IOS102

Introduction to the Xamarin Designer for iOS

Download class materials from
university.xamarin.com

Microsoft

Xamarin University

Information in this document is subject to change without notice. The example companies, organizations, products, people, and events depicted herein are fictitious. No association with any real company, organization, product, person or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user.

Microsoft or Xamarin may have patents, patent applications, trademarked, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any license agreement from Microsoft or Xamarin, the furnishing of this document does not give you any license to these patents, trademarks, or other intellectual property.

Xamarin, MonoTouch, MonoDroid, Xamarin.iOS, Xamarin.Android, Xamarin Studio, and Visual Studio are either registered trademarks or trademarks of Microsoft in the U.S.A. and/or other countries.

Other product and company names herein may be the trademarks of their respective owners.

Objectives

1. Create a single screen application
2. Describe and use Auto Layout
3. Interact with designer-defined views programmatically
4. Navigate between view controllers

Create a single screen application

Xamarin University

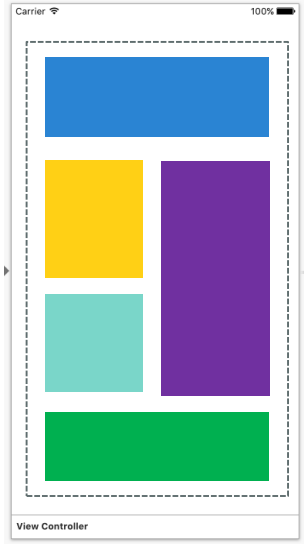
Tasks

1. Describe the iOS Designer
2. Identify controls and properties
3. Demonstrate the designer workflow
4. Lay out subviews

Reminder: UIView

- A **`UIView`** defines a rectangular area on the screen and provides:

Visualization

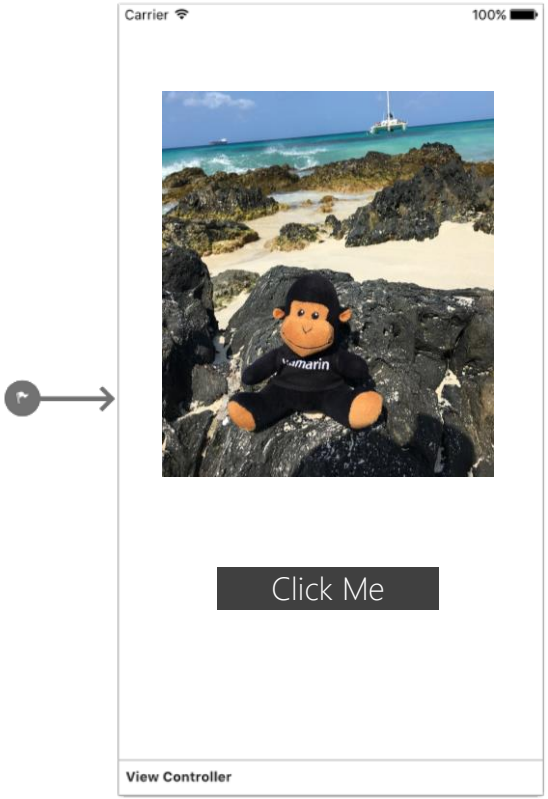


Layout for subviews

Event publishing

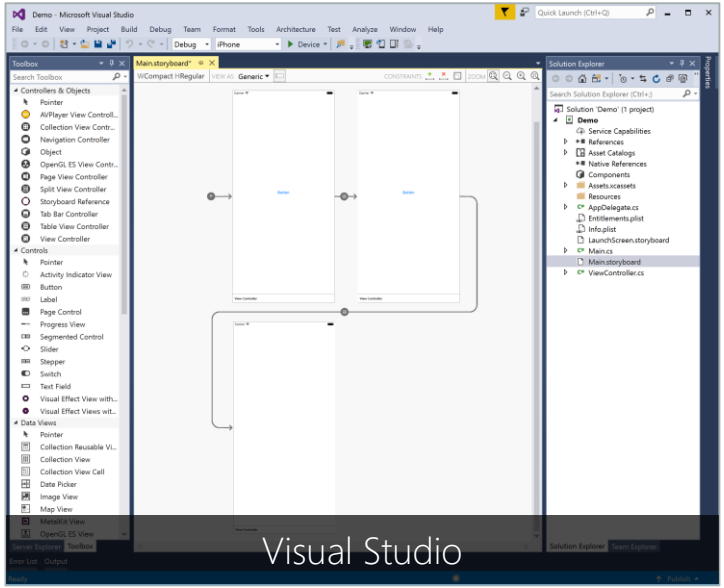
Reminder: View Controllers

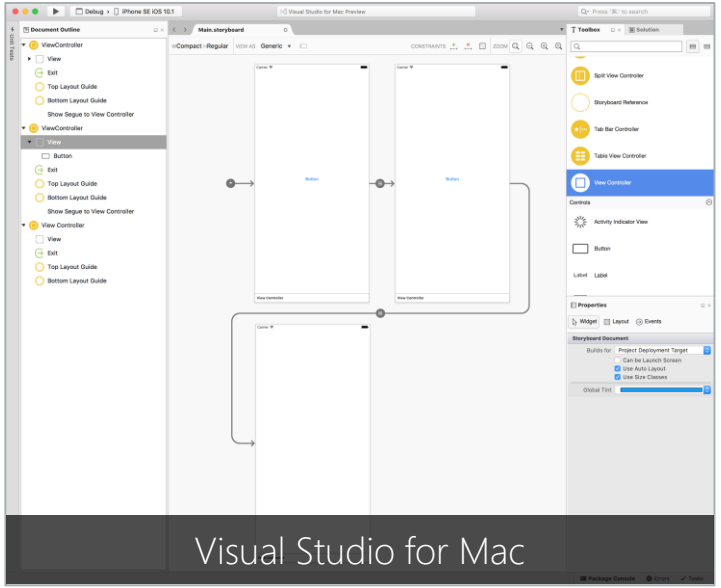
- A **`UIViewController`** provides view management for a single screen
- Owns a **`UIView`** (root view) and receives lifetime notifications from it
- Acts as the mediator between the view(s) and the data/logic/model(s)



The iOS Designer

- The Xamarin.iOS designer is a visual drag + drop editor for creating and editing screens (View Controllers + Views) in your iOS applications

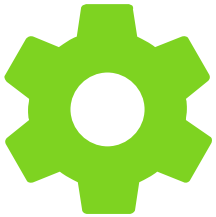




Parts of the Designer

- The iOS Designer has several windows which you use to examine, visualize, design the UI of your application

Document Outline



Properties Explorer

Designer Surface

Toolbox

Designer Toolbar

Demonstration

Tour the Xamarin.iOS designer

Xamarin University

Flash Quiz

Xamarin University

Flash Quiz

① The _____ shows a list of views and view controllers that can be dragged onto the storyboard design surface

a) Toolbox

b) Properties Pane

c) Designer Toolbar

Flash Quiz

① The _____ shows a list of views and view controllers that can be dragged onto the storyboard design surface

a) Toolbox

b) Properties Pane

c) Designer Toolbar

Flash Quiz

② A **`UIView`** is responsible for:

a) Event publishing
b) Visualization
c) Managing subviews
d) All of the above
e) None of the above

Flash Quiz

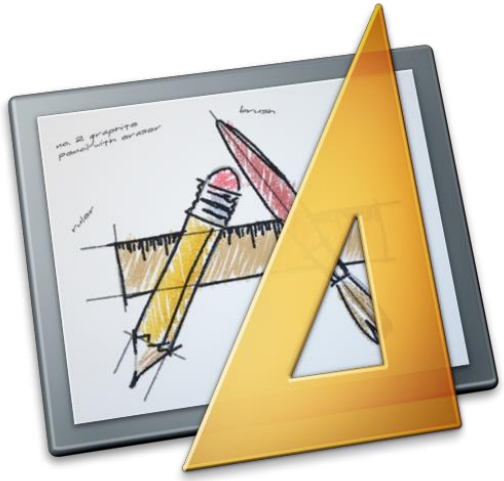
② A **`UIView`** is responsible for:

a) Event publishing
b) Visualization
c) Managing subviews
d) All of the above
e) None of the above

Storyboards vs. XIBs

- iOS supports two designer file formats: Storyboards and XIBs

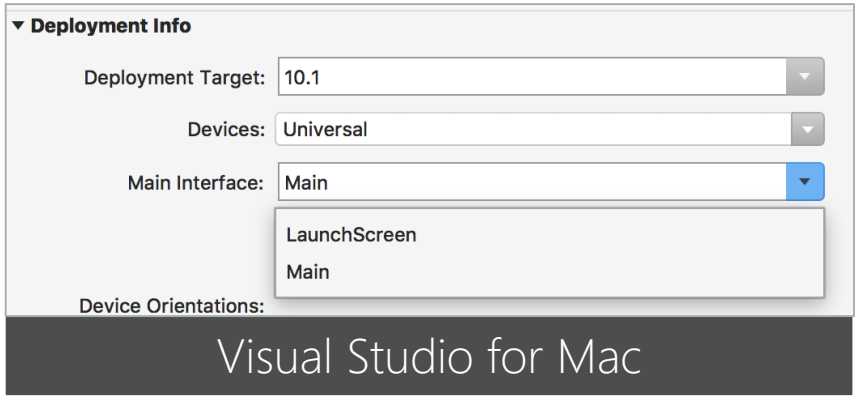
Storyboards let you design multiple screens together with the relationships between them; this is the default file created for your app



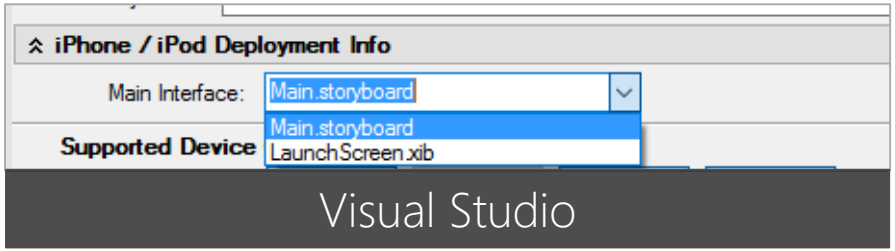
XIB is the original format which defines a single screen or part of a screen; this is used today for the Launch Screen

Using Storyboards

- Applications typically use a single Storyboard to define their UI but it possible to add more to segregate or share portions of the UI



