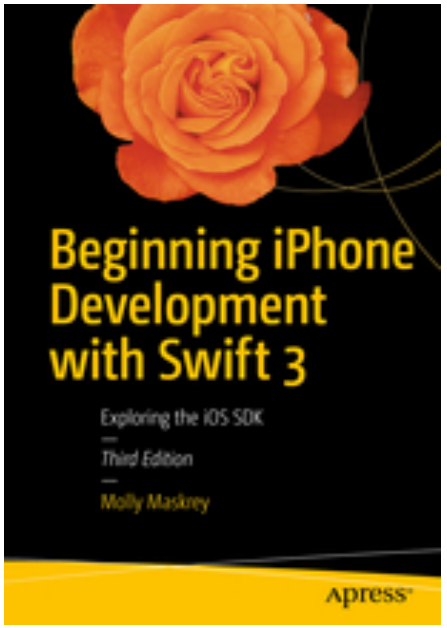
CENTENNIAL COLLEGE



MAPD714 - iOS Development

Week 4

Working with Device Rotations

Working with Device Rotations

- The **iPhone** and **iPad** exude amazing engineering in form, fit and function.

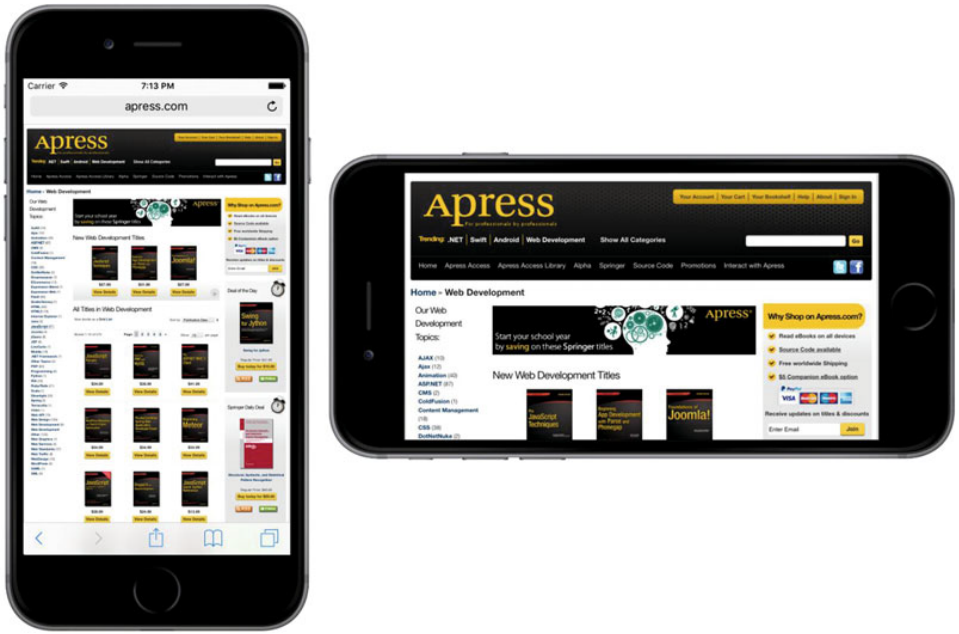
- Apple engineers found all kinds of ways to squeeze maximum functionality into a very small and elegant package.

- One example of this exists in the ability of these devices to be used in either **portrait** (tall and skinny) or **landscape** (short and wide) mode, and how that orientation can be changed at runtime simply by rotating the device.

Working with Device Rotations (continued)

- You see an example of this **autorotation** behavior in the iOS Safari browser, as shown in the following.

- In this lesson, we'll cover rotation in detail, starting with an overview of the ins and outs of **autorotation**, and then move on to different ways of implementing that functionality in your apps.



The Mechanics of Rotation

- The ability to run in both **portrait** and **landscape** orientations might not work for every application.
- Several of Apple's iPhone applications, such as the **Weather app**, support only a single orientation.
- However, iPad applications function differently with Apple recommending that most apps, with the exception of immersive apps like games, should support **every orientation** and most of Apple's own iPad apps work fine in both orientations.
- Many of them use the orientations to show different views of your data.

The Mechanics of Rotation (continued)

- For example, the **Mail** and **Notes** apps use **landscape orientation** to display a list of items (folders, messages, or notes) on the left and the selected item on the right.
- In **portrait orientation**, however, these apps let you focus on the details of just the selected item.
- For iPhone apps, the base rule is that, if autorotation **enhances** the user experience, you should add it to your application.
- For iPad apps, the rule is you should add autorotation unless you have a compelling reason not to.
- Fortunately, Apple did a great job of hiding the complexities of handling orientation changes in iOS and in **UIKit**, so implementing this behavior in your own iOS applications becomes quite easy.

Autorotation and the ViewController

- The view controller authorizes the image to rotate.
- If the user rotates the device, the active view controller gets asked if it's okay to change to the new orientation (which we'll do in this lesson).
- If the view controller responds in the affirmative, the application's window and views **rotate**, and the window and view **resize to fit** the new orientation.

Points, Pixels, and the Retina Display

- You might be wondering why we're talking about "points" instead of pixels .

- The reason for this change is Apple's introduction of the **Retina display**, which is Apple's marketing term for the high-resolution screen on all versions of the iPhone starting with iPhone 4 and later-generation iPod touches, as well as newer variants of the iPad.

- Retina Display **doubles** the hardware screen resolution for most models and almost **triples** it for the iPhone 6/6s Plus.

Handling Rotation

- To handle **device rotation**, we need to specify the correct **constraints** for all of the objects making up our interface.

- **Constraints** tell iOS how the controls should behave when their enclosing view is resized.

- How does that relate to device rotation? When the device rotates, the dimensions of the screen are (more or less) interchanged—so the area in which your views are laid out changes size.

Handling Rotation (continued)

- The simplest way of using constraints is to configure them in **Interface Builder** (IB).

- **Interface Builder** lets you define **constraints** that describe how your GUI components will be repositioned and resized as their parent view changes or as other views move around.

- We did a little bit of this last week and we will delve further into the subject of constraints this week.

- You can think of constraints as **equations** that make statements about view geometry and the iOS view system itself as a “solver” that will rearrange things as necessary to make those statements true.

- You can also add constraints in **code**.

Creating Our Orientations Project

- We'll create a simple app to show you how to pick the **orientations** that you want your app to work with.

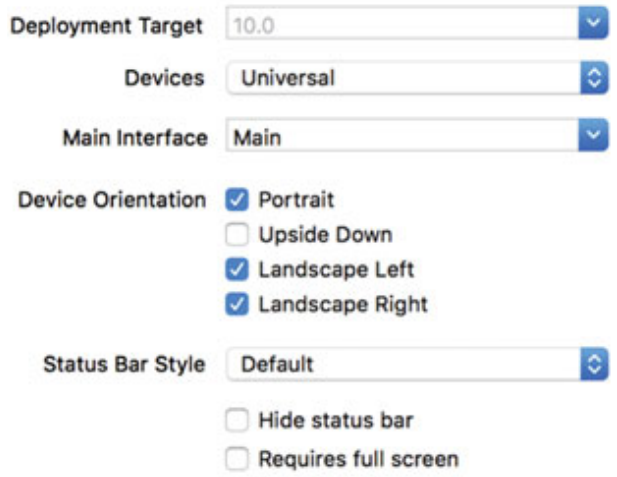
CENTENNIAL COLLEGE

- Start a new **Single View Application** project in Xcode, and call it **Orientations** .
- Choose **Universal** from the Devices pop-up, and save it along with your other projects.
- Before we lay out our GUI in the storyboard, we need to tell iOS that our view supports interface rotation.
- There are actually two ways of doing this. You can create an app-wide setting that will be the default for all view controllers and you can further tweak things for each individual view controller. We'll do both of these things, starting with the app-wide setting.

Supported Orientations at the App Level

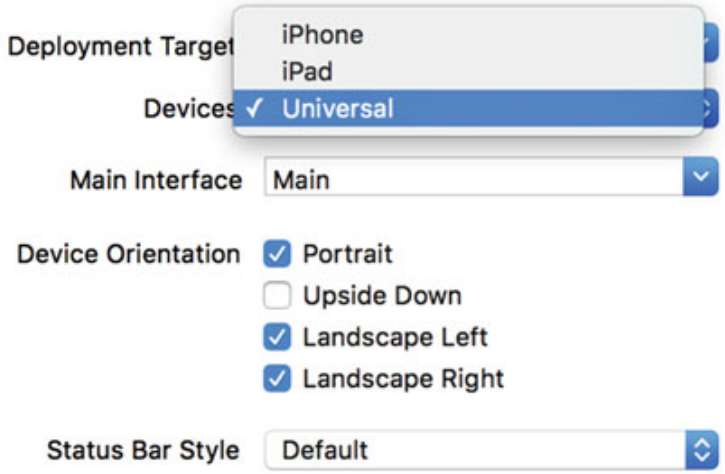
CENTENNIAL COLLEGE

- First, we need to specify which orientations our application supports.
- When your new Xcode project window appeared, it should have opened to your project settings.
- If not, click the top line in the **Project Navigator** (the one named after your project), and then make sure you're on the **General tab**.
- Among the options available in the summary, you should see a section called **Deployment Info**, and within that, a section called **Device Orientation** with a list of check boxes.



Supported Orientations at the App Level (continued)

- This is how we identify which **orientations** our app supports.



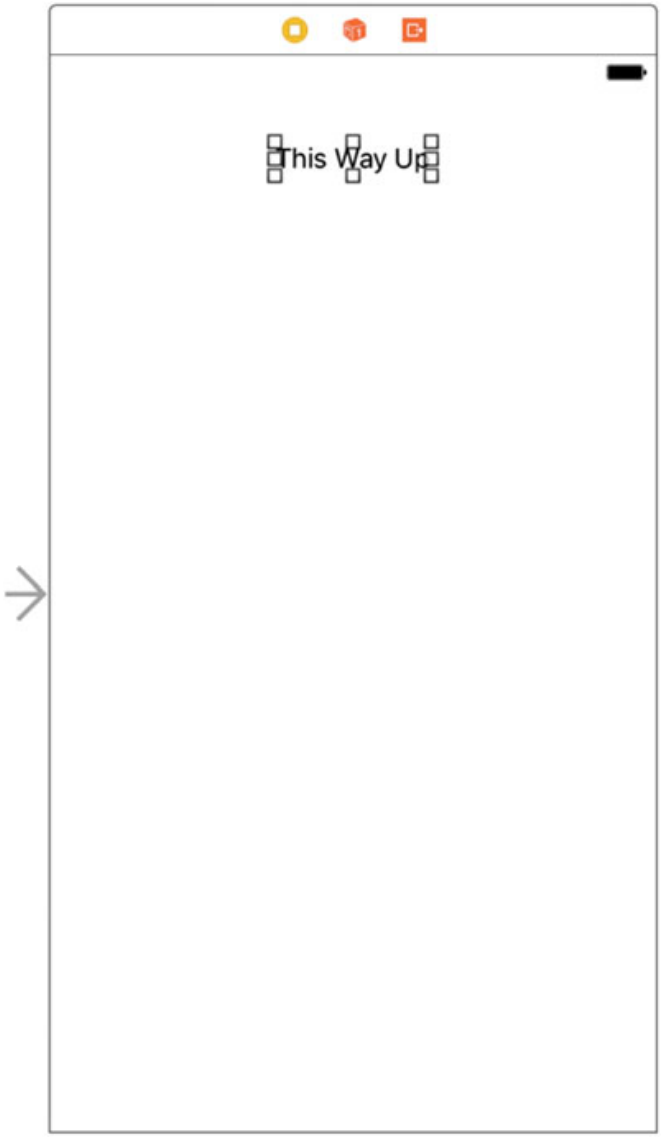
- Notice that the Upside Down orientation is **off** by default. That's because Apple does not encourage the user to hold the phone upside down because if the phone rings while it is in that orientation, the user would have to twist it through a full half turn to answer it.

- Open the **Devices drop-down** that's just above the check boxes and you'll see that you can actually configure separate sets of allowed orientations for the iPhone and the iPad.

- If you choose iPad, you'll see that all four check boxes are selected, because the iPad is meant to be used in any orientation.

Supported Orientations at the App Level (continued)

- Again, we'll work with an iPhone 6s as our device. Now, select **`Main.storyboard`**.
- Find a label in the **Object Library** and drag it into your view, dropping it so that it's horizontally centered and somewhere near the top.
- Select the label's **text** and change it to **This way up** .
- Changing the text may shift the label's position, so drag it to make it horizontally centered again.



Supported Orientations at the App Level (continued)

- We need to add **Auto Layout constraints** to **pin** the label in place before running the application, so **Control-drag** from the label upward until the background of the containing view turns blue, and then release the mouse.
- Hold down the **Shift key** and select **Vertical Spacing to Top Layout Guide** and **Center Horizontally in Container** in the pop-up, and then press **Return**.
- Now, press ⌘ R to build and run this simple app on the iPhone simulator.
- When it comes up in the simulator, try rotating the device a few times by pressing ⌘-**Left Arrow** or ⌘-**Right Arrow**.
- You'll see that the entire view (including the label you added) rotates to every orientation except upside down, just as we configured it to do. Run it on the **iPad simulator** to confirm that it rotates to all four possible orientations.
- We've identified the orientations our app will support, but that's not all we need to do.
- We can also specify a set of **accepted orientations** for each view controller, giving us more **fine-grained control** over which orientations will work in different parts of our apps.

Per-Controller Rotation Support

- Let's configure our **view controller** to allow a different, smaller set of accepted orientations.

- The **global configuration** for the app specifies a sort of **absolute upper limit** for allowed orientations.

- All we can do in the view controller is place **further limits** on what is acceptable.

Per-Controller Rotation Support (continued)

- In the **Project Navigator**, single-click **`ViewController.swift`**. Here we'll implement a method defined in the **`UIViewController`** superclass that lets us specify which subset of the global set of orientations we'll accept for this view controller :


```
override func supportedInterfaceOrientations() -> UIInterfaceOrientationMask {
    return UIInterfaceOrientationMask(rawValue:
        (UIInterfaceOrientationMask.portrait.rawValue
            | UIInterfaceOrientationMask.landscapeLeft.rawValue))
}
```

Per-Controller Rotation Support (continued)

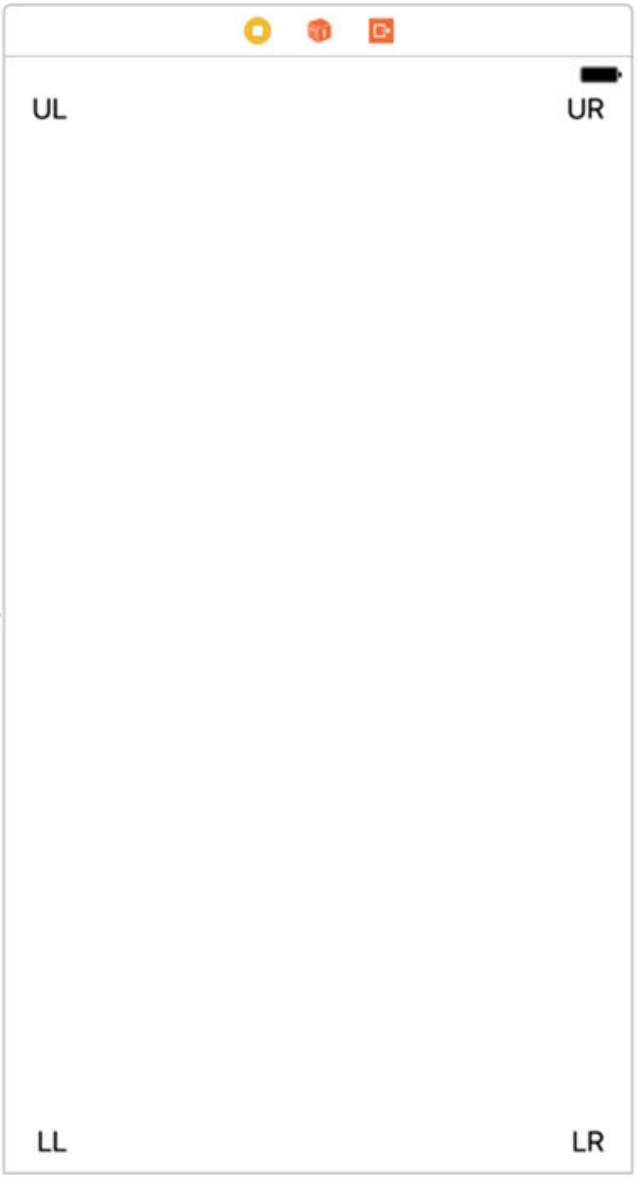
- **`UIKit`** defines the following orientation masks, which you can combine in any way you like using the **`OR`** operator (shown in the preceding example):
  - **`UIInterfaceOrientationMask.portrait.rawValue`**
  - **`UIInterfaceOrientationMask.landscapeLeft.rawValue`**
  - **`UIInterfaceOrientationMask.landscapeRight.rawValue`**
  - **`UIInterfaceOrientationMask.portraitUpsideDown.rawValue`**

- In addition, there are some predefined combinations of these for common use cases. These are functionally equivalent to OR -ing them together on your own, but can save you some typing and make your code more readable:
  - **`UIInterfaceOrientationMask.landscape.rawValue`**
  - **`UIInterfaceOrientationMask.all.rawValue`**
  - **`UIInterfaceOrientationMask.allButUpsideDown.rawValue`**

Creating Our Layout Project

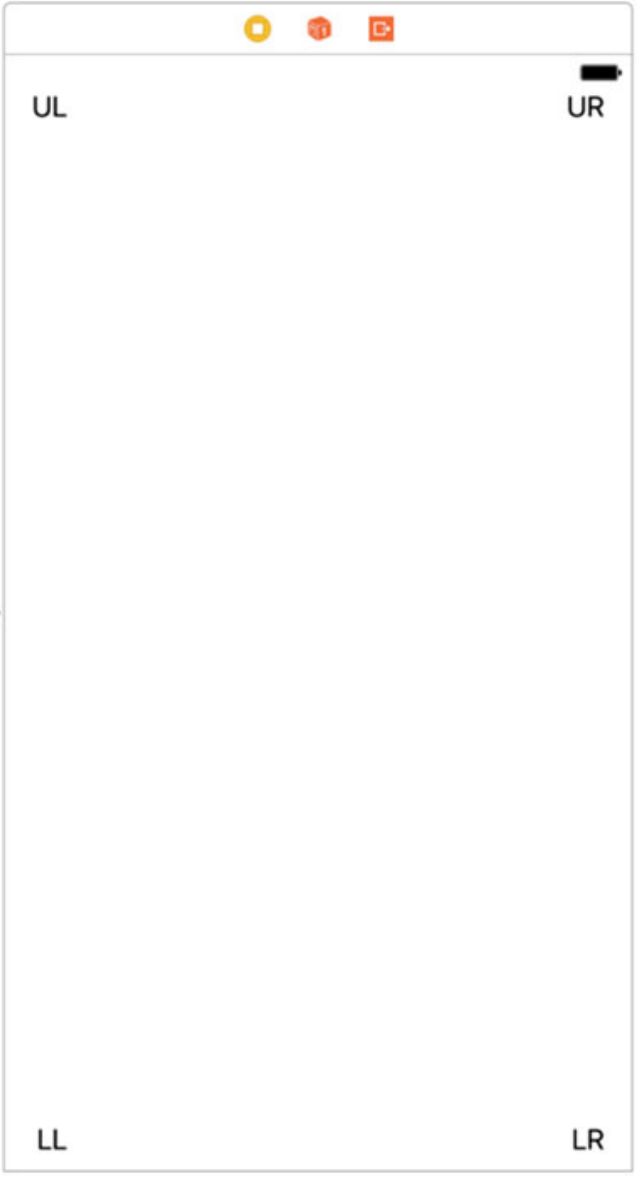
CENTENNIAL COLLEGE

- In Xcode, make another new project based on the Single View Application template and name it Layout.
- Select **`Main.storyboard`** to edit the storyboard in Interface Builder.
- A great thing about **constraints** is that they accomplish quite a lot using very little code.
- To see how this works, **drag four labels** from the library over to your view, and place them, as shown.
- Use the **dashed blue guidelines** to help you line up each one near its respective corner.



