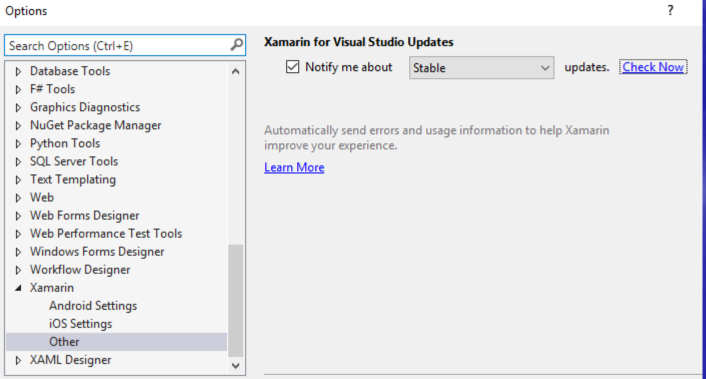
Xamarin.Forms

Pages
Building an Interface

VS and Project Configuration

VS: Tools → Options



Options ?

Search Options (Ctrl+E)

- Database Tools
- F# Tools
- Graphics Diagnostics
- NuGet Package Manager
- Python Tools
- SQL Server Tools
- Text Templating
- Web
- Web Forms Designer
- Web Performance Test Tools
- Windows Forms Designer
- Workflow Designer
- Xamarin
  - Android Settings
  - iOS Settings
  - Other
- XAML Designer

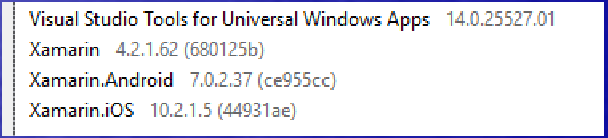
Xamarin for Visual Studio Updates

☑ Notify me about Stable updates. Check Now

Automatically send errors and usage information to help Xamarin improve your experience.

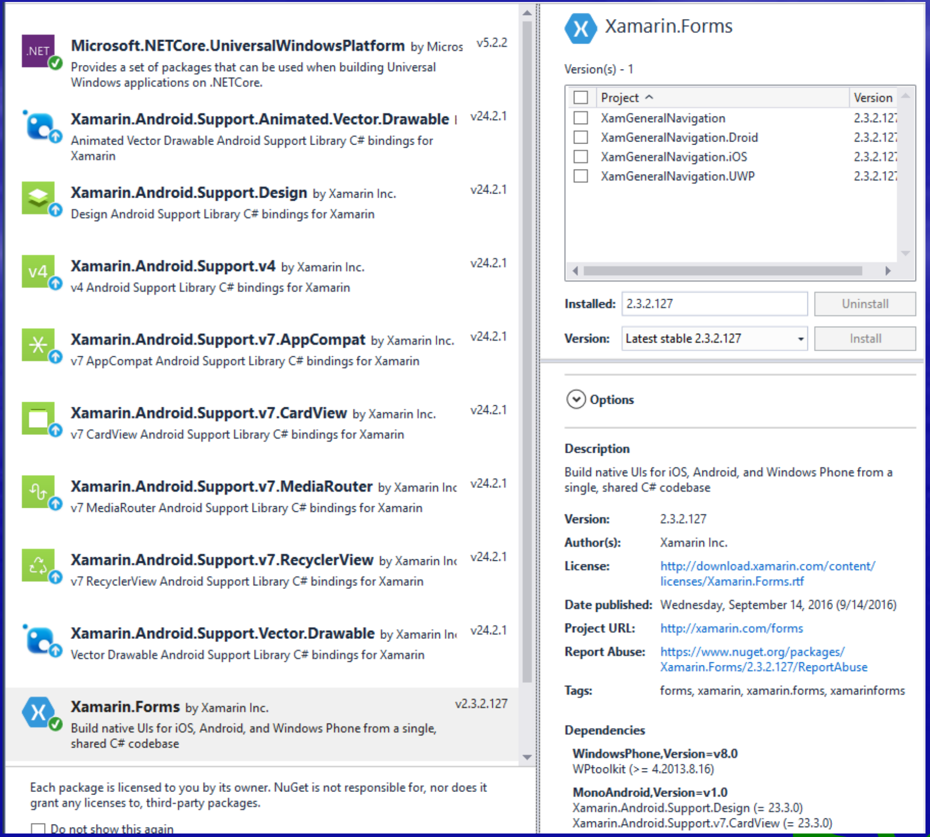
Learn More

Help → About



Visual Studio Tools for Universal Windows Apps	14.0.25527.01
Xamarin	4.2.1.62 (680125b)
Xamarin.Android	7.0.2.37 (ce955cc)
Xamarin.iOS	10.2.1.5 (44931ae)

Solution → Manage Nuget Packages for Solution



only UWP

Microsoft.NETCore.UniversalWindowsPlatform by Micros v5.2.2
Provides a set of packages that can be used when building Universal Windows applications on .NETCore.

only Android

Xamarin.Android.Support.Animated.Vector.Drawable I v24.2.1
Animated Vector Drawable Android Support Library C# bindings for Xamarin