Visual Studio for Mac

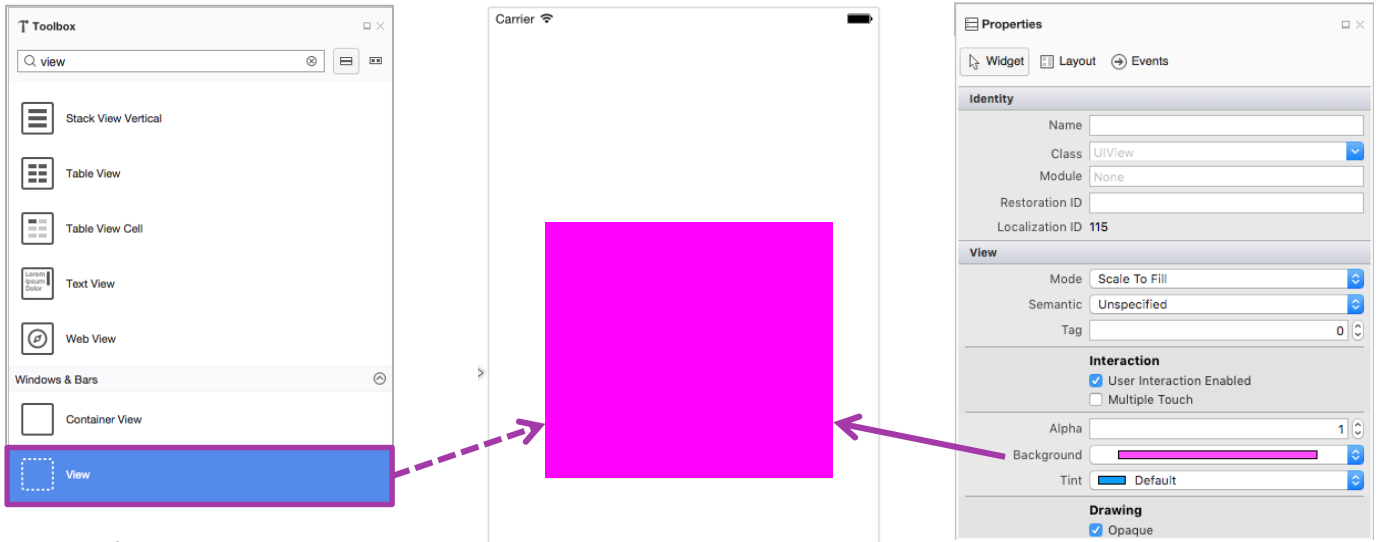


Visual Studio

Info.plist identifies the storyboard to use when the app starts

Workflow [Visual Studio for Mac]

- UI elements are added to a storyboard by dragging items from the Toolbox design surface

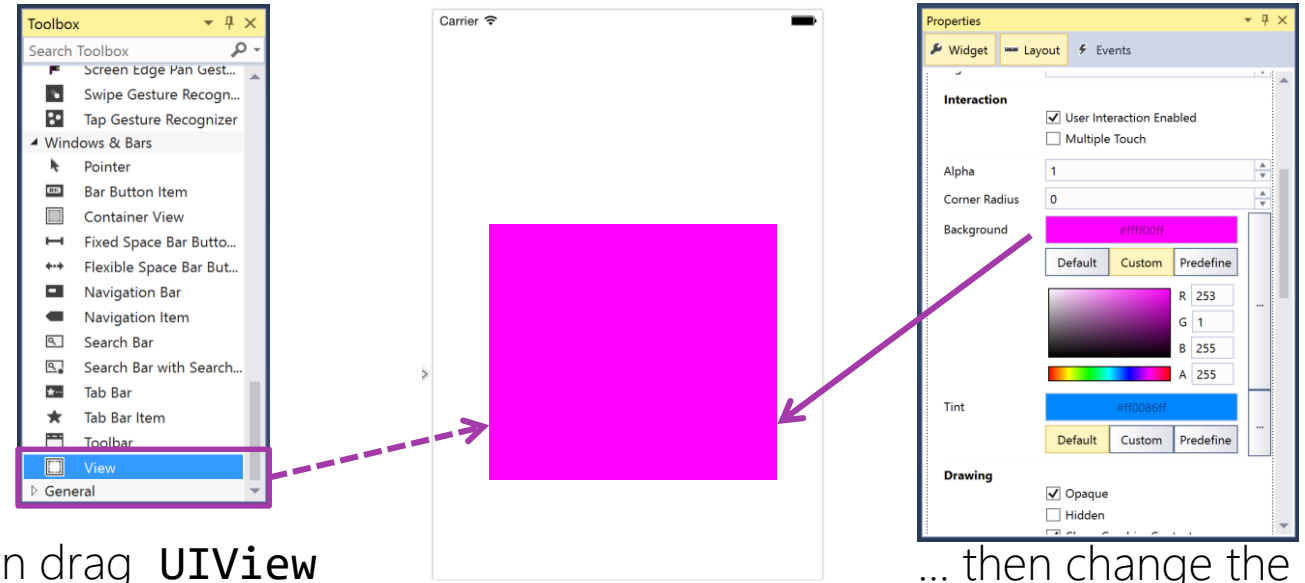


Can drag **`UIView`** from the **toolbox**

… then change the background color

Workflow [Visual Studio]

- UI elements are added to a storyboard by dragging items from the Toolbox design surface

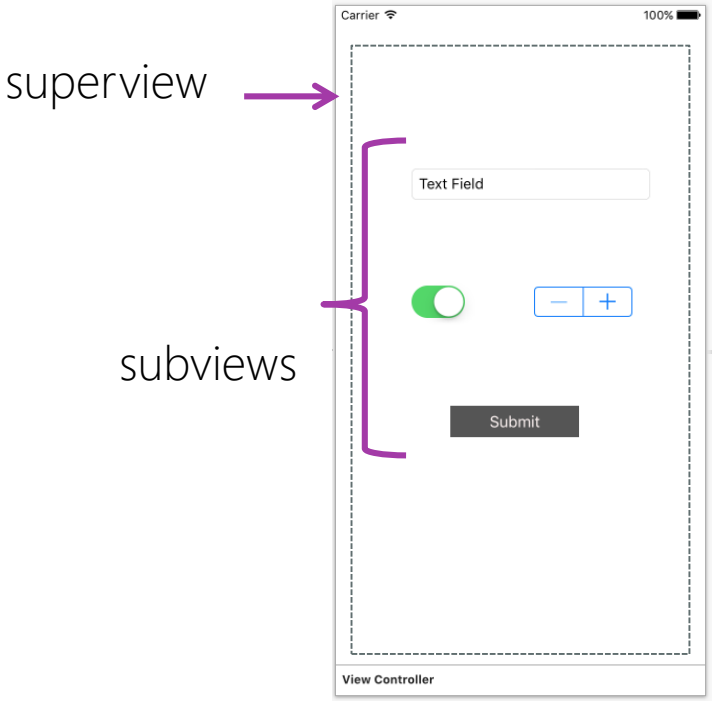


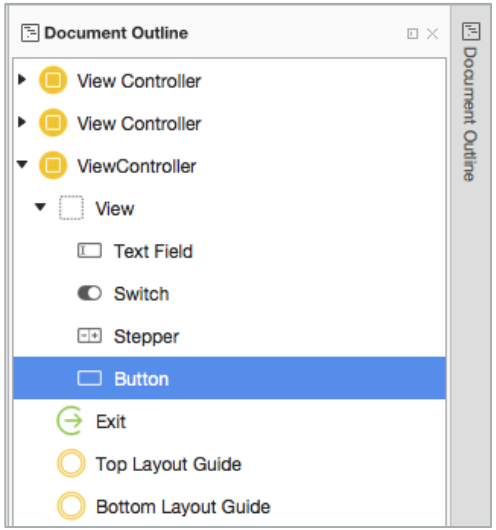
Can drag **`UIView`** from the **toolbox**

... then change the background color

Layout and subviews

- View Controllers have a root **`UIView`** that can hold child views



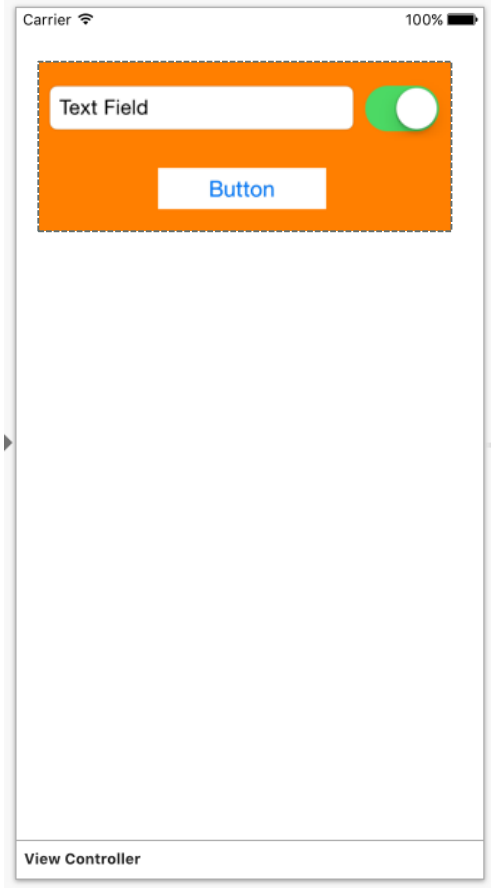


Use the document outline view to see relationships

Composite controls

- Can take advantage of the view architecture to create *composite controls* by nesting controls within a `UIView`

- Composite controls can be made reusable and are easily moved or animated as a group by adjusting the parent view



Individual Exercise

Create the UI for a single view application

Xamarin University

Summary

1. Describe the iOS Designer
2. Identify controls and properties
3. Demonstrate the designer workflow
4. Work with subviews

Describe and use Auto Layout

Xamarin University

Tasks

1. Describe the Auto Layout system
2. Identify constraints
3. Add constraints using the Designer

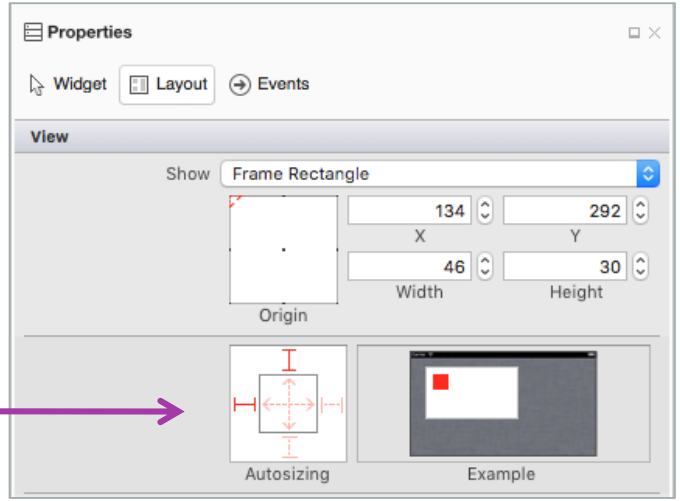
Responsive interface design

- There are several things which can affect the layout of your UI at runtime
  - Device resolution
  - Device form-factor
  - Orientation changes
  - Adding dynamic content
  - User-selectable fonts
  - Localized content

Layout solutions

- Apple has two APIs to manage layout rules in the UI design - the first is *Autoresizing Masks* which provides limited ability to create reactive UIs

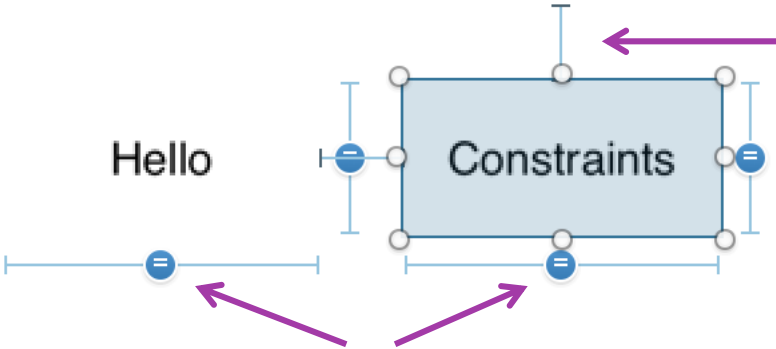
You define each side of the frame with either a "flexible" or "fixed" margin to decide if it stretches with, or is pinned to the parent



Autoresizing Masks

What is Auto Layout?