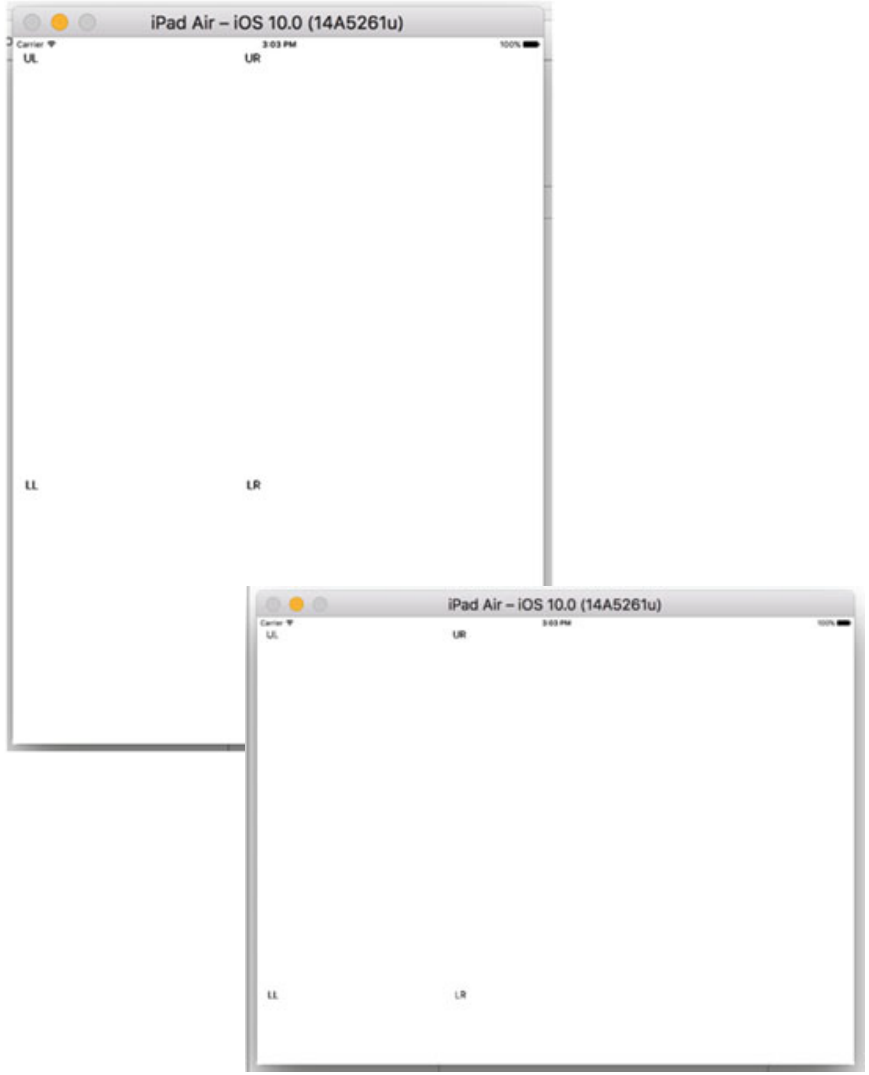
Creating Our Layout Project (continued)

- **Double-click** each label and assign a title to each one so that you can tell them apart later.
- We've used **UL** for the upper-left label, **UR** for the upper-right label, **LL** for the lower-left label, and **LR** for the lower-right label.
- After setting the text for each label, drag all of them into position so that they are lined up evenly with respect to the container view's corners.
- Let's see what happens now, given that we haven't yet set any **Auto Layout constraints** .



Creating Our Layout Project (continued)

- Build and run the app on the iPad Air simulator.
- Once the simulator starts up, you'll find that you can only see the labels on the left—the other two are off-screen to the right.
- Furthermore, the label at the bottom left is not where it should be—right in the bottom-left corner of the screen.
- Select Hardware ➤ Rotate Left, which will simulate turning the iPad to landscape mode.



Creating Our Layout Project (continued)

- As we've seen in earlier lessons, Interface Builder is smart enough to examine this set of objects and create a set of **default constraints** that will do exactly what we want.
- It uses some rules of thumb to figure out that if we have objects near edges, we probably want to keep them there.
- To make it apply these rules, first **select all four labels**.
- With all of them selected, choose **Editor ➤ Resolve Auto Layout Issues ➤ Add Missing Constraints** from the.
- Next, just press the Run button to launch the app in the simulator, and then verify that it works.

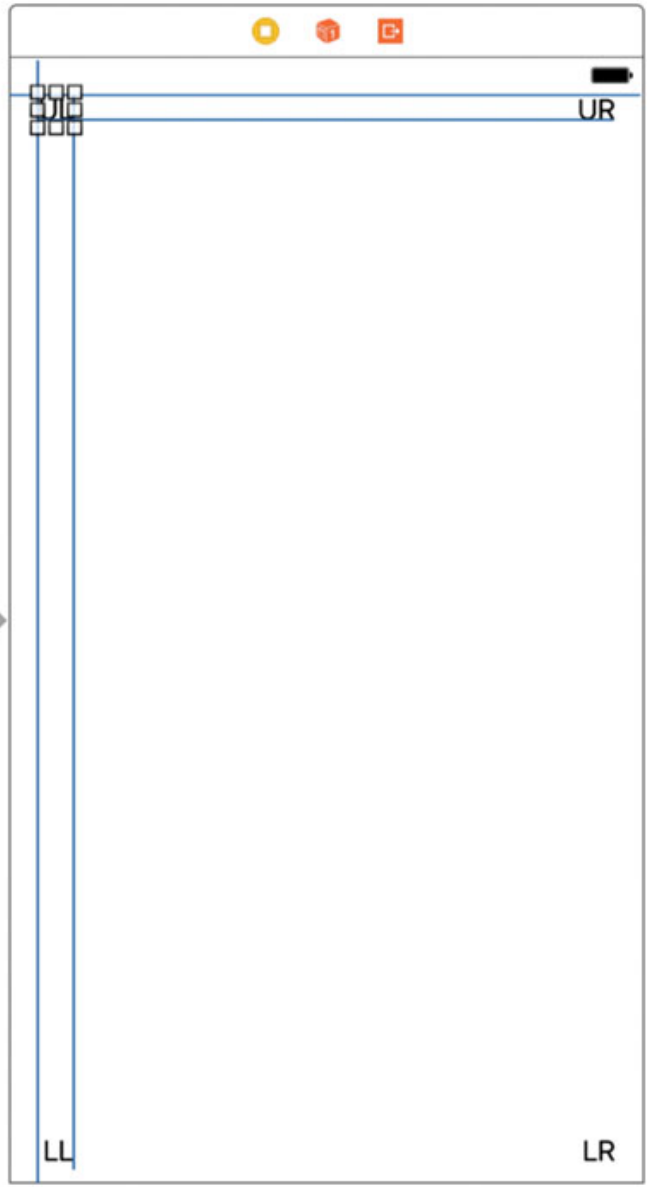
Creating Our Layout Project (continued)

CENTENNIAL COLLEGE

- Knowing that this works is one thing, but to use **constraints** like this most effectively, it's pretty important to understand how it works, too.

- So, let's dig into this a bit. Back in Xcode, click the upper-left label to select it. You'll notice that you can see some **solid blue lines** attached to the label.

- These **blue lines** are different from the **dashed blue guidelines** that you see when dragging objects around the screen, as shown in the following Figure

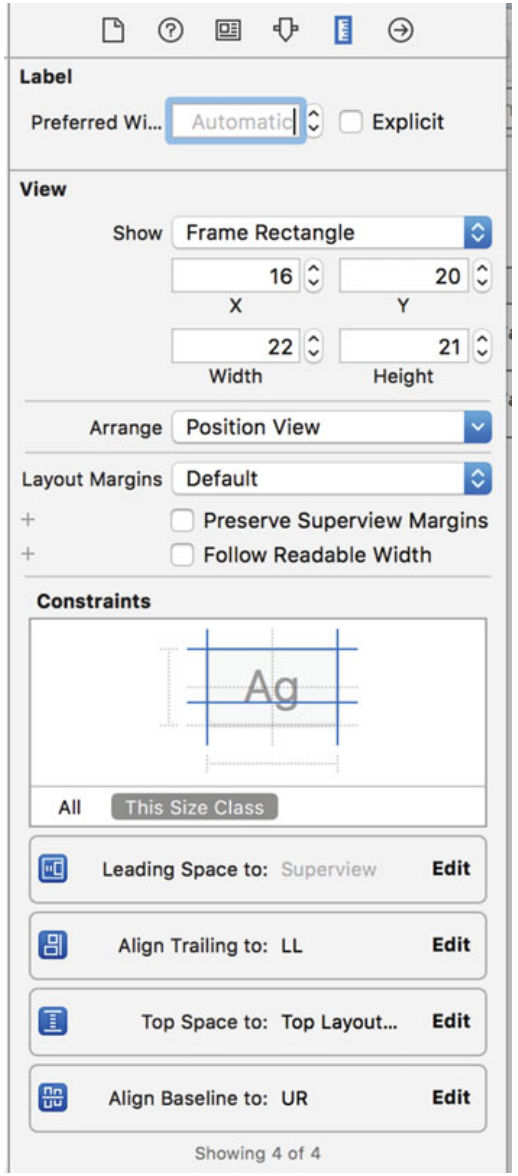


Creating Our Layout Project (continued)

CENTENNIAL COLLEGE

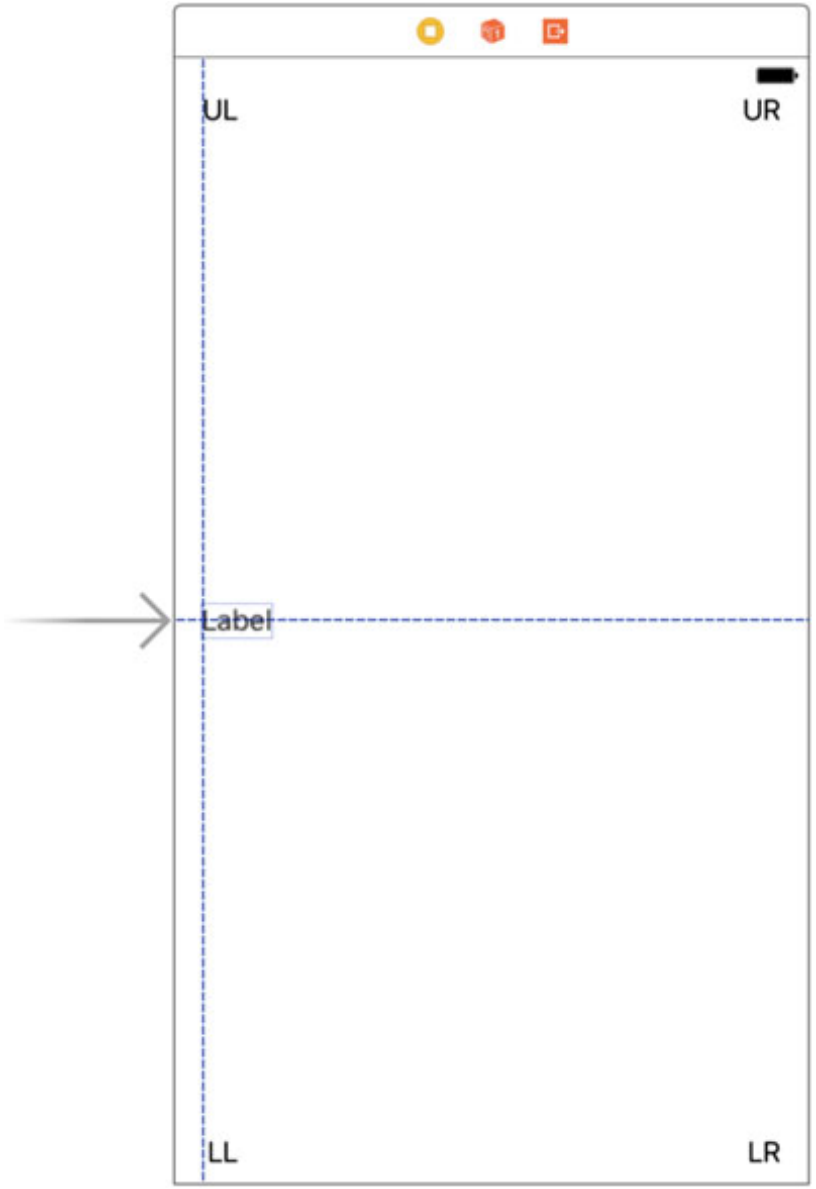
- Each of those solid blue lines represents a constraint. If you now press ⌥⌘**5** to open the **Size Inspector**, you'll see that it contains a list of constraints.

- The following Figure shows the **constraints** that Xcode applied to the UL label in my storyboard, but the constraints that Xcode creates depends on exactly where you placed the labels, so you may see something different.



Overriding Default Constraints

- Grab **another label from the library** and drag it over to the layout area.
- This time, instead of moving toward a corner, drag it toward the left edge of your view, lining up the label's left edge with the left edges of the other labels on the left side, and centering it vertically in the view.
- Dashed lines will appear to help you out. The following Figure shows you what this looks like.



CENTENNIAL COLLEGE

Overriding Default Constraints

- Let's **add a new constraint** to force this label to stay vertically centered.
- Select the **label**, click the **Align icon** below the storyboard, check **Vertically in Container** in the pop-up that appears, and then click **Add 1 Constraint**.
- Now make sure that the **Size Inspector** is on display (by pressing ⌥⌘ **5** if necessary).
- You'll see that this label now has a constraint aligning its center Y value to that of its superview.
- The label also needs a horizontal constraint. You can add this by making sure the label is **selected**, and then choosing **Editor ➤ Resolve Auto Layout Issues ➤ Add Missing Constraints** from the **All Views** section of the menu.
- Press R to run the app again. Do some rotating and you'll see that all the labels now move perfectly into their expected places for the various device types.

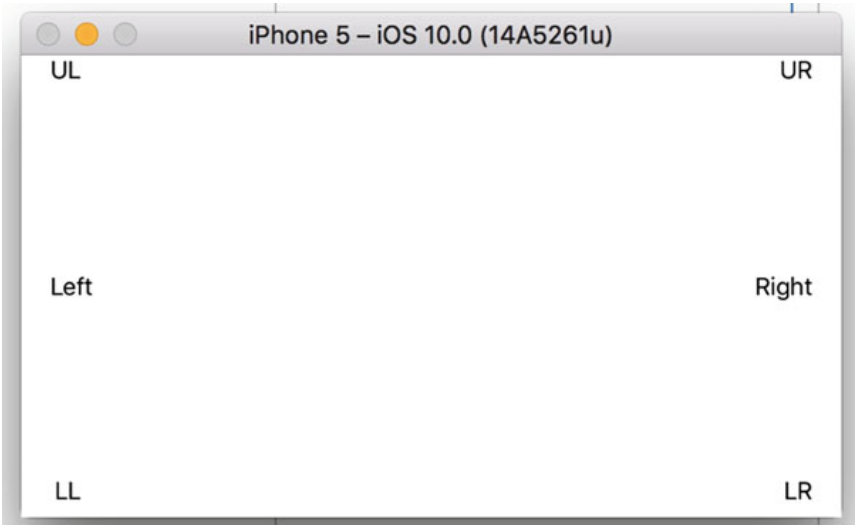
Overriding Default Constraints

CENTENNIAL COLLEGE

- Now, let's complete our ring of labels by dragging out a new one to the **right side of the view**, lining up its right edge with the other labels on the right, and aligning it vertically with the **Left** label.

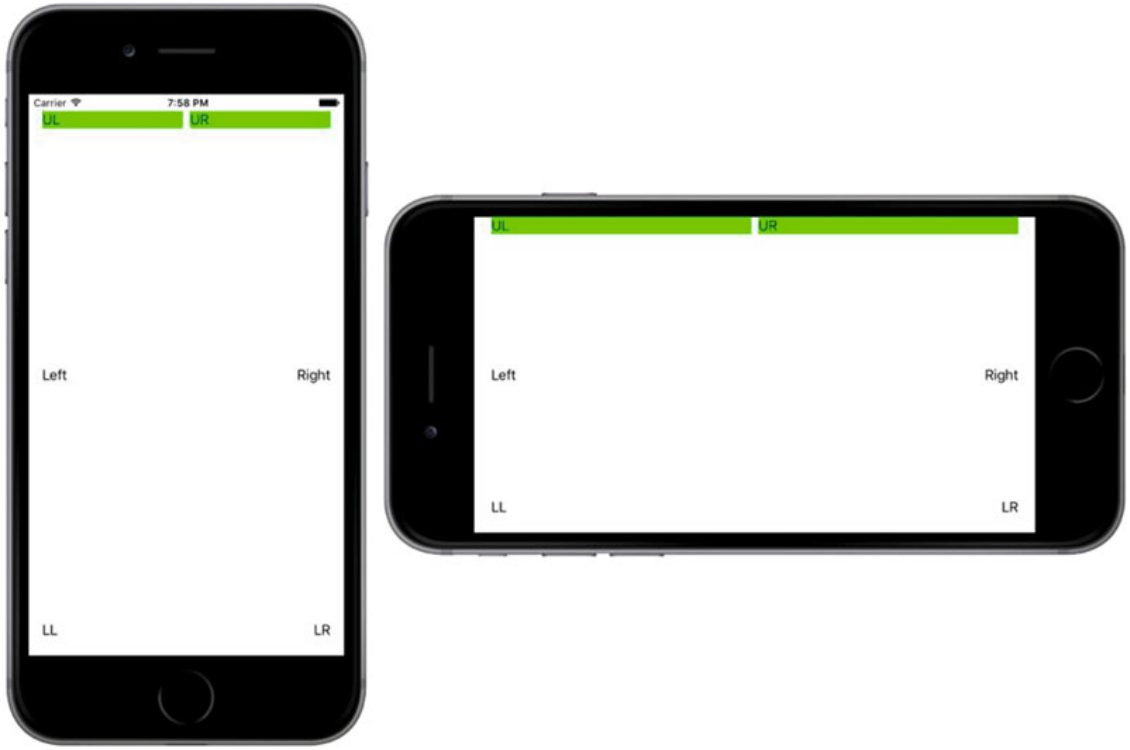
- Change this label's title to **Right**, and then drag it a bit to make sure that the right edge is vertically aligned with the right edges of the other two labels, using the dashed blue line as your guide.

- We want to use the **automatic constraints** that Xcode can provide us with, so select **Editor ➤ Resolve Auto Layout Issues ➤ Add Missing Constraints** to generate them.



Full-Width Labels

- We're going to create some **constraints** that make sure that our labels stay the same width as each other, with tight spacing to keep them stretched across the top of the view even when the device rotates. The following Figure should give you an idea of what we're trying to do.



