Auto Layout

What is Auto Layout and how is it related to Constraints?

"Auto Layout is a system that lets you lay out your app's user interface by creating a mathematical description of the relationships between the elements. You define these relationships in terms of constraints either on individual elements, or between sets of elements. Using Auto Layout, you can create a dynamic and versatile interface that responds appropriately to changes in screen size, device orientation, and localization." [iOS Developer Library, AutoLayout Guide]

To help us visualize that verbose definition, we created a new XCode project called "Autolayout".

Size Classes

Notice at the bottom of the storyboard, clicking on wAny hAny, pulls up this nifty tool. Scrolling around on it show different combinations of Width and Height. Both Width and Height have 3 possible values--Compact, Regular, or Any giving us a total 9 total Size Class options. "A size class identifies a relative amount of display space for the height and for the width" (Developer's Guide). The reason we have all these width and height combinations is to accommodate for all the different device sizes and orientations that are available on all the devices (iphones, ipads).

Compact and Regular

In the example below, if we choose a Size Class of *Regular: Compact, Height:Compact* the description says "For 3.5-inch, 4-inch, and 4.7-inch iPhones in landscape". This means layouts created in this Size Class apply to iphone 4,5, and 6 in the landscape orientation.

Compact Width | Compact Height

Final Values
For 3.5-inch, 4-inch, and 4.7-inch
iPhones in landscape

w Any h Any
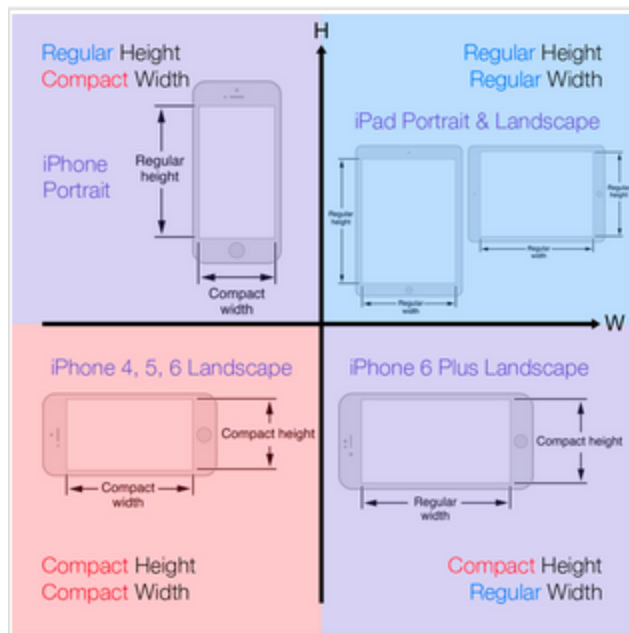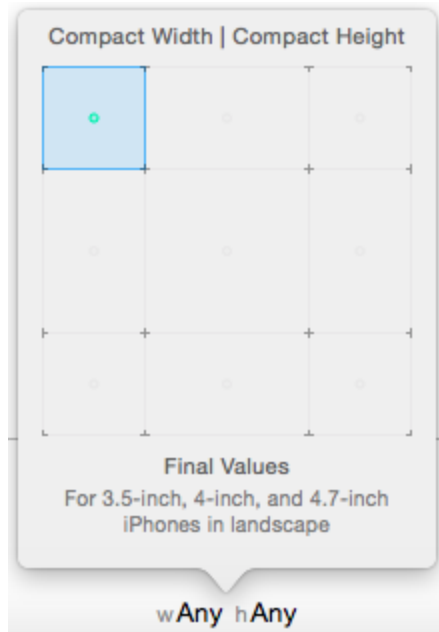
This website has a great visual representation (see image above) of the different combinations Size Classes and which devices they relate to:
http://www.jessesquires.com/adaptive-user-interfaces/

Let's play around with some examples using different Size Classes.

