Autolayout

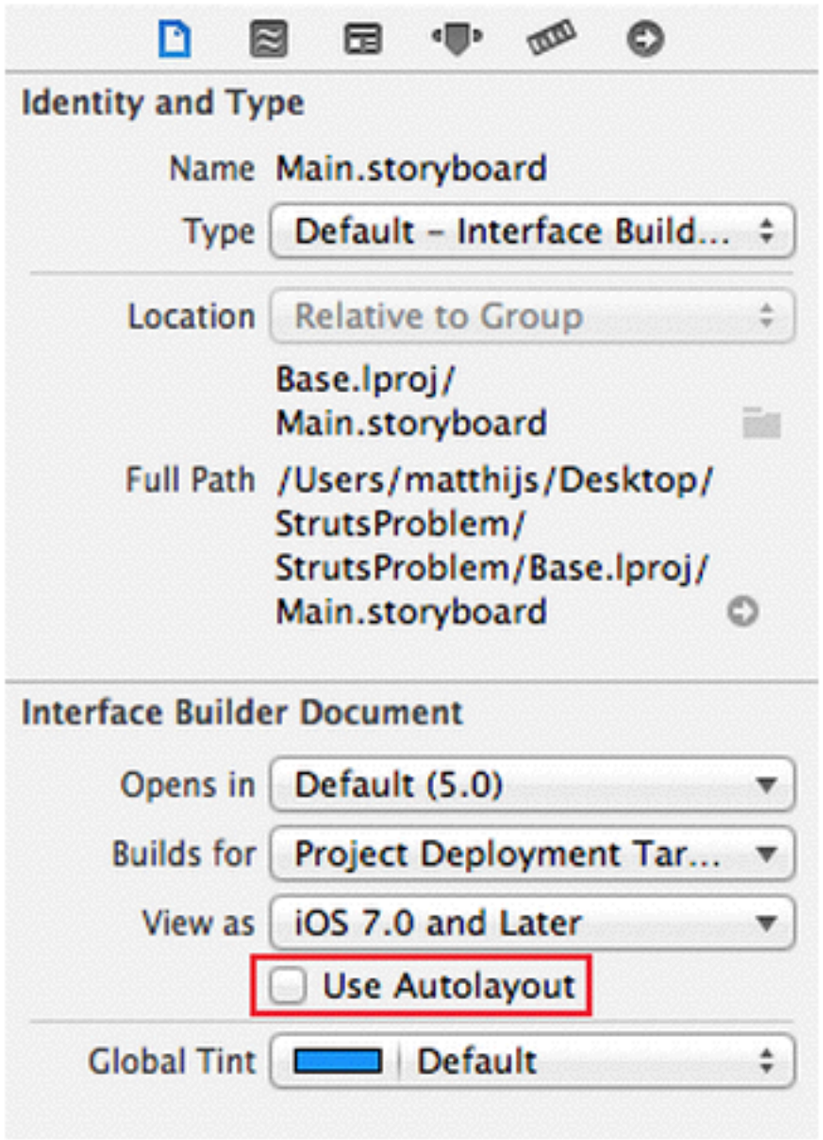
March 11, 2014

Keynote

Feature

- Support different screen sizes in your apps
- Adapt to fit into new dimensions
- no longer have to make new nibs or storyboards for every language that you wish to support
    - Includes right-to-left languages such as Hebrew or Arabic

Enable it


Identity and Type
Name Main.storyboard
Type Default – Interface Build...
Location Relative to Group
Base.lproj/
Main.storyboard
Full Path /Users/matthijs/Desktop/
StrutsProblem/
StrutsProblem/Base.lproj/
Main.storyboard
Interface Builder Document
Opens in Default (5.0)
Builds for Project Deployment Tar...
View as iOS 7.0 and Later
Use Autolayout
Global Tint Default

Key Method:

- [NSLayoutConstraint constraintWithItem: attribute: relatedBy: toItem: attribute: multiplier: constant:]

- [NSLayoutConstraint constraintsWithVisualFormat: options: metrics: views:]

Working with Auto Layout Programmatically

- [NSLayoutConstraint constraintWithItem: attribute: relatedBy: toItem: attribute: multiplier: constant:]
- [NSLayoutConstraint constraintsWithVisualFormat: options: metrics: views:]

http://furnacedigital.blogspot.tw/2013/10/auto-layout.html

Method:

[NSLayoutConstraint constraintWithItem: attribute: relatedBy: toItem: attribute: multiplier: constant:]

這是使用傳統的的方式對每一個物件的限制就加以設定，可以看到輸入的參數非常的多，而且一次只能設定一項限制。

- **constraintWithItem: attribute:**

  代表被限制的物件本身與被限制的屬性。

- **relatedBy:**

  被限制物件的屬性與它參考目標之間的關係 NSLayoutRelationLessThanOrEqual、NSLayoutRelationEqual 或是 NSLayoutRelationGreaterThanOrEqual（小於等於、等於或是大於等於）。

- **toItem: attribute:**

  參考目標本身與參考的屬性

- **multiplier: constant:**

  倍率與偏差值，按照 **NSLayoutConstraint.h** 文件內的說法他們的關係為 view1.attr1 = view2.attr2 * multiplier + constant。

Sample Code:

```objc
//DEMO LOGO
UIImageView *demoLogo = [[UIImageView alloc]initWithImage:[UIImage imageNamed:@"DEMO LOGO.png"]];
[demoLogo setAlpha:0.88];
[self.view addSubview:demoLogo];
[demoLogo setTranslatesAutoresizingMaskIntoConstraints:NO];

//與superview的底部始終差距50個單位高度（不能小於）
NSLayoutConstraint *myConstraint;
myConstraint = [NSLayoutConstraint constraintWithItem:demoLogo attribute:NSLayoutAttributeBottom
                relatedBy:NSLayoutRelationGreaterThanOrEqual
                toItem:[demoLogo superview] attribute:NSLayoutAttributeBottom
                multiplier:1.0f constant:-50.0f];
[self.view addConstraint:myConstraint];

//x軸對齊superview的x軸中心點
myConstraint = [NSLayoutConstraint constraintWithItem:demoLogo attribute:NSLayoutAttributeCenterX
                relatedBy:NSLayoutRelationEqual
                toItem:[demoLogo superview] attribute:NSLayoutAttributeCenterX
                multiplier:1.0f constant:-0.0f];
[self.view addConstraint:myConstraint];

//寬度限制
myConstraint = [NSLayoutConstraint constraintWithItem:demoLogo attribute:NSLayoutAttributeWidth
                relatedBy:NSLayoutRelationEqual
                toItem:nil attribute:NSLayoutAttributeNotAnAttribute
                multiplier:1.0f constant:210.0f];
[self.view addConstraint:myConstraint];

//高度限制
myConstraint = [NSLayoutConstraint constraintWithItem:demoLogo attribute:NSLayoutAttributeHeight
                relatedBy:NSLayoutRelationEqual
                toItem:nil attribute:NSLayoutAttributeNotAnAttribute
                multiplier:1.0f constant:20.0f];
[self.view addConstraint:myConstraint];
```

http://furnacedigital.blogspot.tw/2013/10/auto-layout.html

Method:
constraintsWithVisualFormat

[NSLayoutConstraint constraintsWithVisualFormat: options: metrics: views:]