Full-Width Labels (continued)

- We need to be able to visually verify that we've got the result we want—namely, each label is precisely centered within its half of the screen.
- In order to make it easier to see whether we've got it right, let's temporarily set a background color for the labels.
- In the storyboard, select both the **UL** and **UR** labels, open the **Attributes Inspector**, and scroll down to the **View section**.
- Use the **Background control** to select a nice, bright color. You'll see that the (currently very small) frame of each label fills with the color you chose.

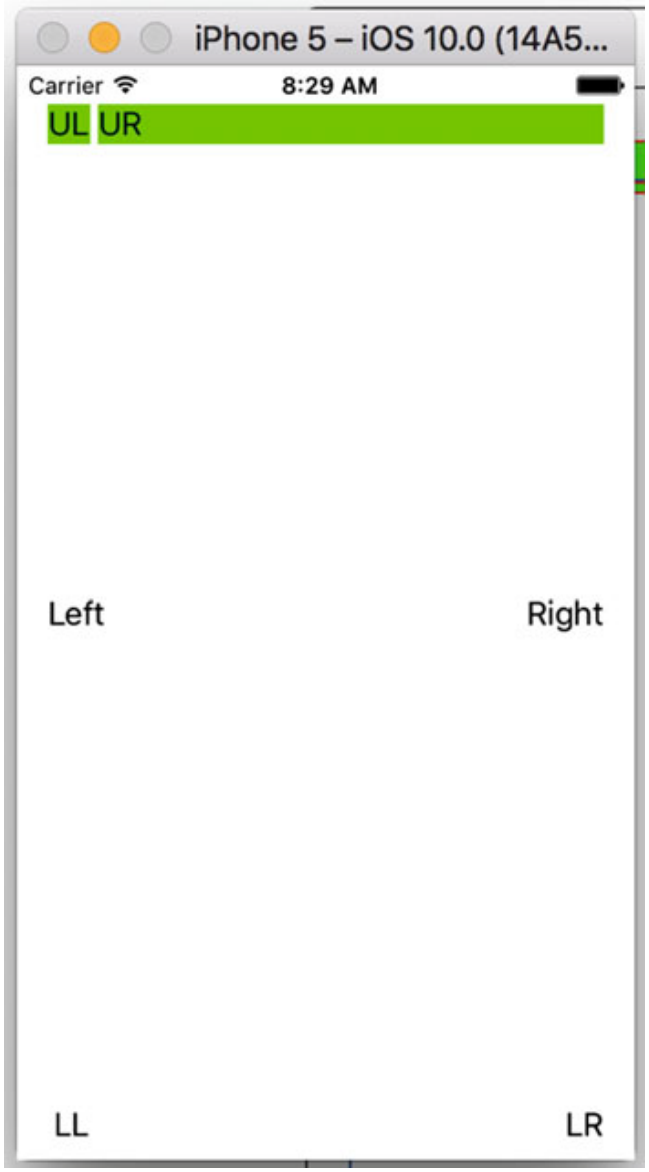
Full-Width Labels (continued)

- Drag the resizing control of the **UL label** from its right edge, pulling it almost to the horizontal **midpoint of the view**. You don't have to be exact here, for reasons that will become clear soon.
- After doing this, resize the **UR label** by dragging its left-edge resizing control to the left until you see the dashed blue guideline appear which tells you that it's the recommended width from the label to its left.
- Now we'll add a constraint to make these labels retain their relative positions.
- Control-drag from the **UL label** until the mouse is over the **UR label**, and then release the mouse. In the pop-up, select Horizontal Spacing and press Return.
- That constraint tells the layout system to hold these labels beside one another with the same horizontal space they have right now. Build and run to see what happens.

Full-Width Labels (continued)

CENTENNIAL COLLEGE

- You should see something like the following Figure; the longer label may appear on the left or right depending upon your configuration.
- That's heading in the right direction but not yet what we had in mind. So what's missing? We've defined constraints that control each label's position relative to its superview and the allowed distance between the two labels, but we haven't said anything about the sizes of the labels.
- This leaves the layout system free to size them in whatever way it wants. To remedy this, we need to add **one more constraint**.



Full-Width Labels (continued)

- Make sure the **UL** label is selected, and then hold down the Shift key (⇧) and click the UR label.

- With both labels selected, you can make a constraint that affects both of them. Click the **Pin icon** below the storyboard and check the **Equal Widths** check box in the pop-up that appears, and then click **Add 1 Constraint**.

Full-Width Labels (continued)

- You'll now see a new constraint appear, as shown in the following Figure.

- You may notice two **orange** lines have appeared below the labels; this means that the current positions and sizes of the labels in the storyboard do not match what you will see at runtime.

- To fix this, select the **View icon** in the **Document Outline**, and then select **Editor ➤ Resolve Auto Layout Issues ➤ Update Frames** in Xcode's menu.

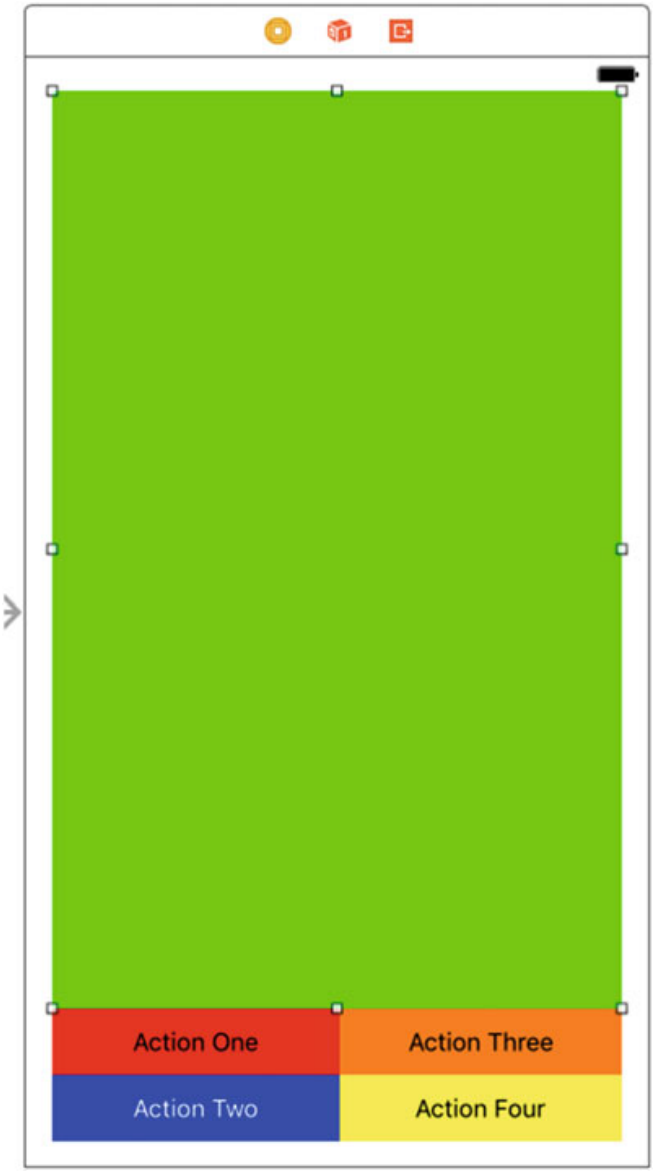
- The **constraints** should change to blue and the labels will resize themselves so that their widths are equal.

Creating Adaptive Layouts

- The **layout** for the simple example that we just created works well in portrait and landscape orientations.
- It also works on both **iPhone** and **iPad**, despite their differing screen dimensions.
- As we already noted, handling **device rotation** and creating a user interface that works on devices with different screen sizes are really the same problem—after all, from the point of view of your application, when the device rotates, the screen effectively changes size.
- In the simplest cases, you handle them both at the same time by assigning **Auto Layout constraints** to make sure that all of your views are positioned and sized where you want them to be.
- However, that's not always possible. Some layouts work well when the device is in **portrait mode**, but not so well when it's rotated to landscape; and similarly, some designs suit the iPhone but not the iPad.

Creating the Restructure Application

- To get started, we'll design a user interface that works well for an iPhone in **portrait mode**, but not so well when the phone is rotated or when the application runs on an iPad.
- Then we'll see how to use **Interface Builder** to adapt the design so that it works well everywhere.
- Start by making a new **Single View project** like you've done before, naming this one **Restructure** .
- We're going to construct a GUI that consists of **one large content area** and a small set of buttons that perform various (fictional) actions.
- We'll place the buttons at the bottom of the screen and let the content area take up the rest of the space, as shown in the following Figure

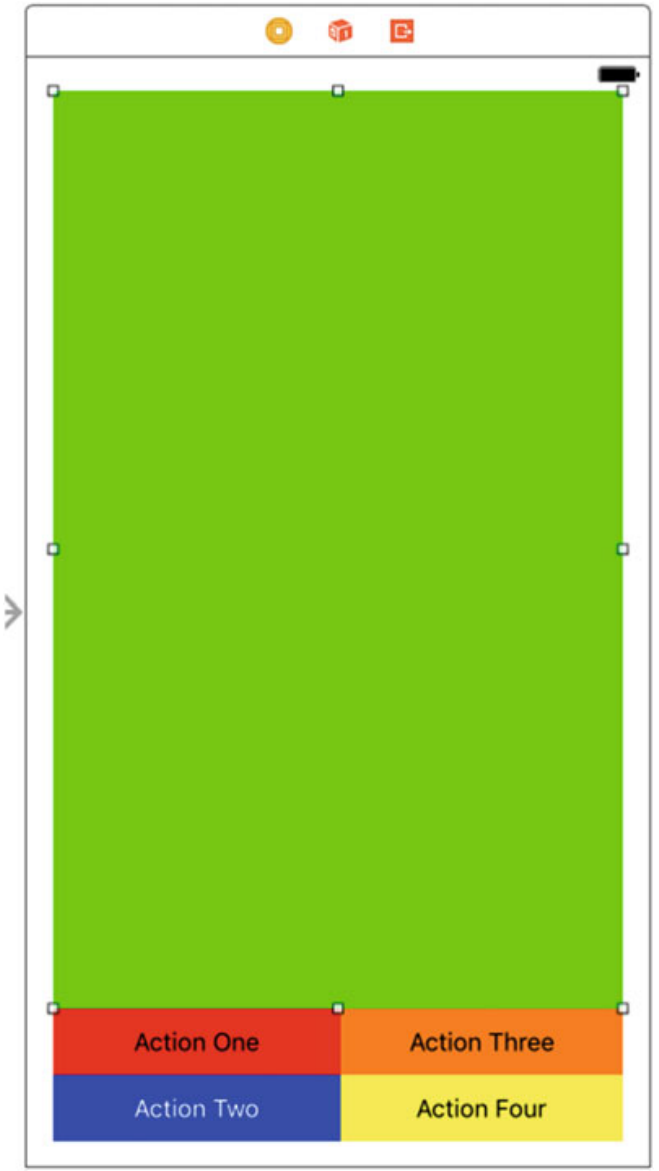


Creating the Restructure Application (continued)

- Select **`Main.storyboard`** to start editing the GUI. Since we don't really have an interesting content view we want to display, we'll just use a large colored rectangle.
- Drag a single **`UIView`** from the **Object Library** into your container view.
- While it's still selected, resize it so that it fills the top part of the available space, leaving a small margin above it and on both sides, as shown in the following figure.
- Next, switch over to the **Attributes Inspector** and use the **Background** pop-up to pick some other background color.
- You can choose anything you like, as long as it's not white, so that the view stands out from the background.
- In the storyboard in the example source code archive, this view is **green**, so from now on we'll call it the **green view**.

Creating the Restructure Application (continued)

- Drag a **button** from the **Object Library** and place it in the lower left of the empty space below the green view.
- Double-click to select the **text** in its label, and change it to **Action One** .
- Now **Option-drag** three copies of this button and place them in two columns, like those in the attached Figure.



Creating the Restructure Application (continued)

- You don't have to line them up perfectly because we're going to use constraints to finalize their positions, but you should try to place the two button groups approximately equal distances from their respective sides of the containing view.
- Change their titles to **Action Two**, **Action Three**, and **Action Four**.
- Also, let's add a different background color to each button so they're easy to see; I've used **red**, **blue**, **orange**, and **yellow** in order, but you can choose any colors you prefer.
- If you use a **dark background** color like blue, you want to make the text lighter as well.
- Finally, drag the lower edge of the green view downward until it just touches the top row of buttons. Use the **blue guidelines** to line everything up.

Creating the Restructure Application (continued)

- Now let's set up the **Auto Layout constraints** . Start by selecting the **green view**. We're going to start by **pinning** this to the **top** and to the **left** and **right** sides of the main view.

- That's still not enough to fully constrain it because its height isn't specified yet; we're going to fix that by anchoring it to the top of the buttons, once we've fixed the buttons themselves.

- Click the **Pin button** at the bottom right of the storyboard editor.

Creating the Restructure Application (continued)

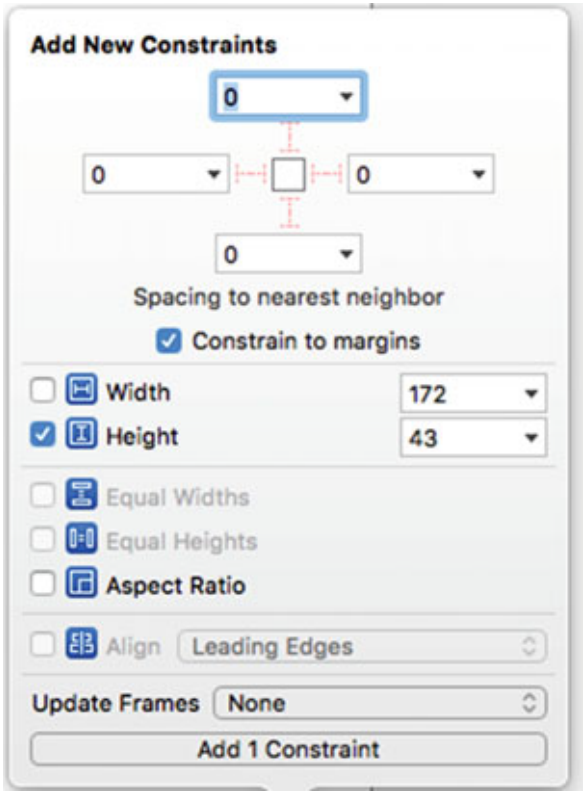
CENTENNIAL COLLEGE

- At the top of the pop-up, you'll see the now familiar group of four input fields surrounding a small square.
- Leave the **Constrain to Margins** check box checked.
- Click the **red dashed lines** above, to the **left**, and to the **right** of the small square to attach the view to the **top**, **left**, and **right** sides of its **superview.**
- Click Add 3 Constraints.

Creating the Restructure Application (continued)

CENTENNIAL COLLEGE

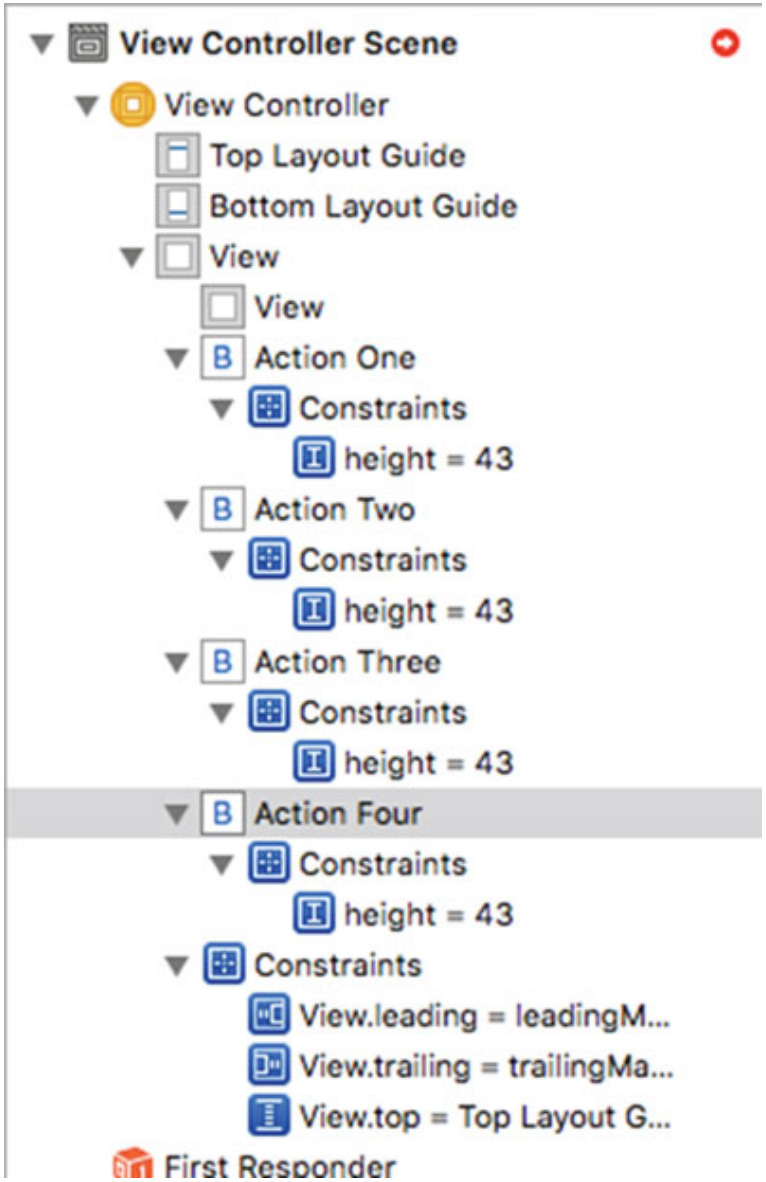
- For now, we'll set a **constant height** for our buttons, starting with **Action One**, as shown in the Figure below.
- Anything around a height of 43 should be okay for what we're trying to do in this example, which is deal with different devices and orientations.
- Then repeat the operation for each of the other three buttons.



Creating the Restructure Application (continued)

CENTENNIAL COLLEGE

- If you performed the operations correctly so far, you should be able to see the results of all of the constraints in the **Document Outline,** as shown in the following Figure, where we see each of our four button heights, as well as the three sides for the green view.

