IOS 350

What's New in iOS8

- Lecture will begin shortly
- Download class materials from university.xamarin.com

Xamarin University

Xamarin University

Objectives

1. Explore the new features of iOS8
2. Identify UIKit API changes
3. Review and experiment with unified storyboards

Explore the new features of iOS8

Xamarin University

Tasks

- New Frameworks
- API changes
- App Extensions
- OS X Integration

iOS8 is a massive update

Xamarin University

- iOS8 is the largest API change since the launch of the App Store adding 15 new frameworks and over 4000 new APIs
- Two main goals with this release: extensibility and adaptive design

What New: APIs

- Several new frameworks have been added to provide new capabilities

Local Authentication

Home Automation

Health and Medical

Photo Editing

Cloud Documents

Camera APIs

Emphasis on Gaming

- Several new frameworks introduced specifically to support 2D/3D games

Sprite Kit

Scene Kit

Metal

Xamarin University

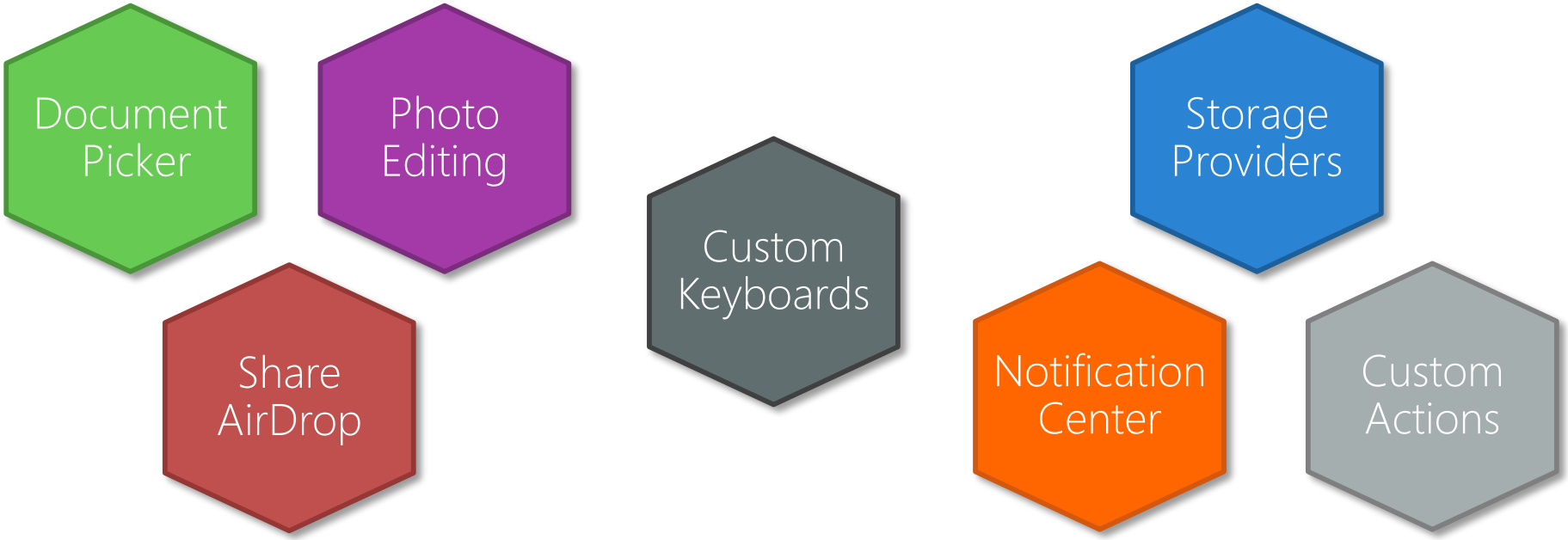
New APIs in existing frameworks

- Several excellent additions to CoreImage Framework for identifying and filtering
- New audio / video features for gaming and audio playback
- … and lots more

Xamarin University

What's New: Extensions

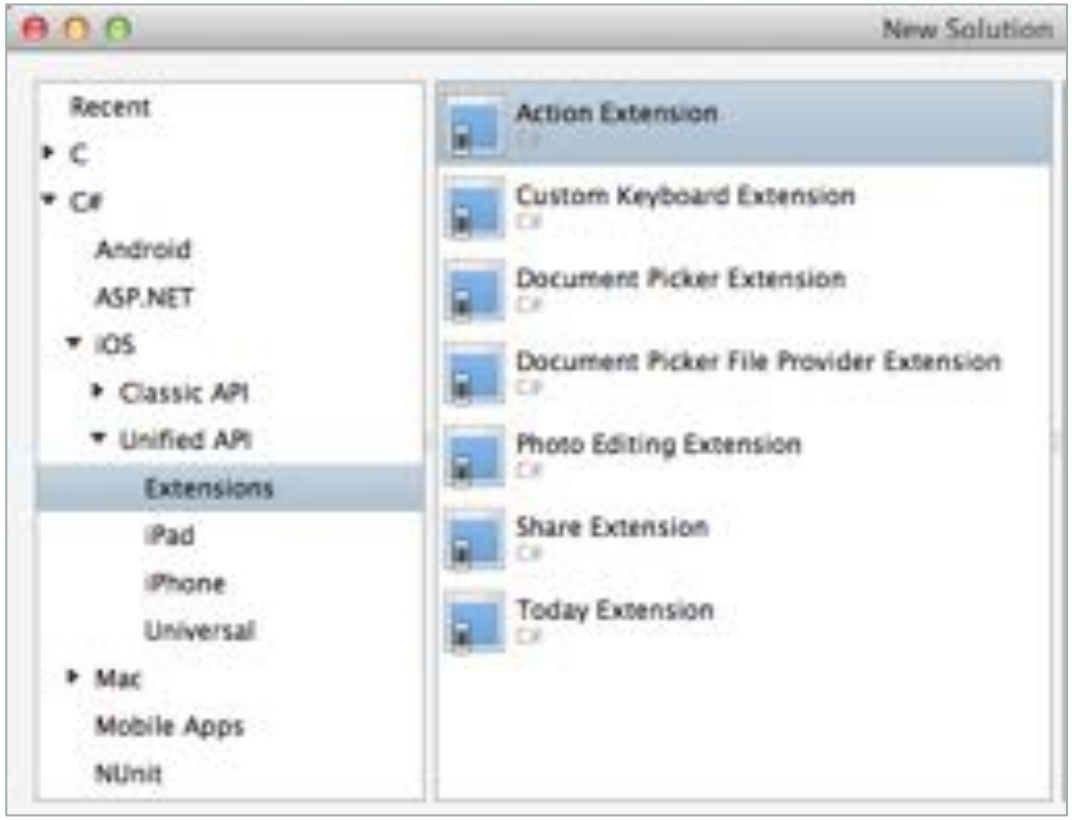
- iOS8 provides an unprecedented level of integration into previously closed features



