Size Classes With Xcode 6: One Storyboard For All Sizes

June 16, 2014

My favorite feature of Xcode 6 so far is the new size classes concept in Interface Builder. Size classes are Apple's solution to the question "How can I easily work with so many different screen sizes and device orientations?" They also enable the condensing of universal apps into one storyboard file. Combined with the new adaptive view controllers, it's easier then ever to rely on interface builder, rather than fight with it, to simplify the creation of your app layouts.

Every view controller in your app gets a trait collection object. This trait collection object has 2 size classes, a horizontal size class and a vertical size class. And each of these classes has 3 possible values: compact, regular, or any. These values can change based on the device and its orientation. Your app will layout its interface for each view controller based on the current size classes. Apple uses a grid to let the user choose which configuration to work in, so here's that same grid but with each device+orientation highlighted in its corresponding size class combo:

John Clem

RT @rwenderlich: @GerardHenninger Yes, we have a cool OS X series by @johnnyclem in the pipeline, stay tuned! :]

3 days ago

John Clem

One interesting thing to note here: iPad has regular x regular for both landscape and portrait. Straight from Apple's 'What's new in iOS8 guide' :

"With the amount of screen space available, the iPad has a regular size class in the vertical and horizontal directions in both portrait and landscape orientations."

— Dr. Dre*

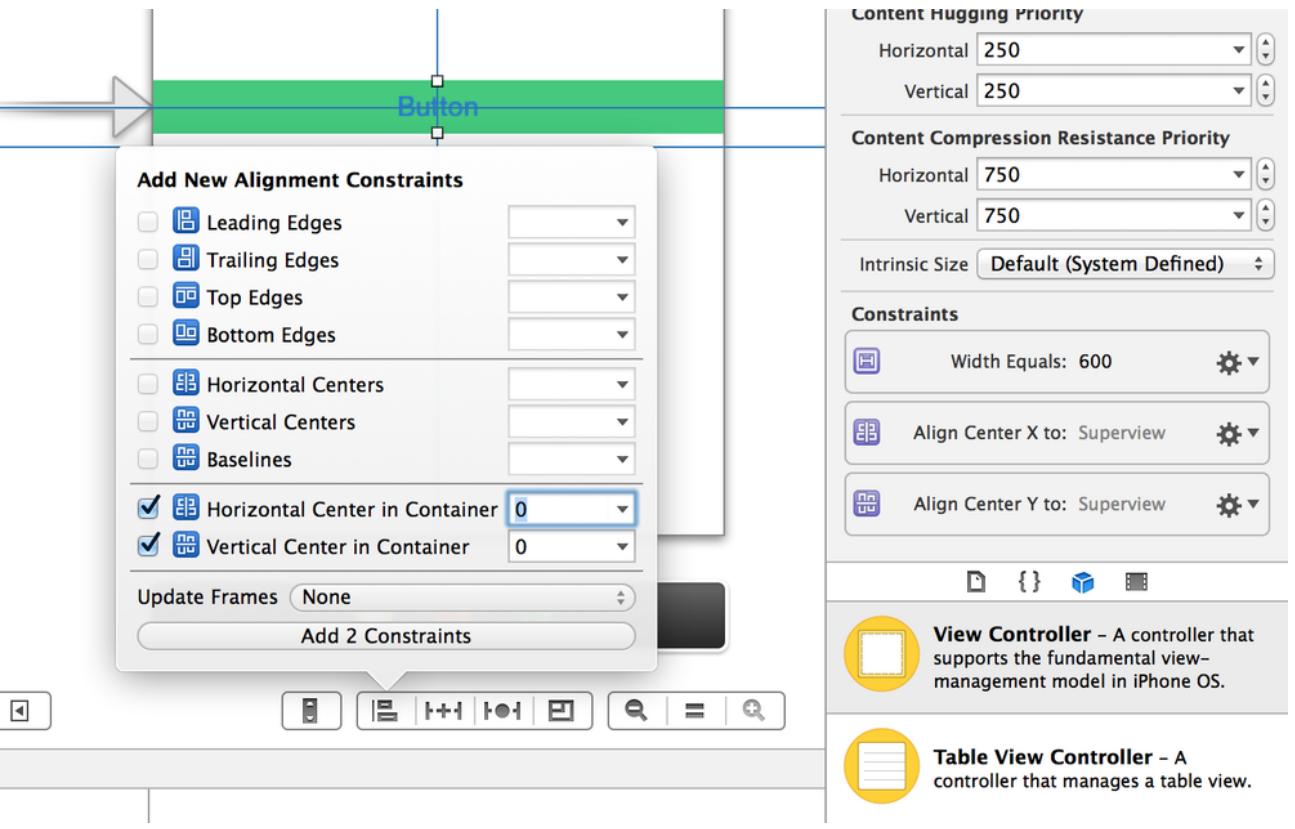
Ok, enough by-the-books stuff, lets open up Xcode 6 and try it out for ourselves. Due to Apple's NDA on Xcode 6 screenshots, the screenshots provided below are actually from Xcode 5, so keep in mind my images are more just a general guide of what screen to be on.

