A Simple Alternative to Auto Layout

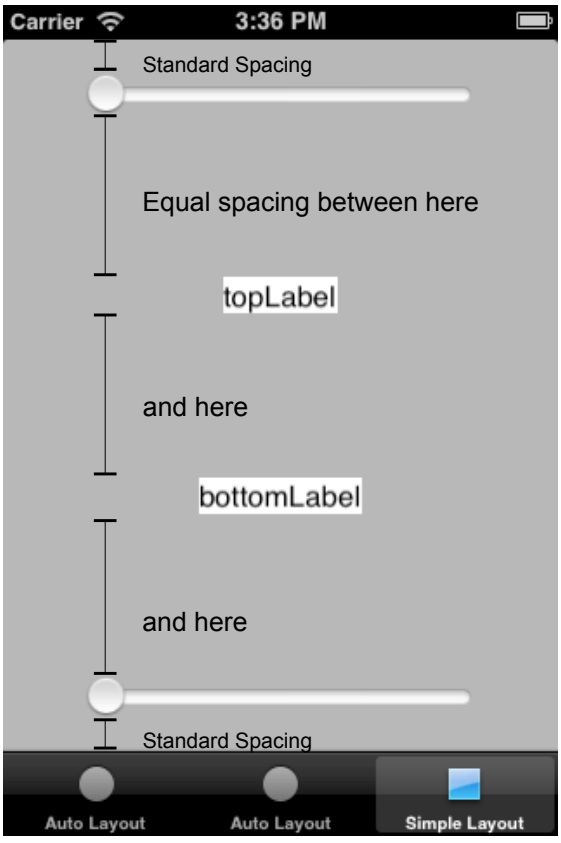
I tried to use auto layout. I really did. But I have weighed and measured it and found it wanting.

I've been working on a new project lately and thought I would explore auto layout for positioning my controls. What I found was that neither auto layout in Interface Builder, nor the visual format string method allowed me to specify the positioning I wanted. That left me with the constraintWithItem constructor. I was able to sort of get close to what I wanted with constraintWithItem, but holy cow – we're talking acres and acres of text to *almost* get the job done.

I wasn't completely satisfied with the results I got from auto layout and I certainly didn't care for the process. Everything about auto layout from its basis in rules to the sheer verbiage it takes to get anything done violates the KISS principle (Keep It Simple, Stupid).

So I began looking for a simpler solution. One that was imperative and not rules-based. And I really liked being able to refer directly to the sides, width, etc. of a view. I finally developed my own solution - a UIView category that solves the problem in a simple, yet elegant way. I call the class SimpleLayout.

As a test and example, I wrote a small demo program. The goal was to layout a simple view with the following requirements: The view should contain four controls – two slider controls and two labels. All controls should be centered horizontally. The sliders should be 70% as wide as the view. The top slider should be positioned with standard spacing from the top of the view. The bottom slider should be positioned with standard spacing from the bottom. The two labels should be positioned with equal spacing between them and between them and the sliders.



I don't think it is possible to achieve those goals with auto layout. And the layout I *was* able to achieve took 111 lines of not very transparent code.

```
-(void) setupAutoLayout {
    NSLayoutConstraint* constraint;

    // === top slider ===
    // Size the top slider 70% of the width of the view
    constraint = [NSLayoutConstraint constraintWithItem:self.topSlider
                                              attribute:NSLayoutAttributeWidth
                                              relatedBy:NSLayoutRelationEqual
                                                 toItem:self.view
                                              attribute:NSLayoutAttributeWidth
                                             multiplier:0.7
                                               constant:0.0];
    [self.view addConstraint: constraint];
    .
    .
    lots more of this...
```

SimpleLayout achieved the exact design goals and did it in just 20 lines of code. This is everything necessary to describe the layout.

```
-(void) viewWillLayoutSubviews {
    // === topSlider ===
    self.topSlider.width = self.view.width * .70;
    self.topSlider.centerX = self.view.centerX;
    self.topSlider.top = self.view.top + kSLEdgeStandardSpace;

    // === bottomSlider ===
    self.bottomSlider.width = self.view.width * .70;
    self.bottomSlider.centerX = self.view.centerX;
    self.bottomSlider.bottom = self.view.bottom - kSLEdgeStandardSpace;

    // Calculate the remaining vertical space
    CGFloat availableSpace = self.bottomSlider.top - self.topSlider.bottom;
    // Then subtract the height of the two labels
    availableSpace -= self.topLabel.height;
    availableSpace -= self.bottomLabel.height;

    // Divide the available space in three for each of the spaces.
    CGFloat oneSpace = availableSpace / 3;

    // === topLabel ===
    self.topLabel.top = self.topSlider.bottom + oneSpace;
    self.topLabel.centerX = self.view.centerX;

    // === bottomLabel ===
    self.bottomLabel.top = self.topLabel.bottom + oneSpace;
    self.bottomLabel.centerX = self.view.centerX;
}
```

The concept of SimpleLayout is very simple indeed. It simply provide setters and getters for the four sides of a view, height and width, and the x and y of center. But this allows layout code to be very descriptive with very little code.

SimpleLayout is Concise

Set a view to a specific location?

self.aView.top = 20;
self.aView.left = 20;

Set two views to be aligned on the left?

self.aView.left = self.bView.left;

SimpleLayout is Flexible

The other power in SimpleLayout lies in the ability to calculate positions and sizes based on the available space.

A UITextView should be 50% of the width and 30% of the height of the parent view.

self.textView.width = self.view.width * 0.50;
self.textView.height = self.view.height * 0.30;

Some Final Words

It's probably obvious, but your SimpleLayout code should be placed in or called from **UIViewController.ViewWillLayouSubviews** or **UIView.layoutSubViews**. SimpleLayout works when it can recalculate the layout when the size of the view changes.

UILabels, UIButtons and some other controls don't know their intrinsic size unless you call sizeToFit. See the code for how this is done.

The other tip I can offer is that you will want to execute code that changes the size of your subviews before positioning them or other subviews that depend on them. Imagine that you place subview B to the right of subview A and then increase the width of A. You might end up obscuring B.

SimpleLayout is provided with a BSD license, so you can use SimpleLayout for all projects free and commercial. But I wouldn't mind if you mentioned to people how nifty SimpleLayout is.

Getting SimpleLayout

Well that's probably all you need to be getting on with. The class and the example project are available at GitHub: **https://github.com/david-a-rogers/SimpleLayout.git**

To get started look at the SimpleLayout class: **UIView+SimpleLayout.m** and **.h** and the example code

in **SLDThirdViewController.m** and **.h**.

David A. Rogers
August 20, 2013