- Apple provides a flexible layout system called *Auto Layout* - Auto Layout organizes the UI views by describing relationships between visual elements



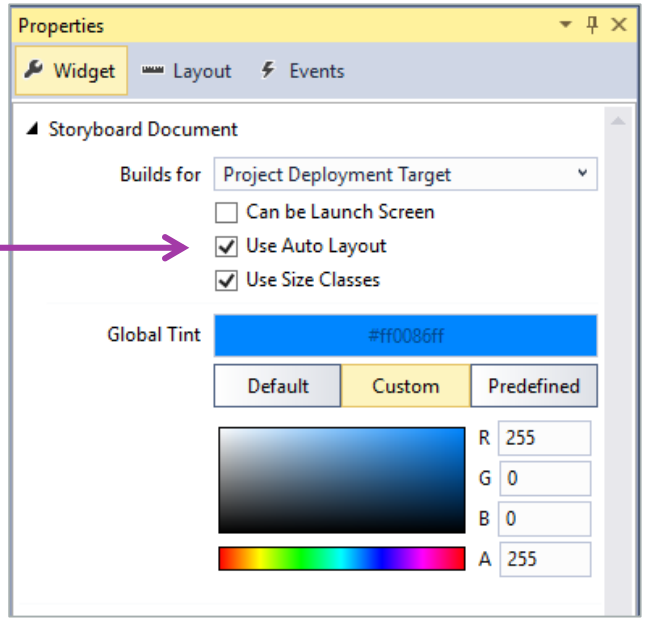
Can set positions relative to other views (or the parent view)

Can set heights and widths

Auto Layout in the Designer

- Storyboard designer allows us to visually manage Auto Layout constraints without writing any code

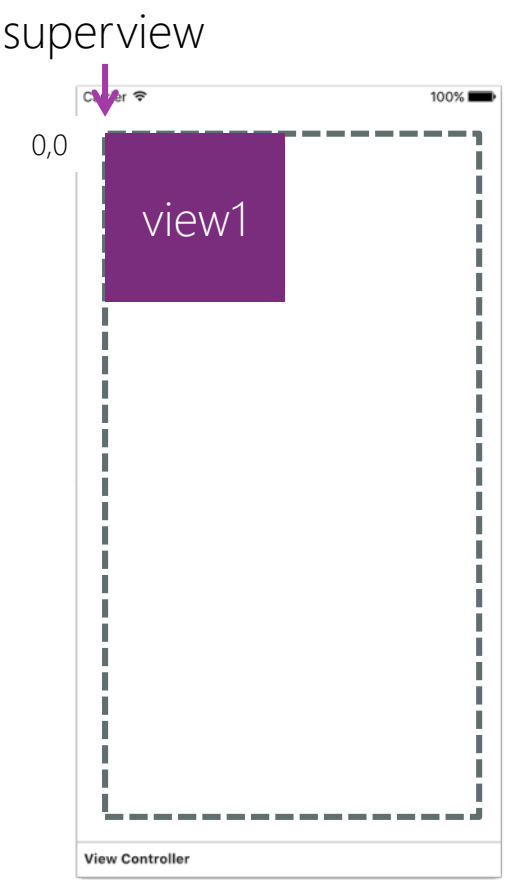
Enabled by default but can be turned on and off in the Storyboard properties



What is a Constraint?

- A Constraint determines *one aspect* of a **`UIView`** position or size and essentially **form the rules** that describe the layout

```
view1.left = superview.left
```

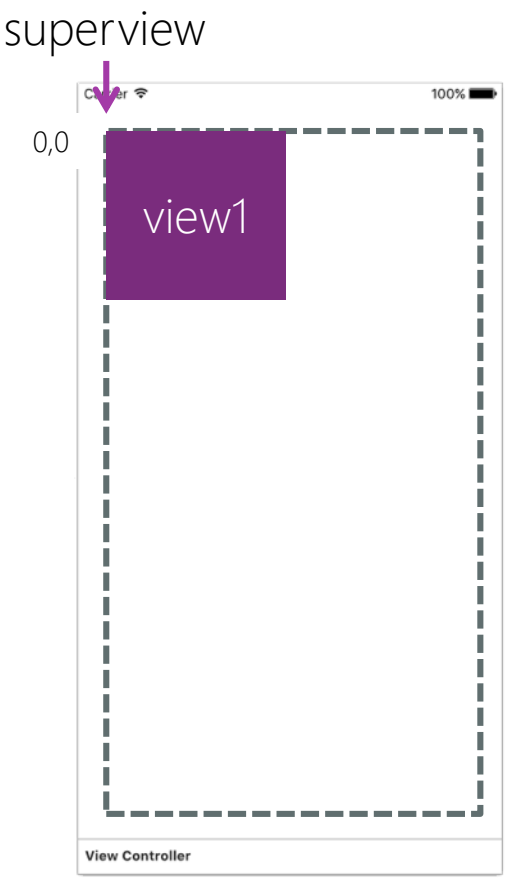


What is a Constraint?

- A Constraint determines *one aspect* of a **`UIView`** position or size and essentially **form the rules** that describe the layout

```
view1.left = superview.left
```

```
view1.top = superview.top + 50
```



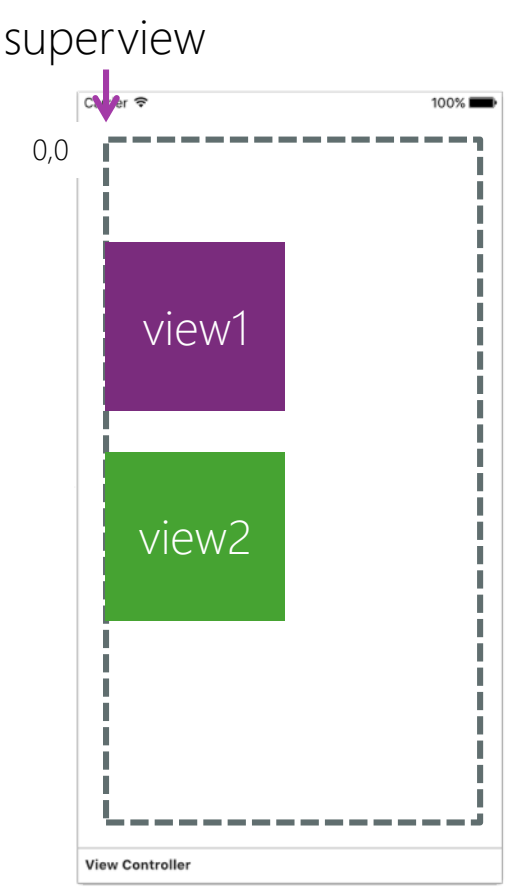
What is a Constraint?

- A Constraint determines *one aspect* of a **`UIView`** position or size and essentially **form the rules** that describe the layout

```
view1.left = superview.left
```

```
view1.top = superview.top + 50
```

```
view2.top = view1.bottom + 8
```



What is a Constraint?

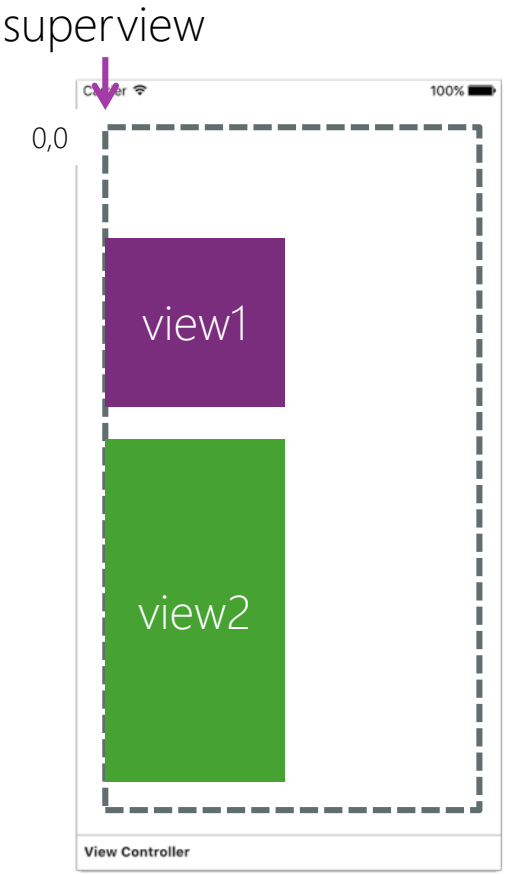
- A Constraint determines *one aspect* of a **`UIView`** position or size and essentially **form the rules** that describe the layout

```
view1.left = superview.left
```

```
view1.top = superview.top + 50
```

```
view2.top = view1.bottom + 8
```

```
view2.height = 0.5 * superview.height
```

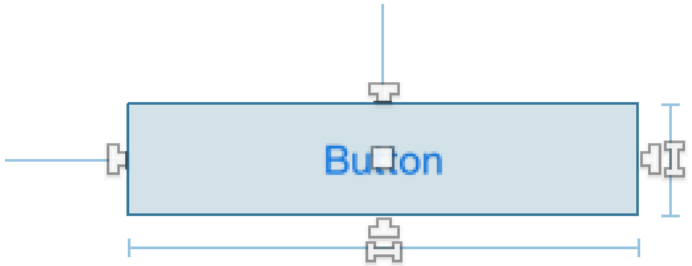


Constraint behavior

- Constraints are applied to views and decide the position and size of the view
- Each constraint defines one property – e.g. **X**, **Y**, **Width** or **Height**
- At runtime, the equation defined by all the constraints is used to define the **`Frame`** for the view

