = Color.Red,
            Font = Font.SystemFontOfSize (20),
            WidthRequest = 200,
            HeightRequest = 30
        };
        var yellow = new Label
        {
            Text = "Slow down",
            BackgroundColor = Color.Yellow,
            Font = Font.SystemFontOfSize (20),
            WidthRequest = 160,
            HeightRequest = 160
        };
        var absLayout = new AbsoluteLayout();
        absLayout.Children.Add(red, new Point(20,20));
        absLayout.Children.Add(yellow, new Point(40,60));
        absLayout.Children.Add(green, new Point(80,180));

        Content = absLayout;
    }
}
```

In XAML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             x:Class="HelloXamarinFormsWorldXaml.AbsoluteLayoutExample"
             Padding="20">

  <AbsoluteLayout>

    <Label Text="Stop"
           BackgroundColor="Red"
           Font="20"
           AbsoluteLayout.LayoutBounds="20,20,200,30" />

    <Label Text="Slow down"
           BackgroundColor="Yellow"
           Font="20"
           AbsoluteLayout.LayoutBounds="40,60,160,160" />

    <Label Text="Go"
           BackgroundColor="Green"
           Font="20"
           AbsoluteLayout.LayoutBounds="80,180,50,50" />

  </AbsoluteLayout>

</ContentPage>
```

PXL IT

The result

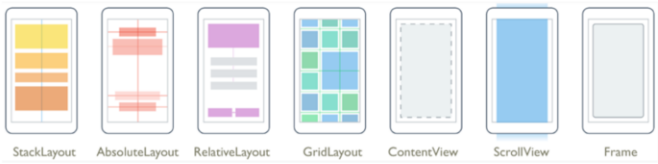


DEMO

AbsoluteLayout

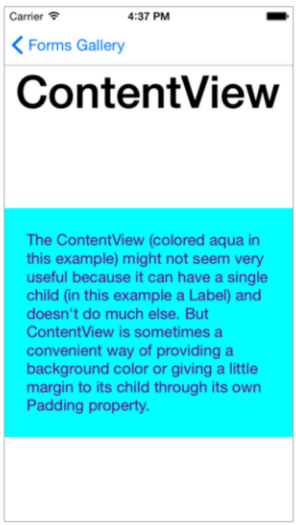
All layouts

- Available layouts
  - StackLayout
  - AbsoluteLayout
  - RelativeLayout
  - GridLayout
  - ContentView
  - ScrollView
  - Frame



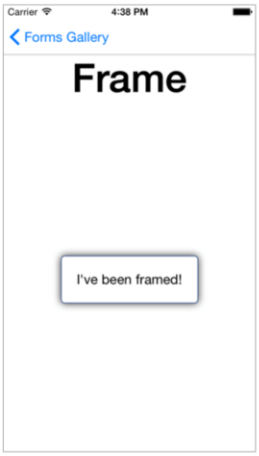
Layout: ContentView

- An element with a single content
  - ContentView has very little use of its own
    - Base class for user-defined compound views



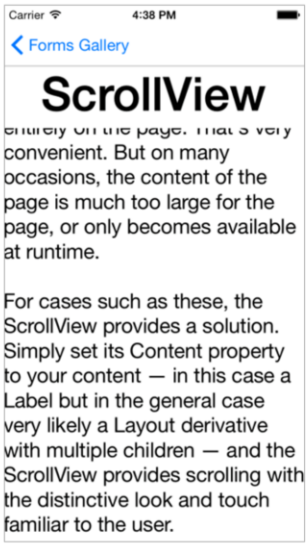
Layouts: Frame

- An element containing a single child, with some framing options
  - Frame has a default Xamarin.Forms.Layout.Padding of 20