Xamarin.Android.Support.Design by Xamarin Inc. v24.2.1
Design Android Support Library C# bindings for Xamarin

Xamarin.Android.Support.v4 by Xamarin Inc. v24.2.1
v4 Android Support Library C# bindings for Xamarin

Xamarin.Android.Support.v7.AppCompat by Xamarin Inc. v24.2.1
v7 AppCompat Android Support Library C# bindings for Xamarin

Xamarin.Android.Support.v7.CardView by Xamarin Inc. v24.2.1
v7 CardView Android Support Library C# bindings for Xamarin

Xamarin.Android.Support.v7.MediaRouter by Xamarin Inc v24.2.1
v7 MediaRouter Android Support Library C# bindings for Xamarin

Xamarin.Android.Support.v7.RecyclerView by Xamarin Inc v24.2.1
v7 RecyclerView Android Support Library C# bindings for Xamarin

Xamarin.Android.Support.Vector.Drawable by Xamarin In v24.2.1
Vector Drawable Android Support Library C# bindings for Xamarin

All projects

Xamarin.Forms by Xamarin Inc. v2.3.2.127
Build native UIs for iOS, Android, and Windows Phone from a single, shared C# codebase

Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third-party packages.

☐ Do not show this again

Xamarin.Forms

Version(s) - 1

☐	Project ^	Version
☐	XamGeneralNavigation	2.3.2.127
☐	XamGeneralNavigation.Droid	2.3.2.127
☐	XamGeneralNavigation.iOS	2.3.2.127
☐	XamGeneralNavigation.UWP	2.3.2.127

Installed: 2.3.2.127 Uninstall

Version: Latest stable 2.3.2.127 Install

Options

Description

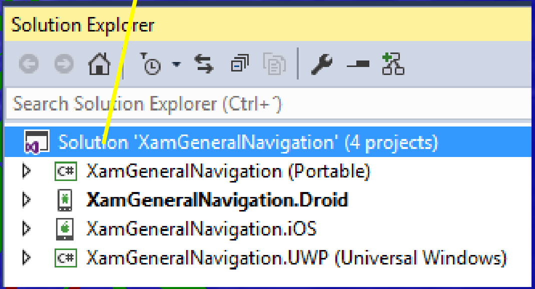
Build native UIs for iOS, Android, and Windows Phone from a single, shared C# codebase

Version: 2.3.2.127
Author(s): Xamarin Inc.
License: http://download.xamarin.com/content/licenses/Xamarin.Forms.rtf
Date published: Wednesday, September 14, 2016 (9/14/2016)
Project URL: http://xamarin.com/forms
Report Abuse: https://www.nuget.org/packages/Xamarin.Forms/2.3.2.127/ReportAbuse
Tags: forms, xamarin, xamarin.forms, xamarinforms

Dependencies

WindowsPhone,Version=v8.0
WPtoolkit (>= 4.2013.8.16)

MonoAndroid,Version=v1.0
Xamarin.Android.Support.Design (= 23.3.0)
Xamarin.Android.Support.v7.CardView (= 23.3.0)

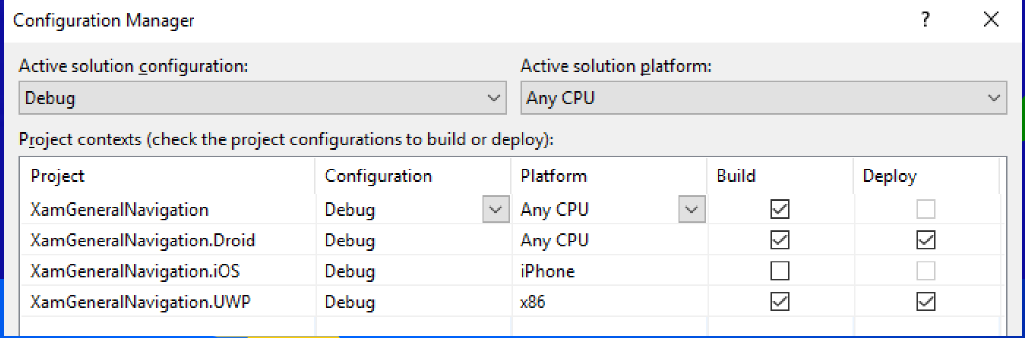


Solution Explorer

Search Solution Explorer (Ctrl+;)

Solution 'XamGeneralNavigation' (4 projects)
- XamGeneralNavigation (Portable)
- **XamGeneralNavigation.Droid**
- XamGeneralNavigation.iOS
- XamGeneralNavigation.UWP (Universal Windows)

Build → Configuration Manager



Configuration Manager ? ×

Active solution configuration: Debug

Active solution platform: Any CPU

Project contexts (check the project configurations to build or deploy):