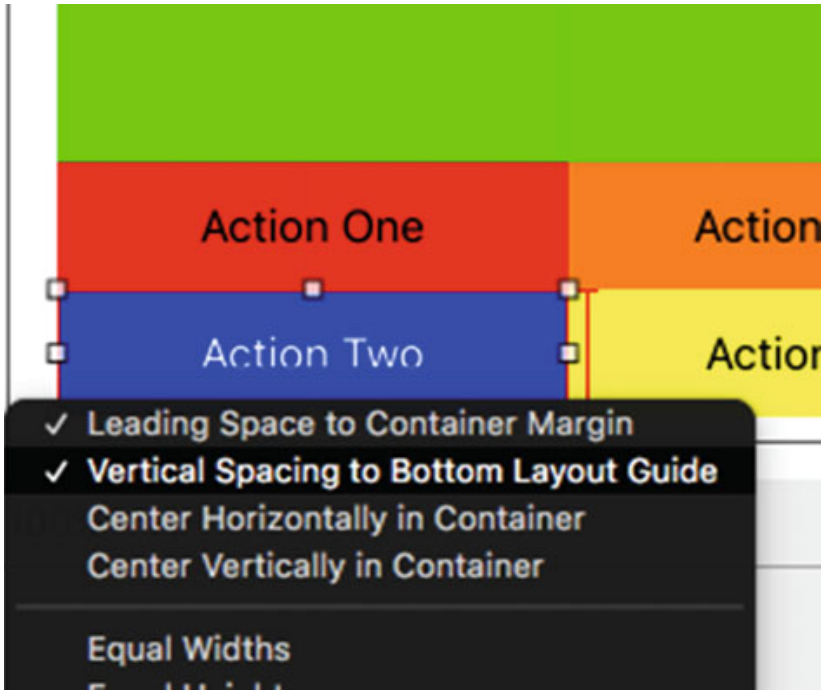


Creating the Restructure Application (continued)

CENTENNIAL COLLEGE

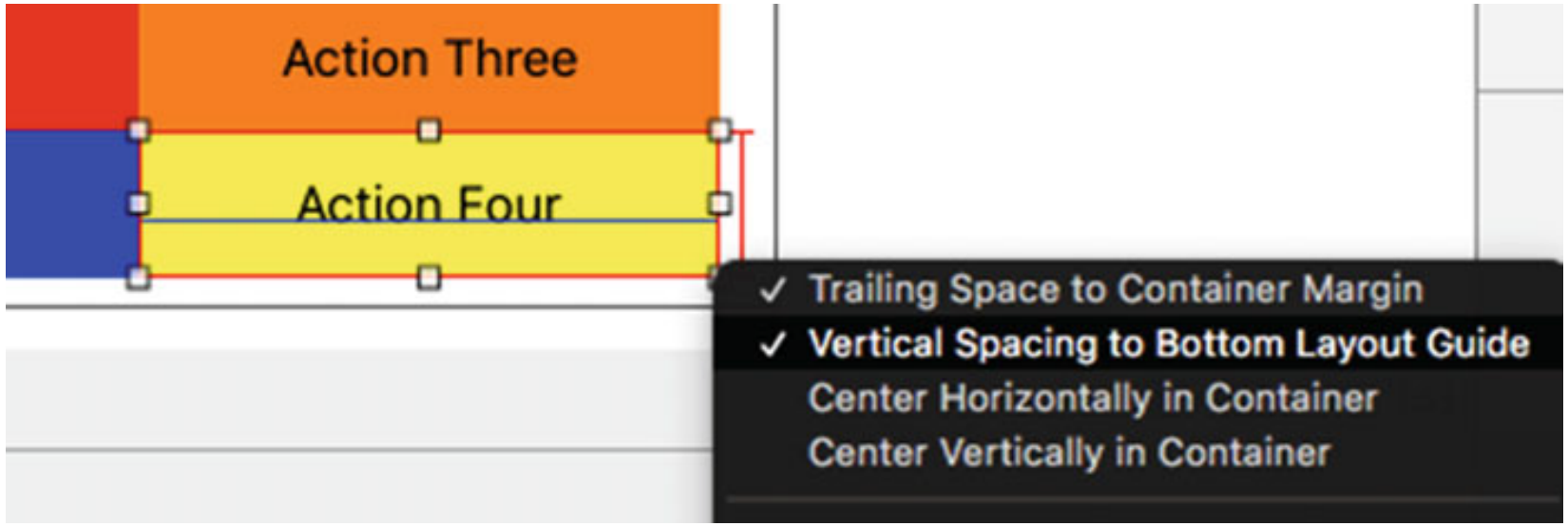
- Next pin the **bottom-left** (Action Two) and **bottom-right** (Action Four) buttons to the lower corners by **Control-dragging** from each button to the lower left and lower right, respectively.

- For **Action Two**, Shift-select the two options, as shown in the following the following Figure



Creating the Restructure Application (continued)

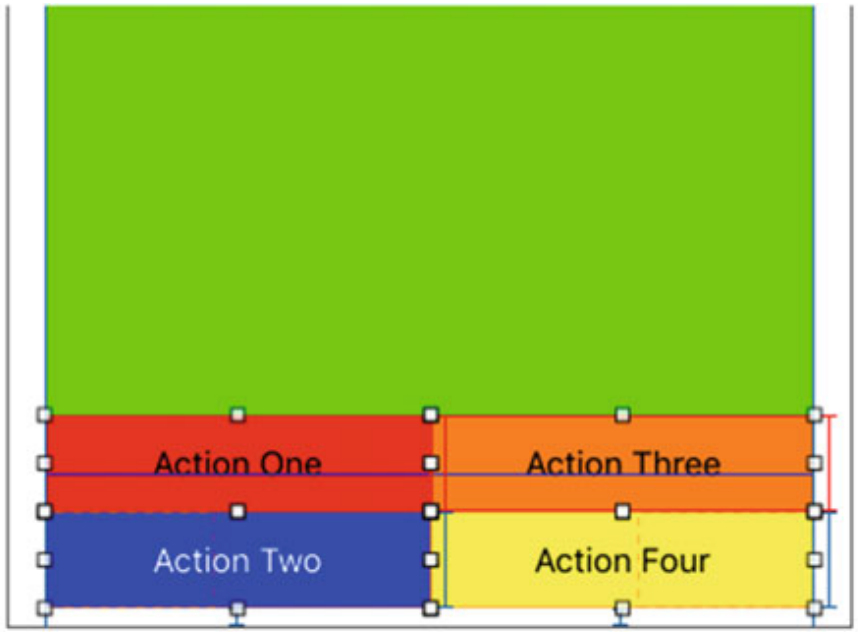
- Perform the similar operation, **Control-dragging** out to the right and down to set the **Leading and Vertical spacing** for the Action Four button, as shown in the following the following Figure

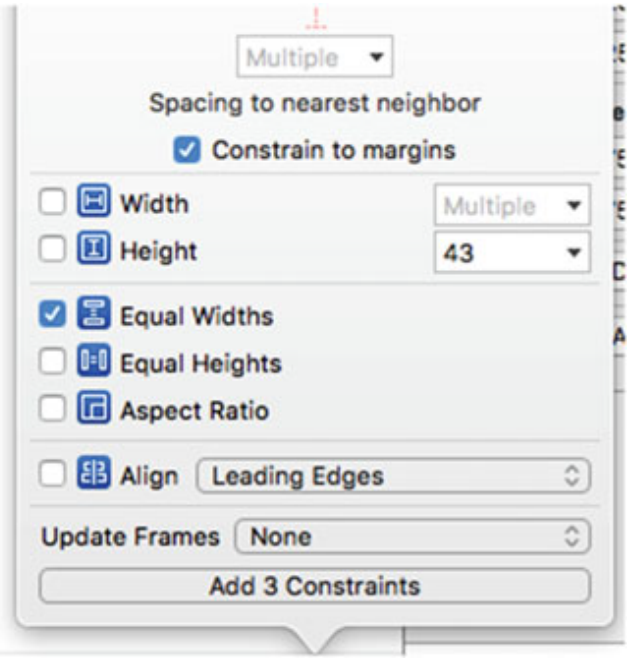


Creating the Restructure Application (continued)

- Next, Shift-select all four buttons, click the **Pin** icon, and set all the widths to be equal.

- Note that we haven't set a **width** yet, so they could vary from small to extremely wide; but in a moment, we'll take care of that through **additional constraints**.

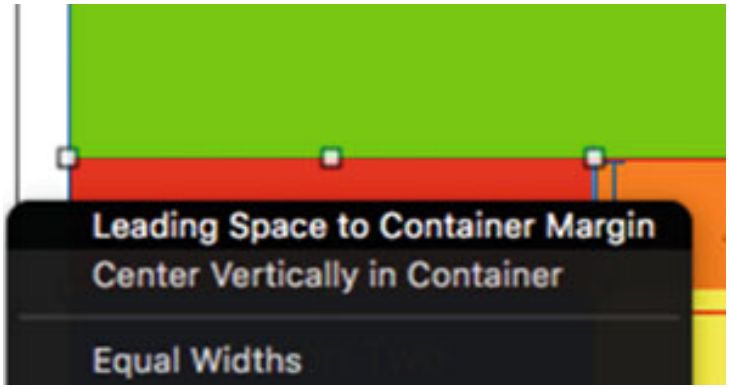


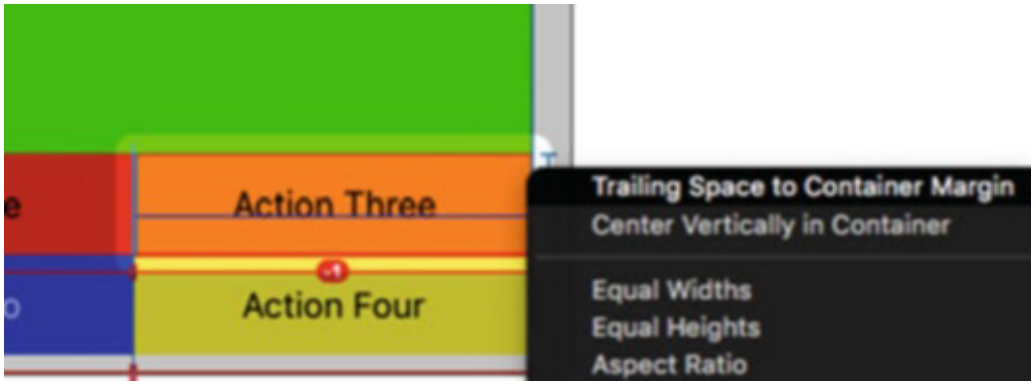


Creating the Restructure Application (continued)

- For the top **row of buttons**, Action One and Action Three, **Control-drag** to the left and right, respectively, to set the **Leading Space to Container Margin** and **Trailing Space to Container Margin**.

- This ties the left edge of Action One and the right edge of Action Three to the edge of the view, setting one of our width anchor points.



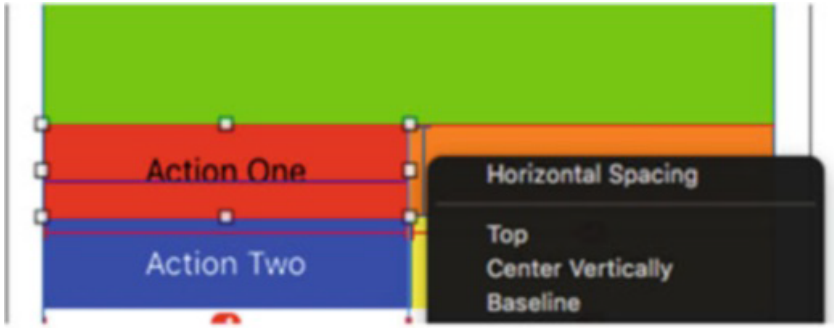


Creating the Restructure Application (continued)

- These final **two constraints** will be all we need to make sure that the buttons are half the width of the green view and match each other.

- **Control-drag** from Action One to Action Three and set **Horizontal Spacing**.

- Do the same thing between **Action Two** and **Action Four**, as shown in following the following Figure

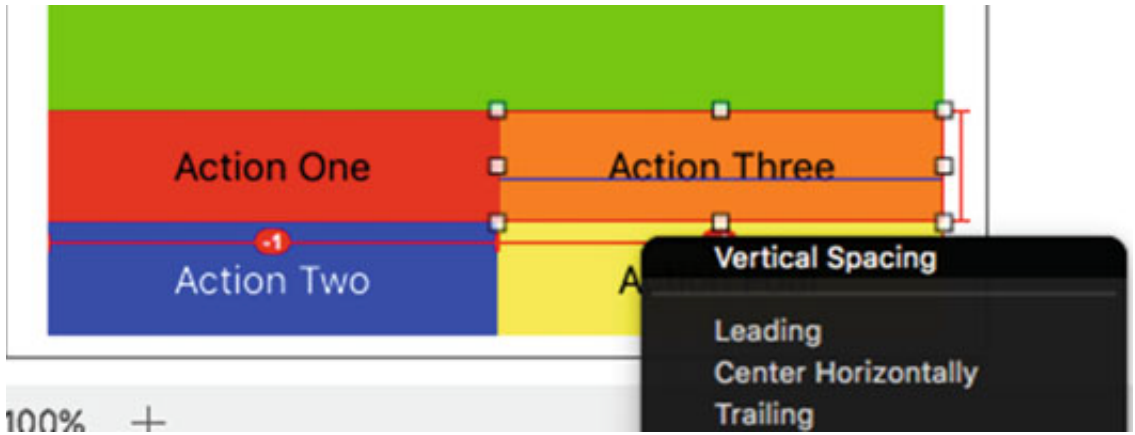


Creating the Restructure Application (continued)

- Just a couple more things and we'll be ready to test out our app in the various devices.

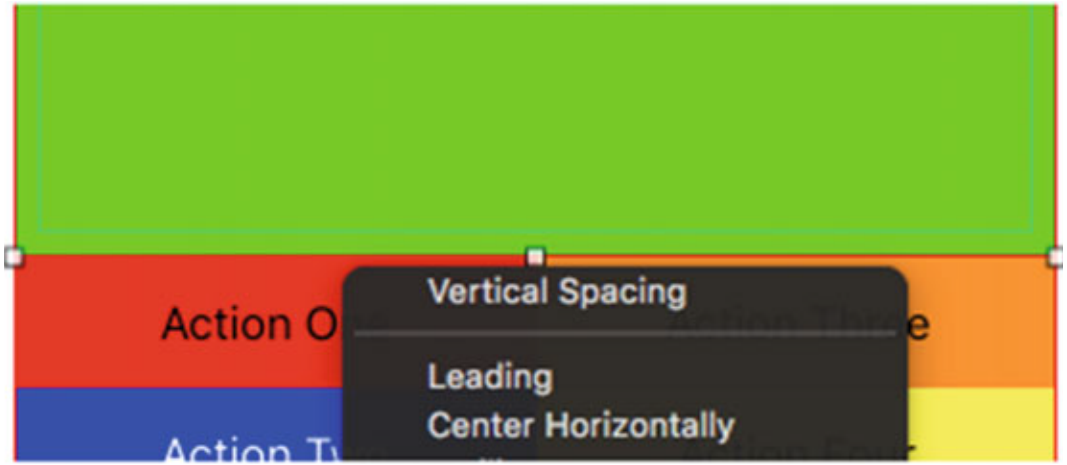
- **Control-drag** from **Action Thre**e to **Action Four** to set the **Vertical Spacing**, like we just did with the **Horizontal Spacing** of rows, as shown in the following Figure.

- Do the same between Action One and Action Two.



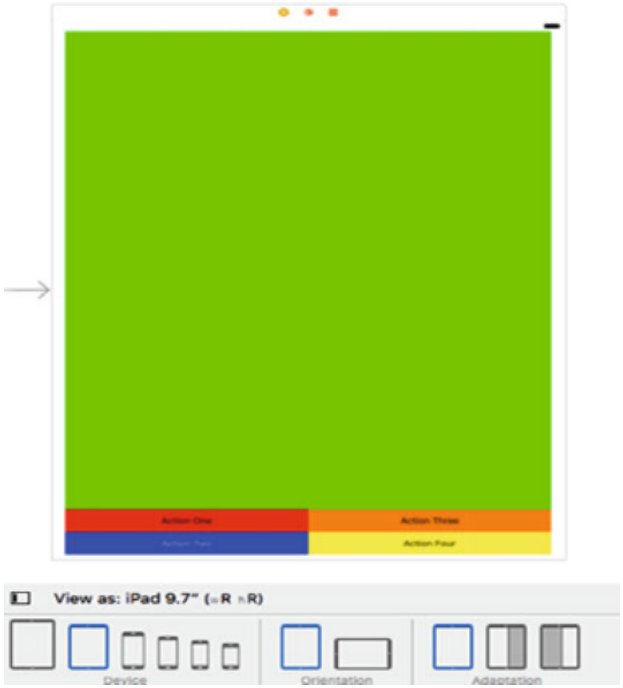
Creating the Restructure Application (continued)

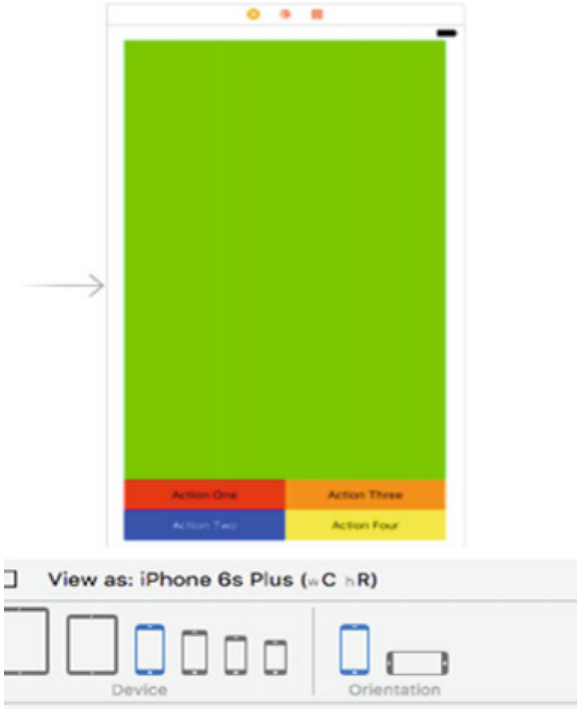
- Finally, **Control-drag** between the green view and the Action One button to set the spacing so that the **green view** and the top row of buttons are adjacent, as shown in the following Figure

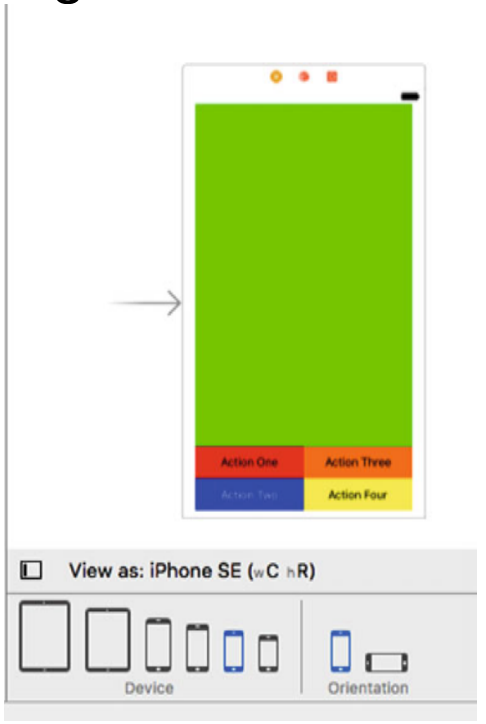


Creating the Restructure Application (continued)

- Now you should be able to select any full screen **iPhone** or **iPad** in either **portrait** or **landscape** orientation and the buttons should maintain their position, as shown in the following Figure







Setting the iPhone Landscape (wC hC) Configuration

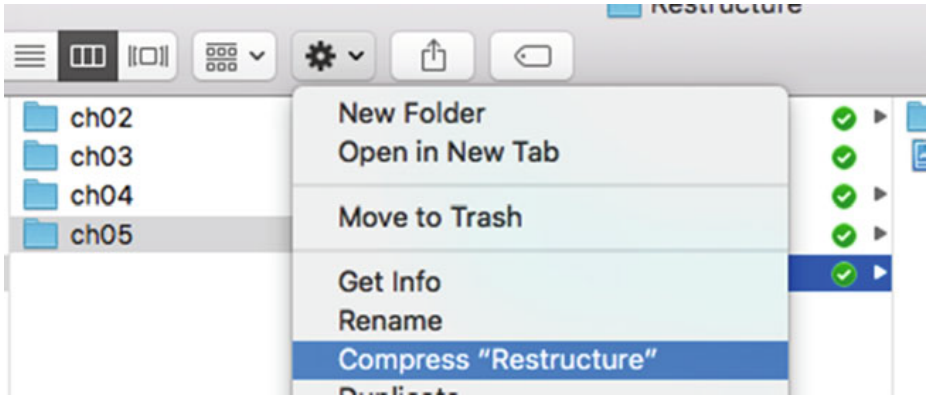
- While these look as we would expect them to, they're not what we're after.
- The notation w- h-, refers to the **width** and **height** of the configuration being considered.
- To simplify things, in **Auto Layout** this will be either **C** for **compact** or **R** for **regular**, so that you have the options: **`wC hC`**, **`wC hR`**, **`wR hC`**, and **`wR hR`**.
- By looking in the **Device Configuration Bar** you can see how these apply to actual Apple devices.
- For an **iPhone** in landscape, still a **`wC hC`** configuration, we want a single column of buttons pressed up against the right side.
- For an **iPad**, in any orientation, **`wR hR`**, we want a single row of buttons against the bottom of the view.