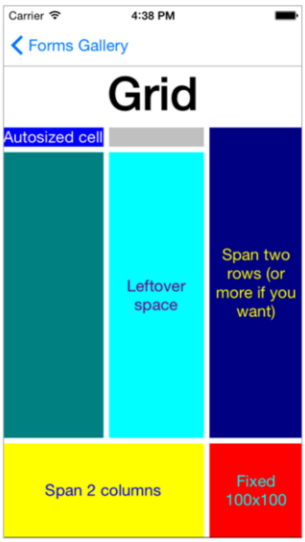
Layouts: ScrollView

- An element capable of scrolling if its Content is larger



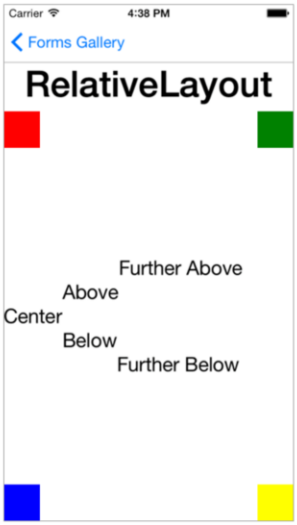
Layouts: Grid

- A layout containing views arranged in rows and columns



Layouts: RelativeLayout

- A Layout that uses Constraints to layout its children

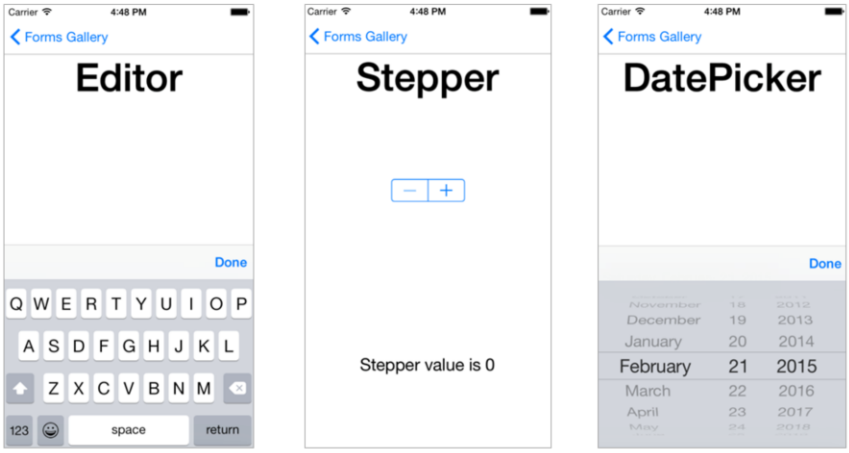


DEMO

Other layouts

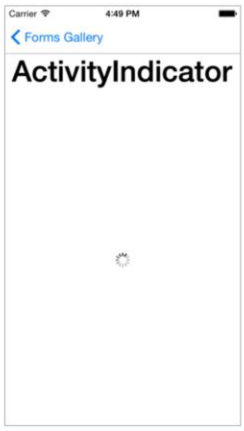
Controls

- View is base class for all controls/widgets (buttons, labels…)
  - Coupled with a renderer to generate native visual element



Controls: ActivityIndicator

- A visual control used to indicate that something is ongoing
  - Gives a visual clue to the user that something is happening, without information about its progress

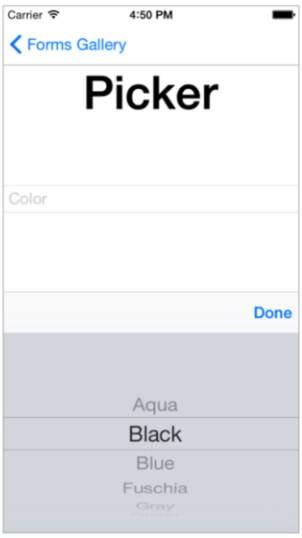


Controls: DatePicker

- Allows date picking (duh)
  - Visual representation of a DatePicker is very similar to the one of Entry
    - Except that a special control for picking a date appears in place of a keyboard