Create a new universal project for this demo. If you want to try it out in an existing Xcode 6 project, then you might have to explicitly enable size classes. You can do this in Interface Builder in the file inspector, right below the enable autolayout checkmark.

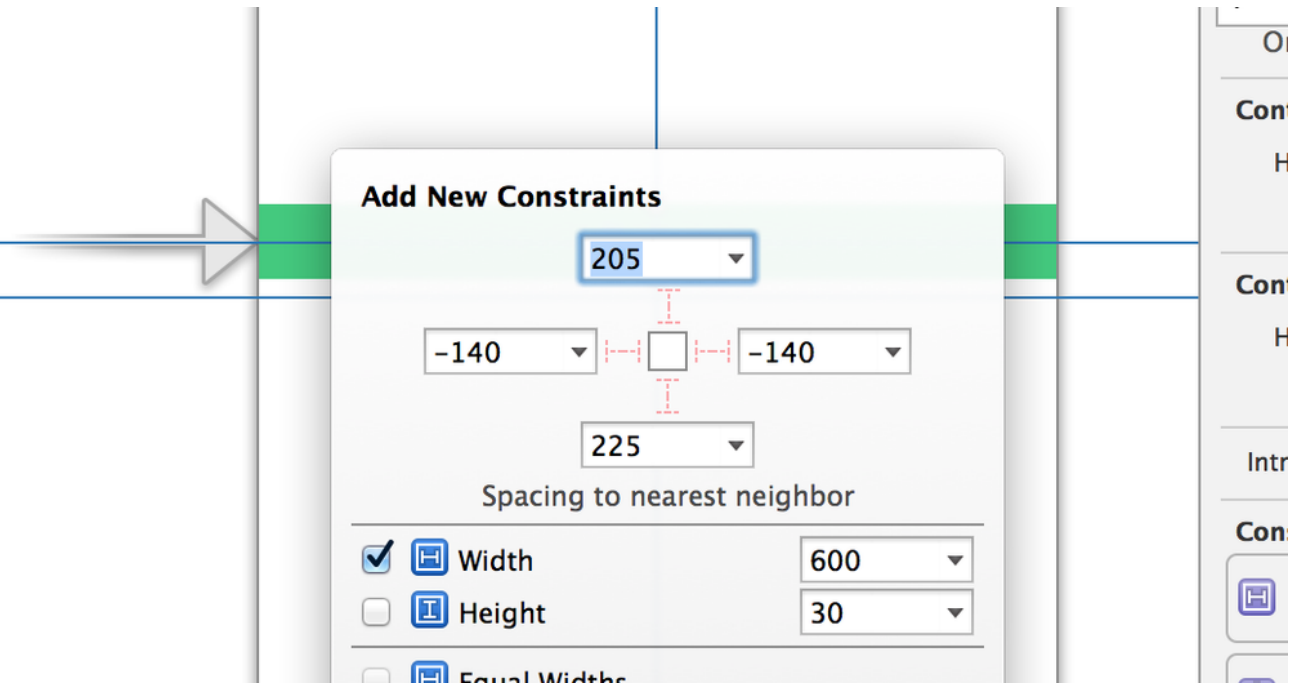
First, let's look at the size class grid in Xcode. This is the area where you can switch your storyboard between the different layout combinations. While viewing your storyboard, look towards the bottom and Click on the label that says 'wAny hAny. You will see something that looks like the grid from the top of this post.