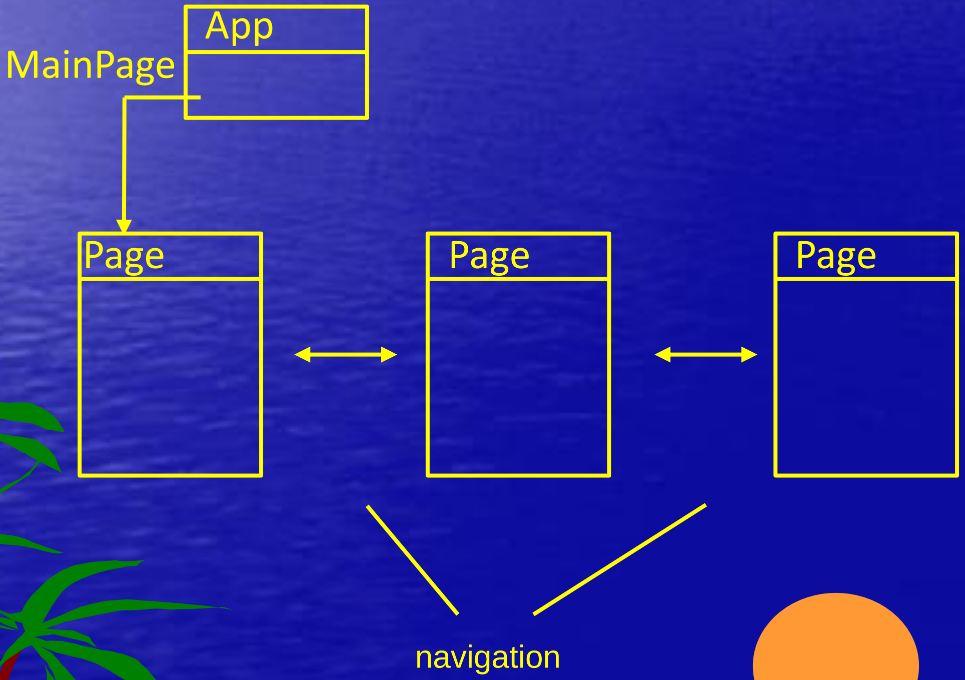
Project	Configuration	Platform	Build	Deploy
XamGeneralNavigation	Debug	Any CPU	☑	☐
XamGeneralNavigation.Droid	Debug	Any CPU	☑	☑
XamGeneralNavigation.iOS	Debug	iPhone	☐	☐
XamGeneralNavigation.UWP	Debug	x86	☑	☑

Pages

Xamarin.Forms interface is defined in the common project and shared by all platforms

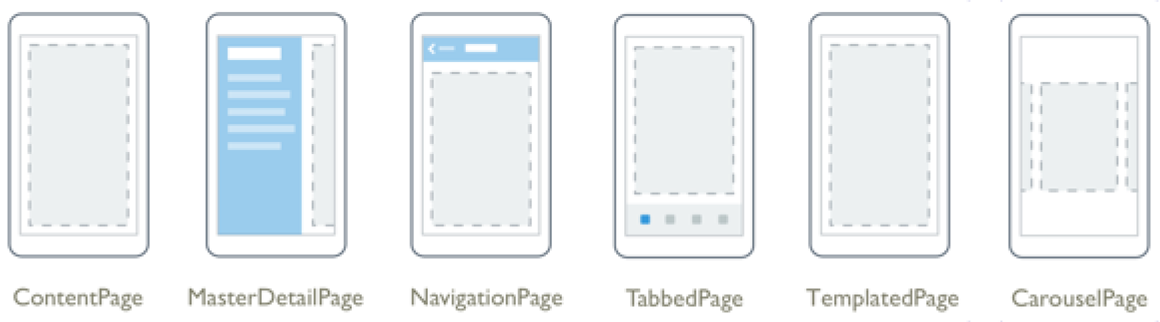
A Xamarin.Forms interface is composed of Pages and a singleton App

The first page presented should be assigned to the MainPage property of the App object



```
public class App : Application {
    public App() {
      // The root page of your application
      var content = new MyRootPage() {
        Title = “MyAppTitle"
      };
      MainPage = content;
    }
    …
    // life cycle handlers
}
```

Pages



Pages contain one single object: a Layout or a View

- Layouts can contain multiple Views or other Layouts
- Specialized Pages like the TabbedPage or CarouselPage can contain a collection of sub-pages

A single page app should contain just a ContentPage

A generic multi-page navigable app should have as the MainPage a NavigationPage that contains a ContentPage. The other pages to where we can navigate are ContentPages.