Setting the iPhone Landscape (wC hC) Configuration (continued)

- We'll get to setting up our **wC hC** landscape configuration quickly, but first save your work and then close the Xcode project.

- You can just click the **red ball** at the upper left of the Xcode window to close this project. You don't have to exit Xcode completely. Go to finder on your Mac to locate the **Restructure folder** and create a compressed version, as shown in the following Figure.

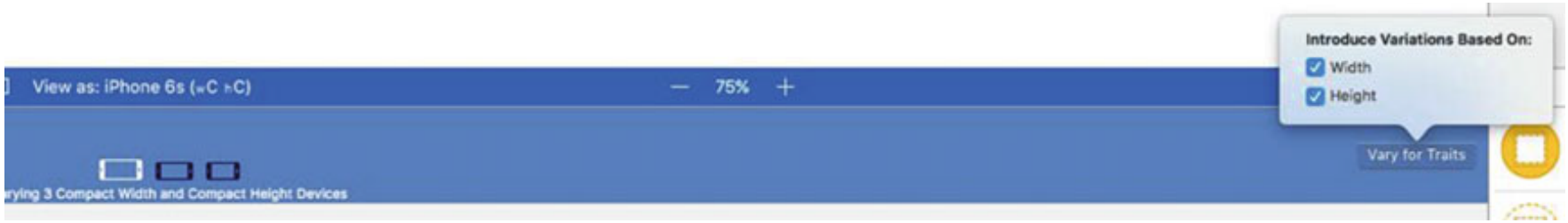
- This creates our "master" copy of the project that we can come back to at any time if, during the following changes, things get out of whack.



Setting the iPhone Landscape (wC hC) Configuration (continued)

CENTENNIAL COLLEGE

- First, let's create our **iPhone** landscape orientation.
- Select the **iPhone 6s** and the landscape orientation.
- To the right of the **Device Configuration Bar**, click the **Vary for Traits** and note that the bar turns blue, as shown in the following Figure



- In the popup select both **height** and **width**. You should see, that now, only **iPhone** devices are shown and only in **landscape orientation**. In this variation of traits, we'll develop our UI just for this configuration.

Setting the iPhone Landscape (wC hC) Configuration (continued)

CENTENNIAL COLLEGE

- Next, and yes this will seem scary, click all **five UI elements** (our green view and the four buttons) and press **Delete**.
- When completed, your canvas will look, as you'd expect, like the following Figure.
- But, if you look at the Document Outline, you'll still see the elements and even the constraints.
- That's because they exist for the **baseline configuration** and we're creating a **new configuration** or a **new trait collection** for just our `wC hC` landscape.

Setting the iPhone Landscape (wC hC) Configuration (continued)

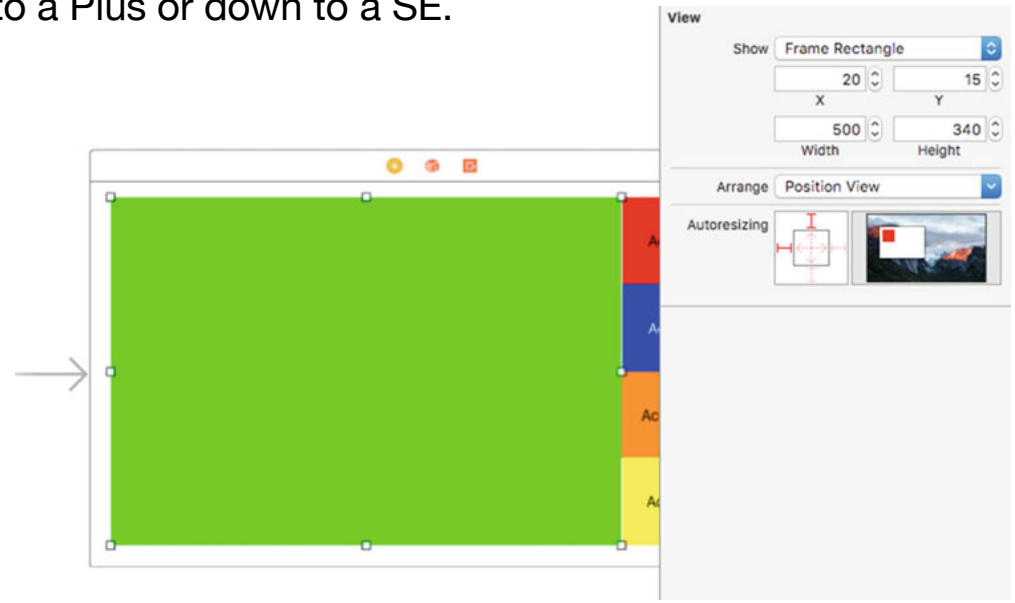
- As we did for our **baseline**, drag out a **`UIView`** and **four buttons** setting them up with colors and titles as we did earlier.

- Place them in the approximate positions shown in the following Figure but don't set any constraints just yet

Setting the iPhone Landscape (wC hC) Configuration (continued)

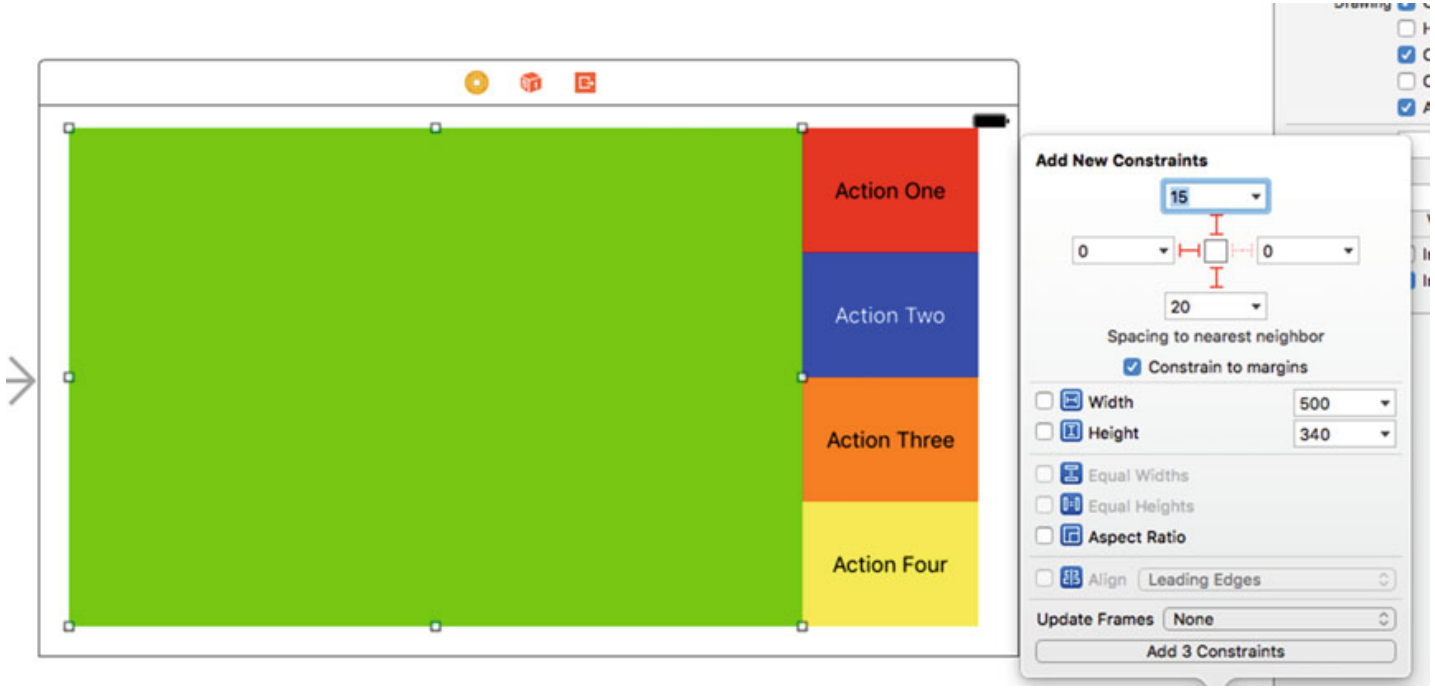
CENTENNIAL COLLEGE

- In this section, let's make our **measurements** a little more accurate.
- Select the **green view**. Using the **Size Inspector**, set the dimensions to **500** x **340** points, as shown in the following Figure.
- The **width** (**500**pts) is arbitrary; the height is **340**, which when divided by **4** comes out to **85**, which is what we will set as the height of the buttons.
- **Note** that this is not necessarily a constraint, but rather, the appearance on the storyboard.
- In fact, we don't want any fixed size for the green view since it will vary depending on whether we go up in size to a Plus or down to a SE.



Setting the iPhone Landscape (wC hC) Configuration (continued)

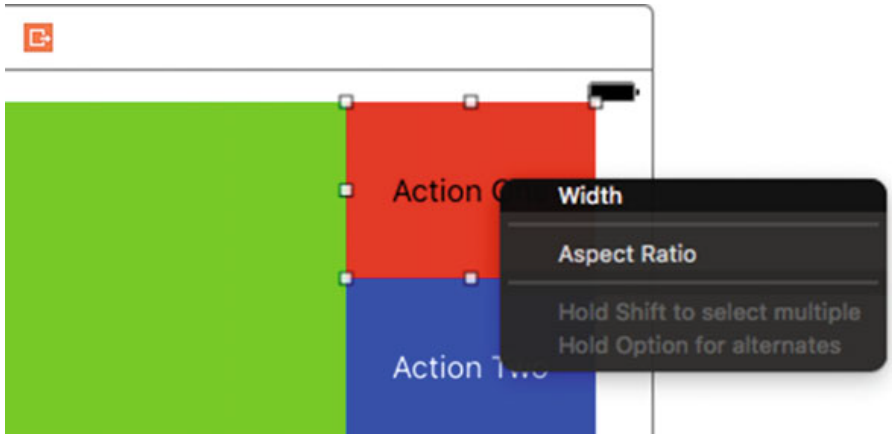
- Select the **green view** and pin the top, left and right sides to the edge, as shown in the following Figure

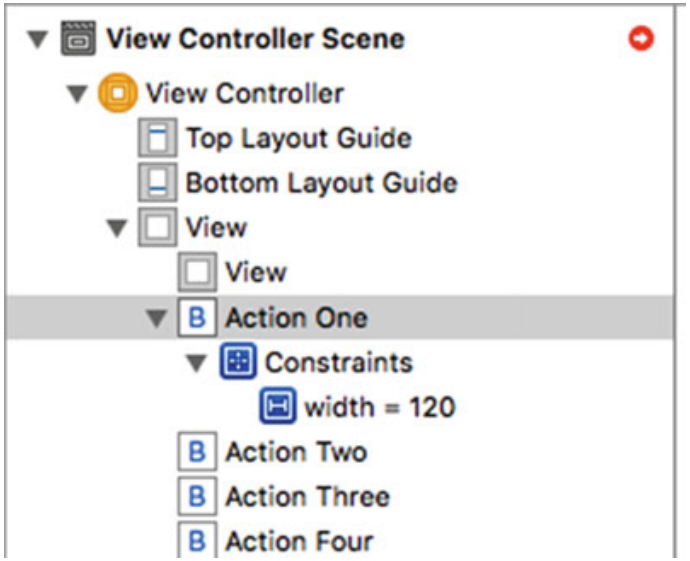


Setting the iPhone Landscape (wC hC) Configuration (continued)

CENTENNIAL COLLEGE

- Next, we're going to fix the **width** of the **Action One** button, but we're going to do it a little differently than previously.
- Inside the Action One button, **Control-drag** from one point to another both the starting and ending points within the red boundary and release.
- You're presented with a similar popover, on which you select **Width**.
- The width of the Action One button should now be 120 points, as seen in following Figures





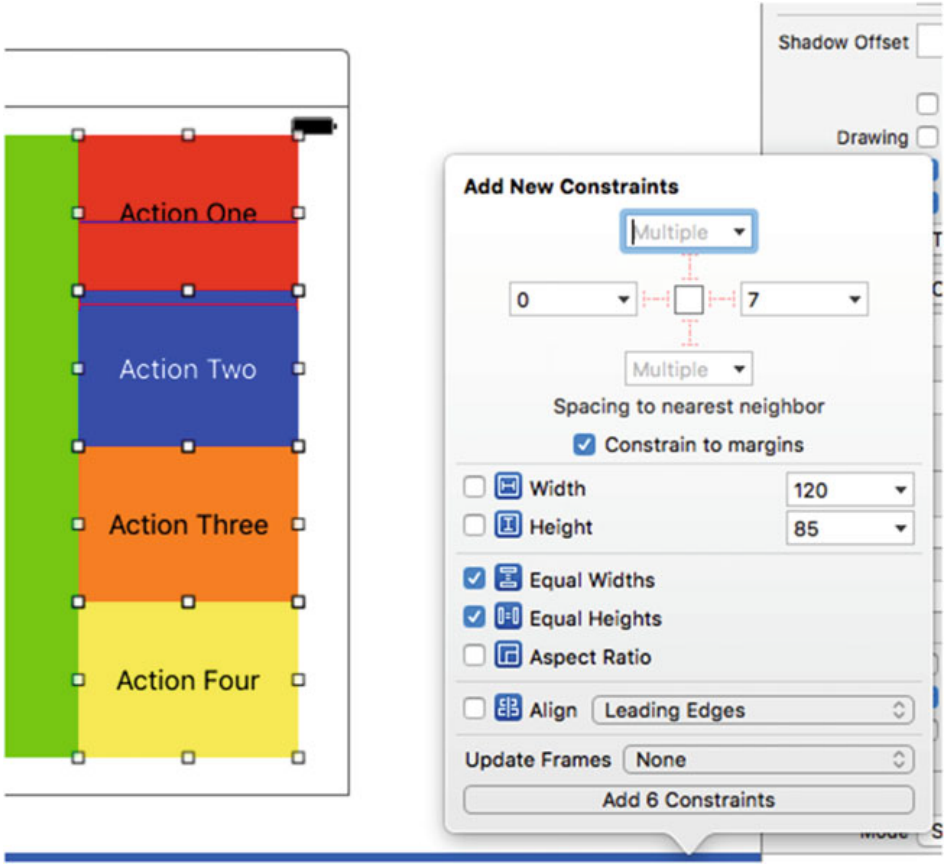
Setting the iPhone Landscape (wC hC) Configuration (continued)

CENTENNIAL COLLEGE

- We want all **four buttons** to be the same **width** and the same **height** — the width being **120** points but the height being adjust dynamically based on the available vertical area of the **iPhone** when in **landscape orientation**.

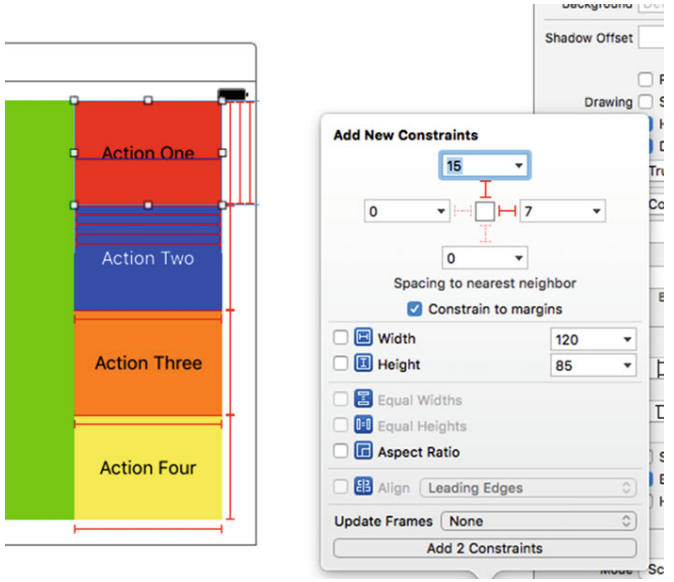
- As before, we'll handle the **equal height** by setting the spacing between the buttons in the column.

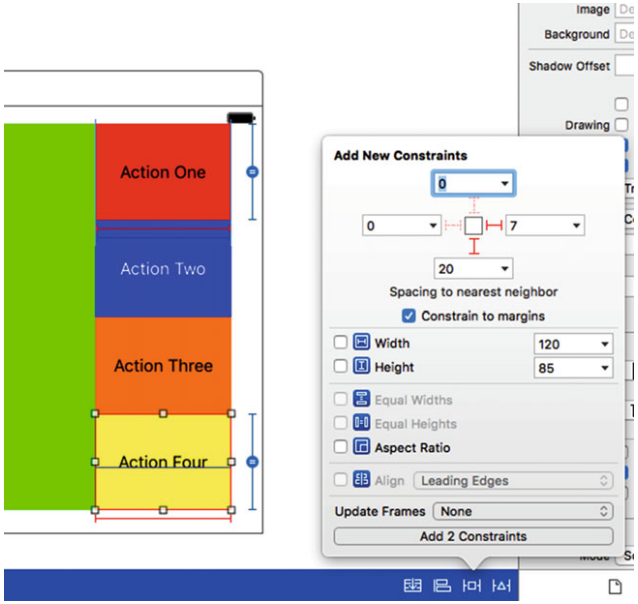
- For now, **Shift-click** all four buttons to select them all, click the **Pin** icon, and set the **Equal Widths** and **Equal Heights** constraints, as shown in the following Figure



Setting the iPhone Landscape (wC hC) Configuration (continued)

- Pin the Action One button to the top and right of its containing view and the Action Four button to the **right** and **bottom**

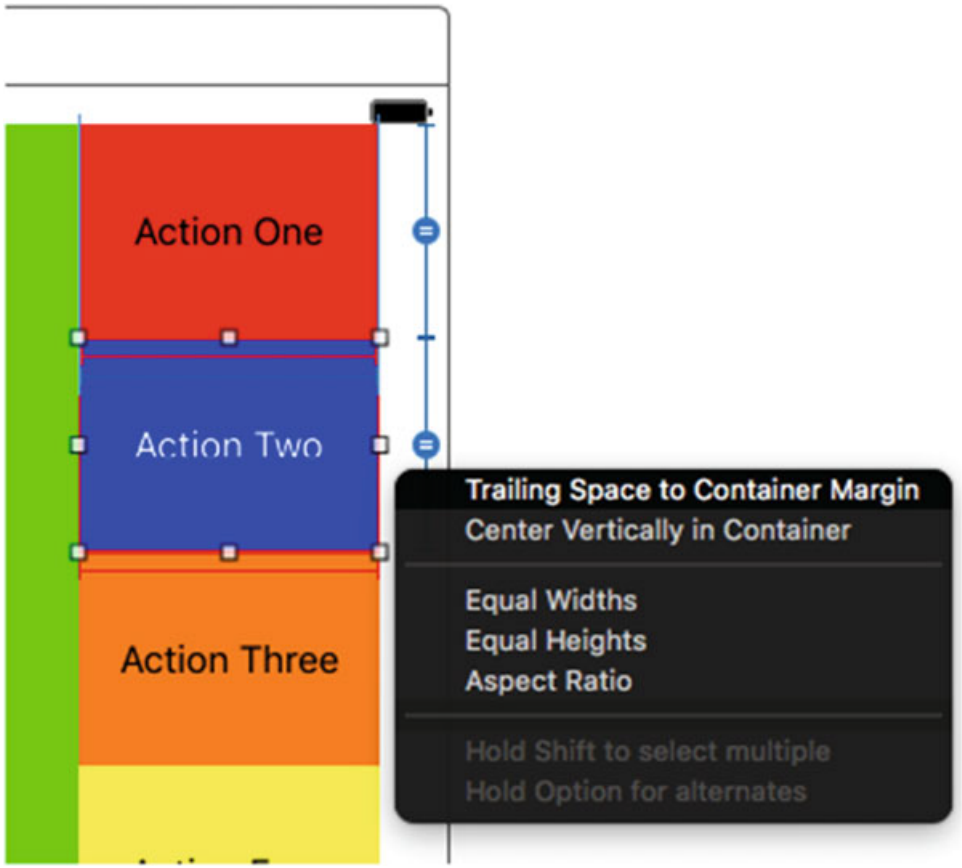




Setting the iPhone Landscape (wC hC) Configuration (continued)

CENTENNIAL COLLEGE

- For the Action Two button, **Control-drag** to the right and choose **Trailing Space to Container Margin** to pin it to the right side of the container.
- Repeat this operation with the Action Three button.