First, the Storyboard in the new project, change the Size Class to *wCompact hCompact*, add a button, and set the button's background to be orange:
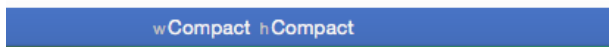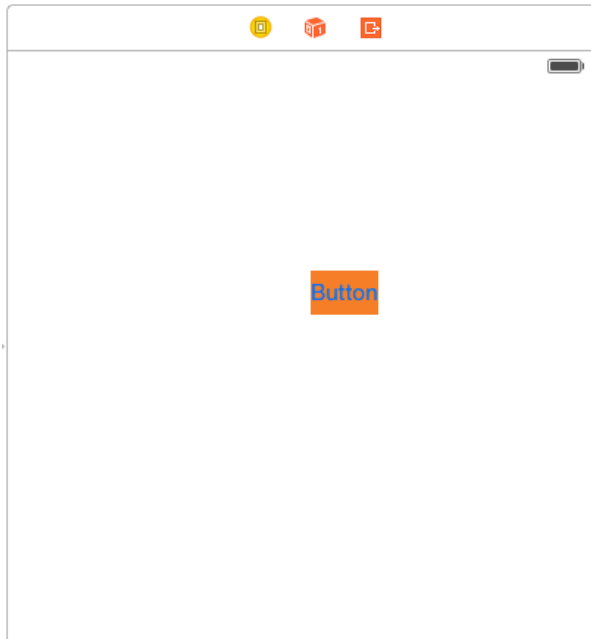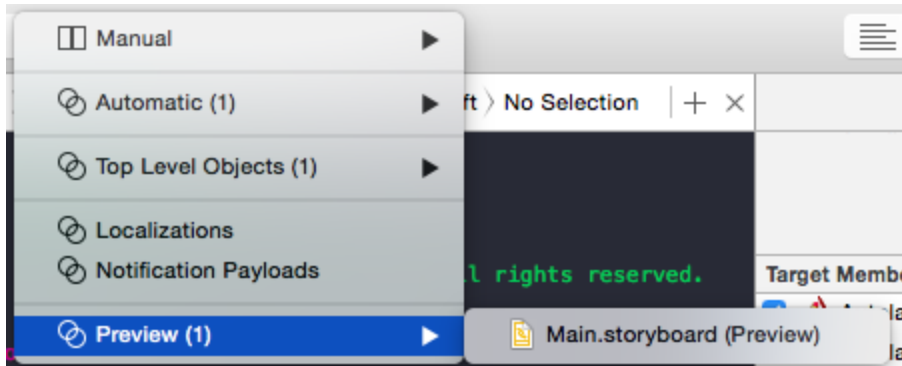

Using Preview

Instead of running the simulator ( ), we can check out what our device will look like in Preview. To access Preview, click the Assistant Editor button ( ) , this will open up the ViewController.swift file. In the toolbar above the file, click on "Automatic"
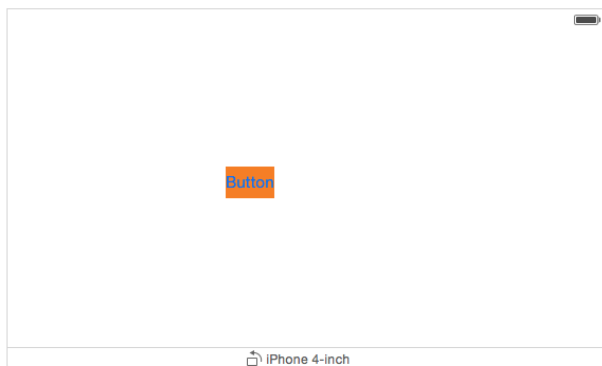

then >Preview>Main.storyboard (Preview)

