Xamarin.iOS Fundamentals: Views and View Controllers

Presented By: Lori Lalonde
Technical Evangelist – Microsoft Canada
@loriblalonde
Blog: solola.ca

Special Thanks To Our Sponsor

Microsoft®

Be what's next.™

Building Blocks of an iOS Application

- UIApplication
  - facilitates interaction between the system and other objects in the app
- UIApplicationDelegate
  - works with the **UIApplication** object to handle app initialization, state transitions, and app events, as well as to set up the app's initial data structures.

Building Blocks of an iOS Application

- UIWindow
  - basic container for the application’s views
- UIScreen
  - represents the screen on the physical device

Building Blocks of an iOS Application

- Views (UIView)
  - represents a rectangular area in the UI, responsible for drawing content, handling events, and managing the layout of subviews
  - **UIView** class is used to create a View
  - Examples: label, button, text field, etc.
  - can be nested inside other views (subviews)
  - Single screen is made up of a view hierarchy

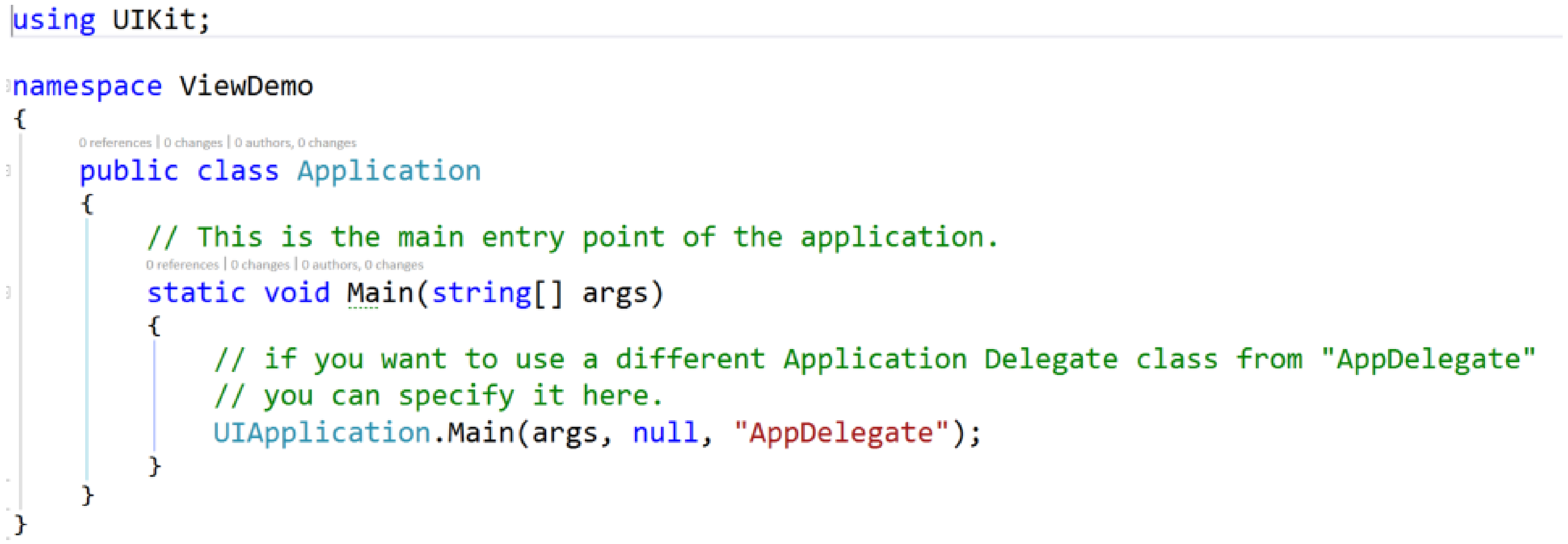
Building Blocks of an iOS Application

- View Controllers (UIViewController)
  - manage a portion of your app's user interface as well as the interactions between that interface and the underlying data
  - facilitate transitions between different parts of your user interface