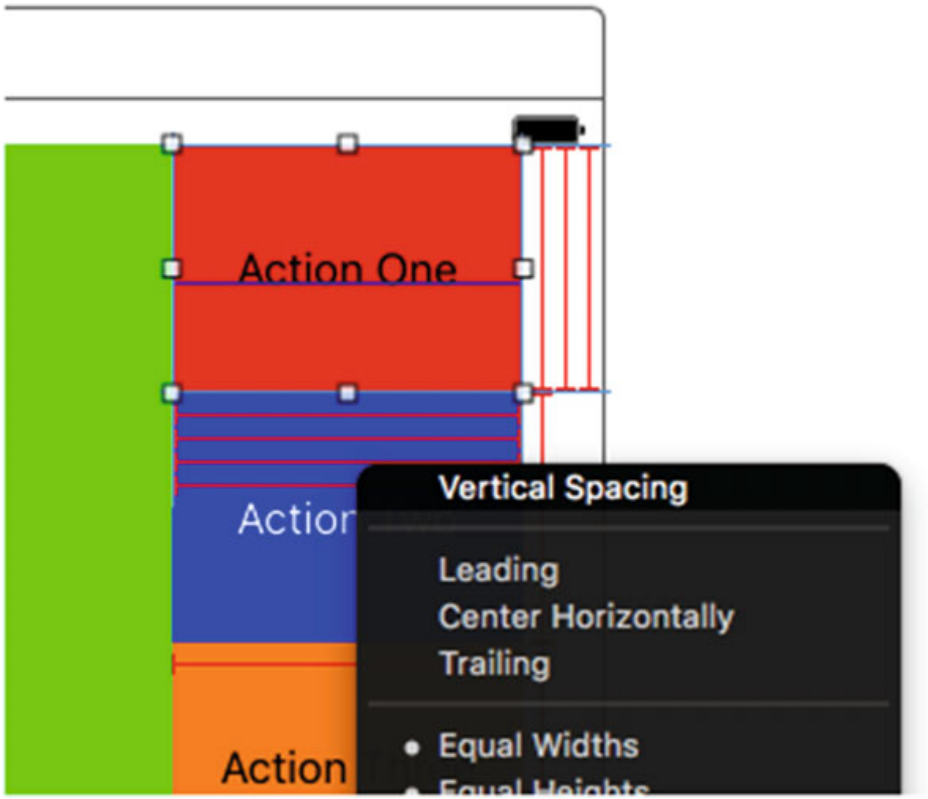


Setting the iPhone Landscape (wC hC) Configuration (continued)

CENTENNIAL COLLEGE

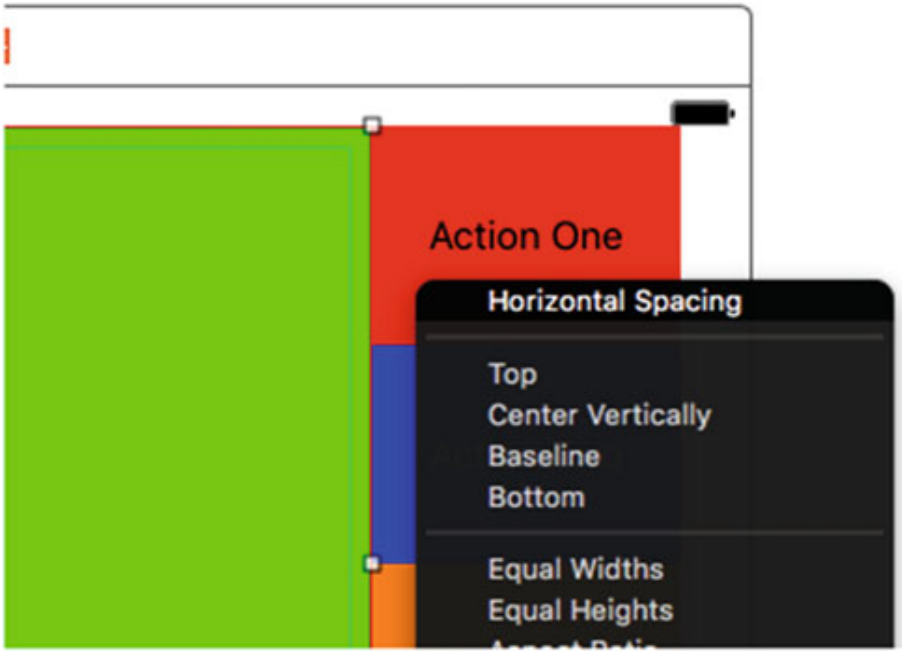
- Between the Action One and Action Two buttons, **Control-drag** to set the **Vertical Spacing**.

- Repeat for the spacing between Action Two and Action Three, and Action Three and Action Four.



Setting the iPhone Landscape (wC hC) Configuration (continued)

- Finally, set the **Horizontal Spacing** between the green view and the Action One button (you could use any of the four buttons)



CENTENNIAL COLLEGE

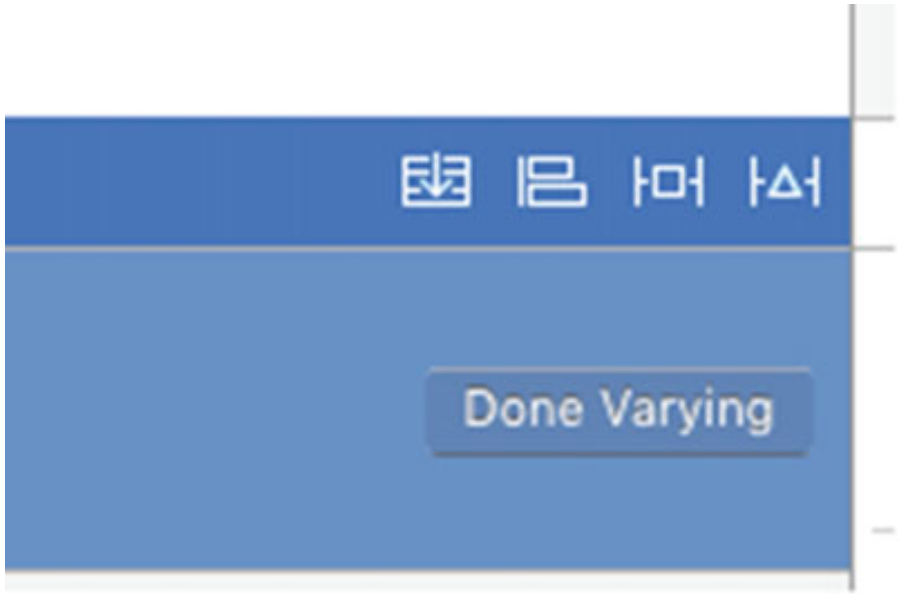
Setting the iPhone Landscape (wC hC) Configuration (continued)

CENTENNIAL COLLEGE

- Click the **Done Varying button** in the **Device Configuration Bar** to finish adding, placing and setting constraints for this iPhone landscape configuration

- The buttons should now be properly placed for any of the three iPhone **`wC hC`** configurations.

- Note that if we were to select a 6/6s Plus, because that is a **`wR hC`** device, we would see the earlier **baseline layout**.



Setting the iPad (iPhone Plus Landscape) (wR hR) Configurations

- For this configuration, to save space, The key steps are in the following Table along with pointing to the **reference figure**s in case you might need a little help with understanding what the step is about.

CENTENNIAL COLLEGE

STEP	Action	Figure
1	Click Vary For Traits and introduce variations based on width.	5-48
2	Delete the five UI elements in the Storyboard. Then add back five new elements from the object library as we did in the last section.	5-49
3	Select the green view we just added and in the Size Inspector, set the width to 728 and the height to 926. These are not constraints; they'll just help us with visually laying out the elements on the Storyboard. (Note: These values work for the iPhone 6s. If using a different device size, you will need to vary your height and width accordingly.)	5-50
4	Select the Action Four button and set its width to 182 using the Size Inspector...again, this is for the iPhone 6s device. This is only for visual placement and not a constraint.	5-51
5	Making sure we still have a blue Device Configuration Bar indicating that we're still in the Vary for Traits mode, align the UI elements as shown: the green view along the top and a single row of buttons along the bottom.	5-52
6	Similar to what we did before, pin the green view to the top, left and right sides of the containing view.	5-53

Setting the iPad (iPhone Plus Landscape) (wR hR) Configurations (continued)

CENTENNIAL COLLEGE

STEP	Action	Figure
7	Pin the Action One button to the lower-left corner of the containing view.	5-54
8	Pin the Action Four button to the lower-right corner of the containing view.	5-55
9	Pin the Action Two button to the bottom edge of the containing view.	5-56
10	Pin the Action Three button to the bottom edge of the containing view.	5-57
11	Add a constraint to set the height of the Action One button to a fixed value. I used 63 points because it fit the layout on my Storyboard. There is no "right" answer; adjust it to your needs for your layout.	5-58
12	Shift-select all four action buttons along the bottom row and set to equal height and width.	5-59
13	Click-drag from the green view to the Action One button and set the Vertical Spacing. This forces the green view to sit against the row of buttons.	5-60
14	Click-drag from Action One to Action Two setting the Horizontal Spacing. Repeat for Action Two to Action Three and Action Three to Action Four. This causes the buttons to sit next to each other on the sides and to be one-quarter the width of the enclosing container.	5-61
15	Click the Done Varying button to end the modifications for this set of traits.	5-62

Setting the iPad (iPhone Plus Landscape) (wR hR) Configurations (continued)

CENTENNIAL COLLEGE

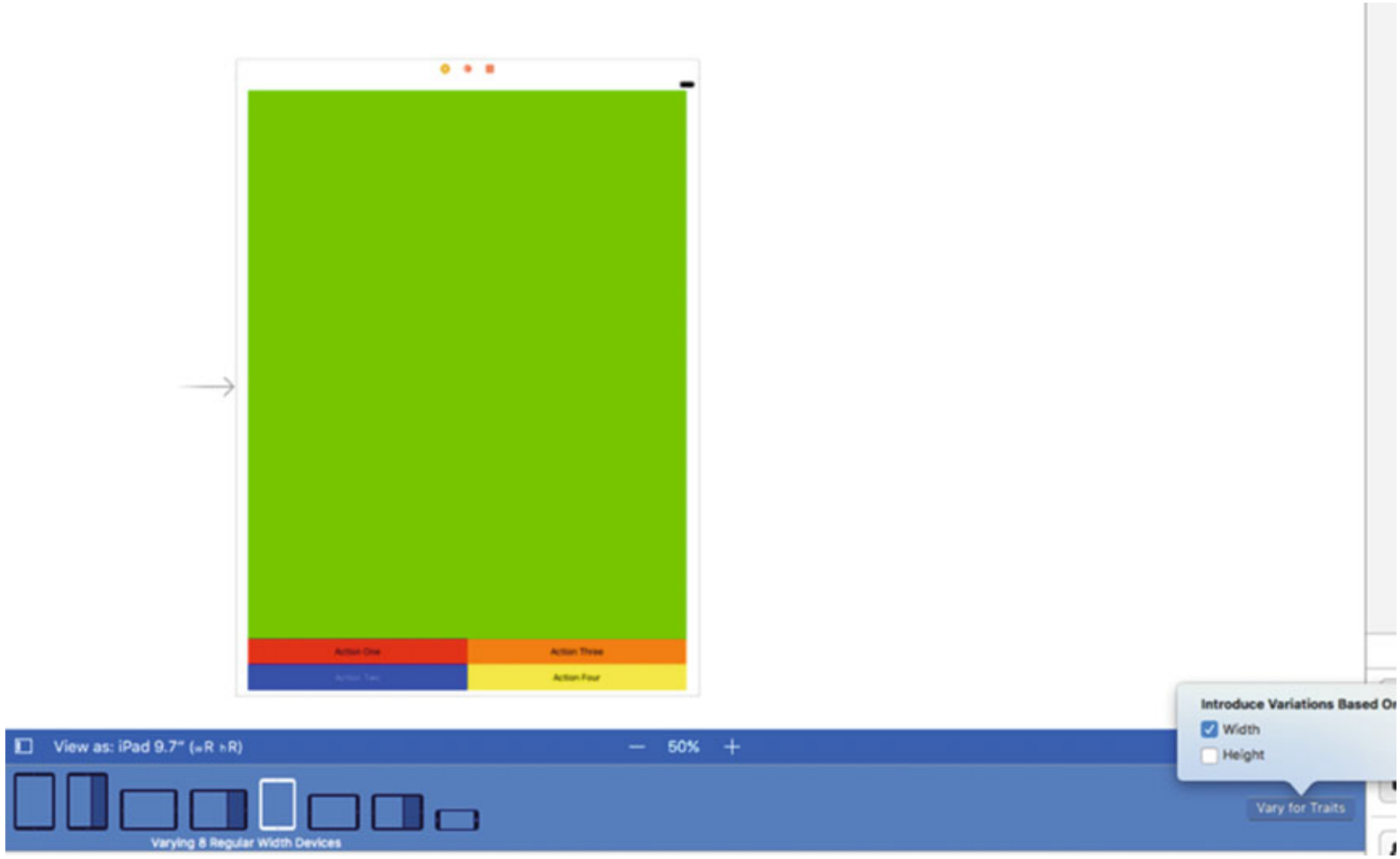


Figure 5-48. Select an iPad, click Vary For Traits, and choose Width

Setting the iPad (iPhone Plus Landscape) (wR hR) Configurations (continued)

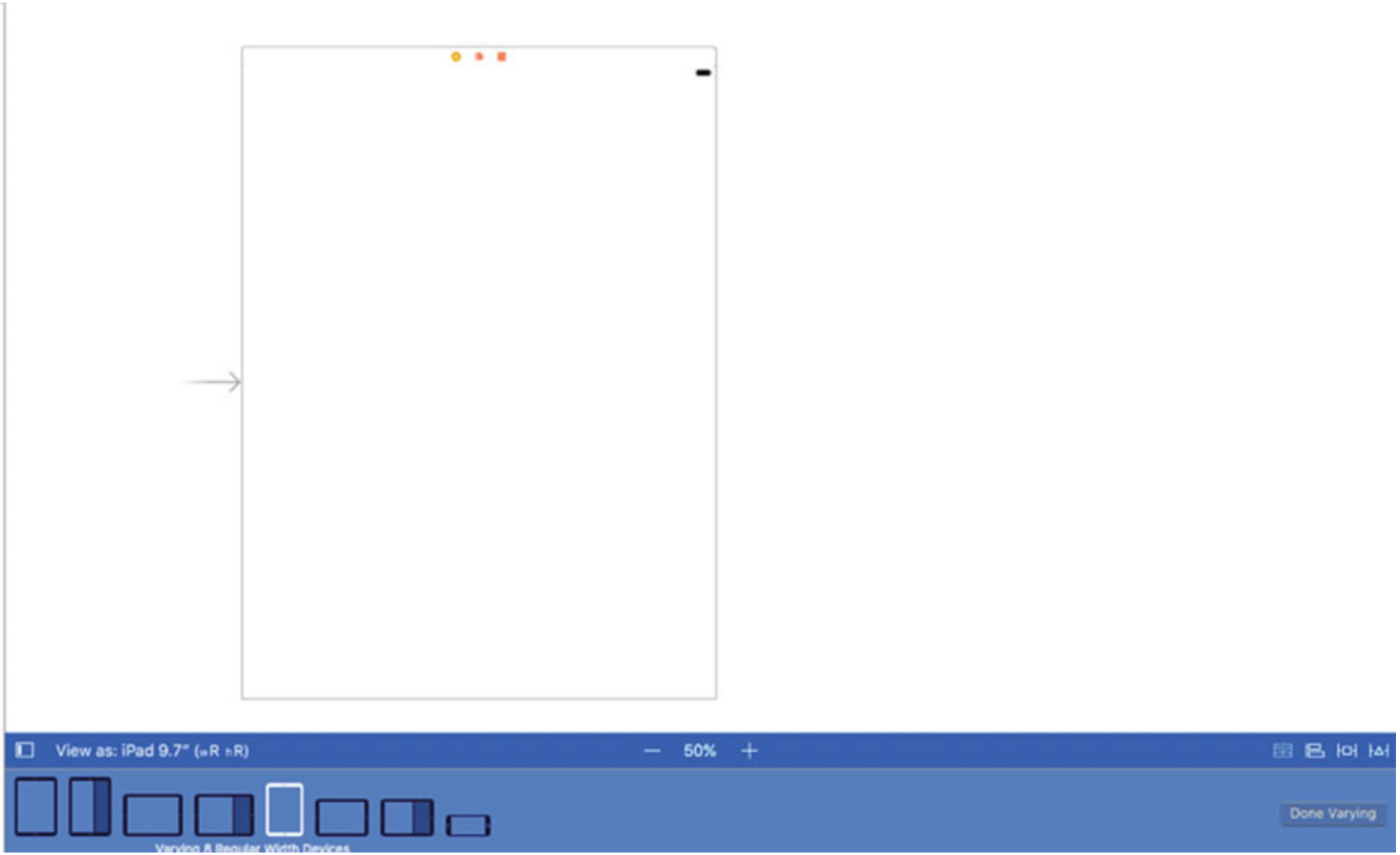


Figure 5-49. Delete the five UI elements

Setting the iPad (iPhone Plus Landscape) (wR hR) Configurations (continued)