" V: " 和 " H: “　　分別代表由垂直或是水平方向來佈局。
" | "　代表 Superview 的邊界。
" - "　代表預設寬度或高度，如果在中間加上數字 " -20- "，則代表限制 20 個單位高度或寬度。
" [ ] "　代表物件本身，括號內包含物件的變數名稱與大小限制，可以使用關係運算子（< ＝ 、> = 或 ＝＝ 等）。
" @ "　優先權，1 至 1000 的整數，優先權較大的條件會優先被滿足，例如 ，[ViewB(>=100@1000)]，物件 ViewB 不可以小於 100 個單位長度或寬度會最優先被考慮。

Sample Code:

[NSLayoutConstraint constraintsWithVisualFormat: options: metrics: views:]

由於傳統的方式過於瑣碎，因此 NSLayoutConstraint 也提供另一種較為直覺的方式來設定這些限制，參考下面範例，我們將繪製兩個 UIView 來擋住我們的 demoLogo，並且給予這兩個 UIView 一些限制。

```
1  UIView *ViewA = [[UIView alloc]init];
2  ViewA.backgroundColor = [UIColor orangeColor];
3  [ViewA setTranslatesAutoresizingMaskIntoConstraints:NO];
4  [self.view addSubview:ViewA];
5
6  UIView *ViewB = [[UIView alloc]init];
7  ViewB.backgroundColor = [UIColor blueColor];
8  [ViewB setTranslatesAutoresizingMaskIntoConstraints:NO];
9  [self.view  addSubview:ViewB];
10
11
12 NSMutableArray *myConstraints = [NSMutableArray array];
13
14 //從水平方向佈局
15 [myConstraints addObjectsFromArray:
16     [NSLayoutConstraint constraintsWithVisualFormat:@"H:|-20-[ViewA(100)]-10-[ViewB(>=100)]-|"
17         options:NSLayoutFormatDirectionLeadingToTrailing
18         metrics:nil
19         views:NSDictionaryOfVariableBindings(ViewA,ViewB)]];
20
21
22 //從垂直方向佈局
23 [myConstraints addObjectsFromArray:
24     [NSLayoutConstraint constraintsWithVisualFormat:@"V:[ViewA(100)]-|"
25         options:NSLayoutFormatDirectionLeadingToTrailing
26         metrics:nil
27         views:NSDictionaryOfVariableBindings(ViewA)]];
28
29 //從垂直方向佈局
30 [myConstraints addObjectsFromArray:
31     [NSLayoutConstraint constraintsWithVisualFormat:@"V:[ViewB(ViewA)]-|"
32         options:NSLayoutFormatDirectionLeadingToTrailing
33         metrics:nil
34         views:NSDictionaryOfVariableBindings(ViewB,ViewA)]];
35
36 [self.view addConstraints:myConstraints];
```

Beginner Topics

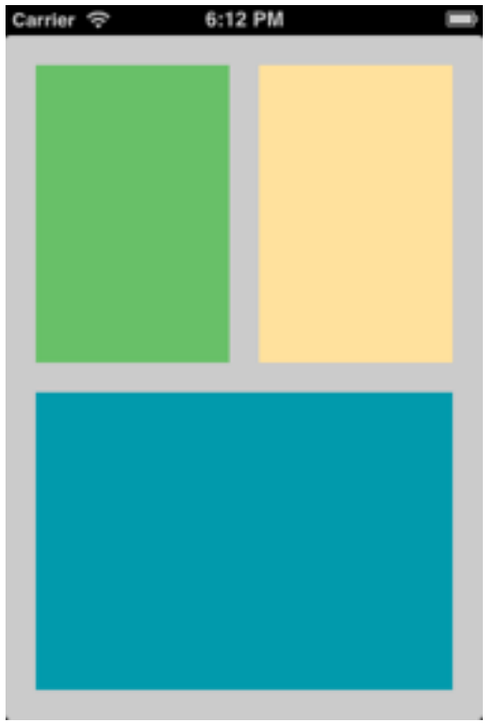
Tutorial

http://www.raywenderlich.com/50317/beginning-auto-layout-tutorial-in-ios-7-part-1