Pages

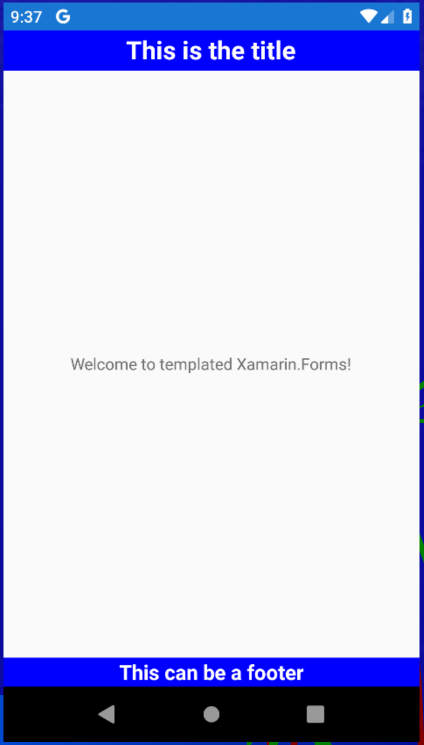
All page classes derive from the base Page, following the class hierarchy:

Page	Basic properties like the size and geometric transforms
TemplatedPage	Allows to define the ControlTemplate property, appearing in all ContentPages
ContentPage	The page used to fill the screen; display also the ControlTemplate of the parent
NavigationPage	Define the navigation methods
MasterDetailPage	A collection of two pages allowing to go forth and back
MultiPage<T> (abstract)	
TabbedPage	A multi-page collection with a visual to select one
CarouselPage	A multi-page collection cycling using swiping

```
<Application ... >
 <Application.Resources>
   <ControlTemplate x:Key="MyPageTemplate">
    <StackLayout>
     <Grid>
      <BoxView HeightRequest="35" BackgroundColor="Blue" />
      <Label TextColor="White" Text="This is the title" HorizontalOptions="Center" VerticalOptions="Center" FontSize="Large" FontAttributes="Bold"/>
     </Grid>
     <ContentPresenter VerticalOptions="FillAndExpand"/>
     <Grid>
      <BoxView HeightRequest="25" BackgroundColor="Blue" />
      <Label TextColor="White" Text="This can be a footer" HorizontalOptions="Center" VerticalOptions="Center" FontSize="Medium" FontAttributes="Bold"/>
     </Grid>
    </StackLayout>
   </ControlTemplate>
  </Application.Resources>
</Application>
```

```
<ContentPage ... ControlTemplate="{StaticResource  MyPageTemplate}">
  <StackLayout>
      <Label Text="Welcome to templated Xamarin.Forms!"  HorizontalOptions="Center"
             VerticalOptions="CenterAndExpand" />
  </StackLayout>
</ContentPage>
```

ContentPage with ControlTemplate



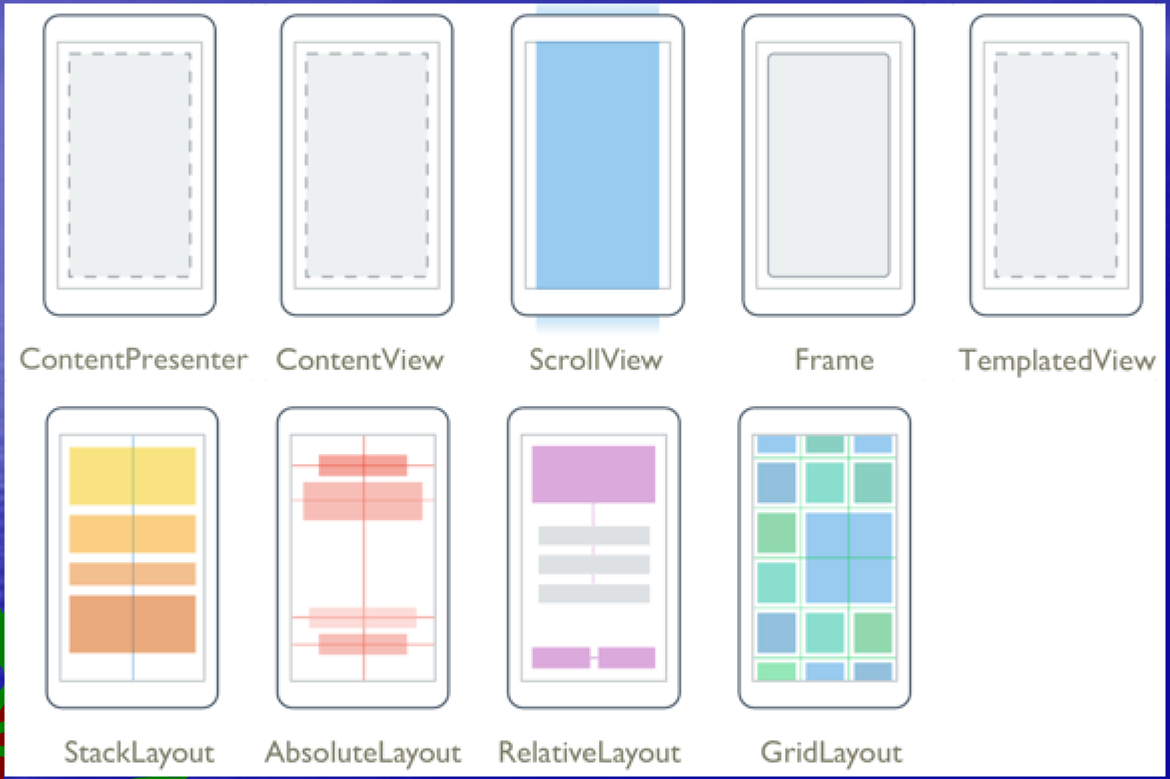
This is the title

Welcome to templated Xamarin.Forms!