By default, we start in our base configuration of any width and any height. Think of this as your master interface, things setup and changed here will be used by both the iPhone and iPad layout by default for all orientations. Apple recommends

doing most of the interface work in this configuration, simply because it would mean less work for you. Let's layout a super-wide button in the center of the screen to play with. Give it a green background color so we can see its true size, center it vertically and horizontally with constraints:



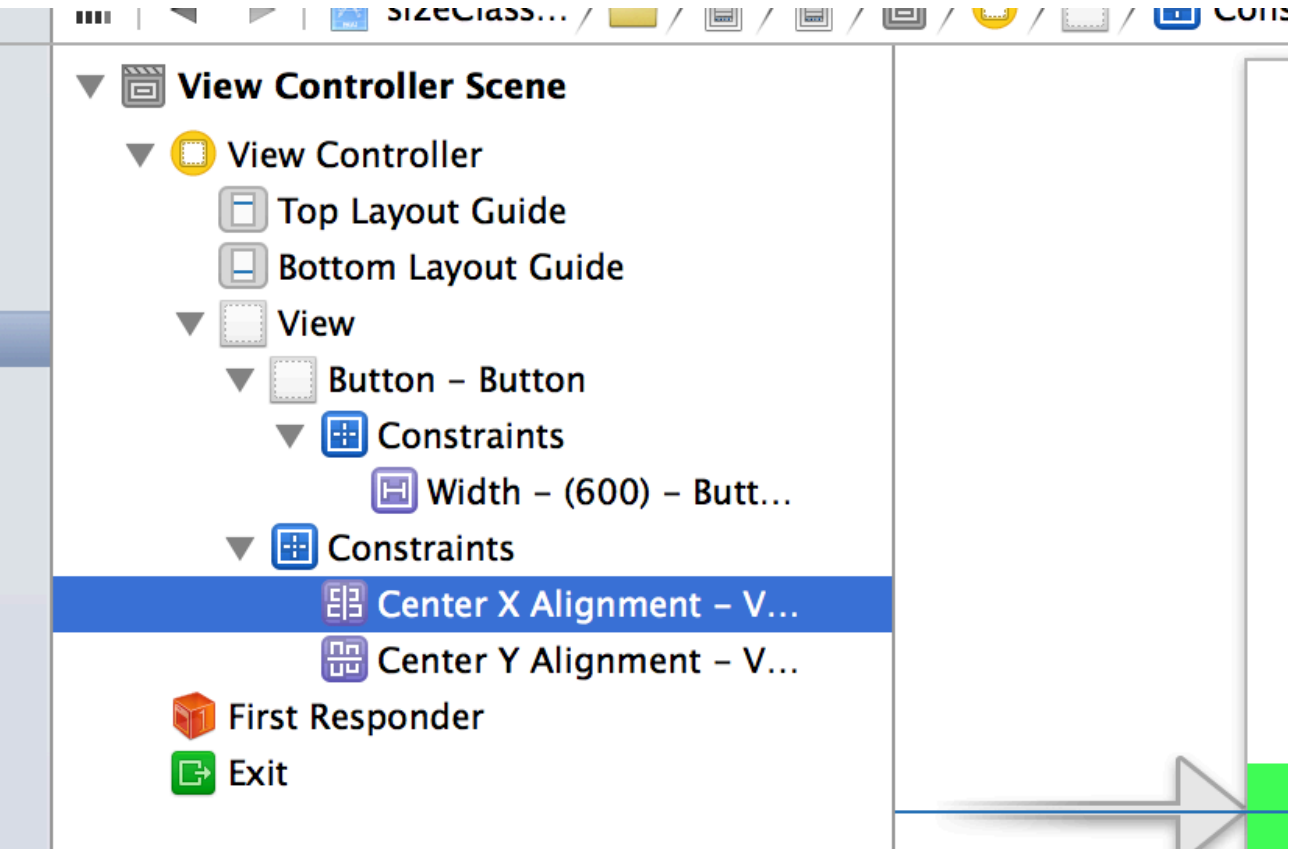
and give it a ridiculous fixed width of 600:



Now run the app in both iPad and iPhone simulators. You will see it centered on both, but its clearly too wide for iPhone in both orientations. Let's use size classes to fix that. Go back to the grid and select the iPhone portrait config, compact width + regular height. Thats the red rectangle from the grid :

You will notice that the bar you click on to access the grid changed from white to blue. This is basically just warning you: "Hey you're not in the base configuration anymore! Some changes you make here will only show when your app is running with these specific size classes. So now this bar is blue!" I say 'some' changes because there are specifically 4 things you can change between size classes in interface builder: 1) constraint constants 2) fonts 3) turning constraints on/off 4)turning subviews on/off.