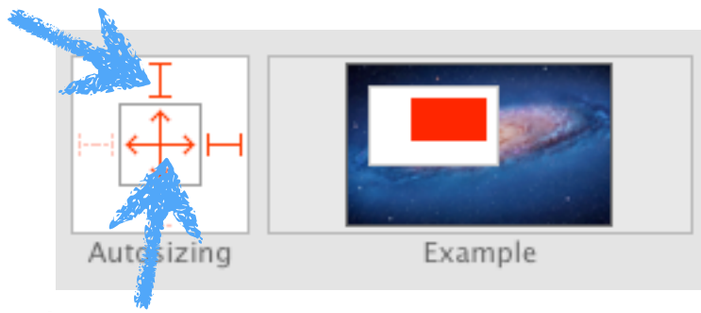
Problem with springs and struts

Springs & Struts

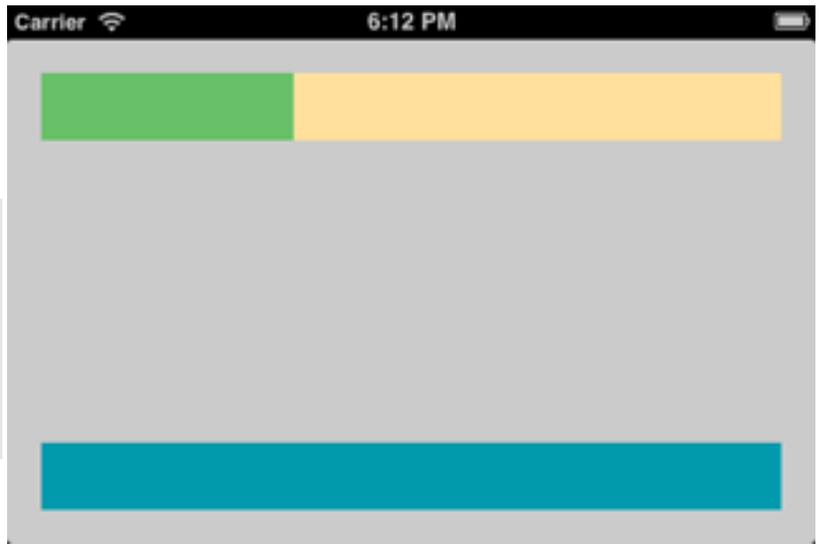
Springs and struts: how to set up

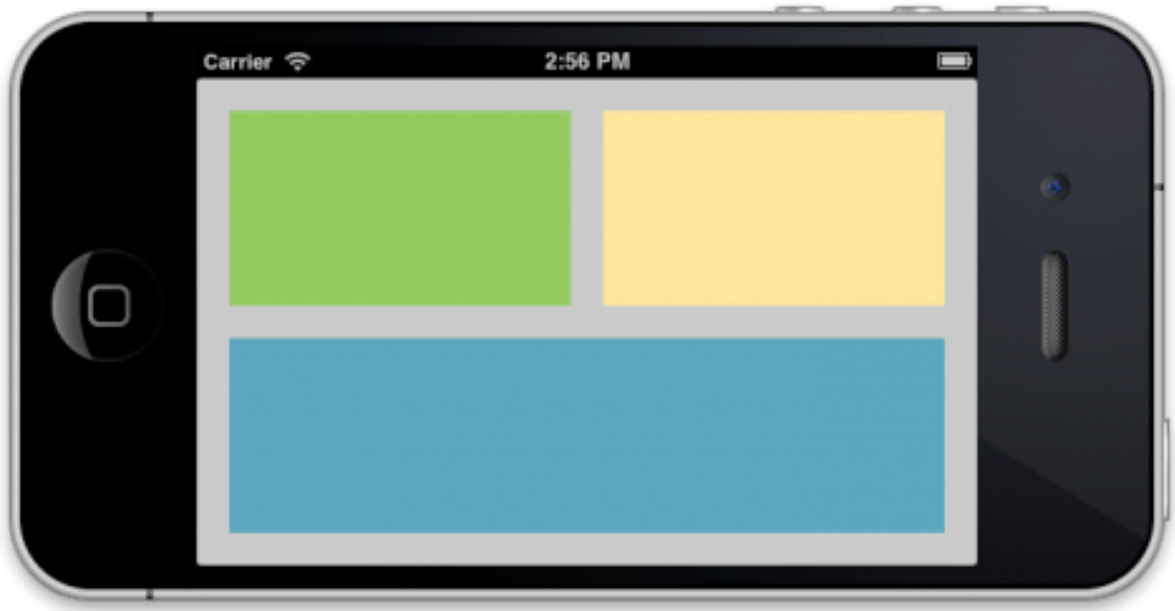


`struts`



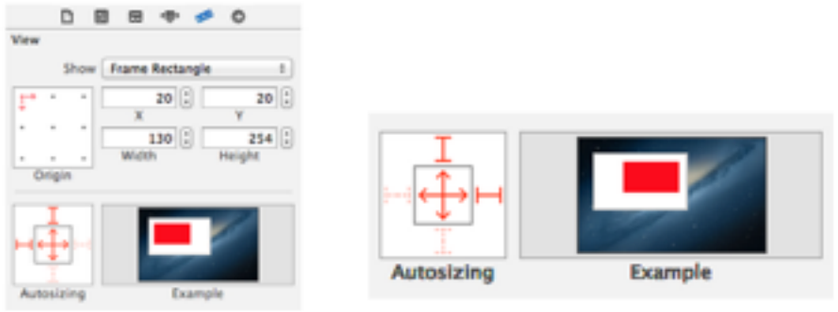
`springs`

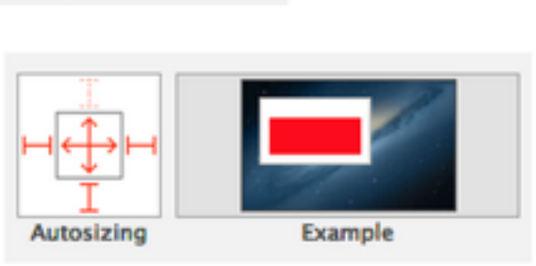


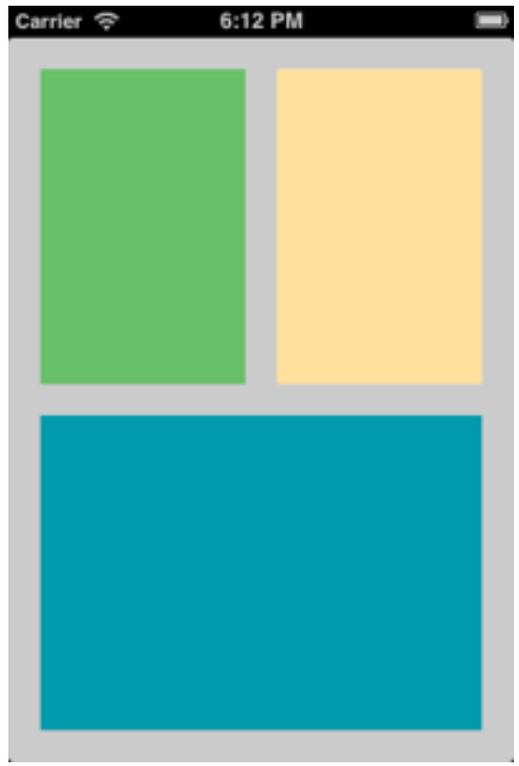


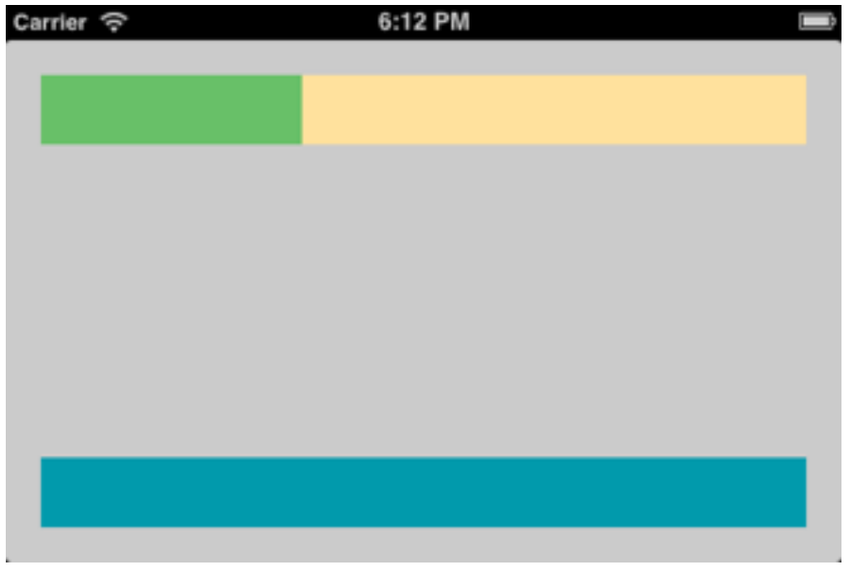
Springs & Struts

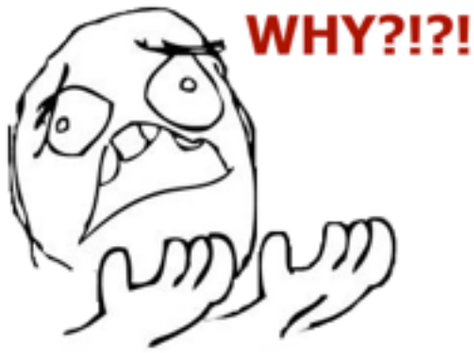
Problem:
not accurate







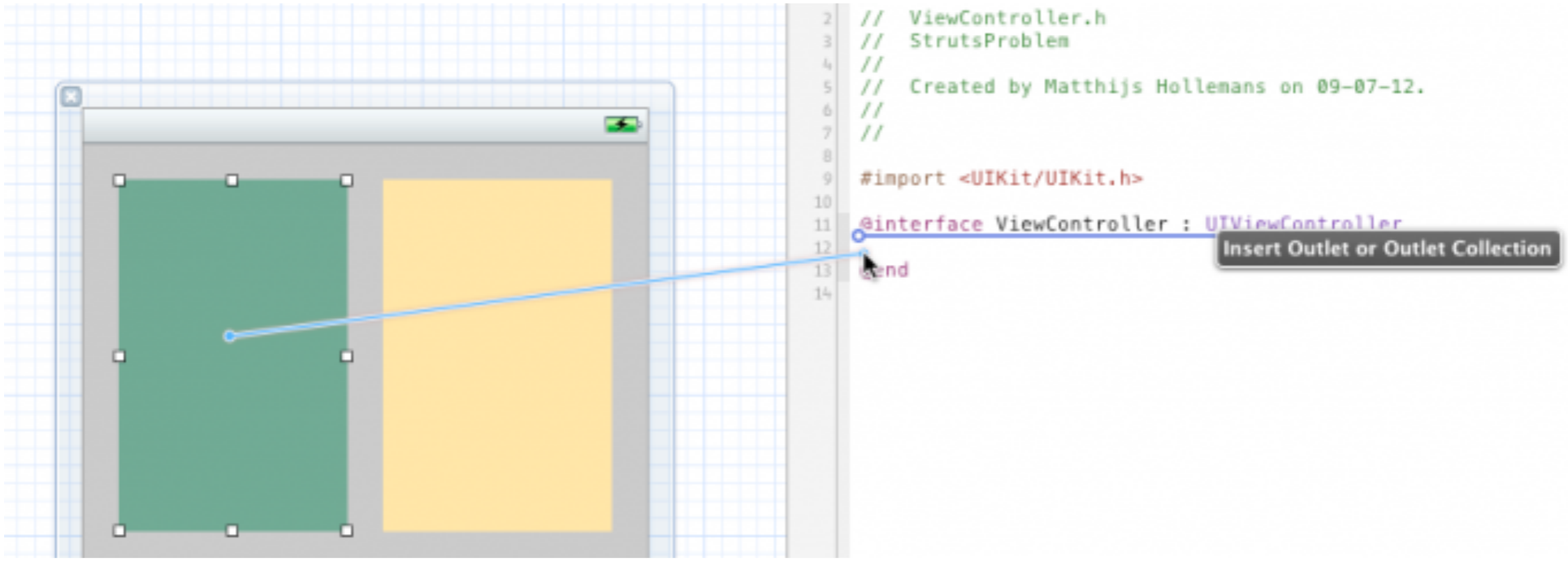




Springs & Struts

Solution:
solve it manually

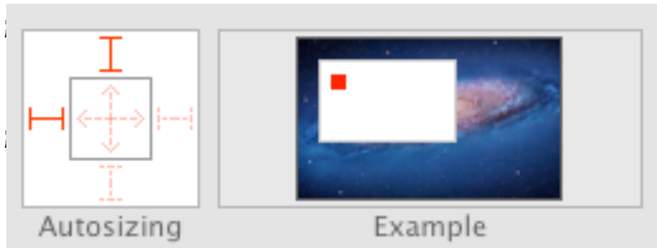
```
#import <UIKit/UIKit.h>

@interface ALKViewController : UIViewController
@property (weak, nonatomic) IBOutlet UIView *topLeftView;
@property (weak, nonatomic) IBOutlet UIView *topRightView;
@property (weak, nonatomic) IBOutlet UIView *bottomView;
@end
```



Springs & Struts

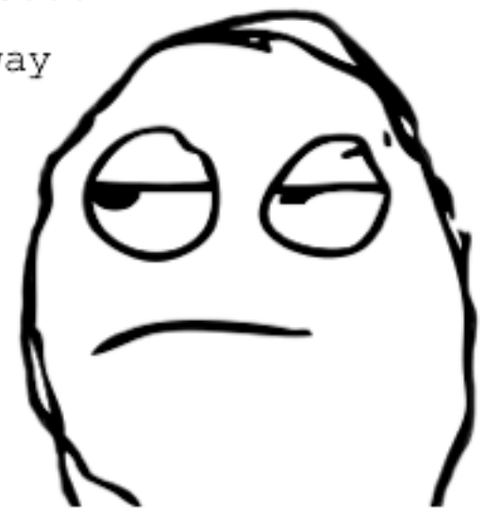
Problem:
Too many codes

```
- (void)willAnimateRotationToInterfaceOrientation:
(UIInterfaceOrientation)toInterfaceOrientation
                                                duration:(NSTimeInterval)duration{
    [super willAnimateRotationToInterfaceOrientation:toInterfaceOrientation
duration:duration];
    if (toInterfaceOrientation == UIInterfaceOrientationLandscapeLeft ||
        toInterfaceOrientation == UIInterfaceOrientationLandscapeRight) {
        //横向模式
        CGRect rect = self.topLeftView.frame;
        rect.size.width = 210;
        rect.size.height = 120;
        self.topLeftView.frame = rect;

        rect = self.topRightView.frame;
        rect.origin.x = 250;
        rect.size.width = 210;
        rect.size.height = 120;
        self.topRightView.frame = rect;

        rect = self.bottomView.frame;
        rect.origin.y = 160;
        rect.size.width = 440;
        rect.size.height = 120;
        self.bottomView.frame = rect;
    }else
    {
        CGRect rect = self.topLeftView.frame;
        rect.size.width = 130;
        rect.size.height = 200;
        self.topLeftView.frame = rect;

        rect = self.topRightView.frame;
        rect.origin.x = 170;
        rect.size.width = 130;
        rect.size.height = 200;
        self.topRightView.frame = rect;

        rect = self.bottomView.frame;
        rect.origin.y = 240;
        rect.size.width = 280;
        rect.size.height = 200;
        self.bottomView.frame = rect;
    }
}
```



There must be...