This can be a footer

Building a Page

Inside a page we can have a single View or, if we need more, a single Layout
Layouts can organize Views (or other Layouts) in the available space



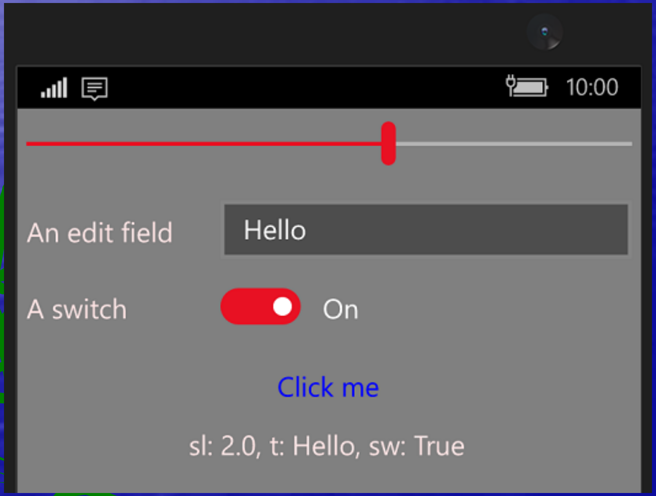
The StackLayout, ScrollView, Grid, RelativeLayout and AbsoluteLayout are the most used.

Building a Page

Pages can be specified in two different ways:

1. In code, deriving from the framework classes, and establishing the hierarchy of Layouts and Views in the constructor.
2. In a dialect of XML called XAML. Associated with this specification, for each page a code file (code behind) is also specified, usually containing the handlers to events triggered by user interactions.

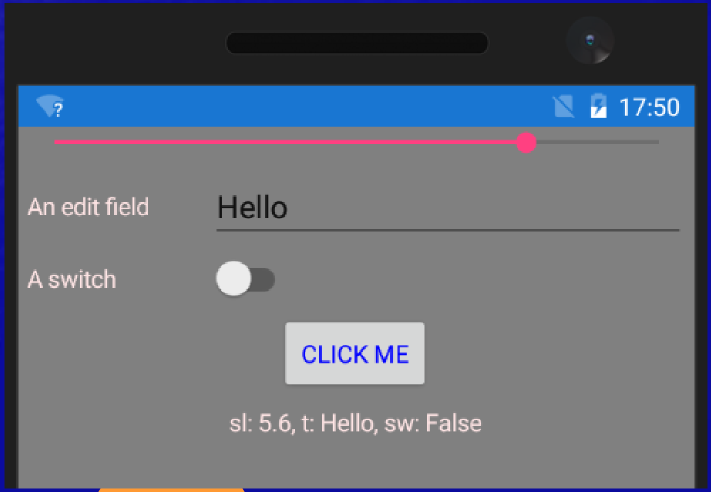
Example:



An edit field Hello

A switch On

sl: 2.0, t: Hello, sw: True



An edit field Hello

A switch

sl: 5.6, t: Hello, sw: False

- ContenPage
  - StackLayout O:V
    - Slider
    - StackLayout O:H
      - Label
      - Entry
    - StackLayout O:H
      - Label
      - Switch
    - Button
    - Label

The code

```
public class RootPage : ContentPage {
  Label lab1, … ;
  Entry entry;
  Button button;
  …

  public RootPage() {    // beginning of constructor

    lab1 = new Label() {
      HorizontalOptions = LayoutOptions.Start,
      VerticalOptions = LayoutOptions.Center,
      Text = "An edit field",
      TextColor = Color.FromRgb(1.0, 0.9, 0.9),
      WidthRequest = 100
    };

    entry = new Entry() {
      HorizontalOptions = LayoutOptions.FillAndExpand,
      Text = "",
      Placeholder = "Write here",
    };

    var stack1 = new StackLayout() {
      Orientation = StackOrientation.Horizontal,
      Children = { lab1, entry }
    };
 …
    button = new Button() {
      HorizontalOptions = LayoutOptions.Center,
      Text = "Click me",
      TextColor = Color.Blue
    };
    button.Clicked += OnButton_Clicked;
  …
```

Constructor

```
…
 Content = new StackLayout() {
       Children = { slider, stack1, stack2, button, labValue }
 };
 Padding = new Thickness(5, Device.OnPlatform(20, 0, 0));
} // end of constructor

/* event handlers */

 private void OnButton_Clicked(object sender, EventArgs e) {
    labValue.Text = String.Format("sl: {0:F1}, t: {1}, sw: {2}",
                                  slider.Value, entry.Text, toggle.IsToggled);
  }
}     // end of class (RootPage)
```