Xamarin.iOS Designer

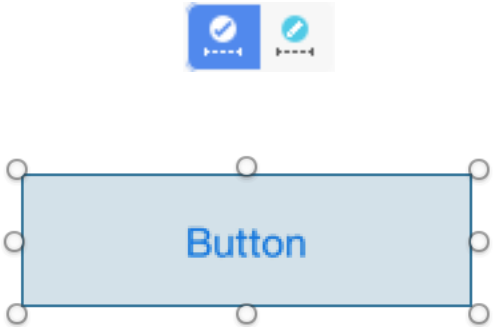
- The Xamarin.iOS Designer adds constraints directly into the Storyboard - iOS will apply them at runtime when the UI is inflated



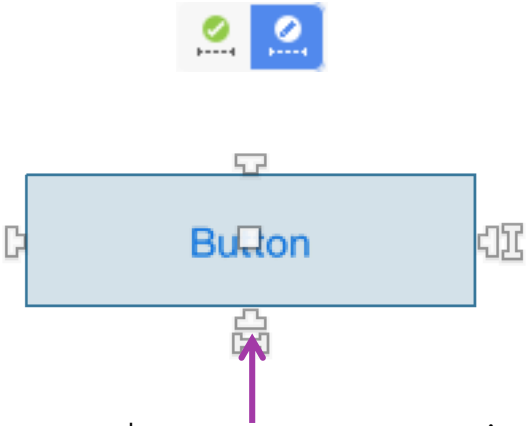
constraints are shown graphically and can be changed either on the designer surface or in the property pad

Positioning views

- In the iOS Designer, **single-tap** views to toggle between editing the frame and editing constraints



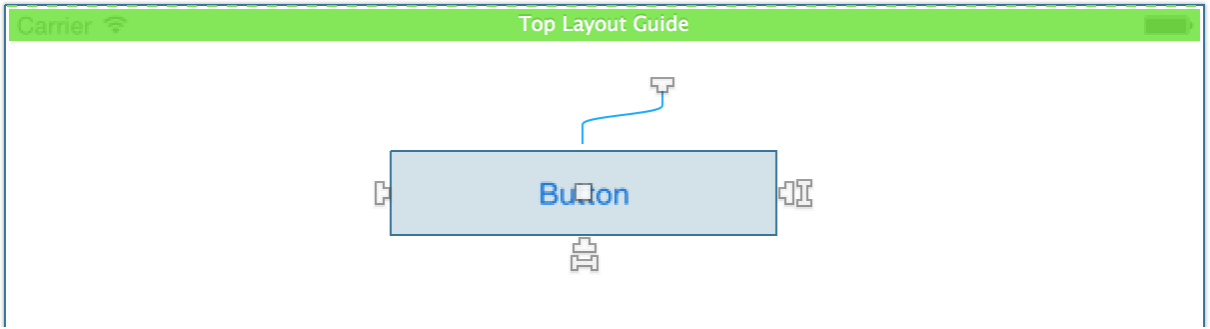
Circles used to adjust the frame



Bars used to create constraints

Adding Constraints

- Use the *dragging control decorators* on a view to create a constraint with itself, the parent, or a sibling view



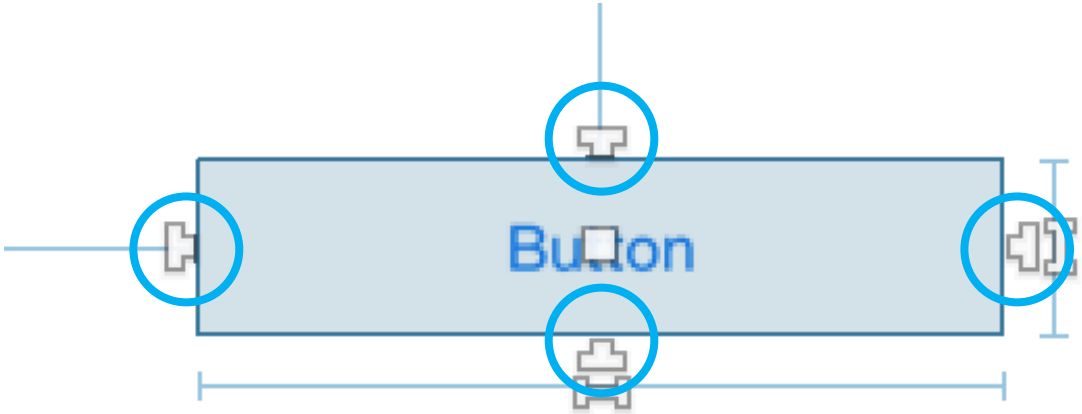
Can select and drag the handles and drop onto the target view

Types of Constraints

- The Xamarin.iOS Designer supports manipulation of three types of constraints to size and position views

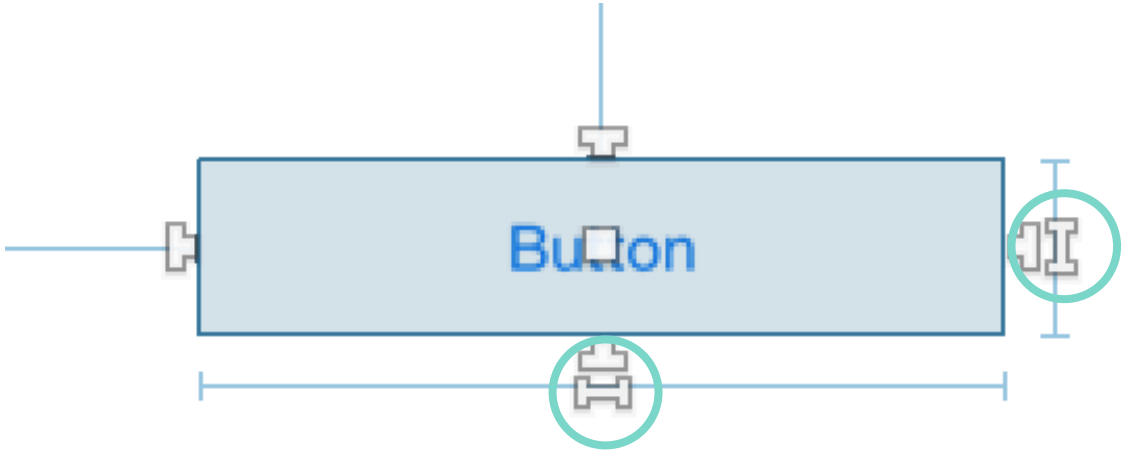
Spacing constraints

- Spacing constraints allow you to position a view relative to another view (or parent) by dragging the T-handle shapes on each edge



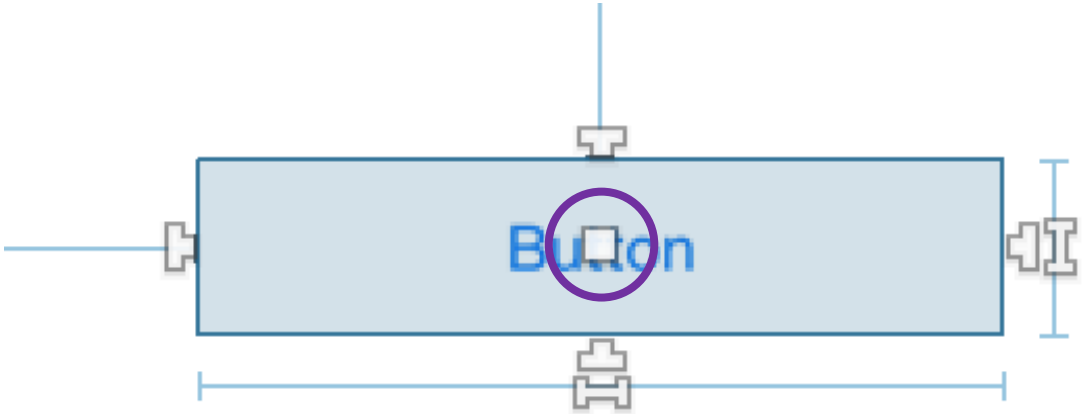
Sizing constraints

- Sizing constraints allow you to control a views width and height (can be a constant, fixed to another constraint, or an inequality) by dragging the center "I" bar shape on the right and bottom edge of the view



Alignment constraints

- Alignment constraints allow you to align a view to the X or Y axis of it's superview or a sibling



Flash Quiz

Xamarin University

Flash Quiz

① Auto Layout _____

a) Is only available in the Designer

b) Describes relationships between visual elements

c) Must be used and cannot be turned off

Flash Quiz

① Auto Layout _____

a) Is only available in the Designer

b) Describes relationships between visual elements

c) Must be used and cannot be turned off

Flash Quiz

② Spacing constraints are used to position a view

a) True

b) False

Flash Quiz

② Spacing constraints are used to position a view

a) True

b) False

Add Recommended Constraints

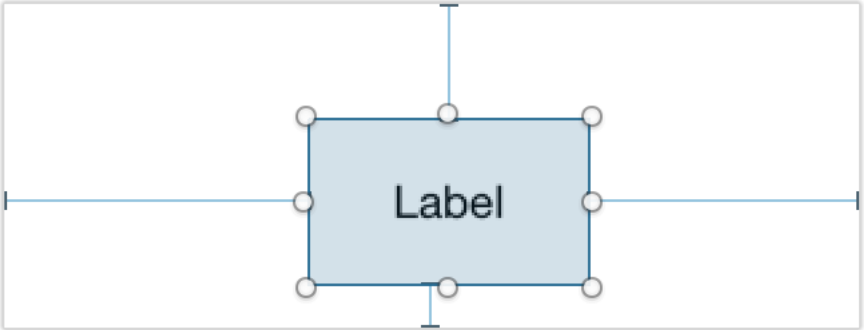
- The Xamarin Designer for iOS can add **recommended constraints** to a View on the design surface

Adds 4 constraints to set position and size

Removes all constraints for a selected view

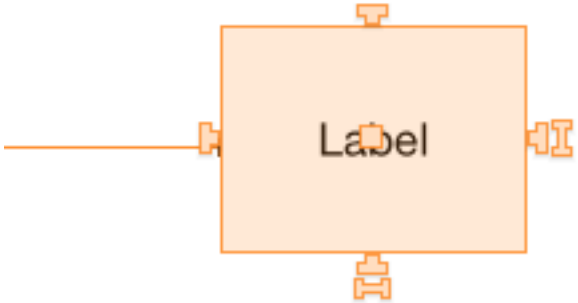
Fully-constrained views

- A *fully-constrained view* has enough constraints to uniquely describe the view's position and size, typically this requires **4 constraints**