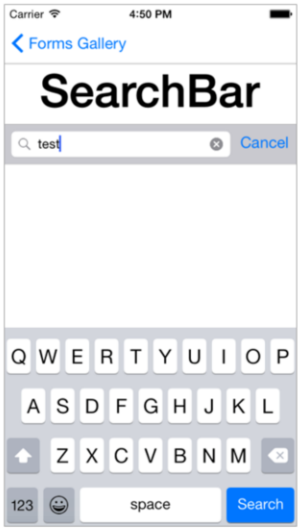
Controls: Picker

- A View control for picking an element in a list



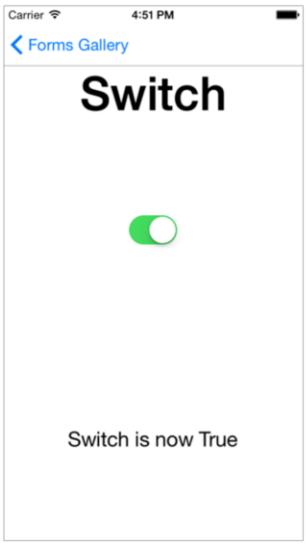
Controls: SearchBar

- A View control that provides a search box



Controls: Switch

- A View control that provides a toggled value



DEMO

Views

PXL IT

Lists

DE HOGESCHOOL
MET HET NETWERK

Hogeschool PXL – Elfde-Liniestraat 24 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

Lists in Xamarin.Forms

- Lists are very common
  - ListView control in Xamarin.Forms
  - Each item is contained in a cell
    - By default, TextCell (renders single line of text)

```
var listView = new ListView
{
    RowHeight = 40
};
listView.ItemsSource = new string []
    {
        "Buy pears",
        "Buy oranges",
        "Buy mangos",
        "Buy apples",
        "Buy bananas"
    };
Content = new StackLayout
{
    VerticalOptions = LayoutOptions.FillAndExpand,
    Children = { listView }
};
```

PXL IT

ListViews are a very common control in mobile applications and deserve to be covered in a bit more detail. The ListView is responsible for displaying a collection of items on the screen; each item in the ListView will be contained in a single cell. By default, a ListView will use the built-in TextCell template and render a single line of text. The code snippet below is a simple example of using the ListView:

Binding to custom class

- TextCell can be used to show text representations of custom classes

```
listView.ItemsSource = new TodoItem [] {
    new TodoItem {Name = "Buy pears"},
    new TodoItem {Name = "Buy oranges", Done=true},
    new TodoItem {Name = "Buy mangos"},
    new TodoItem {Name = "Buy apples", Done=true},
    new TodoItem {Name = "Buy bananas", Done=true}
};
```

```
listView.ItemTemplate = new DataTemplate(typeof(TextCell));
listView.ItemTemplate.SetBinding(TextCell.TextProperty,
"Name");
```

Item selection in a ListView

```
listView.ItemSelected += async (sender, e) => {
    await DisplayAlert("Tapped!", e.SelectedItem + " was tapped.", "OK");
};
```

Custom cells

- We can use a custom cell by subclassing ViewCell

```
class EmployeeCell : ViewCell
{
    public EmployeeCell()
    {
        var image = new Image
                    {
                        HorizontalOptions = LayoutOptions.Start
                    };
        image.SetBinding(Image.SourceProperty, new Binding("ImageUri"));
        image.WidthRequest = image.HeightRequest = 40;

        var nameLayout = CreateNameLayout();

        var viewLayout = new StackLayout()
                         {
                             Orientation = StackOrientation.Horizontal,
                             Children = { image, nameLayout }
                         };
        View = viewLayout;
    }
}
```

PXL IT

It adds an Image and binds it to the ImageUri property of the Employee object. Data binding will be covered in just a moment.

It creates a StackLayout with a vertical orientation to hold the two Labels . The Labels are bound to the DisplayName property and the Twitter property of the Employee object.

It creates another StackLayout that will host the Image and the StackLayout from the previous two steps. It will arrange its children using a horizontal orientation.

Custom cells

- Also, set the type of this class as ItemTemplate for the ListView

```
List<Employee> myListOfEmployeeObjects = GetAListOfAllEmployees();
var listView = new ListView
{
    RowHeight = 40
};
listView.ItemsSource = myListOfEmployeeObjects;
listView.ItemTemplate = new DataTemplate(typeof(EmployeeCell));
```

PXL IT

This code will provide a List<Employee> objects to the ListView. Each cell will be rendered using the EmployeeCell class. The ListView will pass the Employee object to the EmployeeCell as its BindingContext.