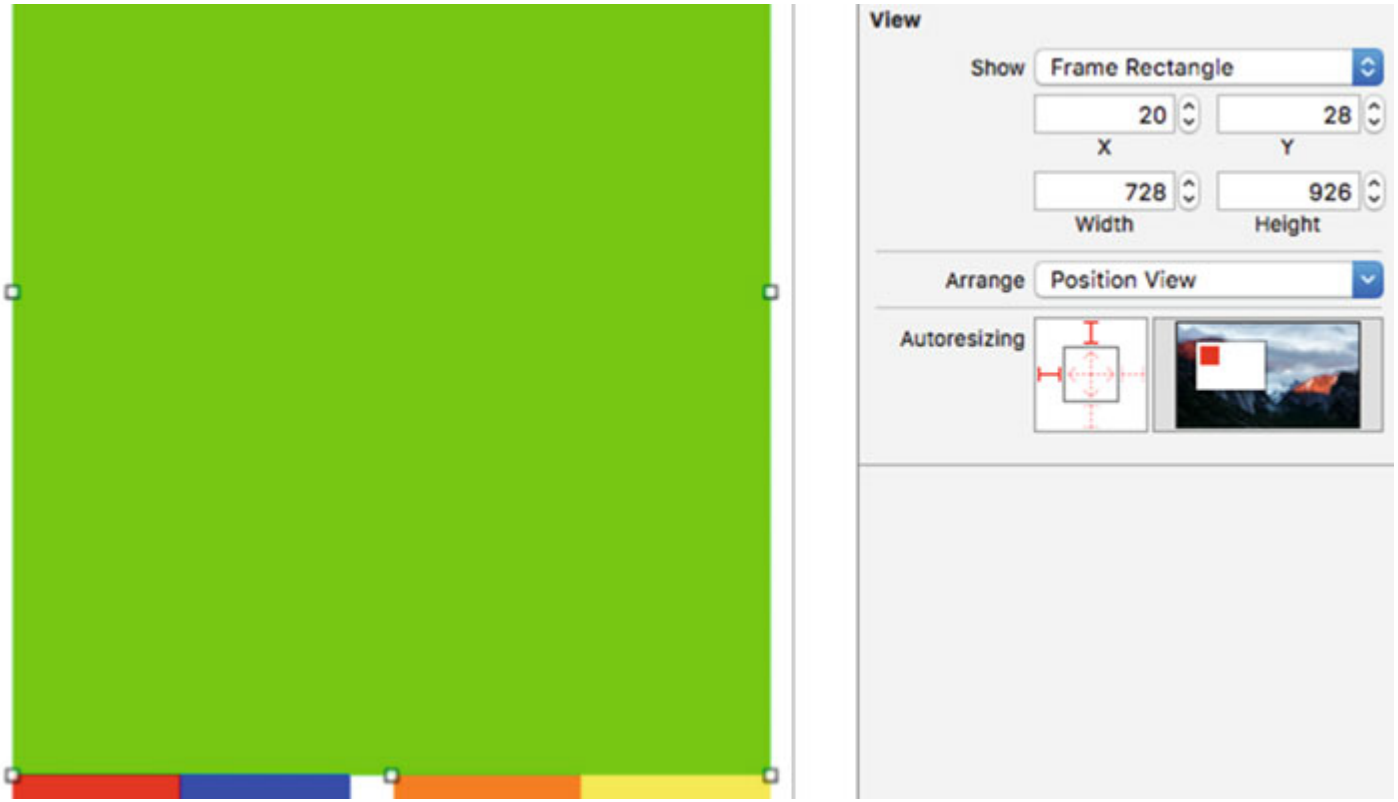
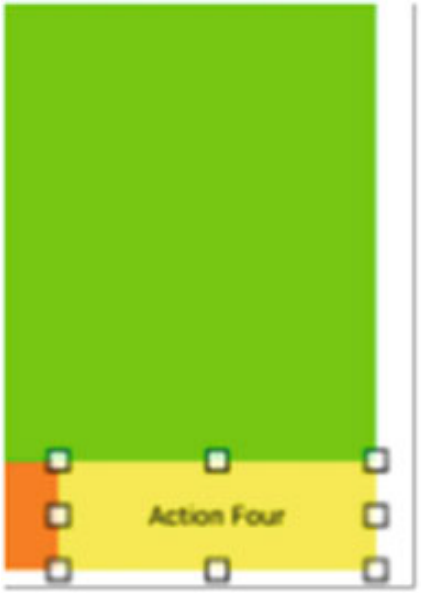


Figure 5-50. Add in the five new UI elements from the UI object library and set the width and height of the green view using the Size Inspector. This does not set constraints, only the visual aspects so we can adjust our Storyboard

Setting the iPad (iPhone Plus Landscape) (wR hR) Configurations (continued)



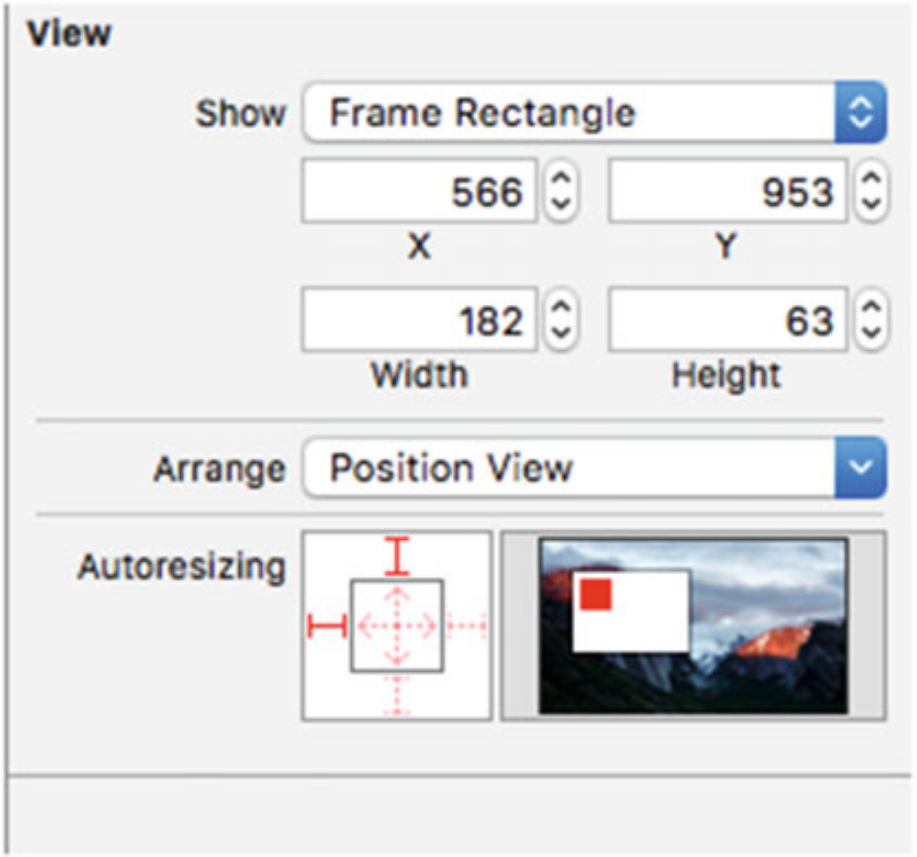


Figure 5-51. Similarly, set the width and height of the Action Four button so we have a visual reference with which to work and manipulate our Storyboard

Setting the iPad (iPhone Plus Landscape) (wR hR) Configurations (continued)

CENTENNIAL COLLEGE

Figure 5-52. Align everything up as shown making sure our Device Configuration Bar is still blue indicating we're working with a particular set of traits

Setting the iPad (iPhone Plus Landscape) (wR hR) Configurations (continued)

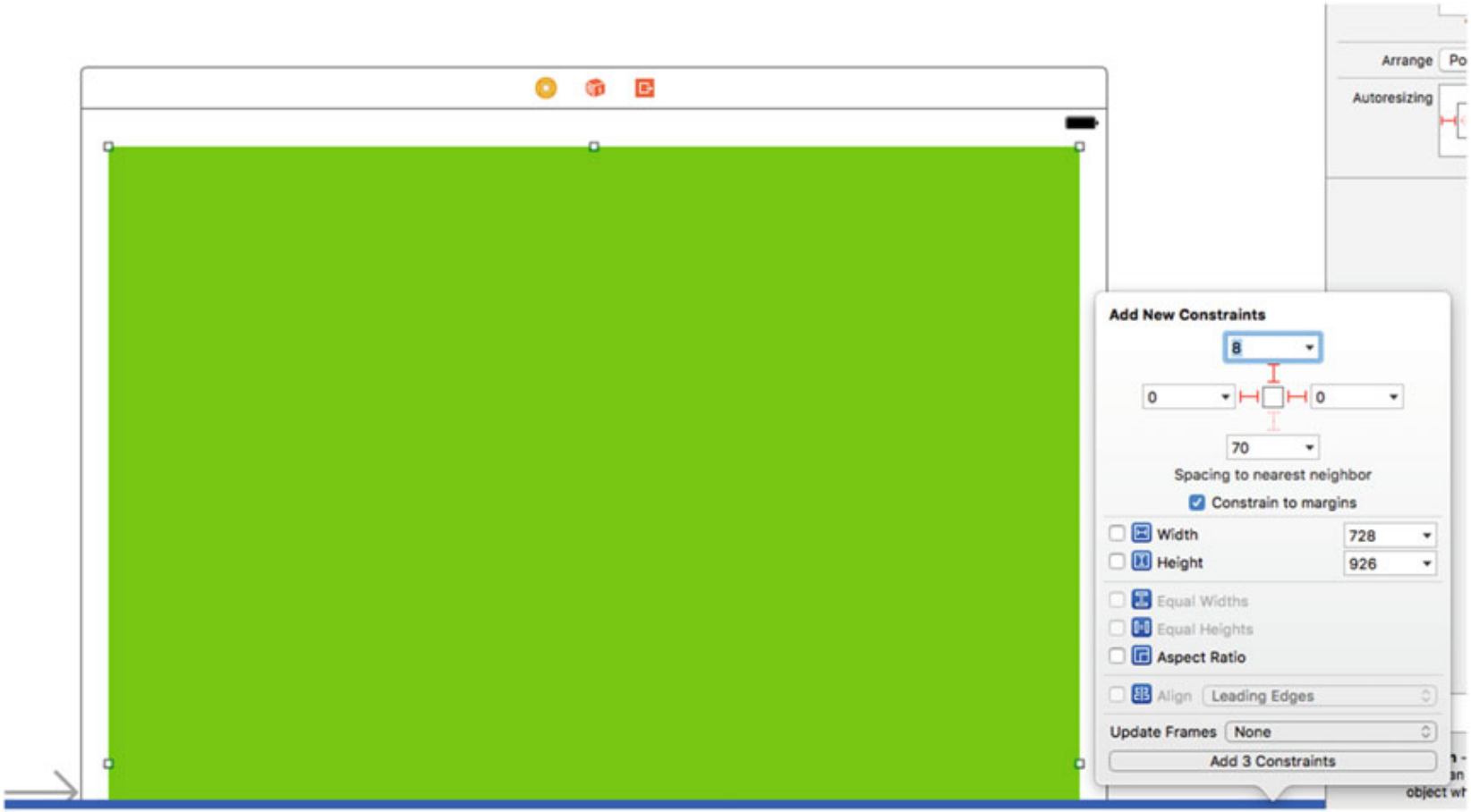


Figure 5-53. Similar to what we did in the previous sections, pin the green view to the top, left and right sides of the containing view

Setting the iPad (iPhone Plus Landscape) (wR hR) Configurations (continued)

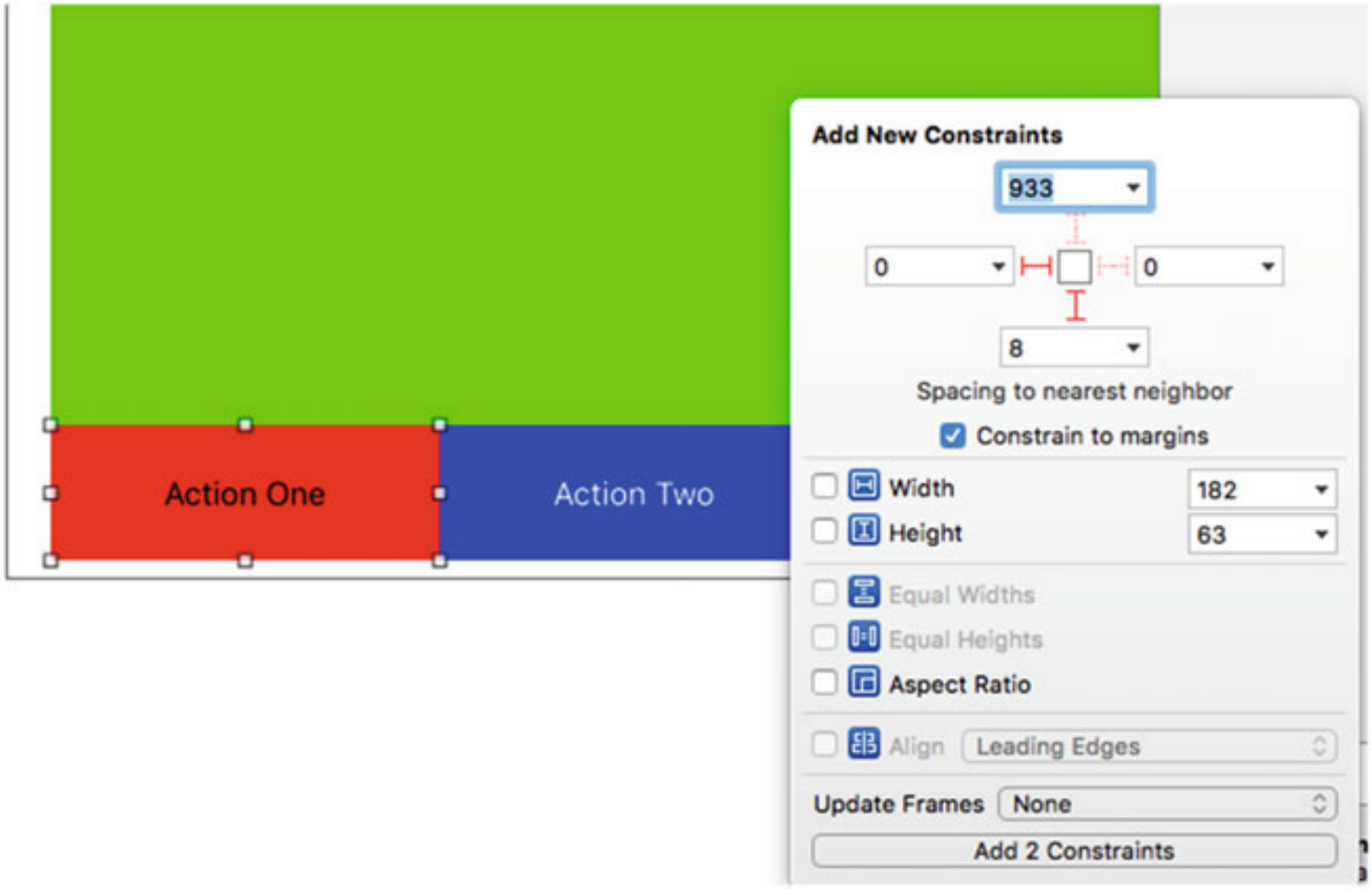


Figure 5-54. Pin the Action One button to the lower-left corner of the containing view

Setting the iPad (iPhone Plus Landscape) (wR hR) Configurations (continued)

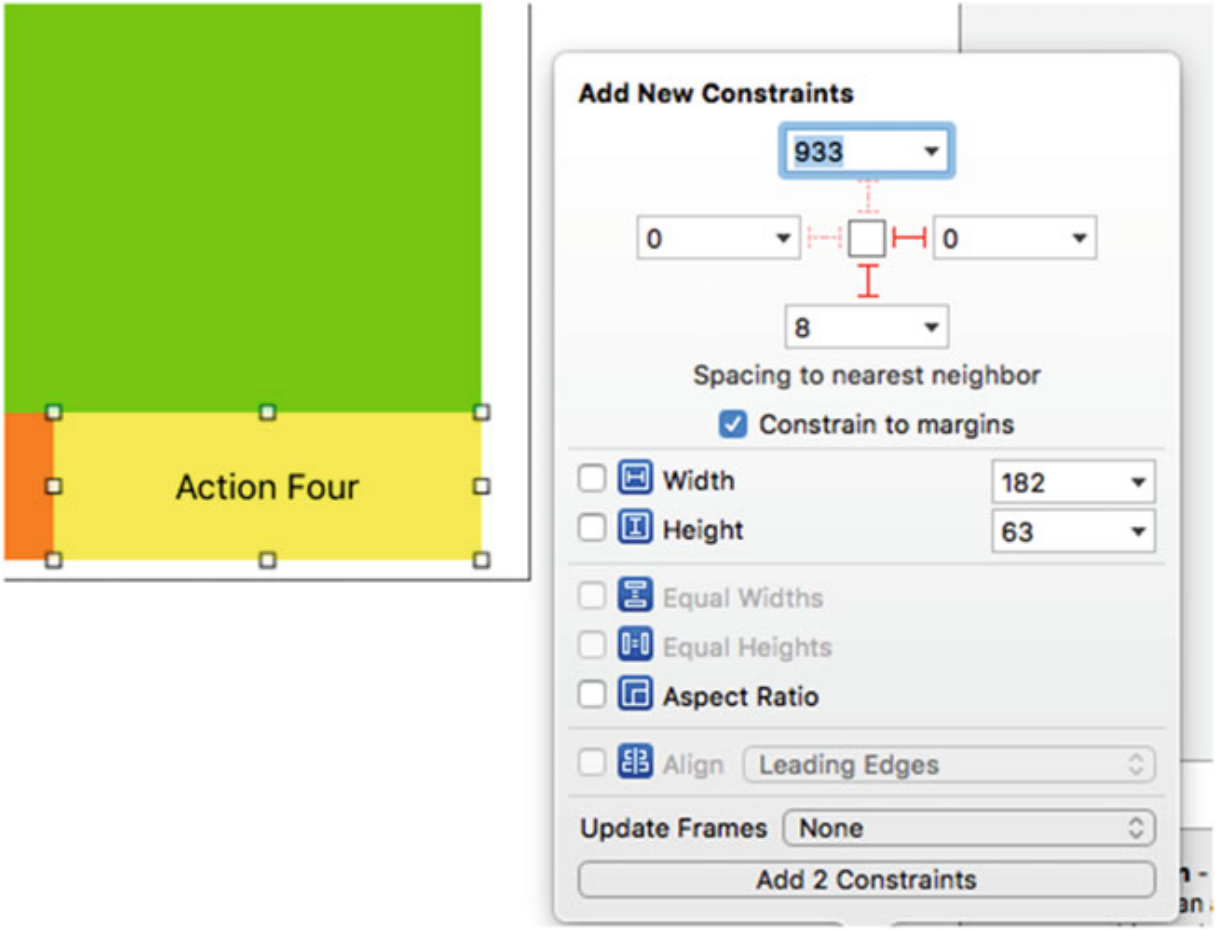


Figure 5-55. Pin the Action Four button to the lower right corner of the containing view

Setting the iPad (iPhone Plus Landscape) (wR hR) Configurations (continued)