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"
         xmlns:local="clr-namespace:XamXSP"
         x:Class="XamXSP.MainPage"
         BackgroundColor="LightYellow">
 <ContentPage.Padding>
  <OnPlatform x:TypeArguments="Thickness" iOS="0, 20, 0, 0"/>
 </ContentPage.Padding>
 <StackLayout>
  <Label HorizontalOptions="Center" TextColor="LawnGreen" Text="A bunch of Xamarin views"></Label>
  <Slider x:Name="slider" HorizontalOptions="Fill" Minimum="-10.0" Maximum="10.0" Value="-5.0"></Slider>
  <StackLayout Orientation="Horizontal">
   <Label HorizontalOptions="Start" WidthRequest="100" TextColor="Coral" Text="An edit field"></Label>
   <Entry x:Name="entry" HorizontalOptions="FillAndExpand" Text="" Placeholder="Write here"></Entry>
  </StackLayout>
  <StackLayout Orientation="Horizontal">
   <Label HorizontalOptions="Start" WidthRequest="100" TextColor="Coral" Text="A switch"></Label>
   <Switch x:Name="toggle" HorizontalOptions="Start" IsToggled="True"></Switch>
  </StackLayout>
  <Button x:Name="button" HorizontalOptions="Center" TextColor="Cyan" Text="Click Me"></Button>
  <Label x:Name="labValue" HorizontalOptions="Center" Text="" TextColor="Coral"></Label>
 </StackLayout>
</ContentPage>
```

Views Geometric Transforms

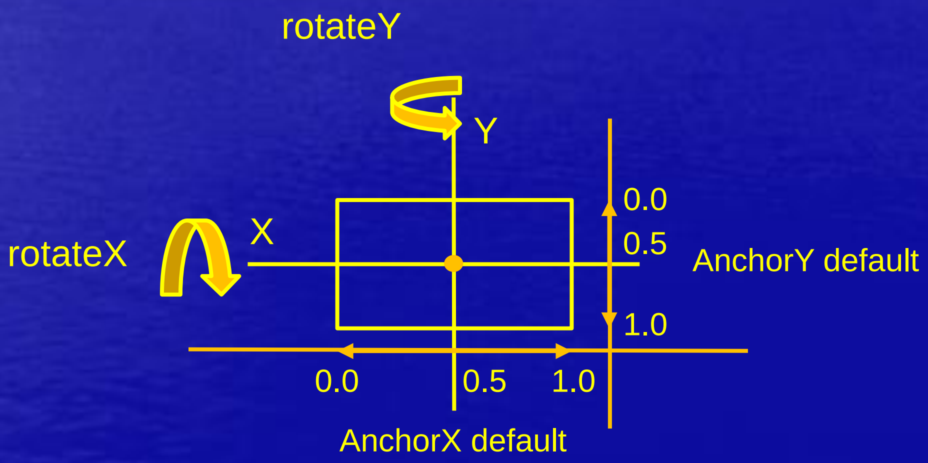
Views have properties that implement any linear geometric transform

- **translation**
- **scale**
- **rotation**

These properties only affect rendering
The reported place and size remain the same

Properties (all double):

- **TranslationX**
- **TranslationY**
- **Scale**
- **Rotation**
- **RotationX**
- **RotationY**
- **AnchorX**
- **AnchorY**

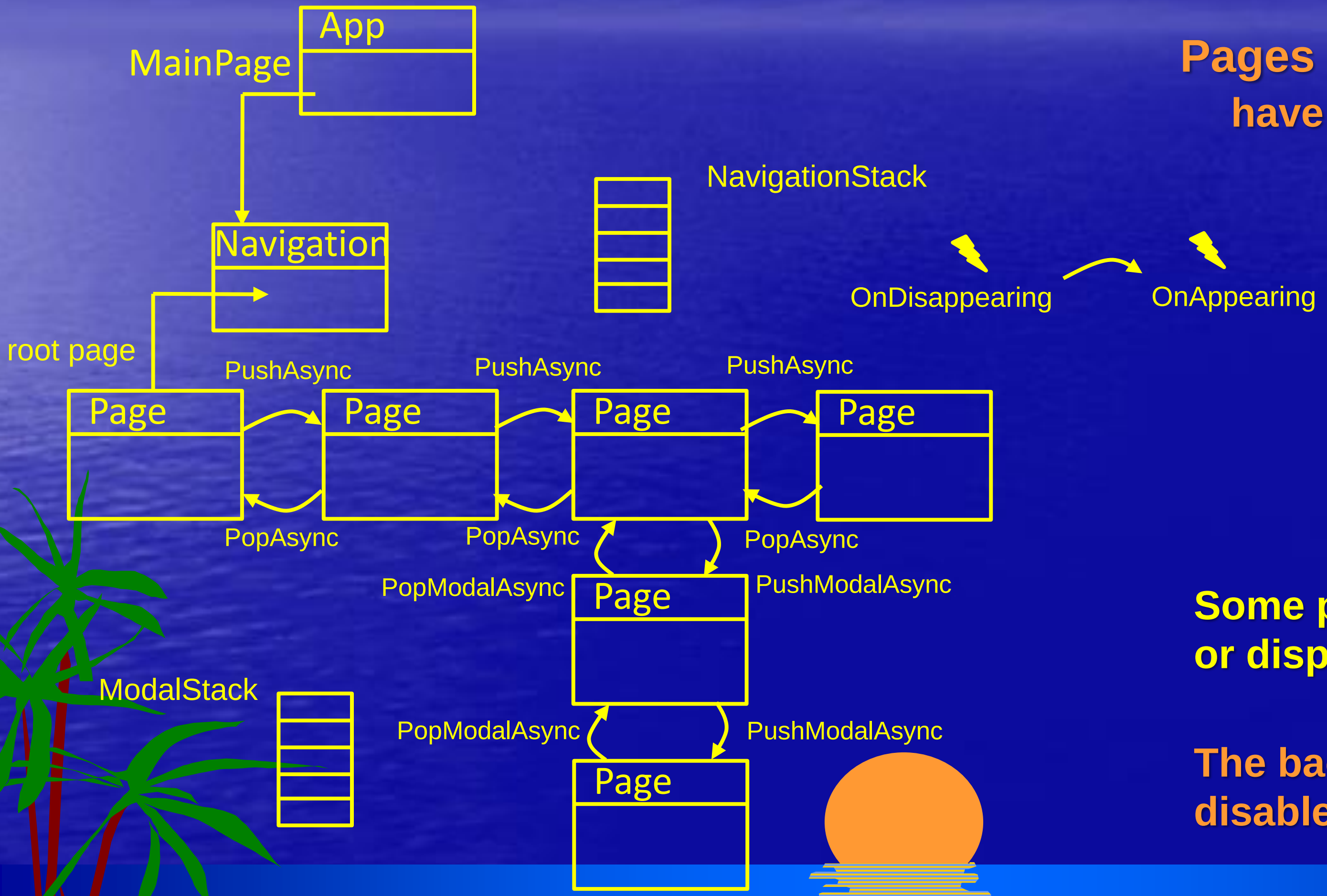


Page Navigation

Two types:

Modeless navigation – go to any other page, put it on a stack, and go back

Modal navigation – go and dismiss (go back) or proceed to another modal page



Pages

have a Navigation property

PushAsync()
PopAsync()
PushModalAsync()
PopModalAsync()
NavigationStack
ModalStack
RemovePage()
InsertPageBefore()
PopToRoot()

Some platforms have also or display a back button

The back button should be disabled in modal navigation

Passing Values between Pages

When we navigate to a Page, usually we construct it first
When we pop a Page, the Page instance disappears

In a navigation to a new page we can pass values using:
- the constructor of the new Page
- properties and methods of the Page

In passing back values to the parent Page we can:
- define an interface for the data to transfer
- implement the interface in the parent
- pass a reference of the parent to the destination page
- call (or set properties) in the destination code using the interface implemented by the parent (modifying the parent)

Other general methods
- using the MessagingCenter class (allows subscriptions and sending messages)
- implementing a DataReady event handled by the recipient Page
- using the singleton App object for global state
- using a ViewModel data object and Binding the Pages
- saving and restoring Page state

Saving and Restoring App Data

Xamarin.Forms Application object has a Properties hash table accessible from every Page

- **We can put values associated with a string key there and get them later**
- **The most portable way is to XML serialize those values into a string**
- **Later we can deserialize them into new objects**

Example:

Allow your data class to Serialize and Deserialize

Note: In this example CurrentInfo (if exists) is an item of the InfoCollection list