This will open up the Preview of an iPhone 4-inch device. Notice that our orange button doesn't appear when the device is in portrait orientation, but if we rotate it by click the rotate symbol:
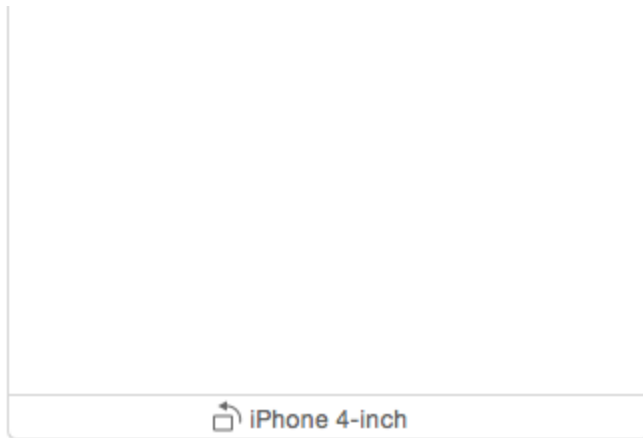
iPhone 4-inch

Our orange button shows up!

Button

iPhone 4-inch

We can add in more devices to preview by clicking the "+" in the left-hand corner of the Preview window:

```
iPhone 3.5-inch
iPhone 4-inch
iPhone 4.7-inch
iPhone 5.5-inch
iPad

+
```
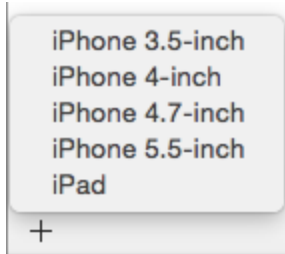
If we preview the 3.5-inch, 4-inch, and 4.7-inch iPhones and rotate them to landscape, our orange button appears as we'd expected because the layout in Storyboard was created for *width:Compact, height:Compact.*

Well this looks great, but does this mean we have to create a different layout for every device/orientation combination? Thankfully not!

wAny, hAny - "One Size Class to rule them all!"

In addition to the *Compact* and *Regular* values, remember we had the width and height can have the value of *Any.* The value *Any* allows us to design "abstractly" so that whatever we create on this w:*Any,*h: *Any* layout will appear in all of the other layouts. If we have specific layouts we need to account for, we can that with the *Compact* and *Regular* values.

From the Developer's guide, "Because much of the layout of an app does not need to change for any available screen size, there is an additional value, *Any*...To create a view controller that uses size classes, begin by laying out your design abstractly—a height and width size class of any. To design for more specific available areas, choose appropriate size classes."

## Demo 2 - Aligning 2 Buttons with Constraints

Now with our Size Class set to *w:Any, h:Any*, let's add in a second button and call it Button 2 and make its background color black. We can drop it anywhere on the storyboard View.

Our goal is to use constraints to pin Button 2 to be vertically, center aligned above Button 1. First ctrl + click from Button 1 to Button 2 and get this list of options. "Top, Center Y, Baseline, Bottom, Left, Center X, and Right" alignments are akin to left justify, center justify, right justify, etc we find when working in a Word document.
For our case, we need "Vertical Spacing" and "Center X", so hold Shift and select both of these.
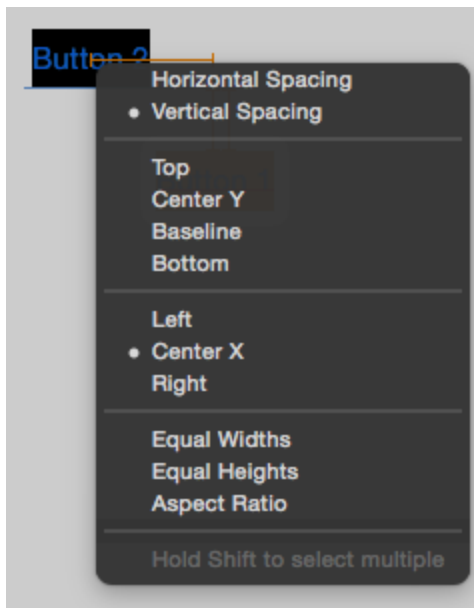