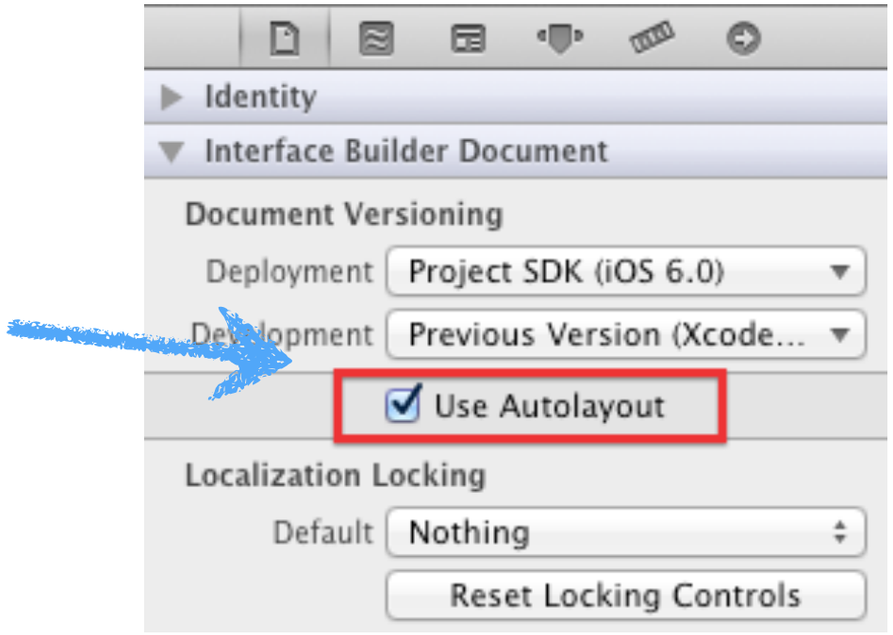
...another way



Auto Layout to the rescue!

Auto Layout

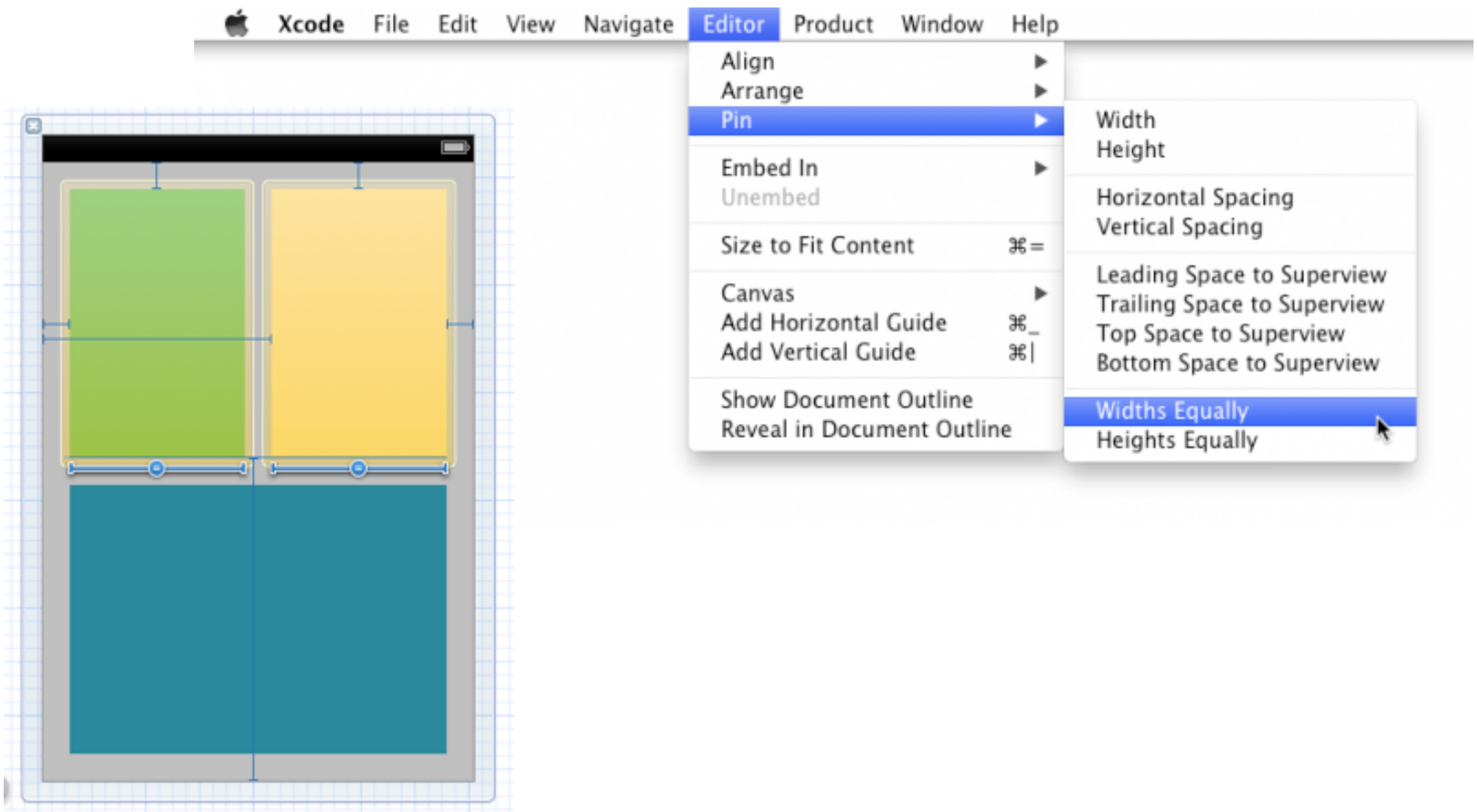
Setup:



Note: Auto Layout is always enabled for the entire nib or storyboard file. All the views inside that nib or storyboard will use Auto Layout if you check that box.