```
public class AppData {
  public AppData() {
    InfoCollection = new ObservableCollection<Information>();
    CurrentInfoIndex = -1;
  }
 public ObservableCollection<Information> InfoCollection { private set; get; }

 [XmlIgnore]
 public Information CurrentInfo { set; get; }

 public int CurrentInfoIndex { set; get; }

 public string Serialize() {
   if (CurrentInfo != null)
     CurrentInfoIndex = InfoCollection.IndexOf(CurrentInfo);
   XmlSerializer serializer = new XmlSerializer(typeof(AppData));
   using (StringWriter stringWriter = new StringWriter()) {
     serializer.Serialize(stringWriter, this);
     return stringWriter.GetStringBuilder().ToString();
   }
 }

 public static AppData Deserialize(string strAppData) {
   XmlSerializer serializer = new XmlSerializer(typeof(AppData));
   using (StringReader stringReader =
                                new StringReader(strAppData)) {
     AppData appData = (AppData)
                                  serializer.Deserialize(stringReader);

     if (appData.CurrentInfoIndex != -1)
       appData.CurrentInfo = appData.InfoCollection[
                                           appData.CurrentInfoIndex];
     return appData;
   }
 }
}
```

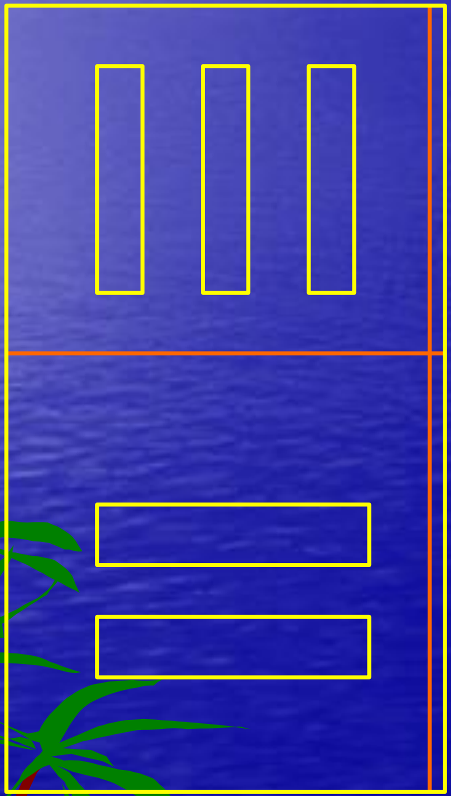
Save and Restore and the Life Cycle