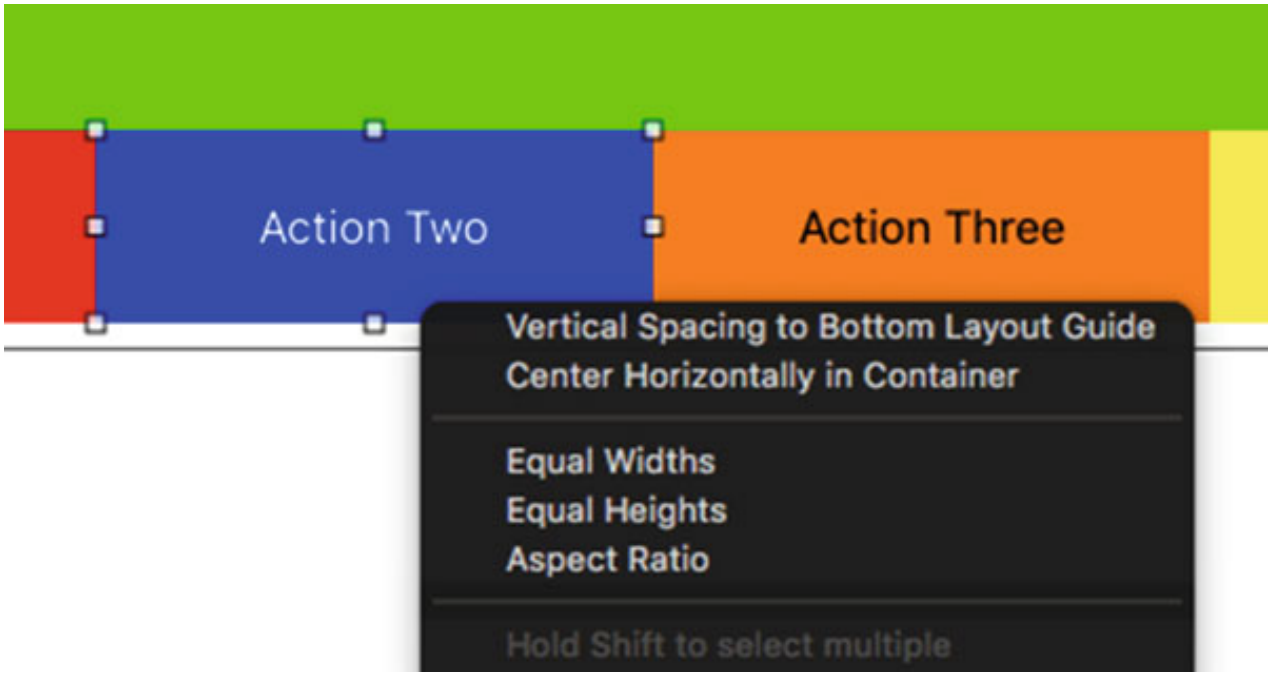


Figure 5-56. Pin the Action Two button to the bottom edge of the containing view

Setting the iPad (iPhone Plus Landscape) (wR hR) Configurations (continued)

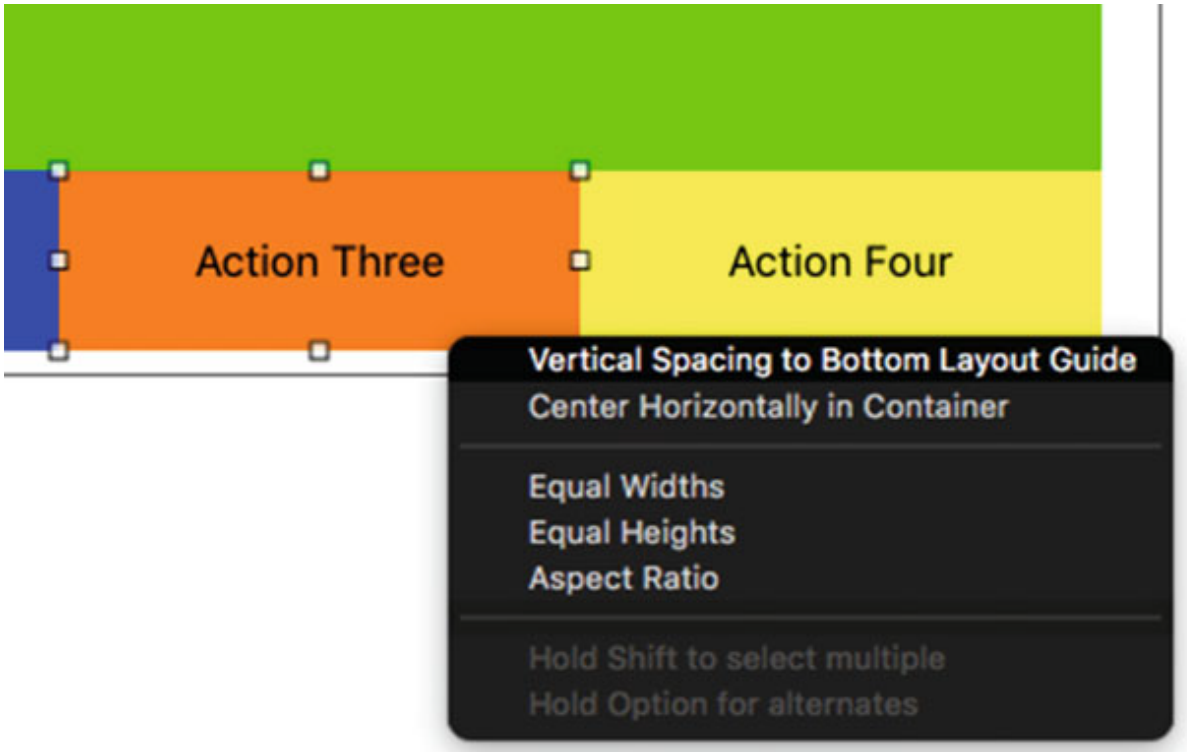


Figure 5-57. Pin the Action Three button to the bottom edge of the containing view

Setting the iPad (iPhone Plus Landscape) (wR hR) Configurations (continued)

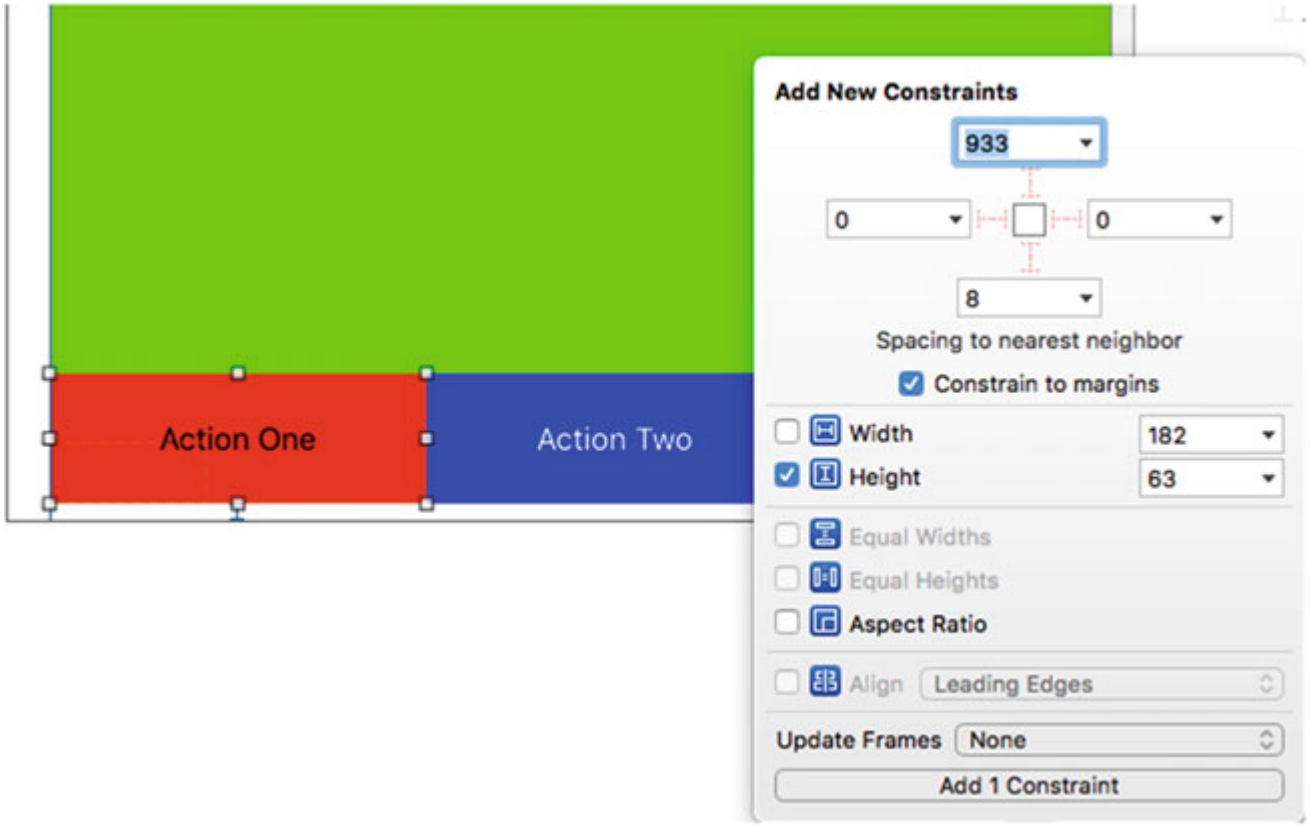


Figure 5-58. Add a constraint to set the height of the Action One button to a fixed value. I used 63 points because it fit the layout on my Storyboard

Setting the iPad (iPhone Plus Landscape) (wR hR) Configurations (continued)

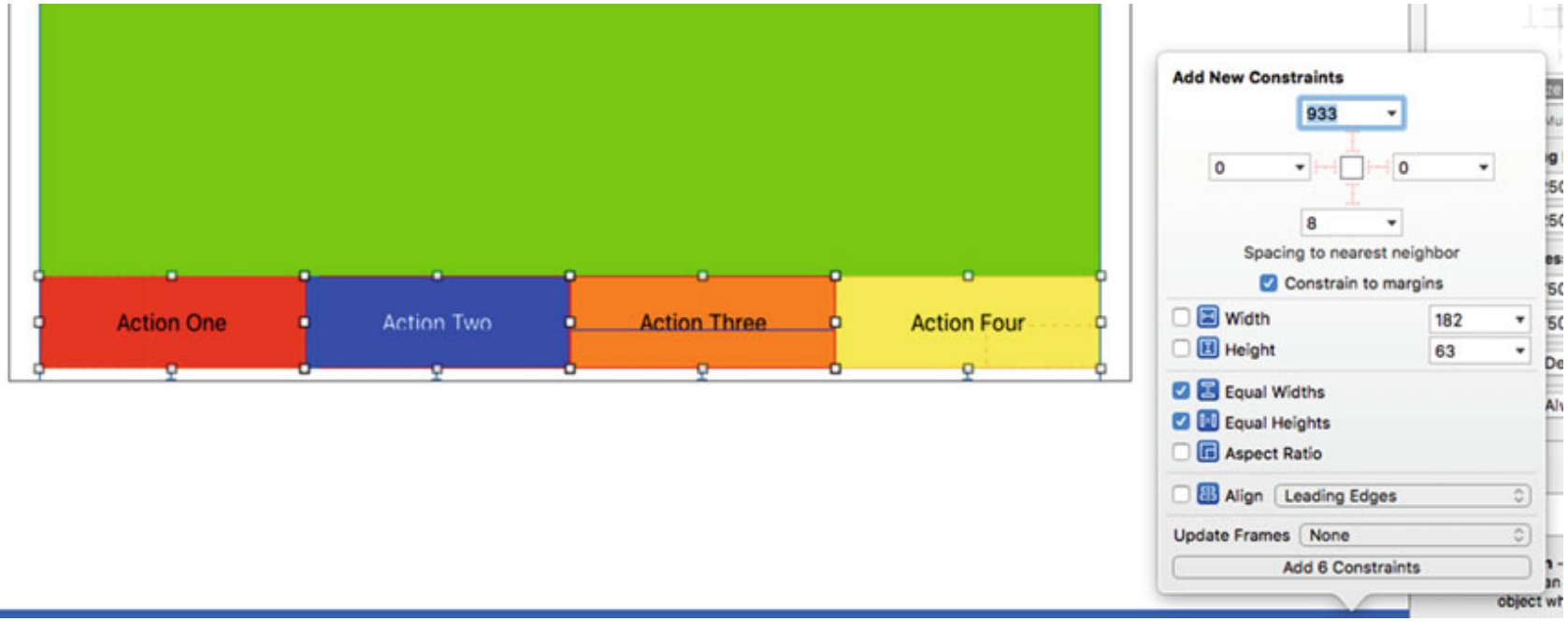


Figure 5-59. Shift-select all four action buttons along the bottom row and set to equal height and width

Setting the iPad (iPhone Plus Landscape) (wR hR) Configurations (continued)

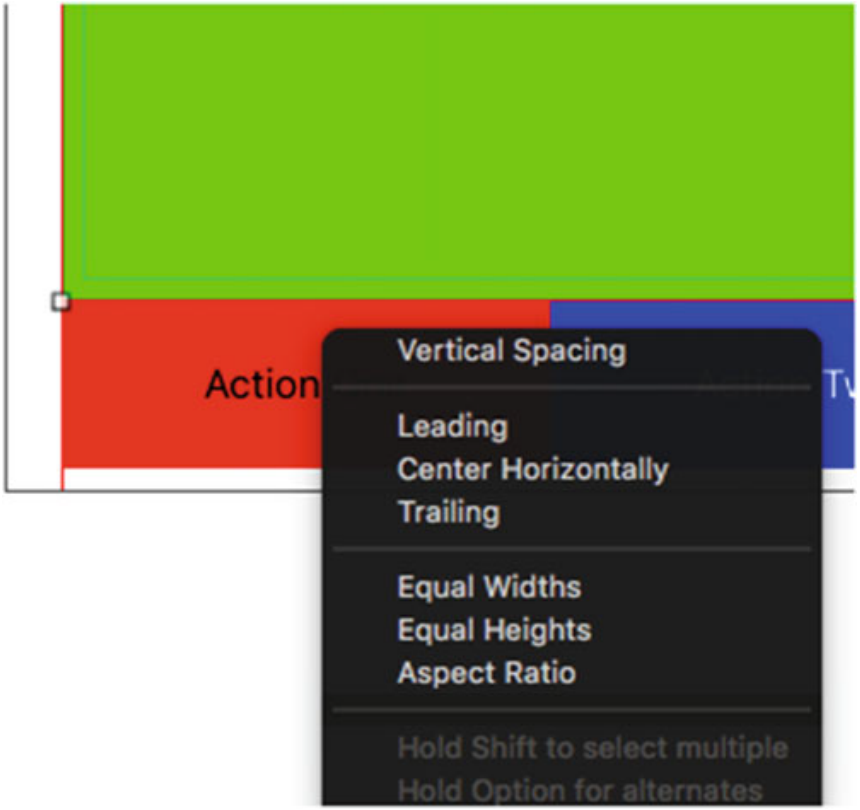


Figure 5-60. Click-drag from the green view to the Action One button and set the Vertical Spacing. This forces the green view to sit against the row of buttons

Setting the iPad (iPhone Plus Landscape) (wR hR) Configurations (continued)

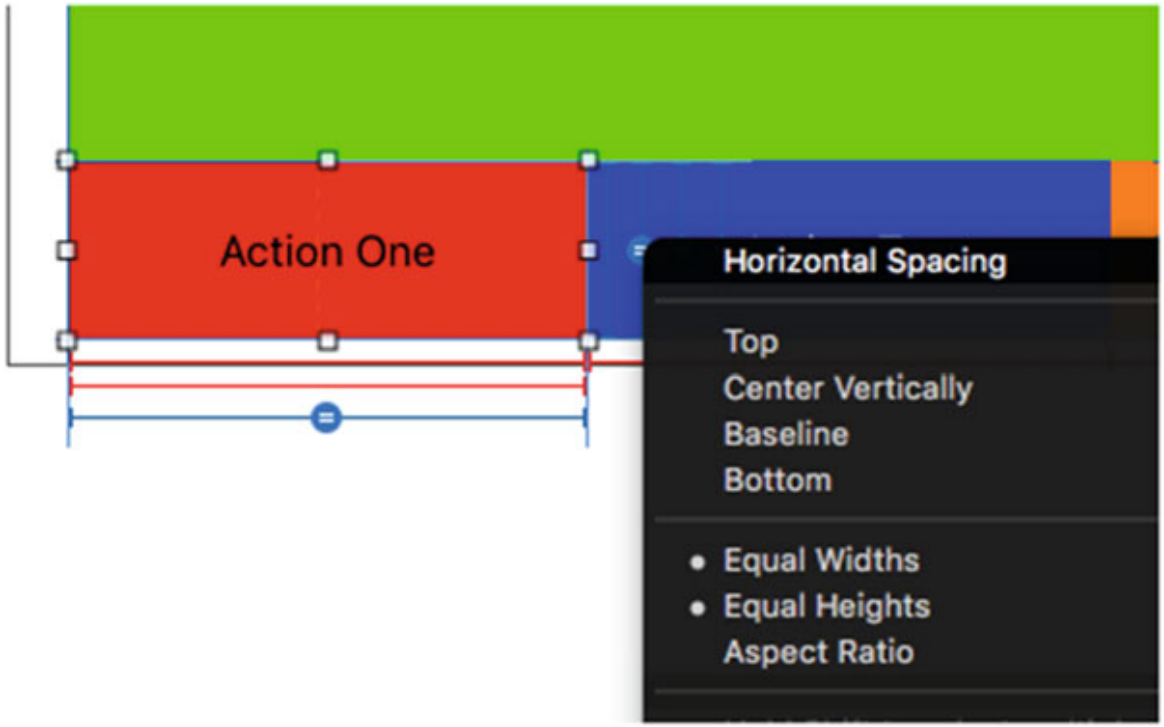


Figure 5-61. Click-drag from Action One to Action Two setting the Horizontal Spacing. Repeat for Action Two to Action Three and Action Three to Action Four

Setting the iPad (iPhone Plus Landscape) (wR hR) Configurations (continued)

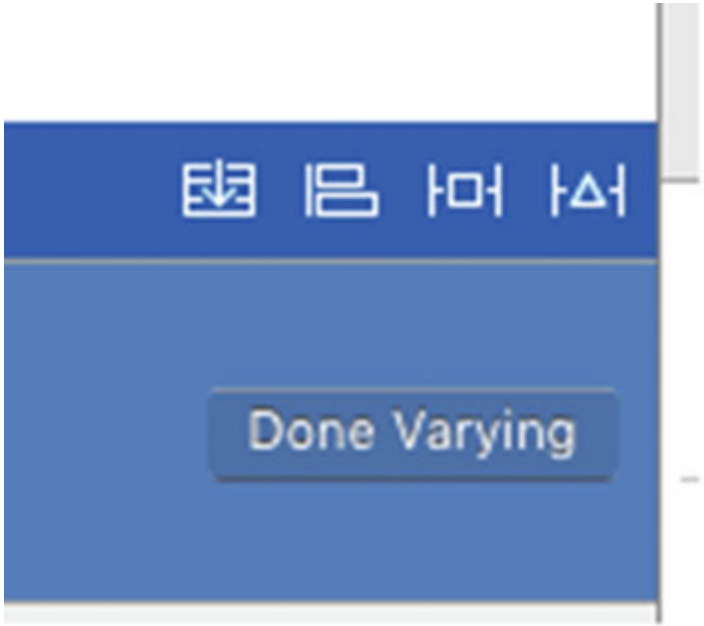


Figure 5-62. Click the Done Varying button to end the modifications for this set of traits