Xamarin University

Creating Extensions in Xamarin

- New project templates included for iOS8 in Xamarin Studio and Visual Studio to build extensions
- Only available for 64-bit Unified API templates



Document Picker

- iOS8 enables apps to reach outside the app sandbox in a standardized fashion with a *Document Picker*

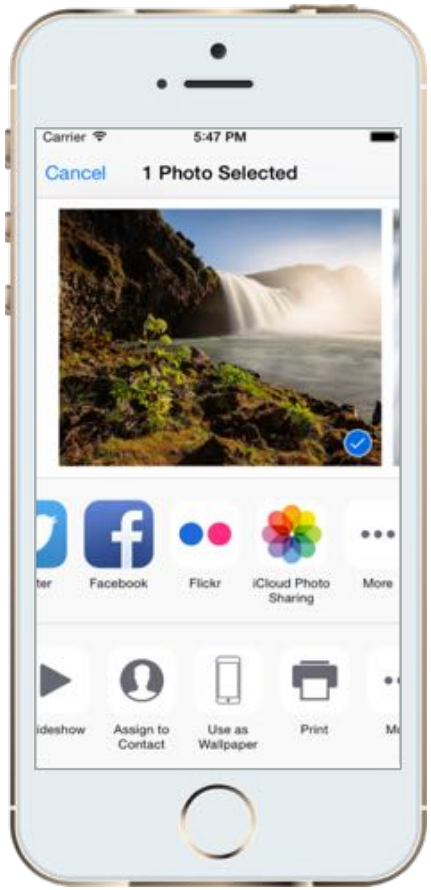
- Picker can be extended with custom storage provider sources – for example Apple provides an iCloud data provider

Sharing Extensions

- Can now integrate with the share sheet to connect independent applications and devices together at a high level

App-based sharing allows content to be passed from one app to another

Actions allow an app to register an activity to perform on the content

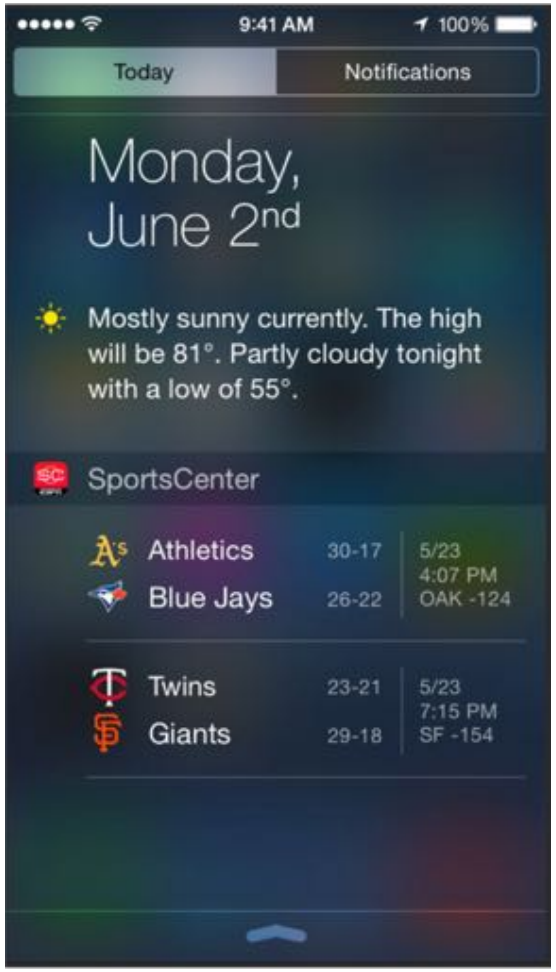


Custom Keyboards

- iOS8 allows 3rd party keyboard to be installed as the default system keyboard
- Some restrictions currently
  - Cannot be used for secure input
  - Not usable in phone app
  - Cannot "push" content into the app (copy / paste)
  - Can be restricted by app
  - No network support by default

Notifications

- Custom Widgets are now supported in Notification Center
- Provides "instant" access to interactive information so you can immediately see and respond to things you are interested in



Continuity Features

- Another driving factor in iOS8 is tighter integration with OS X Yosemite

Handoff is a set of system feature that allows iOS8 and OS X to interact in new ways – for example when a call is received on the phone, it also shows up on your nearby computer and can be answered on either device

Demonstration

Quick look at new frameworks (PhotoKit and Touch ID)

Xamarin University

Xamarin University

Summary

- New and Updated Frameworks
- App Extensions
- OS X Integration

Identify UIKit API changes

Xamarin University

Xamarin University

Tasks

- Working with the Alert Controller
- Navigation Controller condensing
- Creating Popover elements
- Managing a Search Bar
- Utilizing Notification Actions
- Using Custom Effects

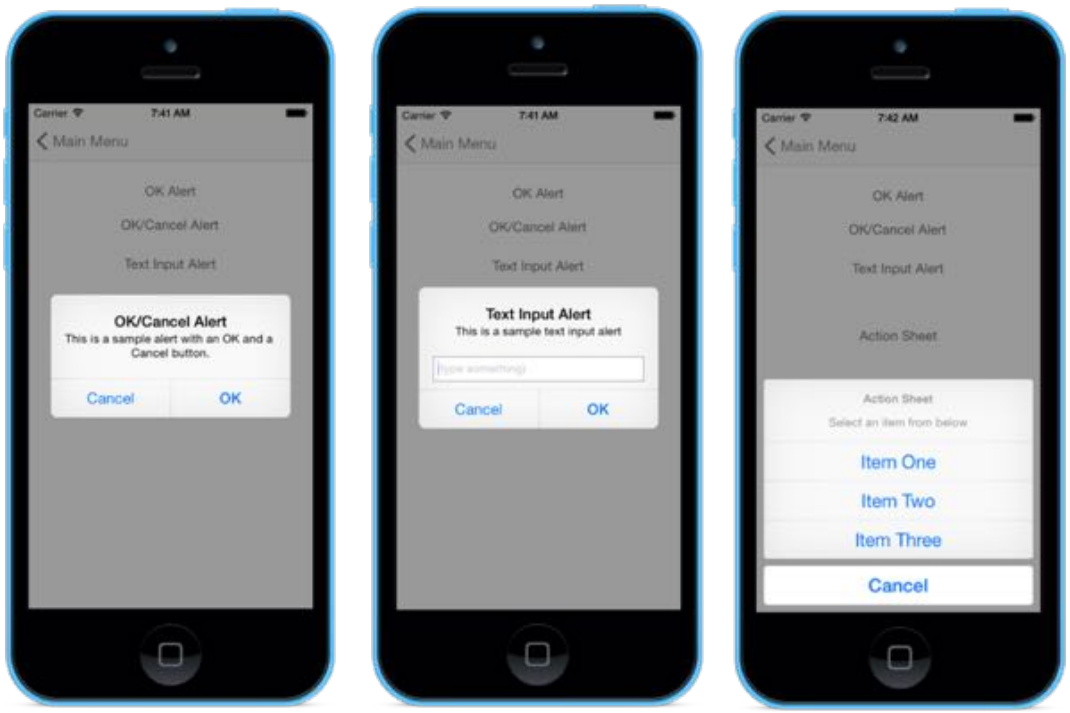
Working with the Alert Controller

- Alerts can be shown in two different styles depending on the prompt and response expected from the user