In storyboard, highlight the constraint and open the Size Inspector ( ) in the right panel.
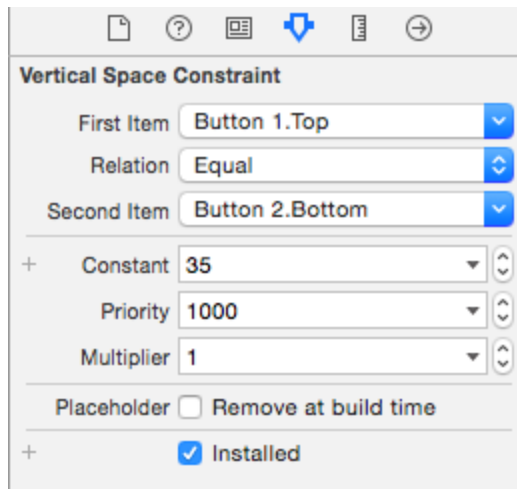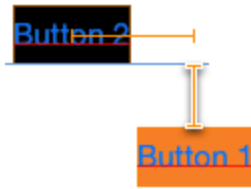
First off, let's discuss the "Constant" value. Think back to algebra when we worked with the equation y=mx+b, where b is the constant value we use to offset the line we are graphing. In our case, because the elements on the storyboard already have an offset, XCode automatically adds this Constant value to adjust for this when the application actually runs.

Let's change the Constant value for the Vertical Spacing to be 40 instead of what's been automatically assigned.

Now let's check out the Center X Constraint. Instead of using the same method above (where we clicked on the constraint in storyboard and viewed its value in the Size Inspector) let's just click on Button 2 in storyboard and view the constraints associated with it in the Size Inspector.
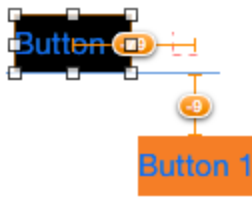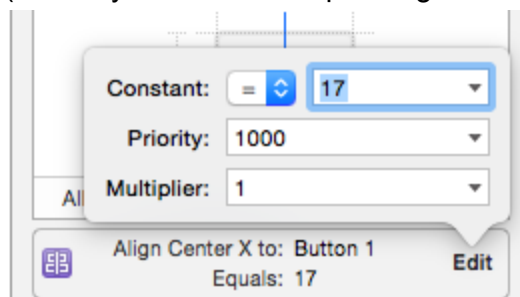
highlight Button 2
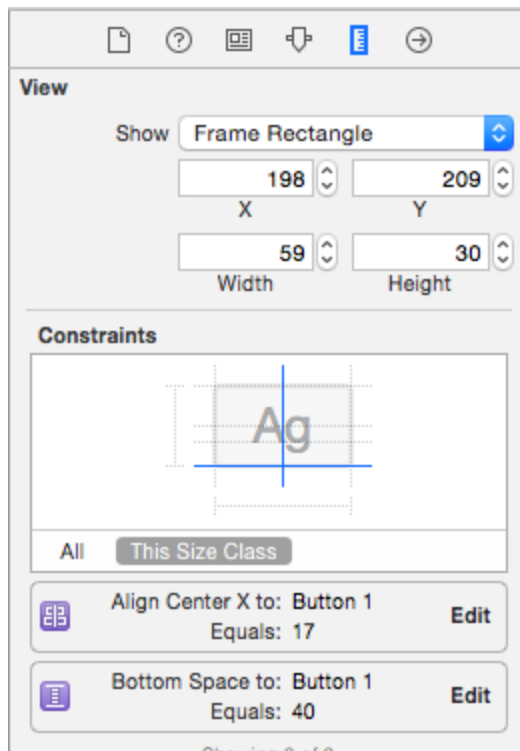
*view associated constraints in Size Inspector*

We are interested in Align Center X. When we click Edit, we see that it has a constant of 17 (this may be different depending on where your Button 2 was initially dropped off).