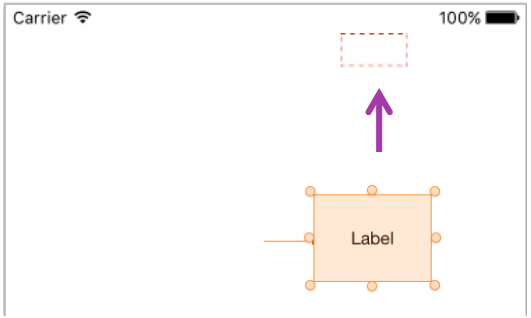


The designer provides feedback

- The designer provides immediate feedback to let you know if your constraints are ambiguous and/or do not match you design position



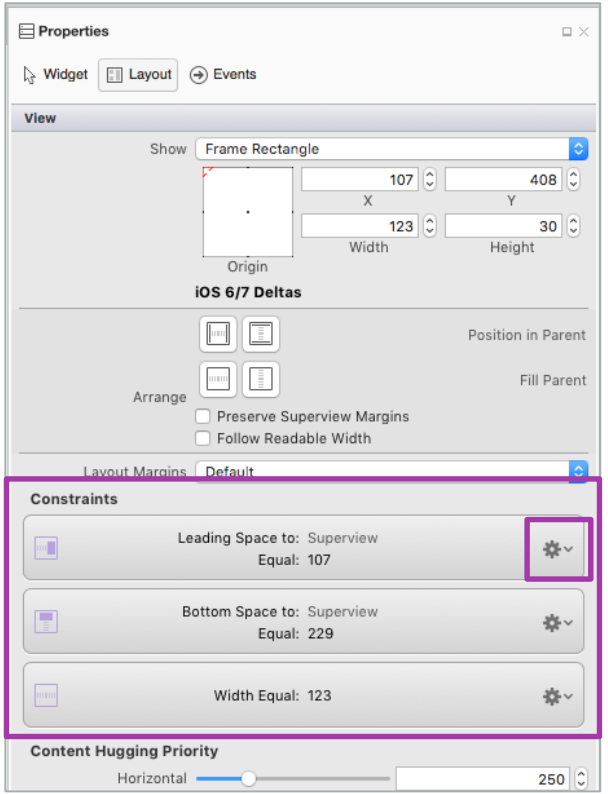
Orange can indicate the view is under constrained



Orange with a dotted rectangle indicates the design and runtime positions don't match

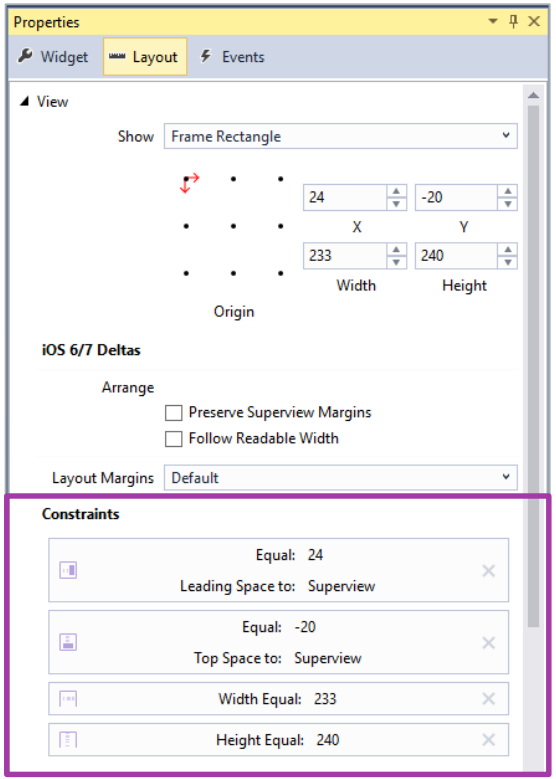
Editing Constraints [macOS]

- **Layout Area in the Properties Pane** provides a more powerful way to edit and manage constraints
  - Provides an overview of all constraints
  - Can "fine-tune" constraints through an inline editor



Editing Constraints [Windows]

- **Layout Area in the Properties Pane** provides a more powerful way to edit and manage constraints
  - Provides an overview of all constraints
  - Can "fine-tune" constraints through an inline editor



Group Exercise

Add constraints to the fireworks app

Xamarin University

Summary

1. Describe the Auto Layout system
2. Identify constraints
3. Add constraints using the Designer

Interact with designer-defined views programmatically

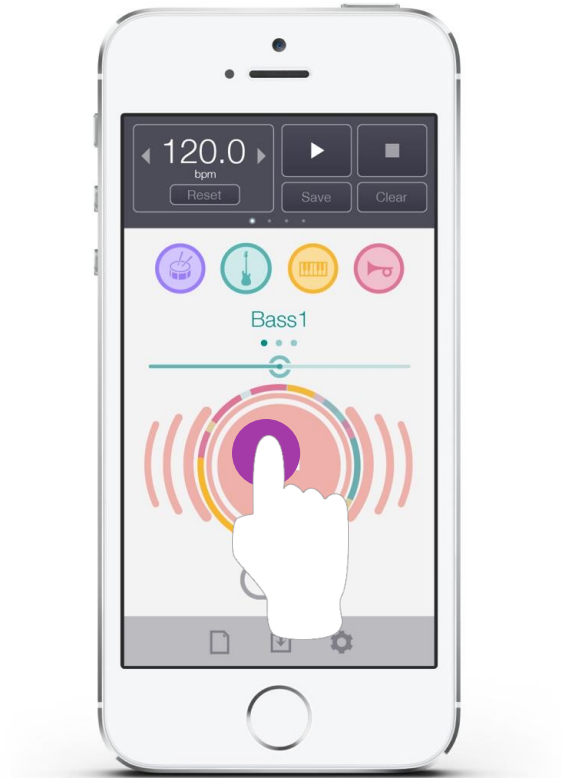
Xamarin University

Tasks

1. Describe the Auto Layout system
2. Identify constraints
3. Add constraints using the Designer

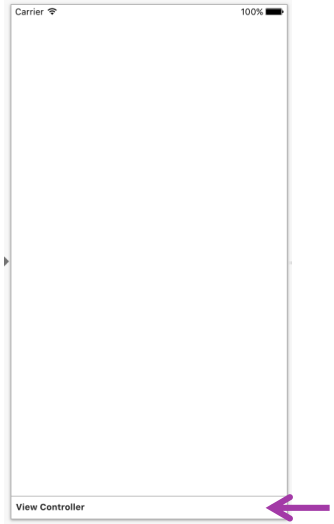
User interaction

- Applying behavior to the views of an app is essential if you want your code to respond to user interactions

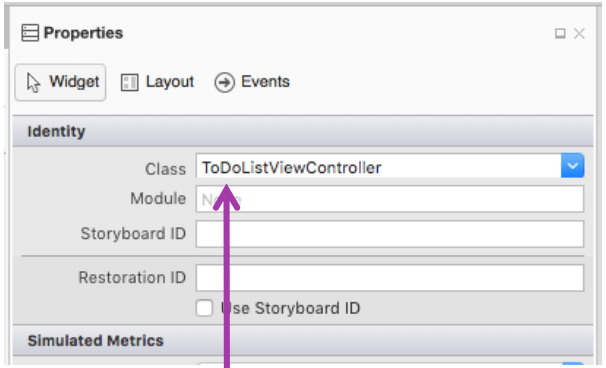


Assign a class

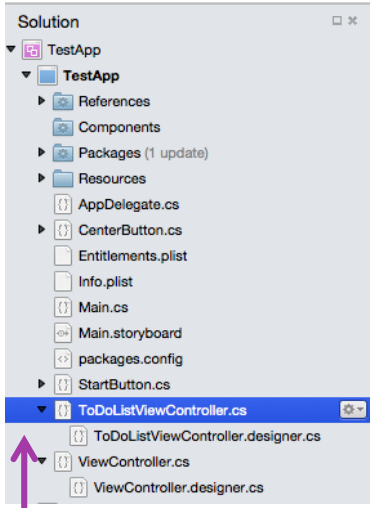
- In order to add behavior programmatically to your controls - the **`UIViewController`** must have an associated class



Select the view controller in the designer



Assign a class name

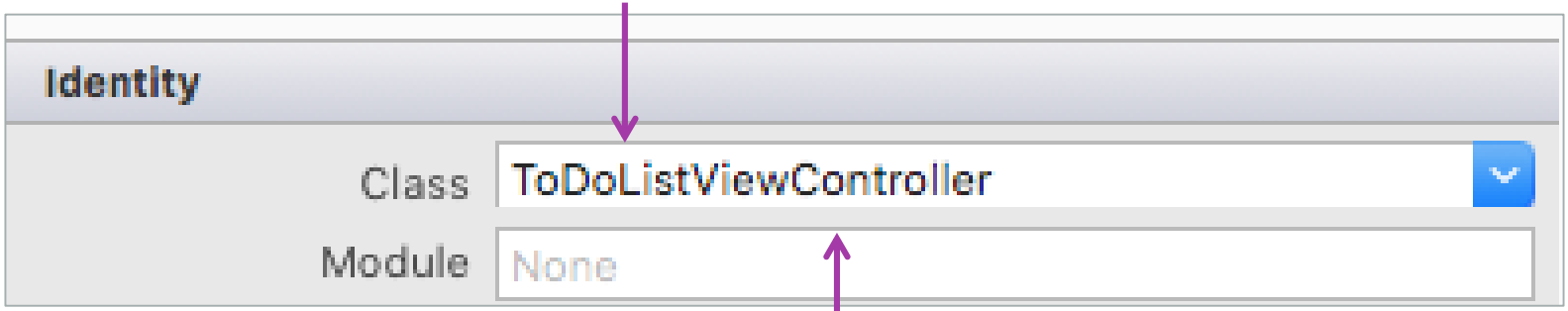


Xamarin will create a C# file in the solution

Naming your view controller

- It's recommended to use a consistent, meaningful names when creating an associated class for View Controller

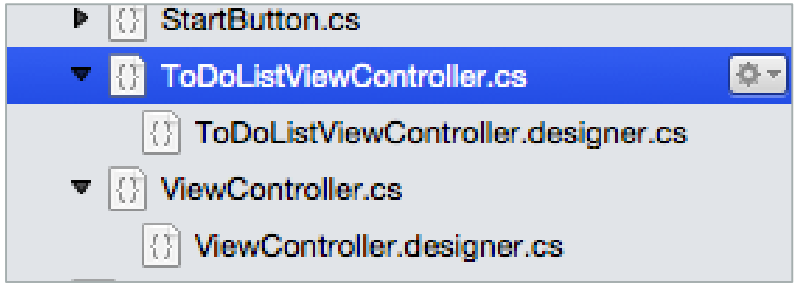
Name should reflect what the screen *does* or *manages*