`UIAlertView`

`UIActionSheet`

Introducing UIAlertController

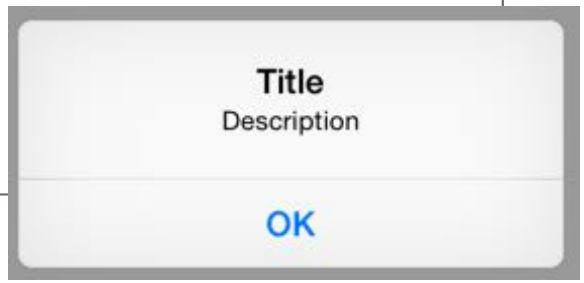


- New `UIAlertController` can be used to display either variation from a single API – this allows for more flexibility in the creation of alerts and prompts

Using UIAlertController

- Style parameter decides how alert is displayed and the contents

```
UIAlertController alert = UIAlertController.Create(
     "Title", "Description", UIAlertControllerStyle.Alert);

// Configure the alert
alert.AddAction(UIAlertAction.Create("OK",
     UIAlertActionStyle.Default, action => { }));

// Display the alert
this.PresentViewController(alert, true, null);
```



Navigation bar size

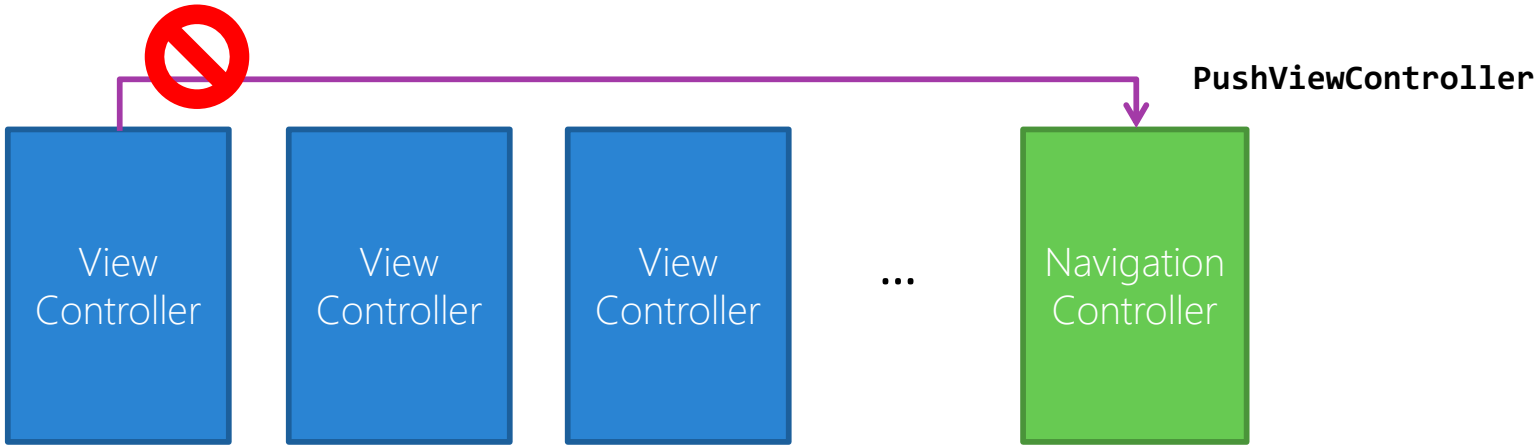
- Can control the size of the navigation bar header through new properties added to the **`UINavigationController`** class

```
P BarHideOnSwipeGestureRecognizer
P BarHideOnTapGestureRecognizer
M ChildViewControllerForStatusBarHidden
P HidesBarsOnSwipe
P HidesBarsOnTap
P HidesBarsWhenKeyboardAppears
P HidesBarsWhenVerticallyCompact
P HidesBottomBarWhenPushed
P NavigationBarHidden
```

Can show / hide the navigation header when the user taps on our app to make more real estate available

Screen transitions

- Navigation often requires intimate knowledge of the parent containers



must have access to our navigation controller to push a new view onto the navigation stack - what if the layout is different when run on an iPad?

PushViewController in iOS8

- Two new methods abstract on **`UIViewController`** provide an abstraction for navigation

Sets the master view, or current navigation view, or modal view

```
public void ShowViewController(
     UIViewController controller, NSObject sender)
```

Replaces the detail view (right side of a split view)

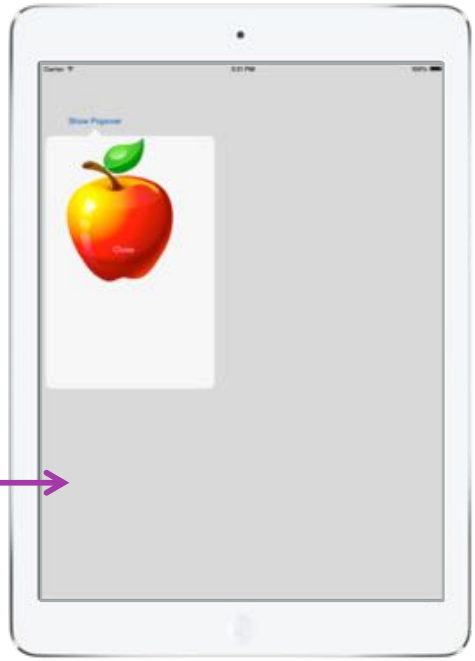
```
public void ShowDetailViewController(
     UIViewController controller, NSObject sender)
```

Creating popover elements

- New **`UIPopoverPresentationController`** can be used to display "popover" UI elements that work on both tablets and phones

When used on an iPhone, it creates a modal popover display

On an iPad the placement can be configured



Working with the Popover Controller

- Popover controller is implied when the presentation style is set to popover

```
var popover = new SomeViewController {
   ModalPresentationStyle = UIModalPresentationStyle.Popover
};

...

this.PresentViewController(popover, true, null);
```