```
public class App : Application {
   public App() {
      // Load previous AppData if it exists.
      if (Properties.ContainsKey("appData"))
        AppData = AppData.Deserialize((string)Properties["appData"]);
     else
        AppData = new AppData();

    // Launch home page.
    Page homePage = new HomePage();
    MainPage = new NavigationPage(homePage);

    // Possibly navigate to info page.
    if (Properties.ContainsKey("isInfoPageActive") && (bool)Properties["isInfoPageActive"])
       homePage.Navigation.PushAsync(new InfoPage(), false);
  }

  public AppData AppData { private set; get; }

   …
   protected override void OnSleep() {
      // Save AppData serialized into string.
      Properties["appData"] = AppData.Serialize();

      // Save Boolean for info page active.
      Properties["isInfoPageActive"] = MainPage.Navigation.NavigationStack.Last() is InfoPage;
   }
  …
 }
```

Adapting to Portrait or Landscape

Not all platforms allow the definition of alternative layouts, automatically set when conditions or devices have different characteristics

Using a Grid view



Put content into a 2x2 Grid

OnSizeChanged

In portrait (W < H) → In landscape (W > H)

Define Grid properties:

Row 0, H: auto
Row 1, H: *

Col 0, W: auto
Col 1, W: 0

Transfer content from
(1, 0) to (0, 1) in the Grid
(changing Row and Column properties)

Redefine Grid properties:

Row 0, H: auto
Row 1, H: 0

Col 0, W: auto
Col 1, W: *

Other way: changing the orientation of a StackLayout

Device Dependent Code

The Device class has static Properties or Methods that have different values or behaviors, depending on the device platform

Device.Idiom (prop) → Phone (w < 600dpi), Tablet, Desktop (UWP)

Device.OS (prop) → iOS, Android, WinPhone(8.1), Windows

Device.OnPlatform() → has 3 arguments of the same type. Returns the first if iOS, the second if Android, and the third if Windows

Ex:

```
Padding = Device.OnPlatform(new Thickness(0, 20, 0, 0),
                            new Thickness(0),
                            new Thickness(0));
```

Device.GetNamedSize() → returns a font size from the size enumerations (micro, small, …)

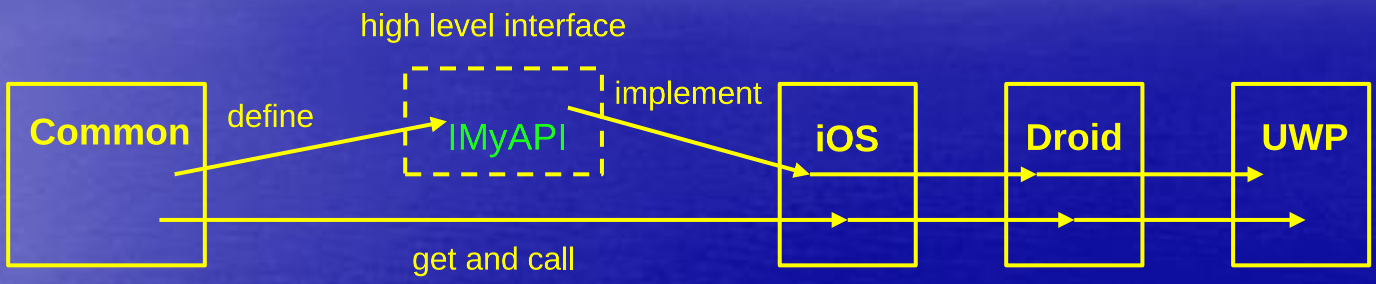
Device.OpenUri() → opens a web page or places a call using the appropriate apps

Device.StartTimer → start a timer using the appropriate way in each platform

Device.BeginInvokeOnMainThread() → allows the supplied function to be executed on the main thread, when called from other thread

From Common to Device Specific Projects

To call code from the Common project, targeting any of the specific platform projects a dependency injection technique can be used:



high level interface

Common

define

IMyAPI

implement

iOS

Droid

UWP

get and call

(in the interface implementation file)

```
iOS
[assembly: Dependency(typeof(MyApp.Droid.MyClassAPI))]
UWP
```

(accessing in common code)

```
…
IMyAPI myAPI = DependencyService.Get<IMyAPI>();
myAPI.method1(…);
Info = myAPI.PropInfo;
myAPI,myEvent += myHandler( … );
…
```