SimpleLayout:
A Simple Approach to iOS View Layout

SimpleLayout is a very small addition to the API already available in iOS. It makes creating dynamic layouts a small and manageable task.

The goal of layout code is to size and position the subviews of the view. Good layout code will adapt dynamically to changes in the size of the parent view.

Sizing Views

Some views know intrinsically how big they ought to be. UILabel and UIButton objects will set their frame size to accommodate their text if you call resizeToFit. For example, when the following code is executed, the topLabel UILabel will have its frame set to the proper size to contain the text assigned.

```
self.topLabel = [[[UILabel alloc] init] autorelease];
self.topLabel.text = @"topLabel";
[self.topLabel sizeToFit];
```

Slider controls know how tall they should be (assuming you're laying it out horizontally). But they don't know how wide to be. You'll have to provide that by arbitrarily choosing a width or (better yet) by setting it in your SimpleLayout code.

Other views won't know what to do for either dimension.

Q: How big should you be Mr. UITextView?
A: I dunno. How big do you want me to be?

I generally call sizeToFit at the point I create my views. Otherwise I do the rest of the view resizing as part of my layout code.

I do recommend, moreover, that you 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.

viewWillLayoutSubviews and layoutSubviews

In order to adapt dynamically, your SimpleLayout code needs to be called whenever the size of your view changes – most often when the user has rotated the device. There are really only two good options. If you have a custom UIView (and you should unless you have a very simple view), you should put your layout code in UIView.layoutSubViews. Or call it from there.

Otherwise, if you are determined to handle everything in your UIViewController, put your layout code in UIViewController.viewWillLayoutSubviews. Or call it from there.

If you want more information about why this is the best place to put orientation/layout code, see Kevin Dew's excellent page on the topic at:

http://kevindew.me/post/18579273258/where-to-progmatically-lay-out-views-in-ios-5-and

SimpleLayout

SimpleLayout is an Objective C category that extends UIView, adding getters and setters for the following (virtual) attributes:

top
bottom
left
right
centerX
centerY
width
height

These simple additions to the UIView class will allow us to quickly and easily layout any view.

Setting **view.top = 10** will position the view so that view.frame.origin.y = 10. SimpleLayout is an example of what is sometimes called "semantic sugar". You could code the same thing directly, but it wouldn't be nearly as succinct. iOS won't let you set a part of the frame. It's all or nothing. So in reality you would have to write:

```
CGRect frame = self.someLabel.frame;
frame.origin.y = 10;
self.someLabel.frame = frame;
```

That's more work to type and less readable.

To use SimpleLayout in your own code, add UIView+SimpleLayout.h and .m to your project. Import the header file in the class where you want to do your layout.

Using SimpleLayout

There are two examples using SimpleLayout in the demo code (and one using auto layout for contrast). They should be all you need to get on with. But here are a few examples to send you off to a good start. These examples all assume that you are working in UIViewController code.

We'll start off with the simplest example – positioning at a specific location.

```
self.someLabel.left = 10;
self.someLabel.top = 10;
```

This is the same thing as writing:

```
self.someLabel.left = self.view.left + 10;
self.someLabel.top = self.view.right + 10;
```

This is true because we are working within the parent view's coordinate system. So self.view.top and

self.view.left will always be 0;

The following code shows how to align two views.

```
self.otherLabel.top = 50;
self.otherLabel.left = self.someLabel.left;
```

Setting the right and bottom attributes do not affect size, only positioning. The following code only moves the right side of the label to be 20 points from the right side of the view.

```
self.someLabel.right = self.view.right - 20;
```

Making Your Layout Dynamic

These examples are useful, but you don't really utilize the power inherent in SimpleLayout until you start making your calculations dependent on the current size of the parent view. Then your layout will adapt as the size of the parent view changes.

```
self.someSlider.width = self.view.width * .70;
```

If you've placed your layout code as recommended above, it will be called every time the size of the view changes. So the slider above will stretch or shrink to be exactly 70% as wide as the parent view.

```
self.otherLabel.top = self.someLabel.bottom - (self.view.height * .08);
```

This is another example of a dynamic layout. The space between someLabel and otherLabel will stretch or shrink depending on the size of the view.

One of the things that auto layout can't do is to provide equal spacing between multiple subviews that grows or shrinks with the view. See the SimpleLayout classes in the demo project to see how that is done.

Well that's probably all you need to be getting on with.

License

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

The class and the example project are available at GitHub:

https://github.com/david-a-rogers/SimpleLayout.git

David A. Rogers
August 26, 2013