Let's change it to 0 because we don't want an offset.

Once both the Vertical and Center X Constraints have been set, we see that Button 2 appears about Button 1 as we expect in Preview. Hooray!:
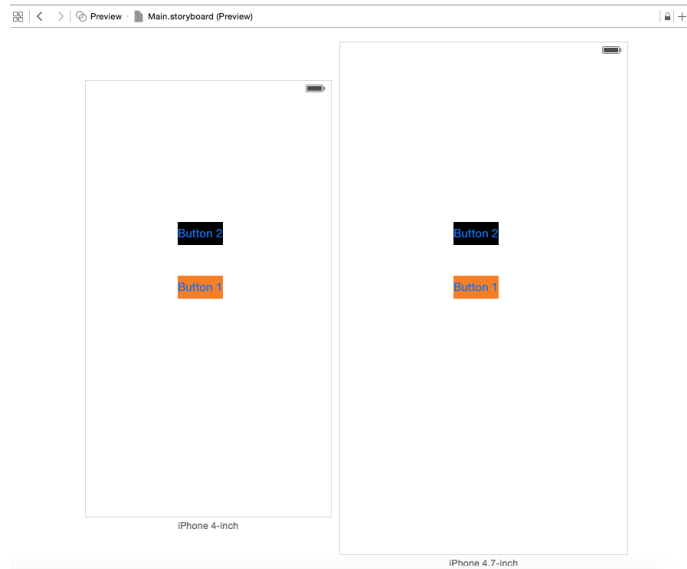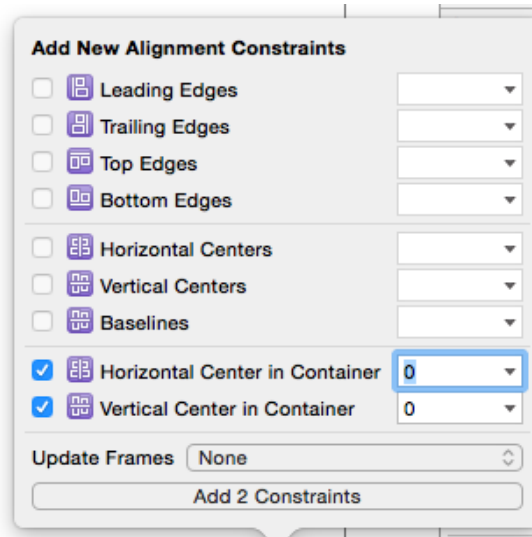

Alternative Ways to Add Constraints

It's important to remember that there are multiple ways to do the same thing in XCode. Try a couple of them and see which one you like best. This applies to adding constraints. I showed above how to add them via *ctrl + dragging* from one element to another, select the required constraints, and modifying the constants in the Size Inspector.

Alternatively, we can use the Autolayout menu (bottom-right corner of the storyboard)
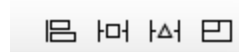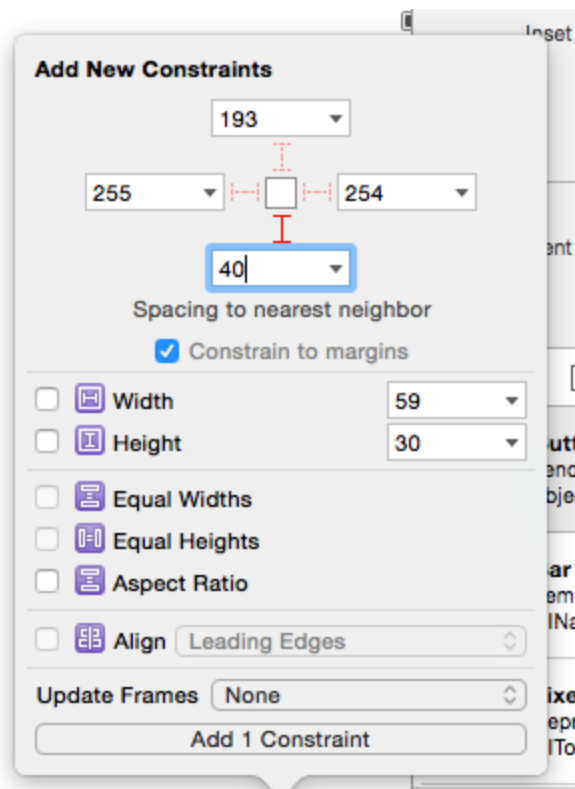