The first two are pretty self explanatory, but let me show you how the last 2 work. Lets try turning a constraint off for the current size class we are in (compact width and regular height). In the document outline, click on the Center X Alignment constraint we setup on our button:



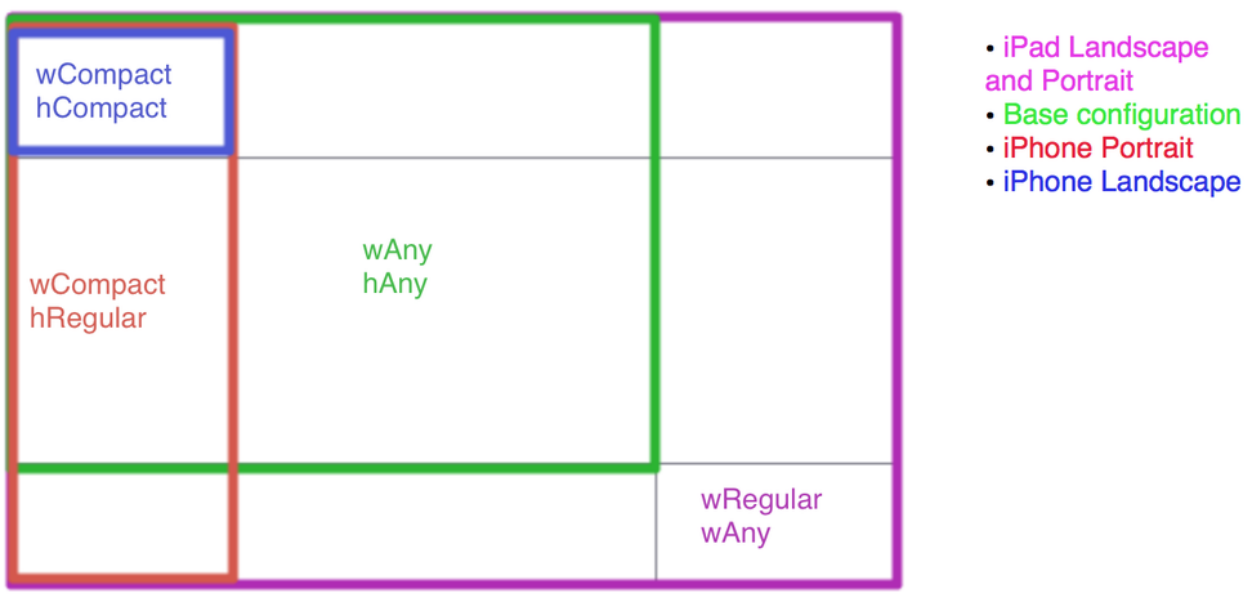
Now look in the attributes inspector, towards the bottom you will see the word installed with a checkmark next to it, and then a plus sign to the left of that. Click the plus sign and click 'Compact Width | Regular Height (current). You will now see 2 check marked items. Uncheck the one we just added (wC hR).

Now our constraint is no longer installed and doing anything for this configuration of size classes. As you can see, Xcode is complaining that our constraints are too ambiguous, and if you run the app on an iPhone sim the button is no longer centered on the X axis. But running it on the iPad shows it still centered, because that constraint is still installed on our base configuration. It will be installed on every configuration except the one we just unchecked. You can even rotate your iPhone sim and the button will magically go back to being centered, since iPhone landscape is a different size class config. Ok lets put the checkmark back in so the button goes back to being centered.

Now lets change the constant we set for the buttons width. Select the button itself, and go to the Size Inspector and scroll to the bottom where all the constraints are. Click the Width Equals one and set it to 100:

Run the app on the iPhone sim now, and you will see the button has the correct width for portrait iPhone. Running on the iPad sim shows the button 600 wide since we didn't change the width on our base config. However, it still doesn't

look good on iPhone landscape, because iPhone landscape is still pulling from our base Any x Any configuration. Lets fix that. In the grid, select compact width and compact height. Thats the blue rectangle in our grid:



Now change the width constant for this configuration the same way we did for compact x regular. Give it a width of 400. Running the app in iPhone and rotating to landscape causes the button to take a 400 width, which looks great (for our learning purposes). Whats nice is you can see a list of all the constants for each constraint for each different configuration. Just select the constraint you want to look at in the document outline, then go its its attribute inspector and they are all listed neatly below the original constant. It labels each one based on which configuration it applied on.