`UIPopoverPresentationController` is created automatically by iOS

Customizing the popover controller

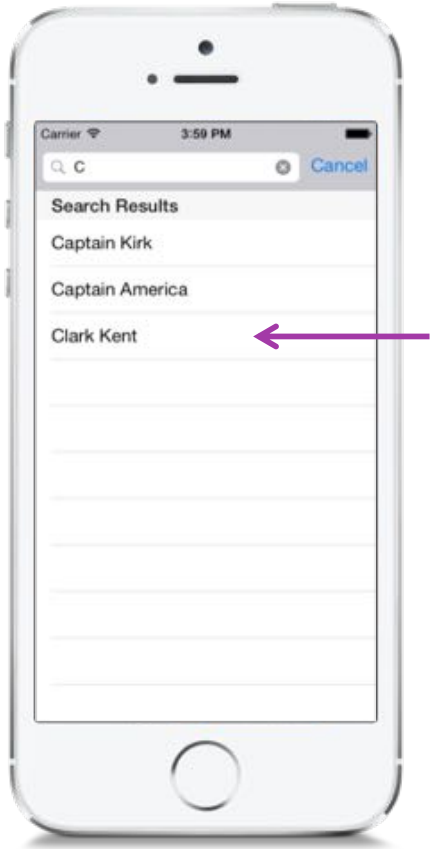
- Can fine-tune the popover controller with new property on view controller

```
var presentationPopover = popover.PopoverPresentationController;

if (presentationPopover != null) {
   presentationPopover.SourceView = this.View;
   presentationPopover.PermittedArrowDirections =
                    UIPopoverArrowDirection.Up;
   presentationPopover.SourceRect = TheButton.Frame;
}

this.PresentViewController(popover, true, null);
```

Property is only valid on an **iPad** when presentation style is **popover**, otherwise it is null

Managing Searches

- New **`UISearchController`** replaces existing search display controller and provides a standard search bar interface as well as live search results



Results can be live and displayed in whatever view we choose – here we are using a table view

Working with the Search Controller

- Search controller coordinates three elements, all configured through properties on the `UISearchController`

1. `SearchBar`

   `UISearchBar` is automatically created but is exposed through a read-only property

2. `searchResultsController`

   Can use any view controller to display results – this is passed to the constructor of the search controller

3. `SearchResultsUpdater`

   The results updater receives notifications about searches and is responsible for updating the UI

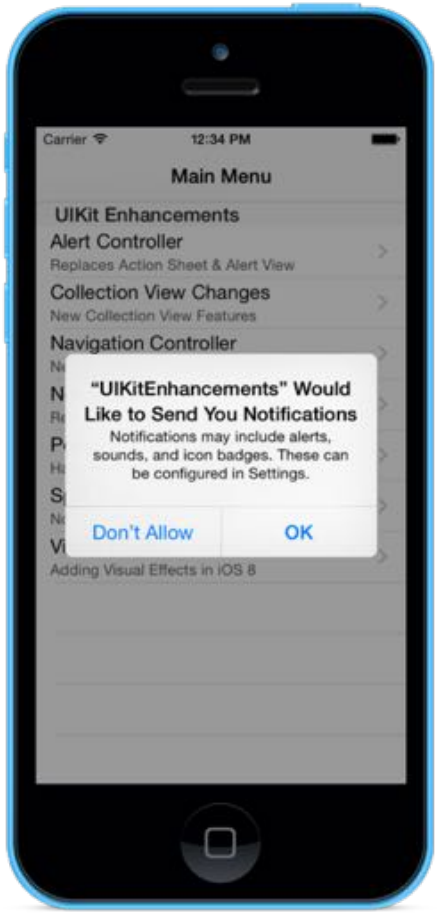
Updating the search results

- Search results are populated by a custom implementation of **`UISearchResultsUpdating`** – this is assigned to the **`SearchResultsUpdater`** property of the **`UISearchController`**

```
public class SearchResultsUpdator : UISearchResultsUpdating
{
    public override void UpdateSearchResultsForSearchController(
                                UISearchController searchController)
    {
        // TODO: update the search results – often done with
        // an event or delegate
    }
}
```

Notification Settings

- New settings object allows app to register the types of notifications they want to use
- iOS matches desire against user preferences and tells application what it can use



Registering notifications

```
public partial class AppDelegate : UIApplicationDelegate
{
   public override bool FinishedLaunching (...)
   {
      ...
      // Want to support both alerts and badges
      UIUserNotificationType type = UIUserNotificationType.Alert |
                                    UIUserNotificationType.Badge;

      // Create the setting for the given types
      UIUserNotificationSettings settings =
          UIUserNotificationSettings.GetSettingsForTypes(type, null);

      // Register the settings
      UIApplication.SharedApplication.RegisterUserNotificationSettings(settings);
   }
}
```

Registering notifications

- iOS calls new method **`DidRegisterUserNotificationSettings`** to let the app know the available notification styles

```
public partial class AppDelegate : UIApplicationDelegate
{
   ...
   public override void DidRegisterUserNotificationSettings(
           UIApplication application,
           UIUserNotificationSettings notificationSettings)
   {
      UIUserNotificationType types = notificationSettings.Types;
      if (types.HasFlag(UIUserNotificationType.Alert)) {
                ...
      }
   }
}
```

Passed types identifies available styles

Notification Actions

- Can register custom responses for notifications which can be selected by the user on the lock screen or when the notification banner is displayed *without switching to your app*

user swipes to reveal possible responses

Creating Notification Actions

- Actions are supplied as *categories* for UI notification settings

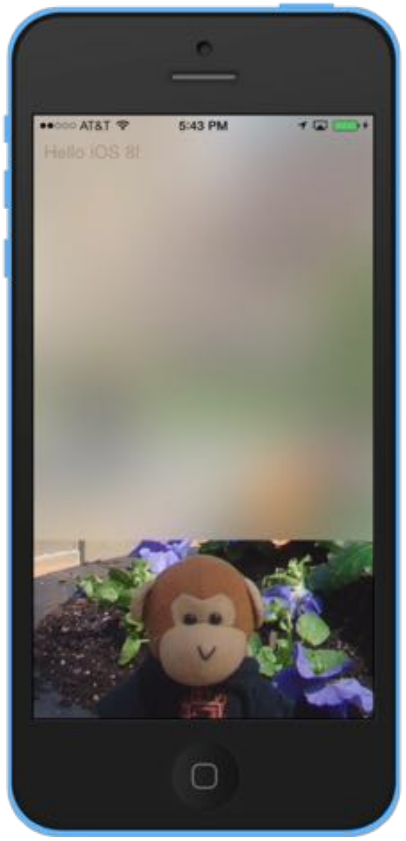
```
var accept = new UIMutableUserNotificationAction {
  Identifier = "ACCEPT_ID", Title = "Accept", AuthenticationRequired = false,
  Destructive = false, ActivationMode = UIUserNotificationActivationMode.Background
};
...
var category = new UIMutableUserNotificationCategory { Identifier = "MESSAGE_ID" };

category.SetActions(new[] {accept, reply}, UIUserNotificationActionContext.Default);
...
UIUserNotificationSettings settings =
          UIUserNotificationSettings.GetSettingsForTypes(type, category);
```

Creating Notification Actions

- Can supply different action lists based on where the notification is displayed: minimal for lock screen or banner, default for alerts

```
var accept = new UIMutableUserNotificationAction { ... };
...
category.SetActions (new[] { accept, trash },
                        UIUserNotificationActionContext.Minimal);
category.SetActions (new[] { accept, reply, trash },
                        UIUserNotificationActionContext.Default);
...
```

Visual Effects



- iOS7 introduced "depth" into the UI through h/w accelerated blurring effects but did not expose an API to actually provide this effect inside your apps
- iOS8 introduces `UIVisualEffects` to do exactly that!