These buttons are called Align, Pin, Resolve Auto Layout Issues, Resizing Behavior, respectively. Let's create the same constraints using these buttons. Be sure to delete all the constraints we made before moving on.

Add New Alignment Constraints

☐ ▣ Leading Edges
☐ ▣ Trailing Edges
☐ ▣ Top Edges
☐ ▣ Bottom Edges

☐ ▣ Horizontal Centers
☐ ▣ Vertical Centers
☐ ▣ Baselines

☑ ▣ Horizontal Center in Container　0 ▼
☑ ▣ Vertical Center in Container　0 ▼

Update Frames　None ↕

Add 2 Constraints

Button 1

First, let's select Button 1 and click the Align button. We will select both "Horizontal Center in Container" and "Vertical Center in Container". We'll keep these values to the right (the offset values) to 0. As we select different constraints, they will be tallied within the button at the bottom of the pop-up window. We should have 2 constraints in queue. Click the button to add them.

Add New Constraints

193 ▼

255 ▼ ⌁ 254 ▼

40 ▼

Spacing to nearest neighbor
☑ Constrain to margins

☐ ▣ Width　59 ▼
☐ ▣ Height　30 ▼

☐ ▣ Equal Widths
☐ ▣ Equal Heights
☐ ▣ Aspect Ratio

☐ ▣ Align　Leading Edges ↕

Update Frames　None ↕

Add 1 Constraint

Button 2

Button 1

Next, using the dashed blue lines, position Button 2 above Button 1. Select the Pin icon and select the bottom I-beam and change the value to 40. This will set the vertical spacing between Button 2 and its nearest neighbor beneath (Button 1) to 40 points. It is important to notice that the only constraints that will be added are those I-beams which are not dashed. So these other values can be ignored, as they are not being added as constraints. Press Add 1 Constraint.
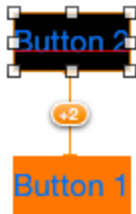

At this point you will most likely see a solid orange line with some value in the middle. This is a warning that Xcode will display if our frame is not yet updated. We'll fix that in a little bit.
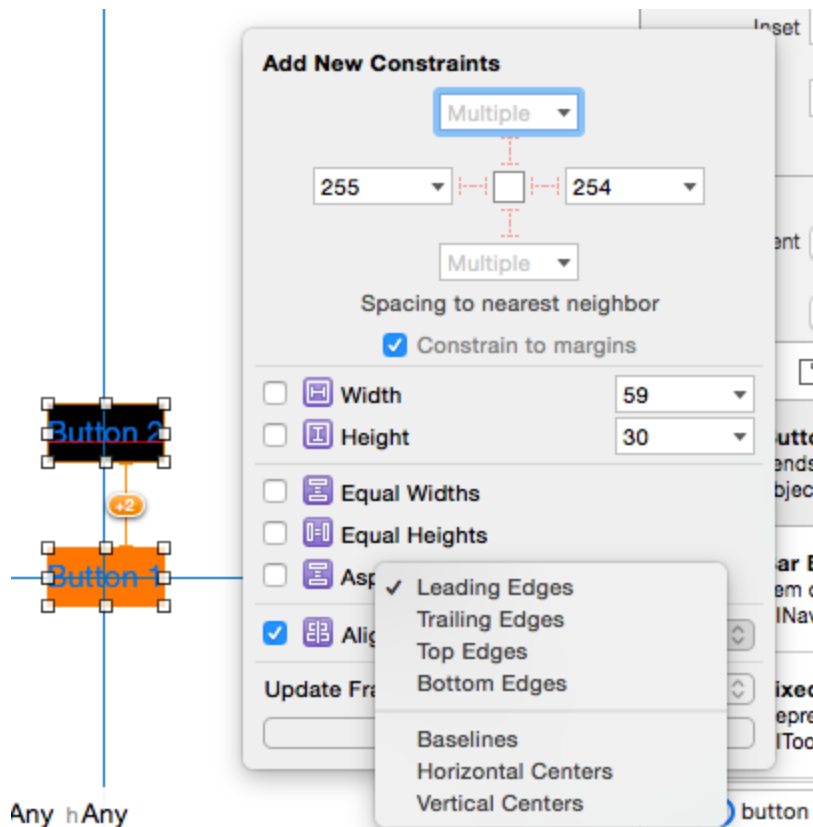