Name should end with "ViewController"

Partial classes [main file]

- When you assign a class to a ViewController using the Xamarin.iOS Designer, the class will be declared as a partial class and split across two files: a **.cs** and a **.designer.cs**

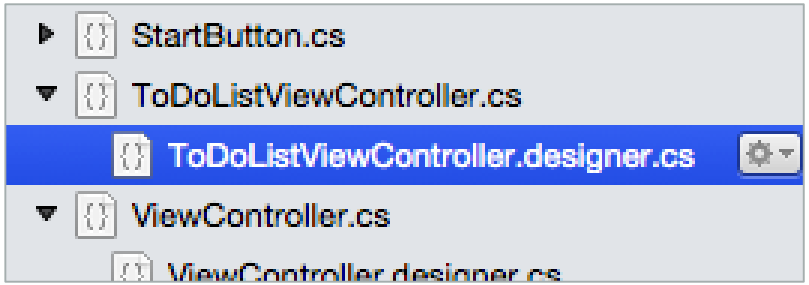



```
namespace TestApp
{
    partial class ToDoListViewController : UIViewController
    {
        public ToDoListViewController (IntPtr handle) : base
        {
        }
    }
}
```

The .cs file is where you will code the behavior of your view controller

Partial classes [designer file]

- When you assign a class to a ViewController using the Xamarin.iOS Designer, the class will be declared as a partial class and split across two files: a **.cs** and a **.designer.cs**




```
namespace TestApp
{
    [Register ("ToDoListViewController")]
    partial class ToDoListViewController
    {
        [Outlet]
        [GeneratedCode ("iOS Designer", "1.0")]
        UIButton MyButton { get; set; }

        [Action ("MyButton_TouchUpInside:")]
        [GeneratedCode ("iOS Designer", "1.0")]
```

The **designer.cs** file is a representation of the storyboard for the compiler – it is auto generated and should not be edited directly

Registering a class with iOS

- Classes that will be instantiated by iOS need to be *registered* with the Objective-C runtime – this is done through a **[Register]** attribute

```
namespace TestApp
{
    [Register ("ToDoListViewController")]
    partial class ToDoListViewController
    {
        [Outlet]
        [GeneratedCode ("iOS Designer", "1.0")]
        UIButton MyButton { get; set; }

        [Action ("MyButton_TouchUpInside:")]
        [GeneratedCode ("iOS Designer", "1.0")]
```

The [Register] attribute is added automatically by the designer

View controller constructor

- iOS uses a custom constructor to create the View Controller

```
public partial class ViewController : UIViewController
{
      public ViewController(IntPtr handle) : base(handle)
      {
      }

      ...
}
```

Must have this constructor and chain to the base if iOS is going to instantiate this View Controller (e.g. load from a Storyboard)

Flash Quiz

Xamarin University

Flash Quiz

① The `.designer.cs` file

a) contains the behavior for a designer-defined view

b) is a representation of the storyboard in code

c) All of the above

d) None of the above

Flash Quiz

① The `.designer.cs` file

a) contains the behavior for a designer-defined view

b) is representation of the storyboard in code

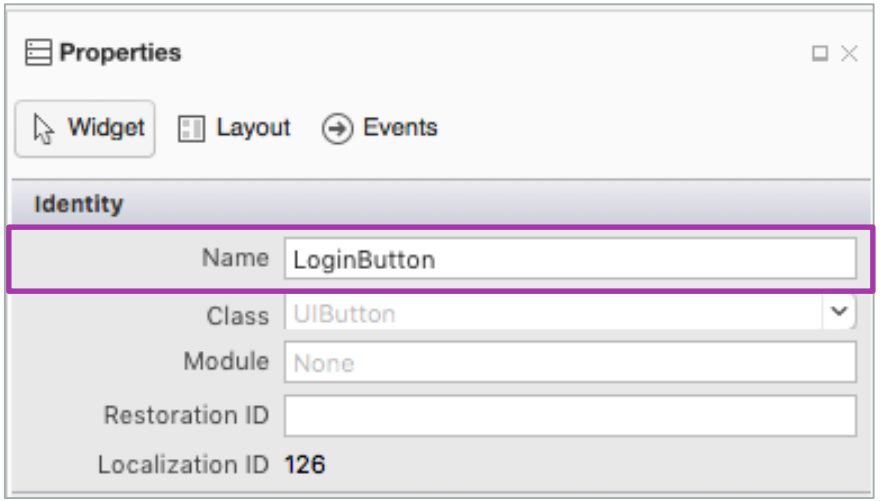
c) All of the above

d) None of the above

Name your view

- Name your designer-defined views to make them accessible in the View Controller associated class

Select the control in the design surface and then set the Name



Hint: as with naming View Controllers, it is advisable to use a name which shows the purpose and the type

What is an Outlet?

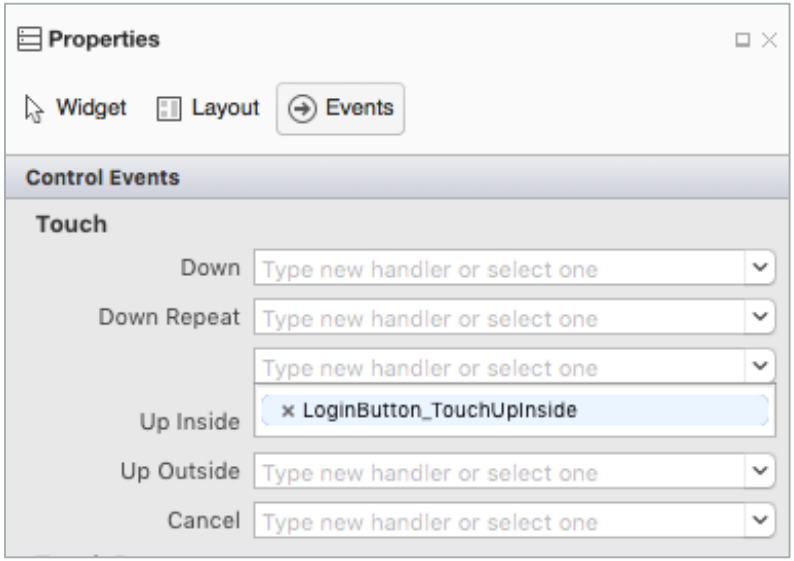
- An **Outlet** defines a property used to access a designer-defined view

```
[Register ("TodoListViewController")]
partial class TodoListViewController
{
    [Outlet]
    [GeneratedCode ("iOS Designer", "1.0")]
    UIButton LoginButton { get; set; }
    ...
}
```

Designer adds this code to your **designer.cs** file when you name a control in the storyboard

What is an Action?

- Actions are methods that are called by a view in response to a runtime interaction or event
- In the Designer you can choose **Events** on the properties pane and associate methods to the actions the selected view raises at runtime
- Can double-click on most controls to add a handler for the "default" action



Implementing Actions

- Actions wired up in the designer are mapped to partial methods defined in the designer portion of your View Controller class and are implemented in the associated class

Created by the designer

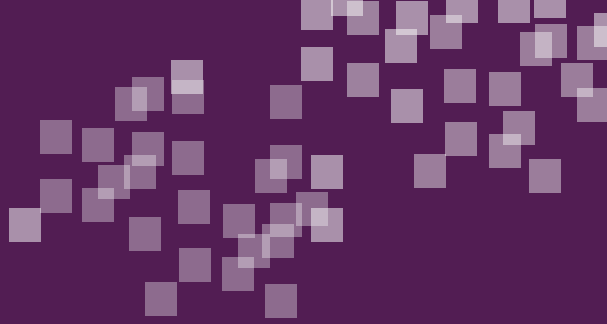
ViewController.designer.cs

```
[Action ("StartLogin:")]
[GeneratedCode ("iOS Designer", "1.0")]
partial void StartLogin (UIButton sender);
```

You add this in the associated .cs file

```
partial void StartLogin(UIButton sender) {
    // TODO: add logic here
}
```

ViewController.cs



Individual Exercise

Code behaviors for your app

Summary

1. Associate a class for the `UIViewController`
2. Identify partial methods
3. Name views
4. Inspect outlets and actions

Navigate between view controllers

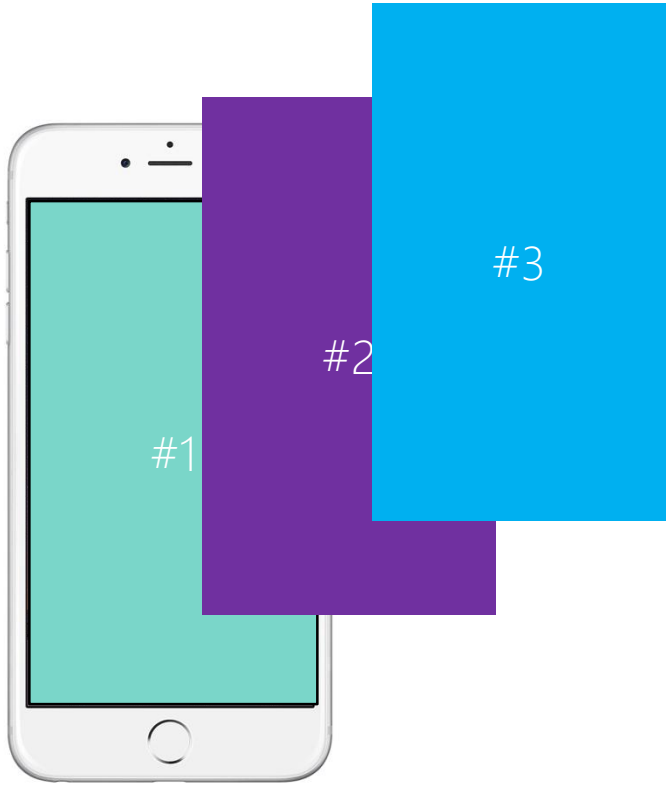
Xamarin University

Tasks

1. Present a view controller
2. Dismiss a view controller programmatically
3. Use segues to perform navigation

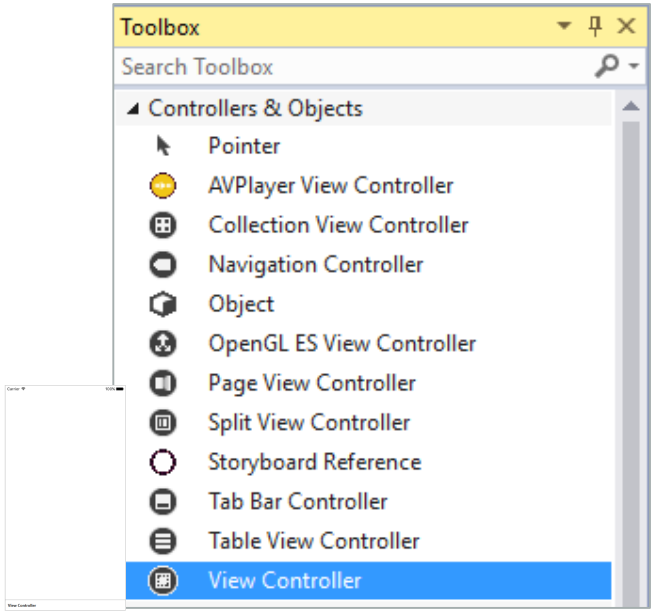
Multi-screen apps

- Most applications consist of more than one screen
- Can define multiple screens in the Storyboard
- Can then display secondary screens through code, or by defining the relationships in the designer