Xamarin.iOS Application

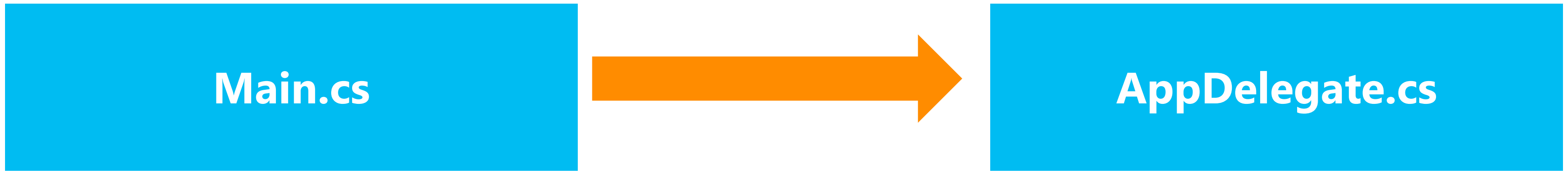
Main.cs

- Main entry point
- Creates a new UIApplication instance, naming the AppDelegate class

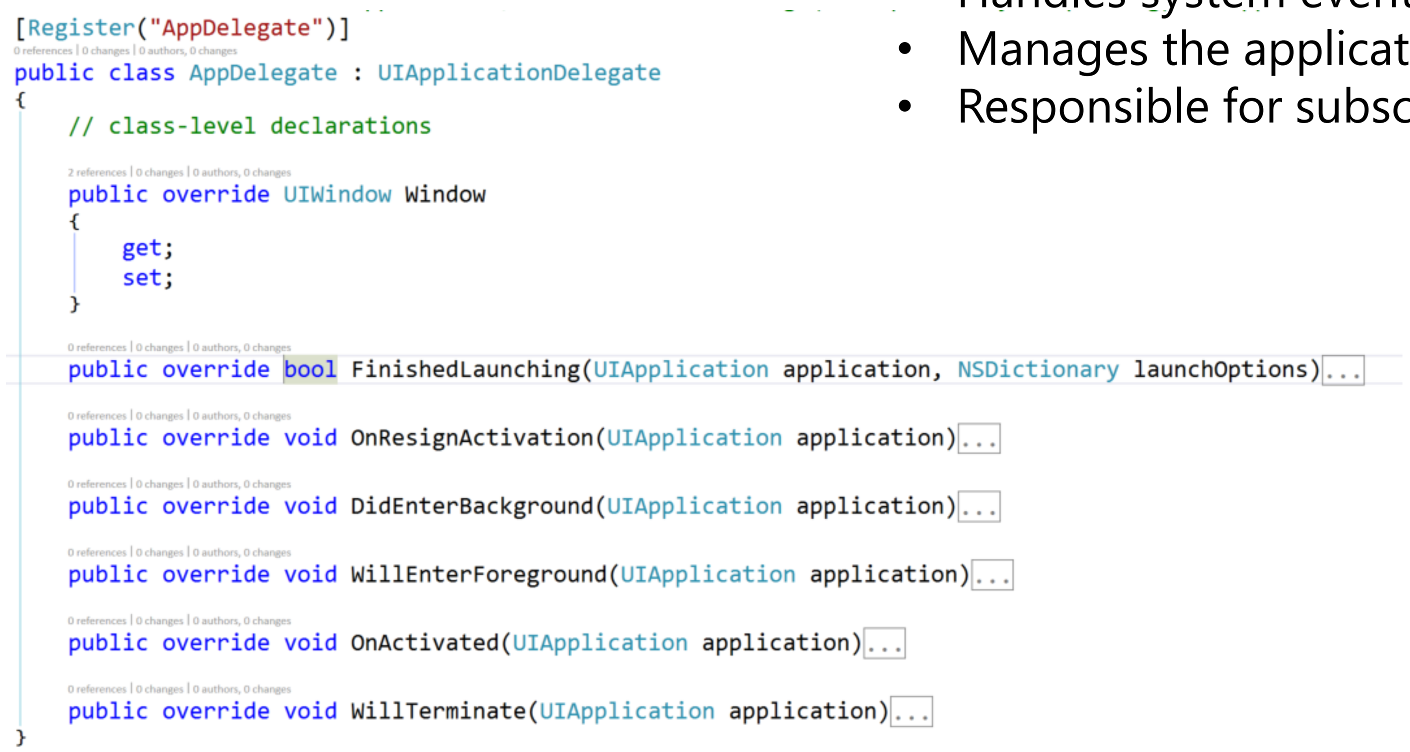


```
using UIKit;

namespace ViewDemo
{
    0 references | 0 changes | 0 authors, 0 changes
    public class Application
    {
        // This is the main entry point of the application.
        0 references | 0 changes | 0 authors, 0 changes
        static void Main(string[] args)
        {
            // if you want to use a different Application Delegate class from "AppDelegate"
            // you can specify it here.
            UIApplication.Main(args, null, "AppDelegate");
        }
    }
}
```