Next, select both buttons, as we want to align them in relation to each other. Click on the Pin icon again, check Align, and choose Horizontal Centers (you may need to scroll down to see this option. Add the constraint. This will find the horizontal center (along the x-axis) of each button, and align those 2 points. Add the constraint.
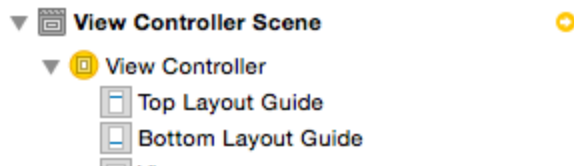
Now let's take care of that warning. There are three ways to do this

.
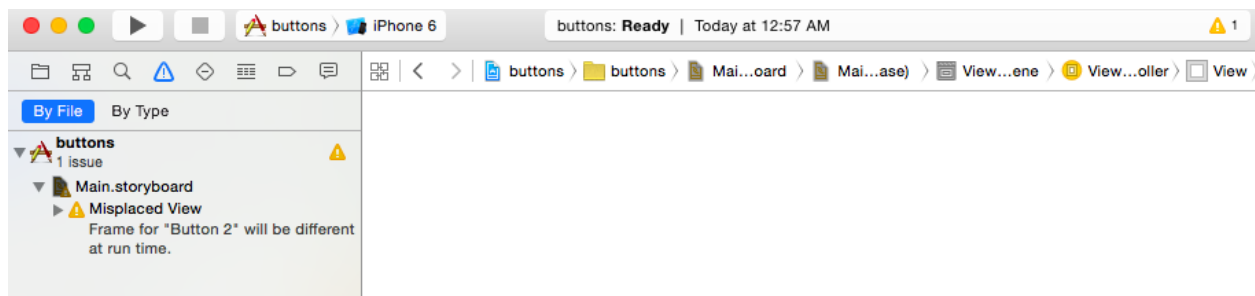
First way: Click the yellow triangle at the top of the screen. This will bring up the issues navigator, which will give you a hint as to what is going on.
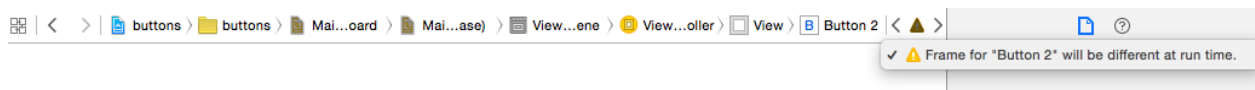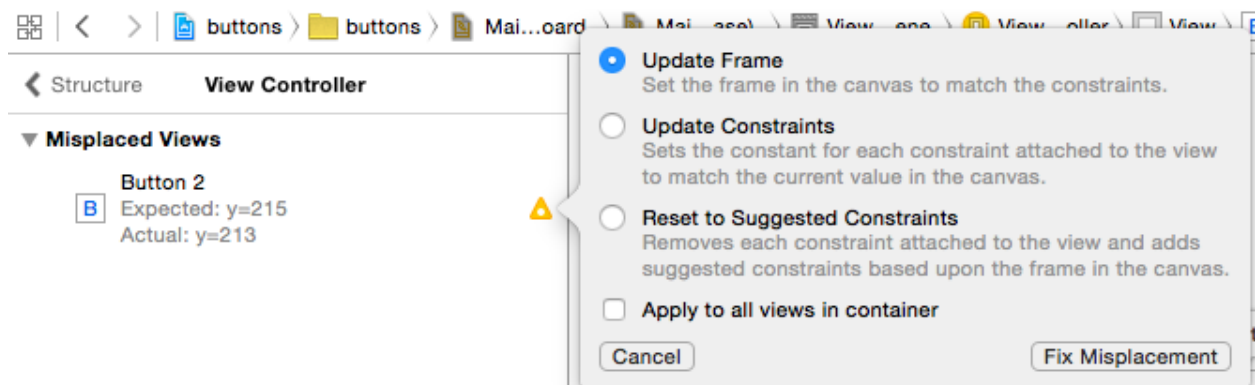