Applying blur effects

- **`UIBlurEffect`** used to blur the background of a view, can apply an optional U**`IVibrancyEffect`** to make a portion of the view "pop"

```
var blur = UIBlurEffect.FromStyle (UIBlurEffectStyle.Light);

// Blur out an image on the screen
var blurView = new UIVisualEffectView (blur);
blurView.Frame = imageView.Bounds;
imageView.View.Add (blurView);
```

More good stuff..

- Collection Views and Table Views are now **self-sizing** using constraints and the applied content
- iPhone support for `UISplitViewController`
- Picker for printer selection – `UIPrinterPickerController`

- … and lots more

Individual Exercise

Working with some of the new UIKit changes in iOS8

Xamarin University

Xamarin University

Summary

- Working with the Alert Controller
- Navigation Controller condensing
- Creating Popover elements
- Managing a Search Bar
- Utilizing Notification Actions
- Using Custom Effects

Review and experiment with unified storyboards

Xamarin University

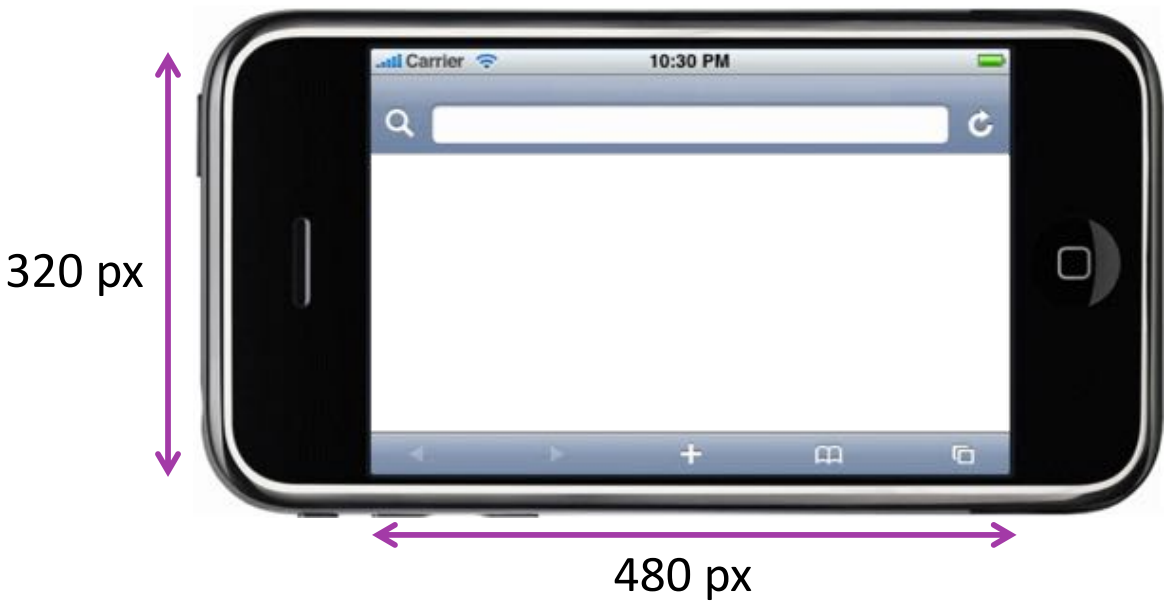
Xamarin University

Tasks

- Explain the purpose of Unified Storyboards
- Device and Size Classes
- Use the Xamarin Designer to create an application using Unified Storyboards

Device Fragmentation

- Flashback to 2007: the iPhone

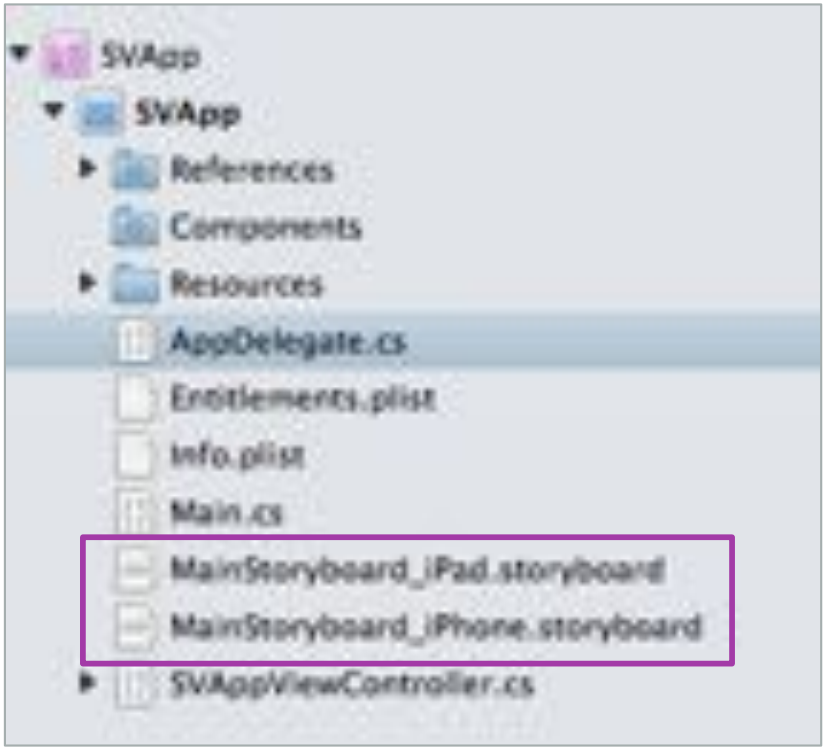


Device Fragmentation

- Now we have a variety of form factors and it's getting harder to build a single application that looks great on every variation