Xamarin.iOS Application

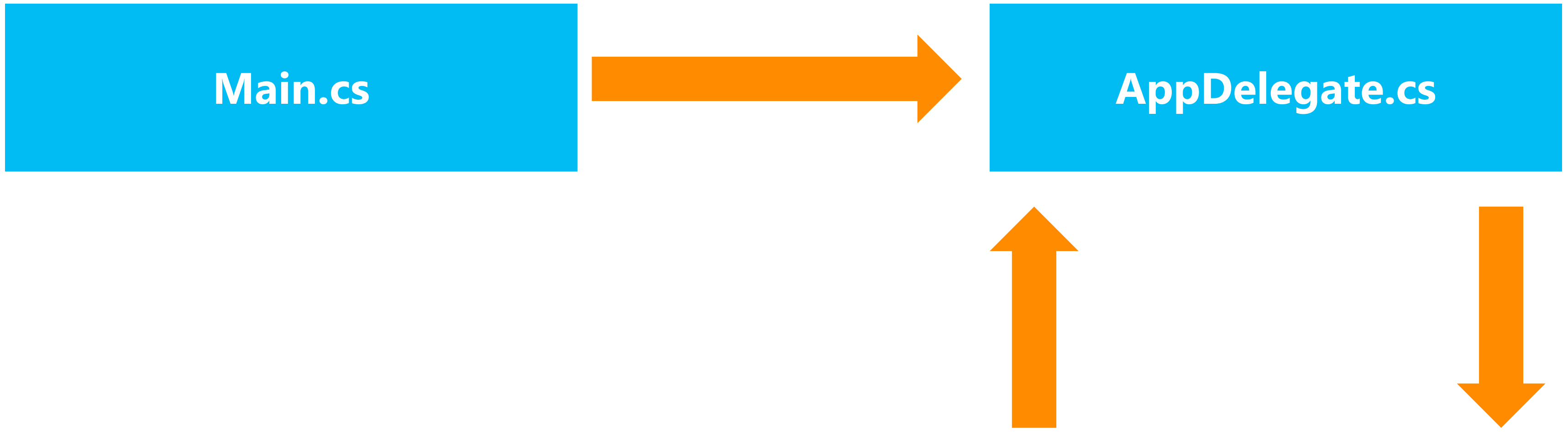


- Handles system events
- Manages the application window
- Responsible for subscribing to system updates



```
[Register("AppDelegate")]
0 references | 0 changes | 0 authors, 0 changes
public class AppDelegate : UIApplicationDelegate
{
    // class-level declarations

    2 references | 0 changes | 0 authors, 0 changes
    public override UIWindow Window
    {
        get;
        set;
    }

    0 references | 0 changes | 0 authors, 0 changes
    public override bool FinishedLaunching(UIApplication application, NSDictionary launchOptions)...

    0 references | 0 changes | 0 authors, 0 changes
    public override void OnResignActivation(UIApplication application)...

    0 references | 0 changes | 0 authors, 0 changes
    public override void DidEnterBackground(UIApplication application)...

    0 references | 0 changes | 0 authors, 0 changes
    public override void WillEnterForeground(UIApplication application)...

    0 references | 0 changes | 0 authors, 0 changes
    public override void OnActivated(UIApplication application)...

    0 references | 0 changes | 0 authors, 0 changes
    public override void WillTerminate(UIApplication application)...
}
```