Second way: Click the yellow triangle at the far right of the breadcrumb trail of the main storyboard.


Third way: Open up the document outline and click the yellow circle at the top of it. This third way will end up giving us lots of information and ways to handle it.

Click on the yellow triangle to the right of the warning for a few options.

Let's talk about these three options. First is Update Frame. Think of this as a refresh button on your browser. This will simply have Xcode look at the constraints you have made, and then put the items into their proper places on the storyboard. The second option, Update Constraints, looks at where the items are in the view, and will alter the constants of the constraints you have already made so what is on your device will look similar to the storyboard. The third option is to Reset to Suggested Constraints. This one is fun. It will delete all your constraints and create new constraints to try interpret what you are trying to do on the storyboard. Sometimes Xcode will do a good job with this, other times it won't. Finally, if you

have multiple items, clicking "Apply to all views in view container" will apply the selected method you have chosen to all items in the view container.
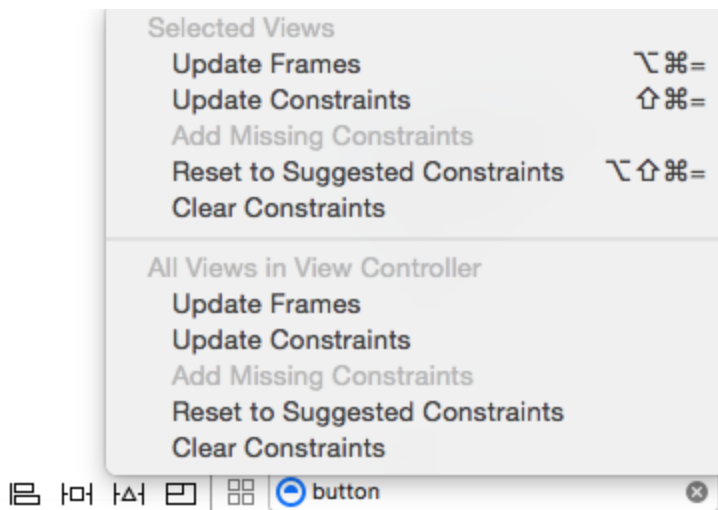
There is yet another way to fix these issues. This is by selecting Resolve Auto Layout Issues icon at the bottom of the main storyboard. It is the icon that has a triangle surrounded on either side by a line.

Conclusion

We hope this exposure to auto layout has been helpful. We've only scratched the surface. The best way to become comfortable with auto layout and constraints is to get your hands dirty and practice. It can be a powerful tool in your iOS programming toolbox.

Extra Exercise:

We have one more demo of constraints that may be viewed here