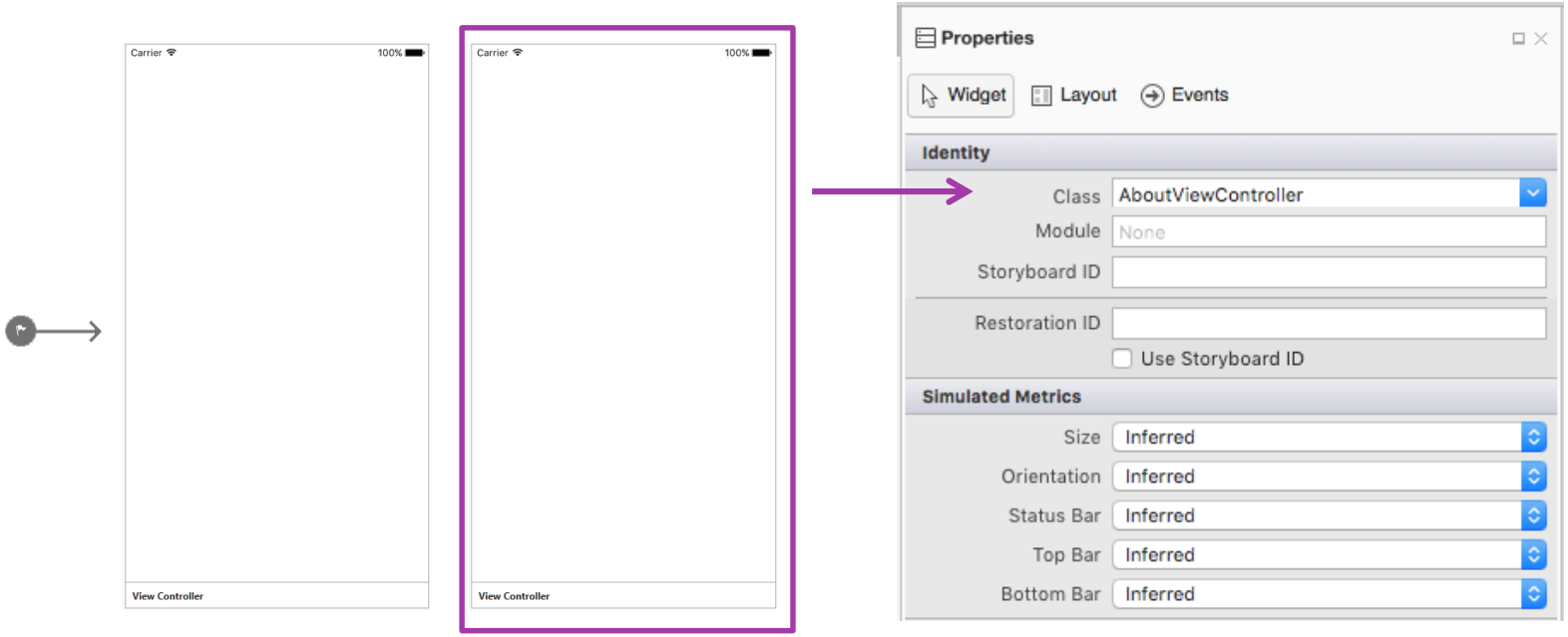
Adding multiple View Controllers

- You can add additional screens to your storyboard by dragging additional view controllers onto the storyboard design surface



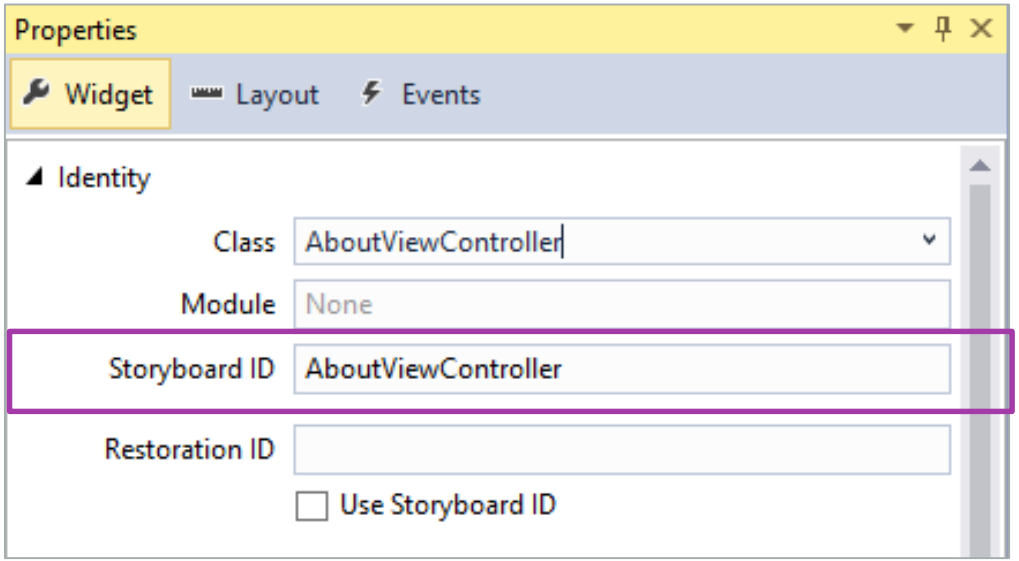
View Controller associated classes

- Can associate classes to each view controller as required



Naming a View Controller

- Must set the Storyboard ID on the View Controller to identify the View Controller in code – a good practice is to set the Class and the Storyboard ID to the same value



Instantiating a view controller in C#

- View Controllers defined in Storyboards must be created through the Storyboard APIs to get the proper views created

```
partial void ShowAboutPage(UIButton sender) {

   UIStoryboard storyboard = this.Storyboard;
   AboutViewController viewController = (AboutViewController)
     storyboard.InstantiateViewController("AboutViewController");
   ...
}
```

Instantiate a view controller in C#

- Can instantiate designer-defined view controllers programmatically using the Storyboard APIs

```
partial void ShowAboutPage(UIButton sender) {

   UIStoryboard storyboard = this.Storyboard;
   AboutViewController viewController = (AboutViewController)
     storyboard.InstantiateViewController("AboutViewController");
   ...
}
```

Designer-defined view controllers contain a reference to their storyboard

Use the **InstantiateViewController** method, passing in the name of the view controller as defined in the storyboard

Present the view controller

- Can use the **`PresentViewController`** method to display a new View Controller in a modal fashion on top of your existing screen

```
partial void ShowAboutPage(UIButton sender)
{
    UIStoryboard storyboard = this.Storyboard;
    AboutViewController viewController = (AboutViewController)
      storyboard.InstantiateViewController("AboutViewController");

    this.PresentViewController(viewController, true, null);
}
```

Dismiss a modal view controller

- To return to the previous View Controller, use the **`DismissViewController`** method in your active view controller

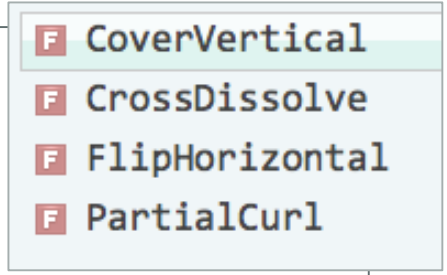
```
partial class AboutViewController : UIViewController
{
    ...
    partial void OnGoBack(UIButton sender)
    {
        this.DismissViewController(true, null);
    }
}
```

Changing the transition style

- Can customize the animation used to transition to the new controller through the **ModalTransitionStyle** property

```
partial void ShowAboutPage(UIButton sender)
{
    AboutViewController viewController = ...;
    viewController.ModalTransitionStyle =
            UIModalTransitionStyle.PartialCurl;

    this.PresentViewController(viewController, true, null);
}
```



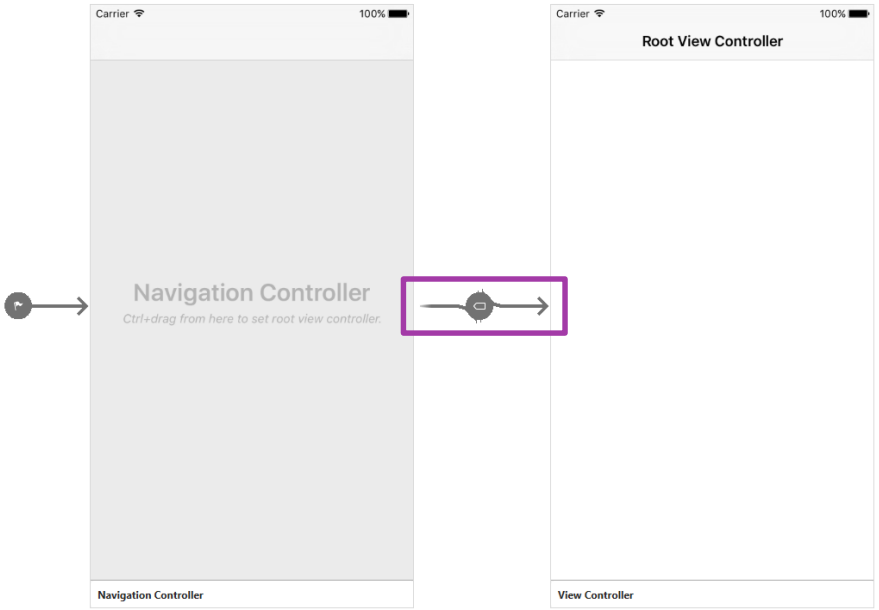
Individual Exercise

Add a second screen to your app
and code a button to navigate to it

Xamarin
University

What is a Segue?