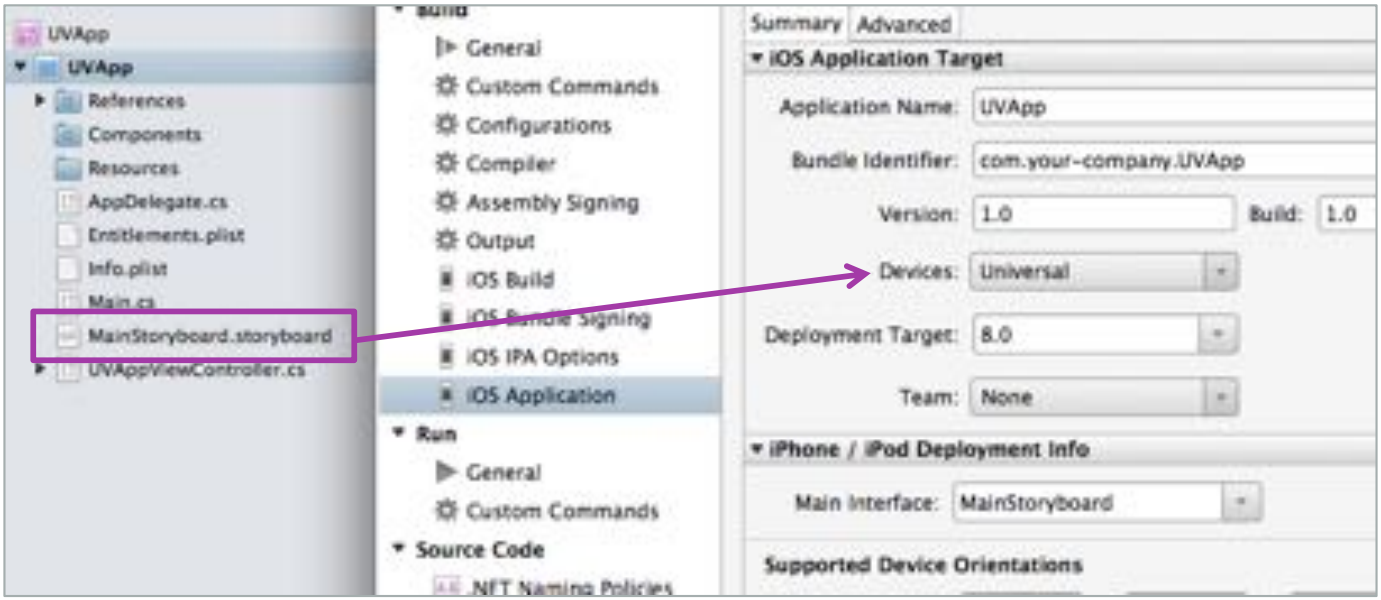
< iOS8 Solutions

- Prior to iOS8, "Universal" apps would have a unique storyboard for iPhone and iPad to allow for differences in the form factor
- Monitor orientation change notifications for dynamic layout
- Use Layout Constraints to manage the different sizes within a single device class (i.e. 3.5" vs. 4" display)



Introducing Unified Storyboards

- With iOS8, Apple is reversing course and instead using a single storyboard for all devices, they call this *unified storyboards*



Dealing with orientation changes

- When designing layout, we have always been interested in two things:
  1. What type of device are we dealing with (phone vs. tablet)?
  2. What orientation is the device in (portrait vs. landscape)?

- But the *real* question we need the answer to is:

How big is my drawing surface?

Xamarin University

Introducing "Size Classes"

"Size classes for iOS 8 enable designing a single universal storyboard with customized layouts for both iPhone and iPad. With size classes you can **define** common views and constraints **once**, and then **add variations for each supported form factor**. iOS Simulator and asset catalogs fully support size classes as well."

– What's New in Xcode 6, Apple documentation

What is it good for?

- Size Classes allow you to define your UI in a *single storyboard* where you have variations of the same UI defined for each supported size class

- If you want to have a *completely* different look for your iPhone 4 vs. iPhone 5 app, or you want to support iOS 7 or below, then size classes are not the solution

Size Class Definitions

- Content area is determined by how much space is available horizontally and vertically; each dimension can be one of two different values

Indicates that the specified dimension has more space available,
either because the device can "scroll" or because it has more pixels

Regular

Compact

Compact indicates that the dimension is constrained and has limited viewing capability

Mapping Size Classes to Devices

Horizontal Size Class

Vertical Size Class

	Regular	Compact
Regular	**iPad** ▪ Portrait ▪ Landscape	**iPhone** ▪ Portrait
Compact	**iPhone 6+** ▪ Landscape	**iPhone** ▪ Landscape

iPad Size Classes

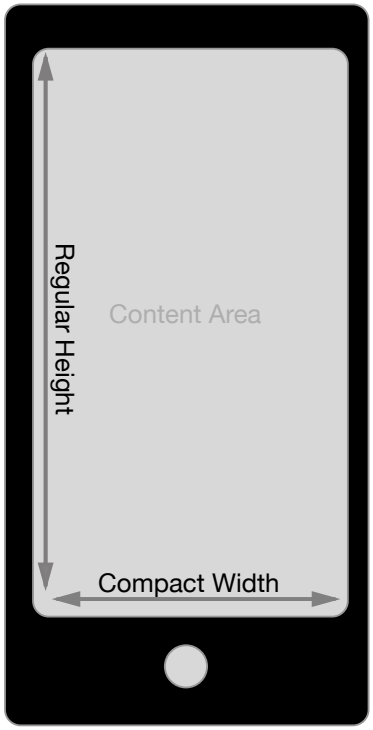
- iPad has large amount of screen space in <u>both</u> dimensions and <u>both</u> orientations and so always uses a *regular size class*

iPad Portrait View

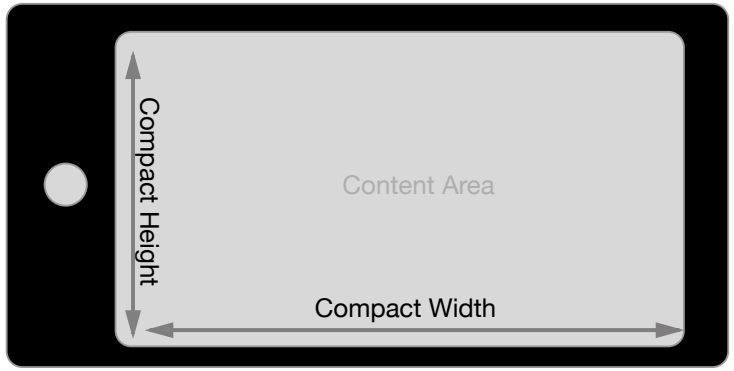
iPad Landscape View

iPhone Size Classes

- iPhone uses a compact size class horizontally in portrait, but a regular size class vertically
- iPhone is always compact in landscape orientation



iPhone Portrait View



iPhone Landscape View

Defining our UI for different sizes

- UI definition is based on *traits* – these define how content and layout change as the environment changes