DEMO

ListViews

PXL IT

Navigation

DE HOGESCHOOL
MET HET NETWERK

Hogeschool PXL – Elfde-Liniestraat 24 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

Navigation

- Navigation is last-in, first-out
  - Pages are being pushed onto a stack with forward navigation
  - Pages are retrieved from the stack on backward navigation
- Handled in Xamarin.Forms with the INavigation interface

```
public interface INavigation
{
    Task PushAsync(Page page);
    Task<Page> PopAsync();
    Task PopToRootAsync();
    Task PushModalAsync(Page page);
    Task<Page> PopModalAsync();
}
```

PXL IT

Now that we understand how to create pages and arrange controls, lets discuss how to navigate from one page to another. Navigation can be thought of as a last-in, first-out stack of Page objects. To move from one page to another an application will push a new page onto this stack. To return back to the previous page the application will pop the current page from the stack. This navigation in Xamarin.Forms is handled by the INavigation interface which provides the following methods:

Navigation

- Xamarin.Forms has a NavigationPage that implements this interface
  - Also adds navigation bar at the top of the page with title and back button
- Typically, a NavigationPage wraps the first page in the app

```
public static Page GetMainPage()
{
    var mainNav = new NavigationPage(new EmployeeListPage());
    return mainNav;
}
```

PXL IT

Xamarin.Forms has a NavigationPage class that implements this interface and will manage the stack of Pages. The NavigationPage class will also add a navigation bar to the top of the screen that displays a title and will also have a platform appropriate Back button that will return to the previous page. The following code shows how to wrap a NavigationPage around the first page in an application:

Navigation

- To show the next page, we use PushAsync()

```
await Navigation.PushAsync(new LoginPage());
```

- To navigate backward, we use PopAsync()

```
await Navigation.PopAsync();
```

DEMO

Navigation

Platform features

DE HOGESCHOOL
MET HET NETWERK

Hogeschool PXL – Elfde-Liniestraat 24 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

Platform tweaks

- The Device class allows detecting the used platform in code
  - Small platform tweaks become possible
- Device.Idiom
  - Used to alter layouts or functionality depending on whether the device is a phone or a tablet
    - Enum value: Phone or Tablet

```
if (Device.Idiom == TargetIdiom.Phone) {
    // layout views vertically
} else {
    // layout views horizontally
}
```

Xamarin.Forms provides a number of methods that allow functionality and layout to be altered for each platform.

Xamarin.Forms provides many different ways to take advantage of the underlying platform APIs, such as custom renderers and the DependencyService. For simpler platform-specific tweaks, such as layout changes, the Device class allows developers to detect the platform in shared code.

References:

https://developer.xamarin.com/guides/xamarin-forms/custom-renderer/

https://developer.xamarin.com/guides/xamarin-forms/dependency-service/

Platform tweaks

- Device.OS can be set to one of the TargetPlatform values
  - iOS, Android or WinPhone
  - Allows small changes to be made in the layout

```
if (Device.OS == TargetPlatform.iOS) {
    // move layout under the status bar
    stackLayout.Padding = new Thickness (0, 20, 0, 0);
}
if (Device.OS == TargetPlatform.iOS) {
    label.Font = Font.OfSize("MarkerFelt-Thin", NamedSize.Medium);
} else {
    label.Font = Font.SystemFontOfSize(NamedSize.Medium);
}
```

Platform tweaks

- Device.OnPlatform
  - Generic method that has 3 optional parameters: iOS, Android and WinPhone

```
// left and right padding: 5; top padding: 20 (only on iOS)
layout.Padding = new Thickness (5, Device.OnPlatform(20,0,0), 5, 0),
```

  - Also has Action parameters

```
Device.OnPlatform(
    Android: () => {
         PositiveBalance = PositiveBalance.AddLuminosity(0.3);
         NegativeBalance = NegativeBalance.AddLuminosity(0.3);
         SubTitle = Color.FromRgb(115, 129, 130);
        },
    WinPhone: () => {
          PositiveBalance = PositiveBalance.AddLuminosity(0.3);
          NegativeBalance = NegativeBalance.AddLuminosity(0.3);
        }
    );
```

Platform tweaks

- Device.OnPlatform can also be used from XAML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="XamlSamples.GridDemoPage">
  <ContentPage.Padding>
    <OnPlatform x:TypeArguments="Thickness">
      <OnPlatform.iOS>0, 20, 0, 0</OnPlatform.iOS>
    </OnPlatform>
  </ContentPage.Padding>
```

Platform tweaks

- Device.OpenUri can be used to open another app based on a protocol

```
Device.OpenUri(new Uri("http://xamarin.com/evolve"));
```

DEMO

Platform tweaks