Auto Layout

Setup "Widths Equally":

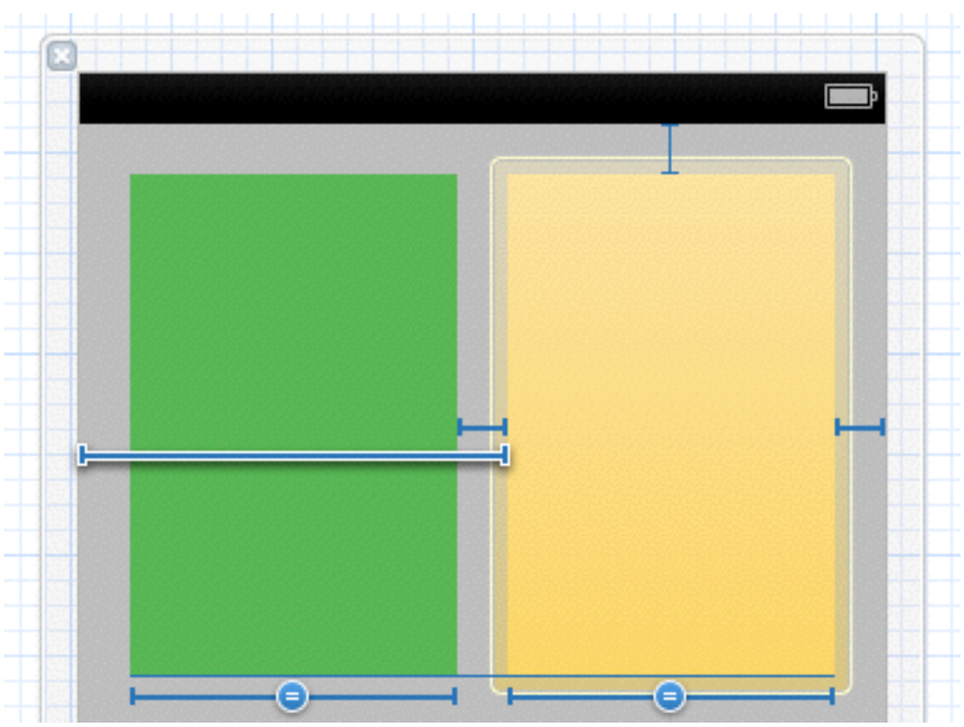


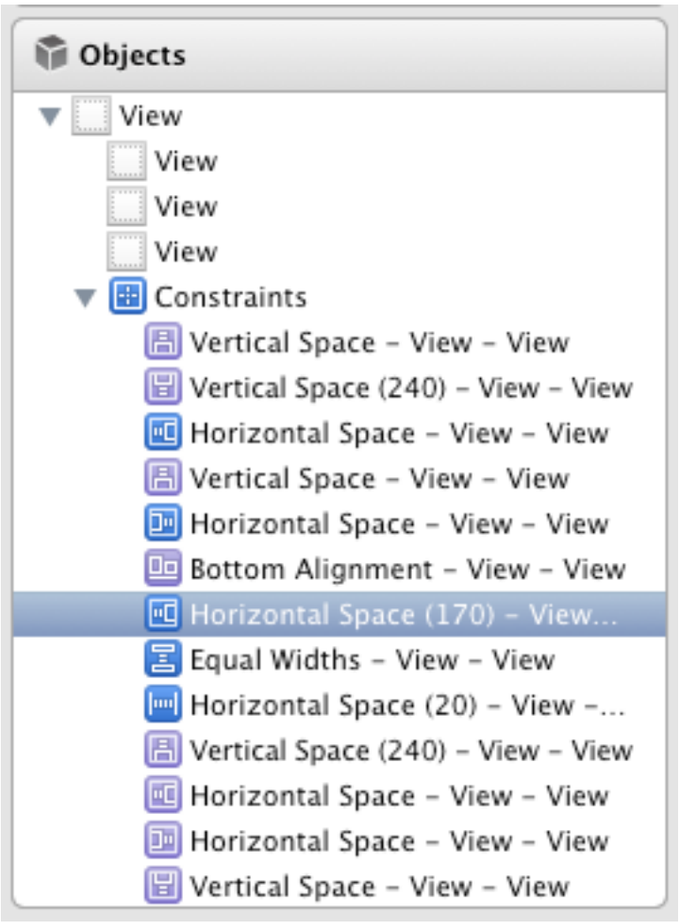
Hold down the ⌘ key while you click on the two views on the top (the green and yellow ones), so that both are selected. From Xcode's Editor menu, select Pin\Widths Equally:

Auto Layout救星

Springs & Struts 的問題

在左邊的文檔概要圖中, 你會注意到有一個新的section名叫 「Constraints」
當你在nib文件中啟用Auto Layout，這個section會被自動加入。
在這篇文檔的下一部分你會了解到這些Contraints是什麼以及他們是如何操作的。
現在把一個名叫Horizontal Space (170)的section從Constraints列表裡面刪除：