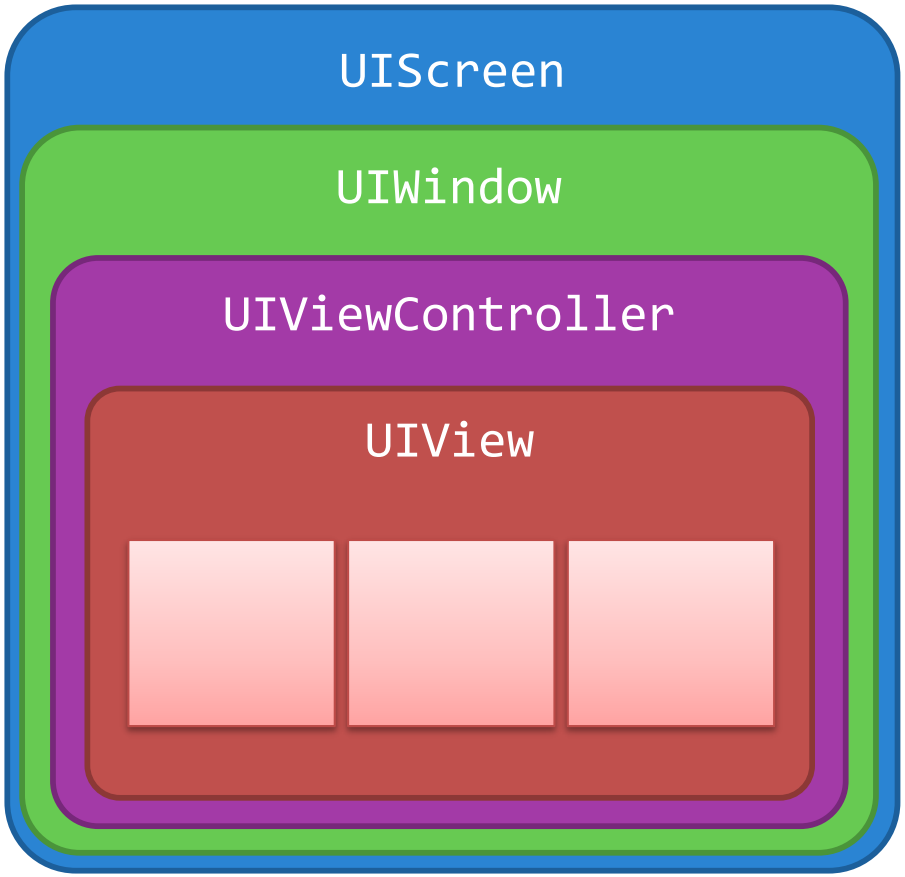
traits are contained in a **`UITraitCollection`**

Trait Collection
Horizontal Size Class
Vertical Size Class
UI Idiom (phone vs. tablet)
Display Scale (@2x, etc.)

What size is it?
Where is it positioned?
Is it present / visible?
Which image is shown?
…

Trait Collections

- Each UI component defines a trait collection, which together make up the *trait environment*
- Inherited from parent > child
- Can override **`TraitCollectionDidChange`** on view or view controller to handle changes in code, or use the designer



Designing UI with traits

- Do not design to explicit screen sizes, instead UI should be designed for each supported device class

Landscape Phone

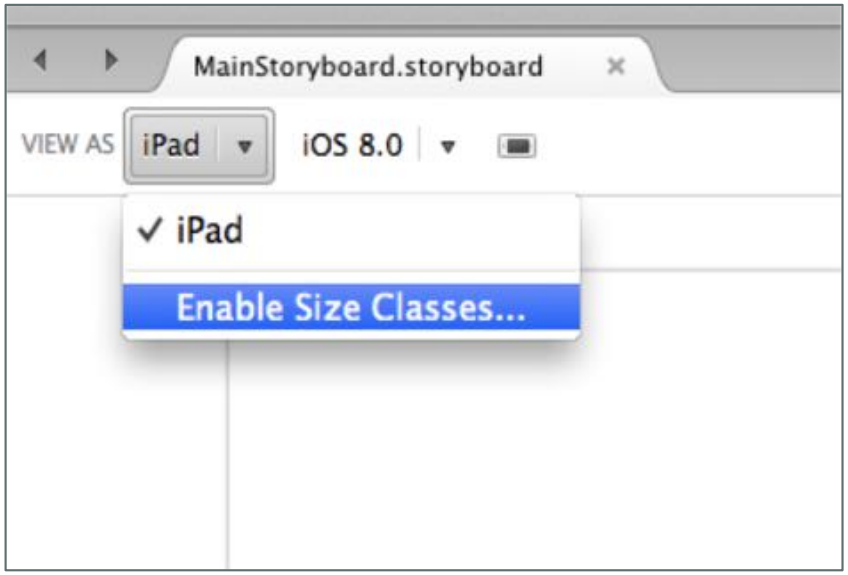
Portrait Phone

Landscape Tablet

Portrait Tablet

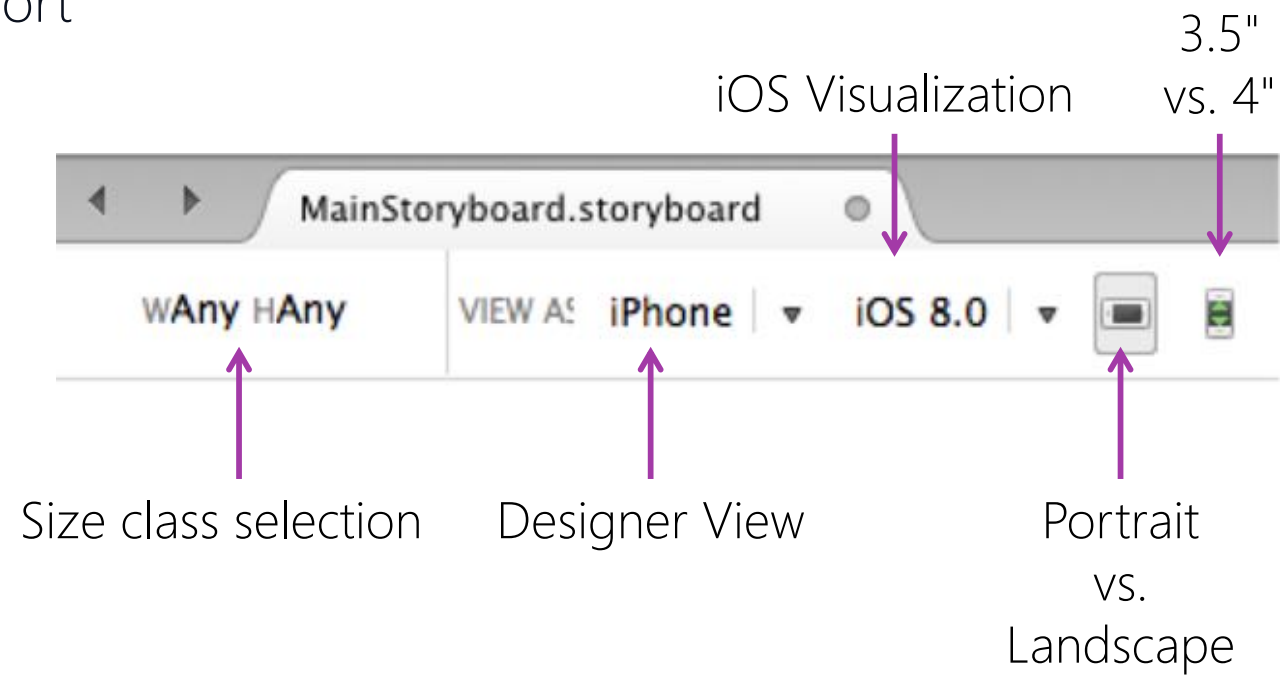
Xamarin Designer

- Designer has been updated to support size classes
  - Must target iOS8
  - Set Storyboard to Universal
  - Enable Size Classes in the Storyboard toolbar