What if we decided we wanted the button to disappear only when the iPhone was in landscape mode? Well with size classes we can uninstall views just like we uninstalled a constraint. Select our UIButton, scroll down to the very bottom of our attributes inspector, and add a new installed option for our current configuration by hitting the plus button, and then uncheck it.

As you can see, the view instantly disappears from the storyboard because we uninstalled it on the configuration we

are currently looking at. Running the app, you can see its gone in portrait iPhone, but it comes back when you rotate to landscape. And it will still be installed on the iPad since our iPad still uses the base configuration.

The last thing I want to show is how we can be notified in code when these configurations are changed by something (usually a rotation). Sizes classes are meant to replace the use of UIInterfaceOrientation & UIUserInterfaceIdiom, so its important to understand how to work with them in your code. The first you will need to do is set your view controller(s) to conform to the UITraitEnviroment protocol:

```
class ViewController: UIViewController, UITr
aitEnvironment {

 override func viewDidLoad(){
  //code
 }
}
```

This protocol has a required method that notifies your view controller when the trait collection has changed:

```
    override func traitCollectionDidChange(p
reviousTraitCollection: UITraitCollection!)
{
        if previousTraitCollection? {
          //print out the previousTrait's in
fo
        println(previousTraitCollection)
        }
    }
```

A UITraitCollection provides details about the attributes of a view controller. Heres what printing them out shows:

```
<UITraitCollection: 0xb4072a0;
                    _UITraitNameUserInterface
Idiom = 0,
                    _UITraitNameDisplayScale
 = 2.000000,
                    _UITraitNameTouchLevel =
 0,
                    _UITraitNameInteractionM
odel = 1,
                    _UITraitNameHorizontalSi
zeClass = 1,
                    _UITraitNameVerticalSize
Class = 2>
```

Those last two describe the vertical and horizontal size class of the trait collection we just changed from. Heres how that enum looks:

```
enum UIUserInterfaceSizeClass : Int {
    case Unspecified
    case Compact
    case Regular
}
```

Hopefully this has helped you get a basic understanding of size classes and how they can greatly simplify your work in interface builder. As always, we've uploaded the code from this tutorial to Github. Let us know if you have any questions

or input in the comments below.

*probably not Dr. Dre

Brad Johnson 10 Comments

7 Likes Share

SWIFT IN SEATTLE

USING NSNOTIFICATIONCENTER IN

COMMENTS (10) Newest First Subscribe via e-mail

Preview Post Comment…

K Hopp 12 hours ago

Nicely done! Thank you for this great introduction!

Ubo 23 hours ago

Great job !

sravan 3 weeks ago

Its very helpful and awesome tutorial who learning size classes concept.

Andrew A month ago

Awesome tutorial. Thanks very much for posting this. I guess I'll need to use Storyboards more now but I prefer to do almost everything in code. It just seems much easier with the visual tools.

Rajesh 4 months ago

Great tutorial on Size Classes, Thanks!

Brian Fritz 4 months ago

How can I accomplish the same kind of thing on an iPad? When the device is rotated I want to change how my app is laid out. However, since an iPad is defined as having a regular height and regular width the events for traitCollectionDidChange do not get fired when the device is rotated.

Brad Johnson 4 months ago

Hey Brian,

In your view controller, you can override this

method:

func viewWillTransitionToSize(size: CGSize, withTransitionCoordinator coordinator: UIViewControllerTransitionCoordinator!)

and then based on the size parameter passed in you can make your changes in code.

In addition, you could try making your landscape iPad layout in the wRegular hCompact configuration, and then manually setting your view controller's trait collection size to wRegular hCompact upon detection of landscape orientation in the method I mentioned above. To manually set a view controllers trait collection, use this method in its parent view controller:

func setOverrideTraitCollection(collection: UITraitCollection!, forChildViewController childViewController: UIViewController!)

I've only seen this done with a split view controller, but let us know if you try it and how it works.

Piotr A month ago

> How can I use setOverrideTraitCollection method to change trait collection of a root controller of a view? I've been calling it with self as childViewController but it seems to have no effect.

Petar A month ago

This is probably a good solution, however I cannot understand why this kind of functionality is not included in interface builder. Since this can be very easily achieved for the iPhone, I would assume it should have been possible for the iPad too. A bit confused here of why Apple have done it like this.

Brian Fritz 4 months ago

Brad, thanks for the reply. Yes, seems that willTransitionToSize does what I need. Thanks for pointing that out.