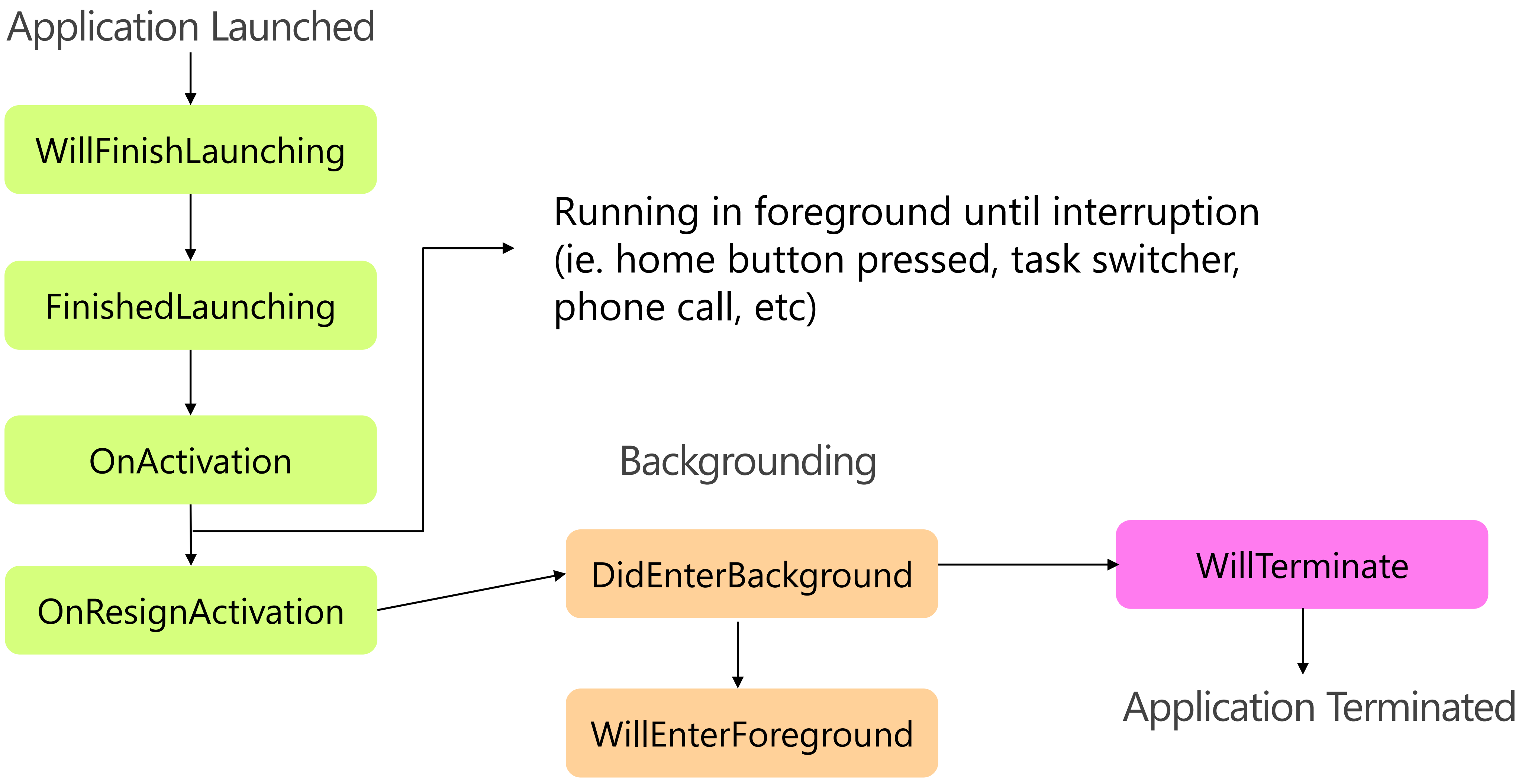
Xamarin.iOS Application



Receives system updates from iOS

- Creates UIWindow and attaches to UIScreen
- UIWindow.RootViewController is defined
- User interface is loaded

Application Lifecycle Events



View Lifecycle Events

- ViewDidLoad
  - Use this to perform initial setup of your view
- ViewWillAppear
  - Use this to restore view state
- ViewDidAppear
  - View is added to the view hierarchy
- ViewWillDisappear
  - Use this to clean up resources and save state
- ViewDidDisappear
  - View is removed from the view hierarchy

User Interface Controls

- Derive from UIControl, which derive from UIView
- Common UI Controls:
  - UIScrollView
  - UILabel
  - UIButton
  - UITextField
  - UITextView
  - UIImageView
  - UISlider