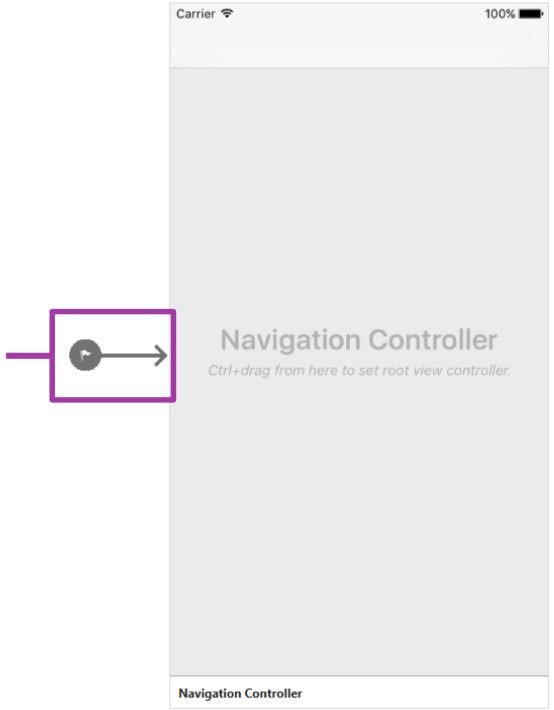
- *Segues* ("segways") define the transitions between the screens of our application in the Xamarin.iOS Designer

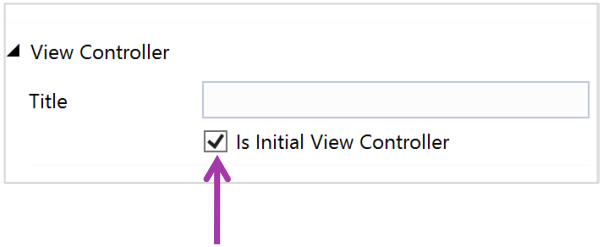


Sourceless segue

- The **sourceless segue** indicates the root (initial) view controller

Click+Drag to move the **sourceless segue** to a different screen



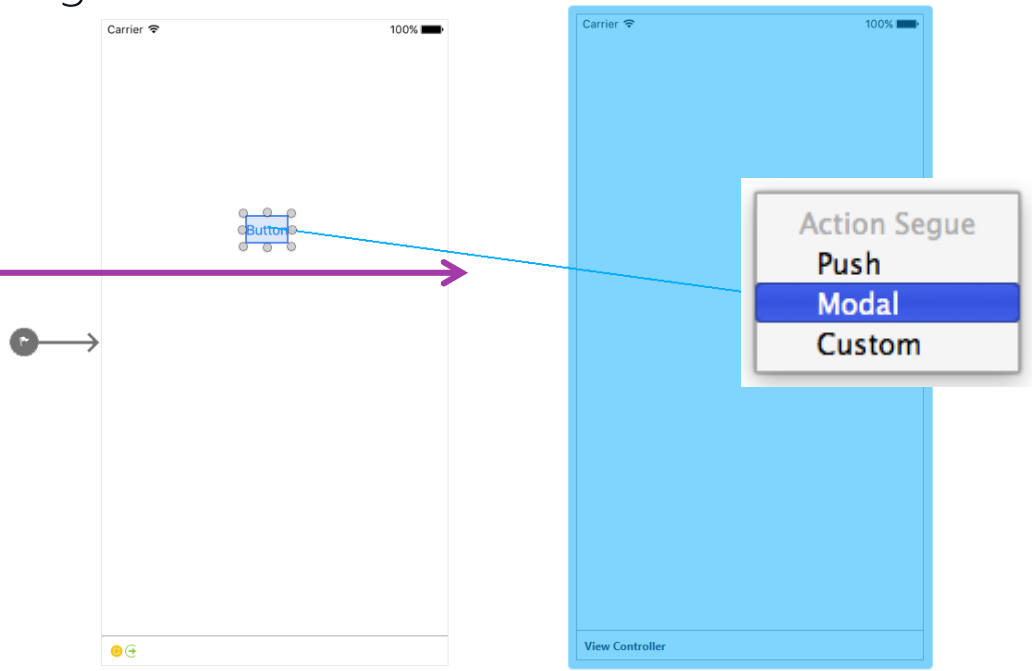


… or select the view controller and check Is Initial View Controller in the properties

Create a segue relationship

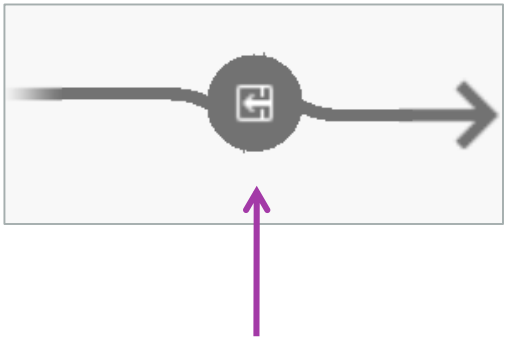
- Use **`Ctrl+Drag`** to create segues between two screens

The blue connector appears as you drag your mouse from a control to the target screen

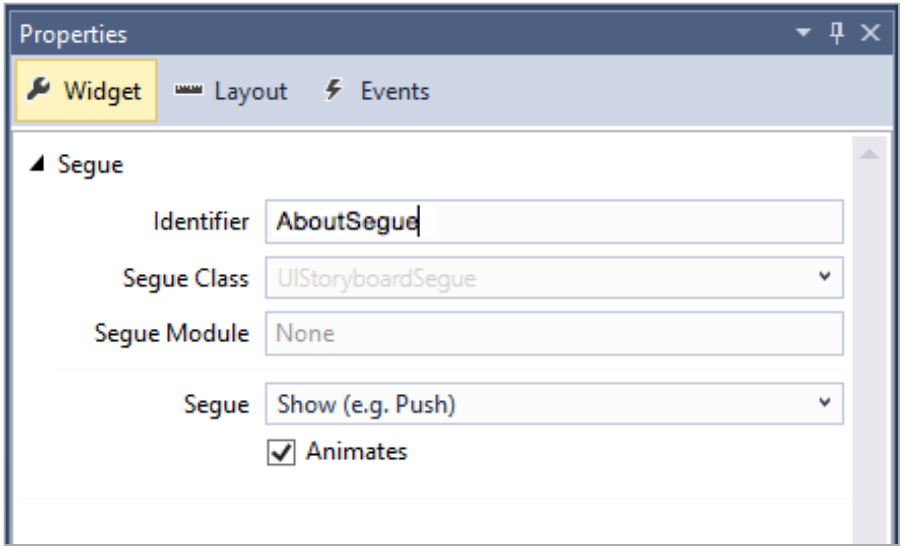


Segue properties

- Segues define properties that can be used to control the Segue behavior and reach it programmatically

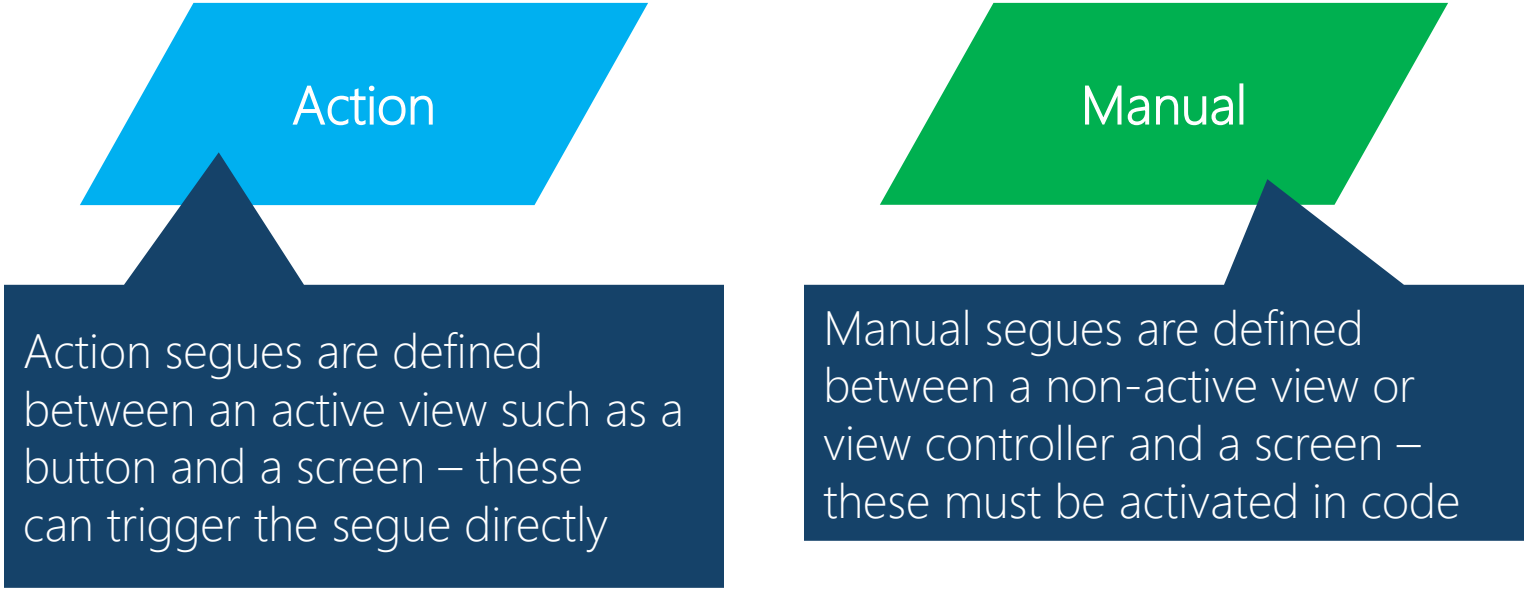


Select the segue on the storyboard to view segue options



Relationship types

- There are two types of Segue relationships that can be created:



Run a segue from code

- Can use **PerformSegue** in a View Controller to initiate a segue from code – this allows you to define the transition in the Storyboard, but decide when to run it based on your application logic

```
partial void ShowAboutPage(UIButton sender)
{
    this.PerformSegue("AboutSegue", this);
}
```

Takes the identifier of the segue

.. And the sender

Stopping a segue

- Sometimes you need to stop a segue from occurring due to some application state

```
public override bool ShouldPerformSegue(
          string segueIdentifier, NSObject sender)
{
    if (segueIdentifier != "AboutSegue")
       return false; // do not run any segue except About

    return true; // allow segue
}
```

Influence a segue

- Sometimes you need to just setup the target screen with some data from the source – can use **PrepareForSegue** override

```
public override void PrepareForSegue(UIStoryboardSegue segue,
                                      NSObject sender)
{
    if (segue.Identifier == "AboutSegue") {
        var vc = segue.DestinationViewController as
                        AboutViewController;
        vc.RegisteredUserName = ...; // Some custom property
    }
}
```

Individual Exercise

Add segues to define the navigation

Xamarin University

Summary

1. Present a view controller
2. Dismiss a view controller programmatically
3. Use segues to perform navigation

Thank You!

Please complete the class survey in your profile:
university.xamarin.com/profile