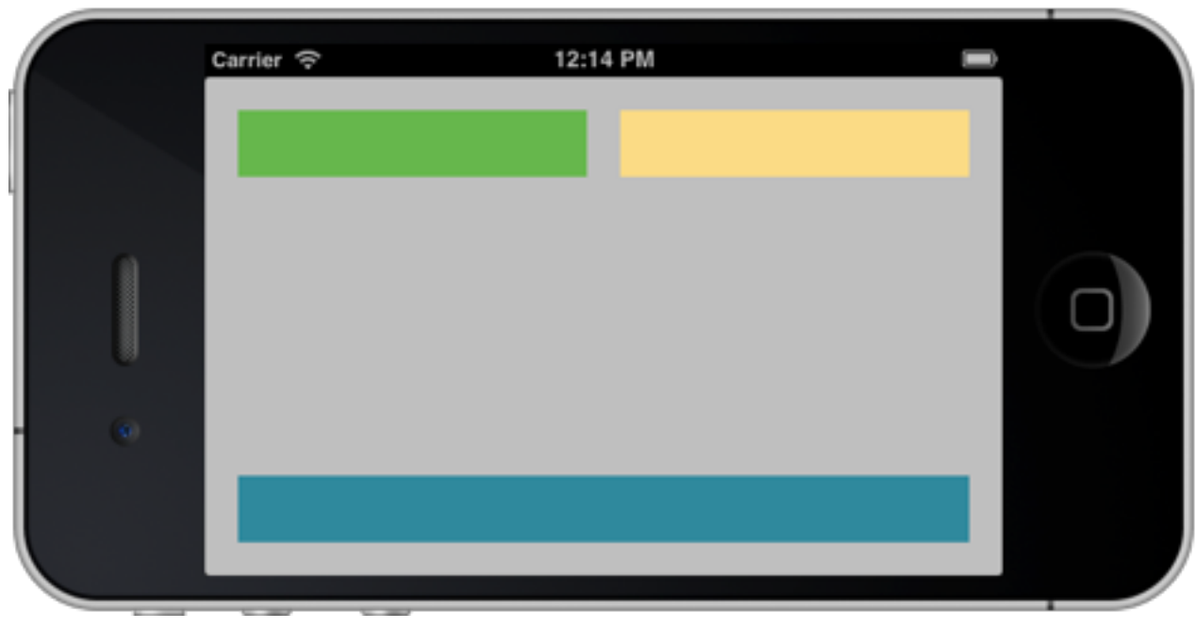




Auto Layout救星

Springs & Struts 的問題

現在看起來寬度正常了



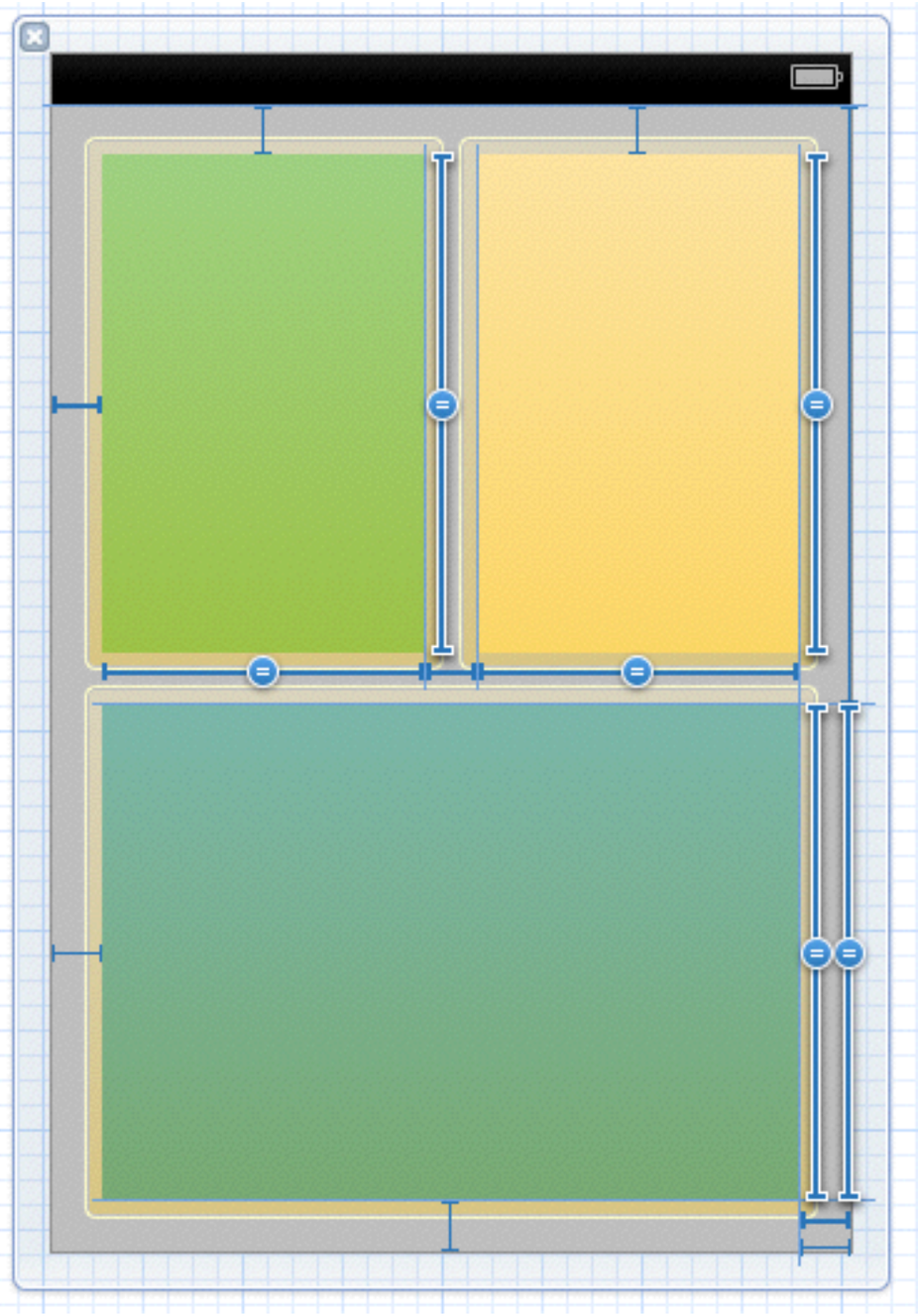
Auto Layout救星

Springs & Struts 的問題

1.接著選取三個view，使用
Editor > Pin > Heights Equally

2.現在還是按住Cmd鍵同時選中左上角的以及底部的view,
然後使用Editor > Pin> Vertical Spacing。

3.最後把「Vertical Space (240)」 從constraint列表裡面刪除



Auto Layout救星

Springs & Struts 的問題

酷！很簡單的設定，達到我們需要的結果

但剛才你究竟做了什麼呢?

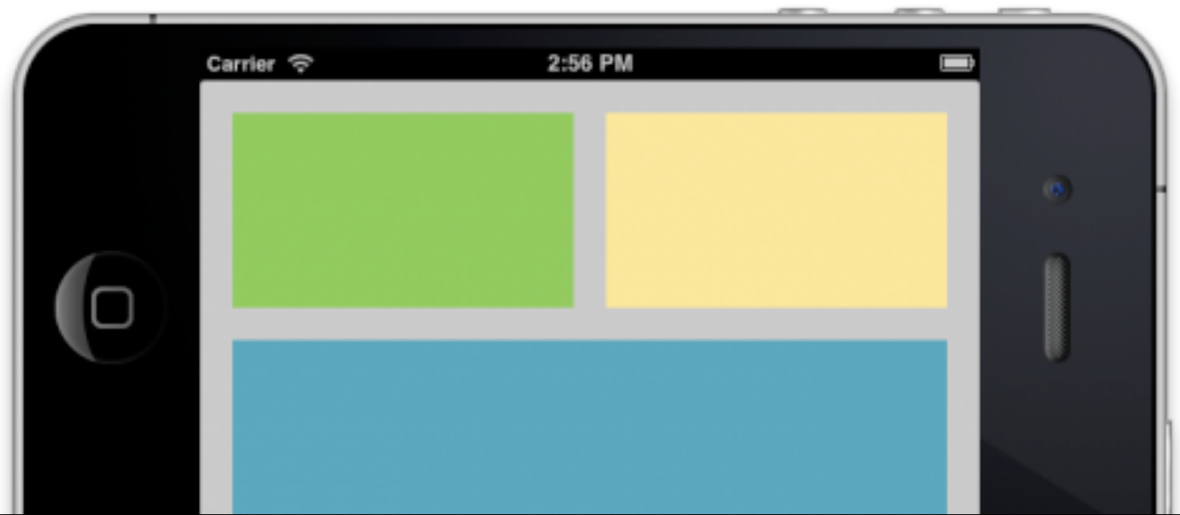
Auto Layout 能使你簡單地表達清楚頁面布局中

各個view之間的關係而不會讓你為了各種view有多大以及他們該定位在哪裡硬編許多代碼 。

你剛才做了如下的關系操作 – 也就是 constraints – 在頁面布局裡:

- 左上角和右上角的view (也就是第一次的pin widths equally 操作).
- 在左上角view和右上角view之間有20-point的間距 (相應的操作是 pin horizontal spacing).
- 所有的view是相同的高度 (相應的操作是pin heights equally).
- 在頂部兩個view與底部的view之間有一個20-point的間距 (the pin vertical spacing).