Designing the User Interface

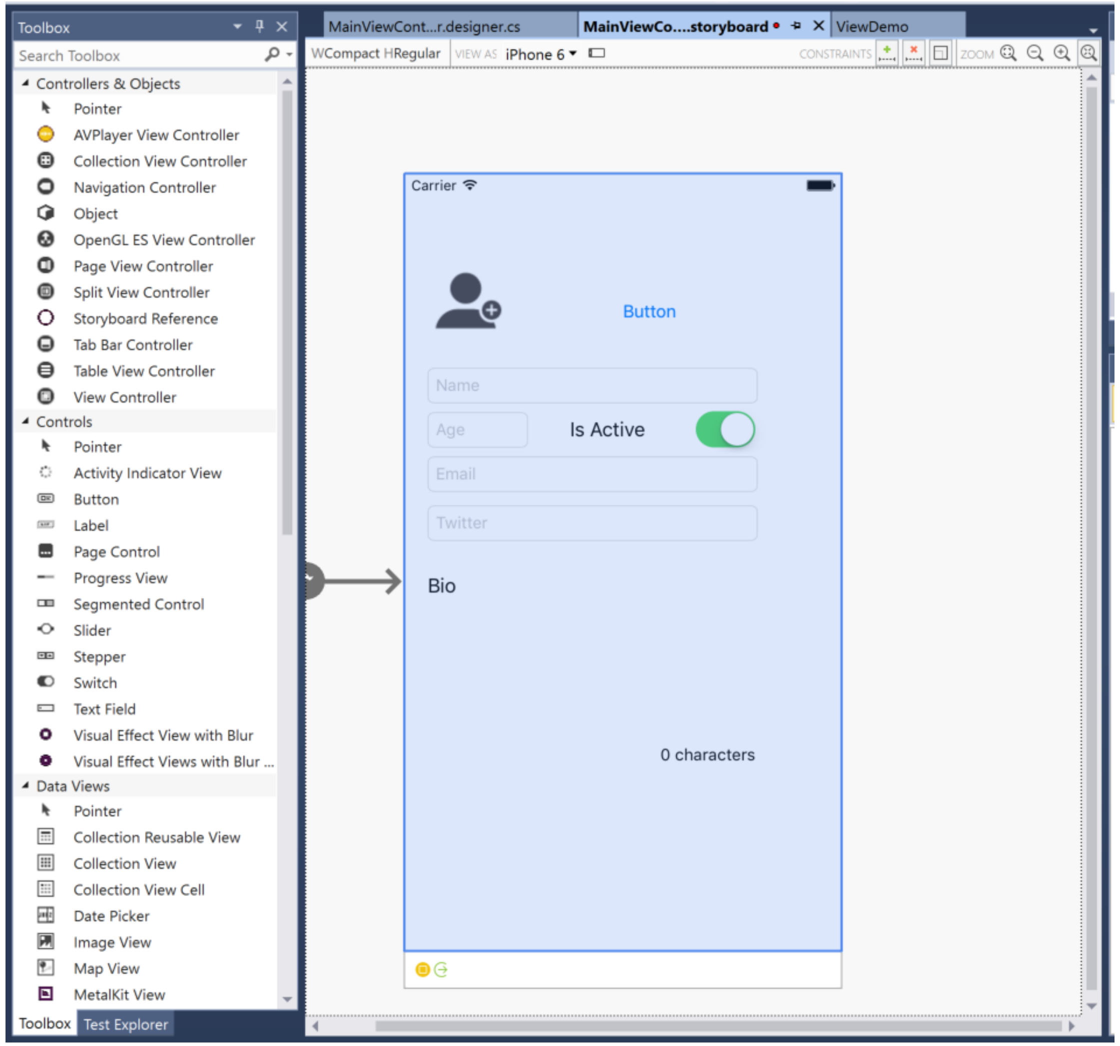
- Add views to a ViewController's layout using
  - Xamarin.iOS Designer
  - Xcode Interface Builder
  - Programmatically in C#
- Register event handlers for views which require user interaction

UIControl Events

- TouchCancel
- TouchDown
- TouchDownRepeat
- TouchDragEnter
- TouchDragExit
- TouchDragInside
- TouchDragOutside
- TouchUpInside
- TouchUpOutside
- EditingChanged
- EditingDidBegin
- EditingDidEnd
- EditingDidEndOnExit
- ValueChanged

Designing UI in the Storyboard

- Drag and drop controls to the View Controller from the Toolbox
- Configure constraints, auto-layouts*
- Create Segues*