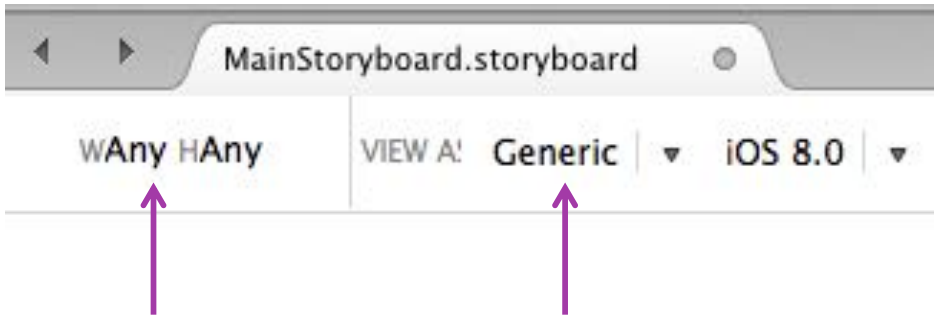
Exploring the Toolbar

- Toolbar has been reorganized for size classes and better constraint support



Create the standard UI

- Start with the Generic UI + default size class; this is *base UI definition*



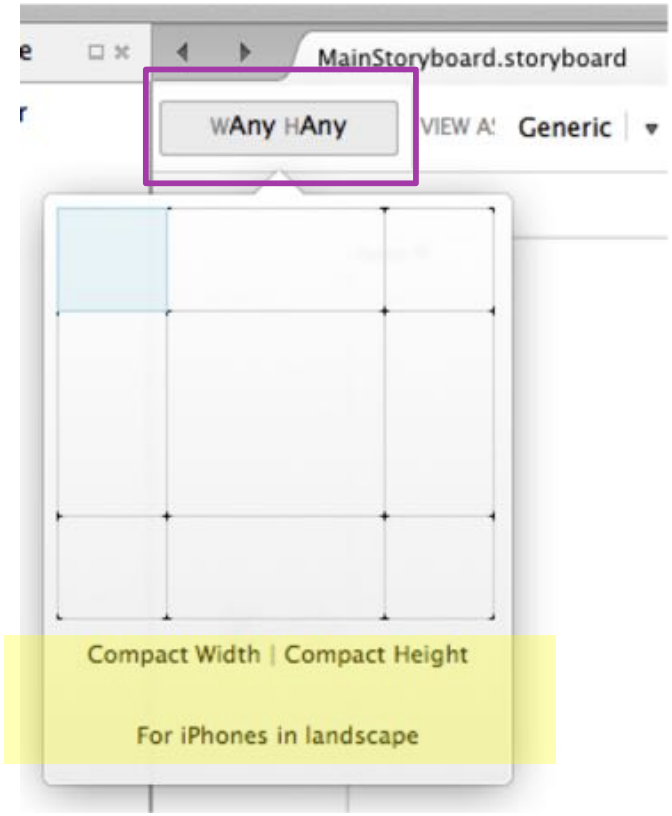
Any + Any means any vertical / horizontal size class is a match

Generic means it matches all device idioms

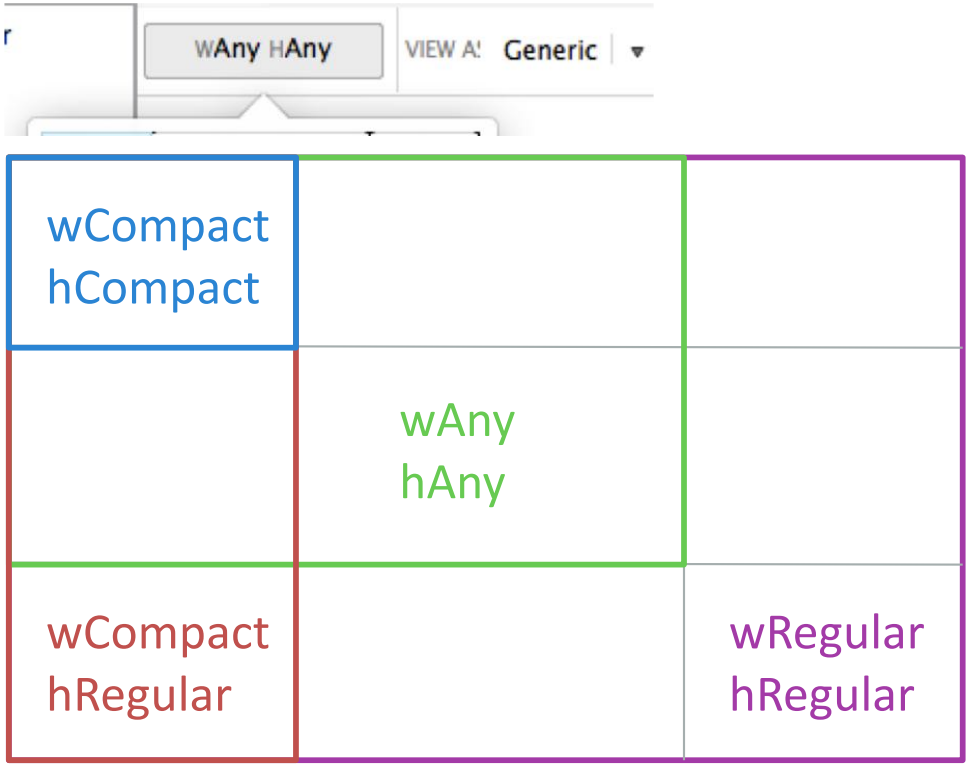
Adding Size Class Variations

- Can use the new size class tool to select an alternative device class – changes made to the UI design will then be applied when that trait environment is matched at runtime

- Can also add specific constraints into the UI which are only applied when that size class is matched



Designer Size Classes



- iPad – Landscape + Portrait
- Base Configuration
- iPhone Portrait
- iPhone Landscape

Selecting a UI at runtime

- iOS will identify the trait environment at runtime and then select the best appearance proxy to render the UI based on the trait collection
- This also determines which image(s) to load from your image assets, support is built into the **`UIImage`** class or you can use the new **`UIImageAsset`** class for more fine-grained control

HorizontalSizeClass	Compact
VerticalSizeClass	Regular
UserInterfaceIdiom	Phone
DisplayScale	2.0

This would define an iPhone (retina) in Portrait orientation

Group Exercise

Working with Unified Storyboards in Xamarin Studio

Xamarin University

Xamarin University

Summary

- Explain the purpose of Unified Storyboards
- Device and Size Classes
- Use the Xamarin Designer to create an application using Unified Storyboards

Thank You!

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