* Out of scope for today's workshop

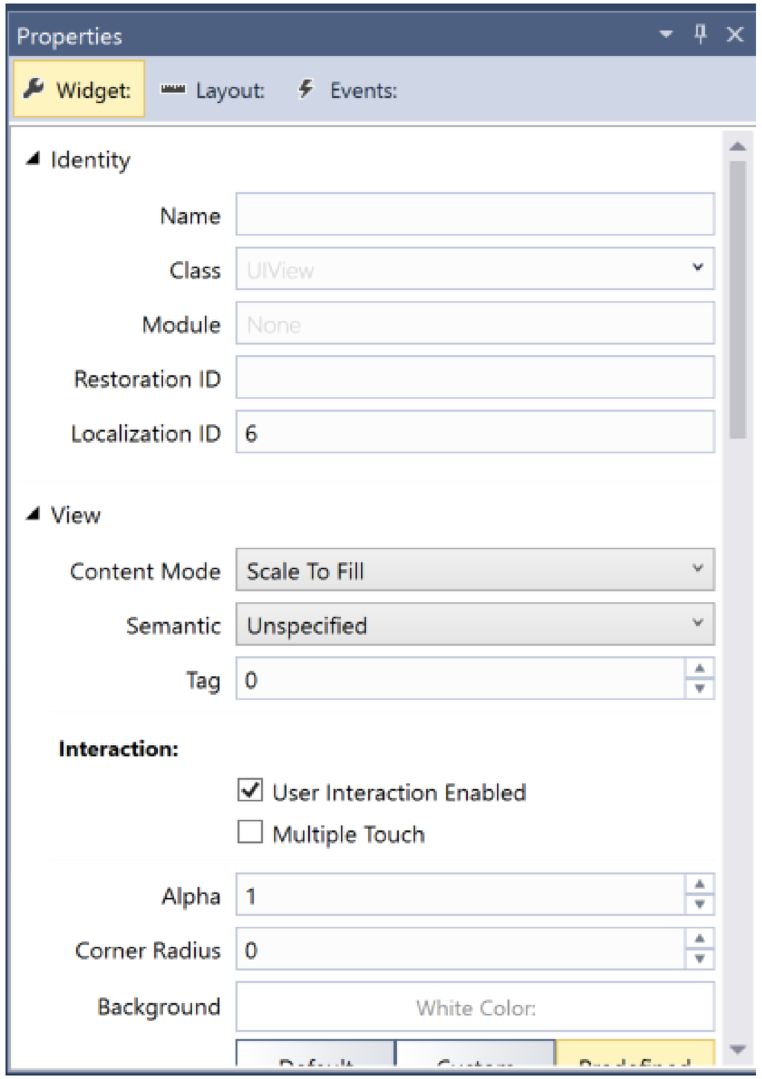
@loriblalonde

solola.ca

Designing UI in the Storyboard

- Set properties on the control within the Widgets
- When a control's name is defined, it is added as an outlet in the ViewController.designer.cs file to expose the control in code

```
[Outlet]
[GeneratedCode ("iOS Designer", "1.0")]
4 references | 0 changes | 0 authors, 0 changes
UIKit.UILabel CharsEnteredLabel { get; set; }
```


Properties

Widget: Layout: Events:

Identity

Name	
Class	UIView
Module	None
Restoration ID	
Localization ID	6

View

Content Mode	Scale To Fill
Semantic	Unspecified
Tag	0

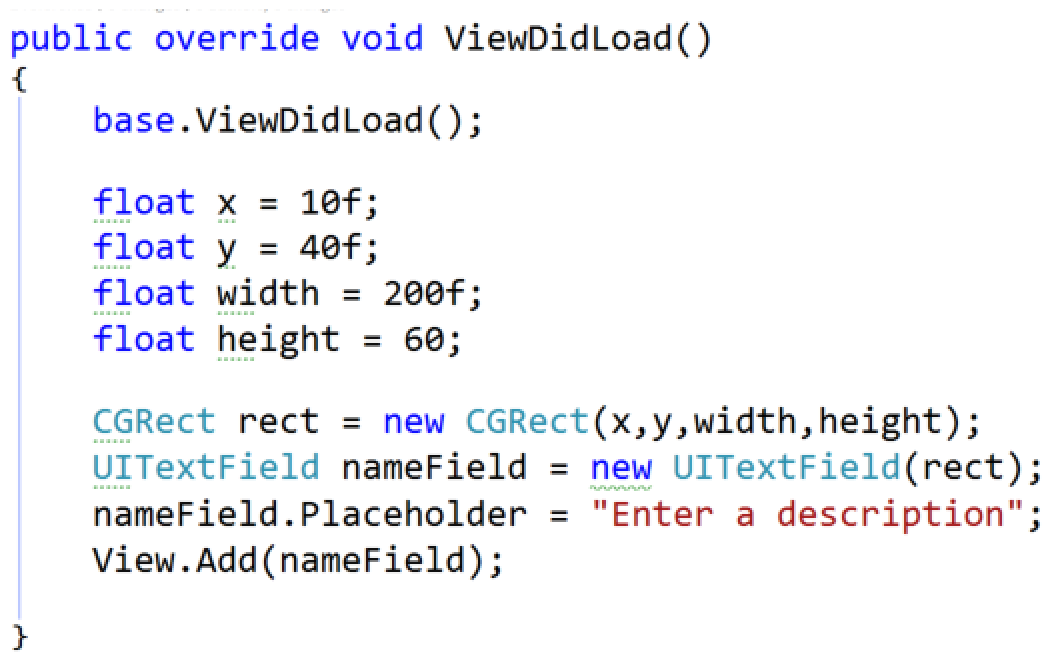
Interaction:

- [✓] User Interaction Enabled
- [ ] Multiple Touch

Alpha	1
Corner Radius	0
Background	White Color:

Designing UI in C#

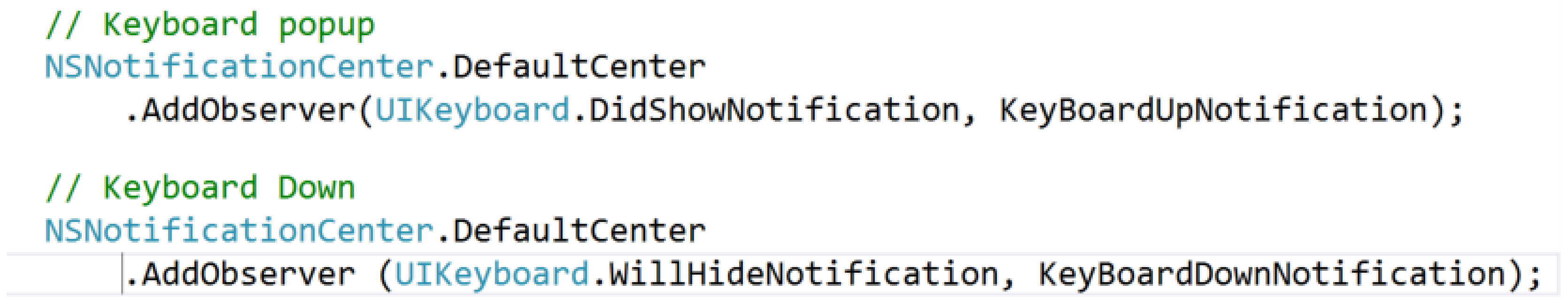
- Declare class level field for each UI control
- Create a rectangle that defines the frame of the control
- Initialize the UI control
- Configure the properties of the control
- Register any event handlers
- Add control to the view hierarchy



```
public override void ViewDidLoad()
{
    base.ViewDidLoad();

    float x = 10f;
    float y = 40f;
    float width = 200f;
    float height = 60;

    CGRect rect = new CGRect(x,y,width,height);
    UITextField nameField = new UITextField(rect);
    nameField.Placeholder = "Enter a description";
    View.Add(nameField);

}
```

Broadcast Notifications

- NSNotificationCenter
  - Hub that is used to listen to broadcast messages and post broadcast messages
  - Post is synchronous, blocking execution until all notification handlers have completed running
  - NSNotificationCenter.DefaultCenter is where system notifications are posted for system-level events
  - To register for notifications, use the AddObserver method



```
// Keyboard popup
NSNotificationCenter.DefaultCenter
    .AddObserver(UIKeyboard.DidShowNotification, KeyBoardUpNotification);

// Keyboard Down
NSNotificationCenter.DefaultCenter
    .AddObserver (UIKeyboard.WillHideNotification, KeyBoardDownNotification);
```

Workshop: iOS Views (45 mins)

Workshop (45 mins)

- Clone: https://github.com/llalonde/XamarinWorkshops.git
- Follow the steps in iOSViewsWorkshop.pdf

Additional Resources

- iOS Developer Resources: https://developer.apple.com/
  - **UIView:** ~/reference/uikit/uiview
  - **UIViewControllers:** ~/reference/uikit/uiviewcontroller

- Xamarin Developer Documentation:

  https://developer.xamarin.com/guides/ios/
  - NSNotificationCenter:
    https://developer.xamarin.com/api/type/MonoTouch.Foundation.NSNotificationCenter/