以上這些就足以展示，當屏幕尺寸變化時，Auto Layout如何放置布局裡的各種view以及它是如何工作的。

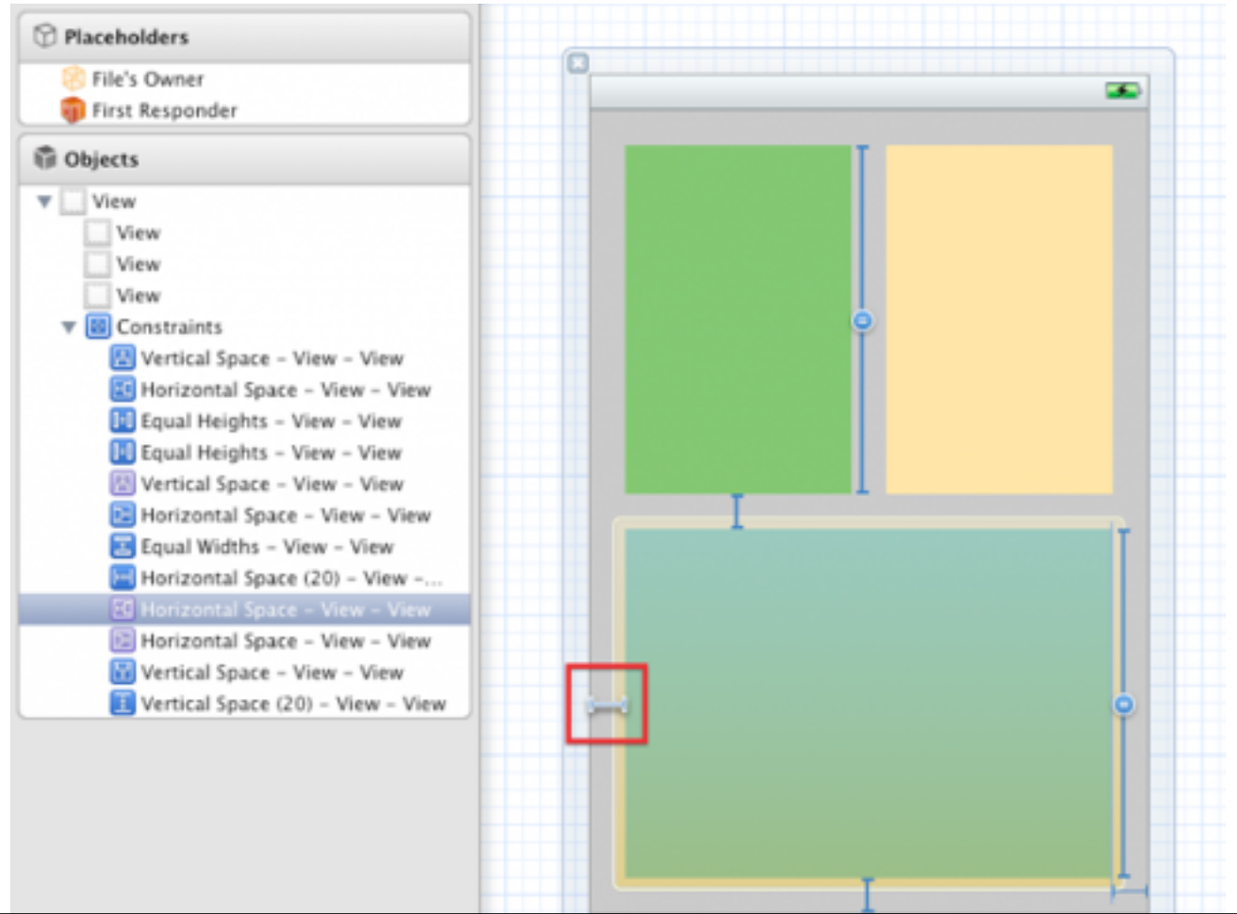


Auto Layout救星

Springs & Struts 的問題

提示： springs-and-struts布局模式也會帶來一些其他限制當你從它切換至「Use Autolayout」模式時。對於各個view和屏幕邊緣之間的邊距都基本會有一條限制，是這麼說的：「這個view總是和頂部/底部/左邊/右邊保持著20-points的距離。」

你可以看到你的所有contraints在文檔概要裡。如果你在文檔概要裡點擊一個constraint，Interface Builder會在contraint在view中所體現的地方通過畫一條白色的邊框並且對之添加一個陰影使其高亮顯示：

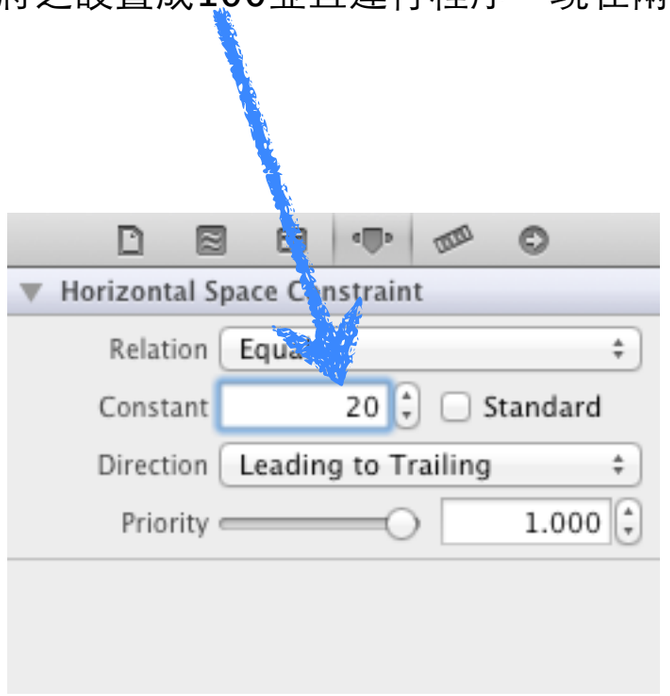


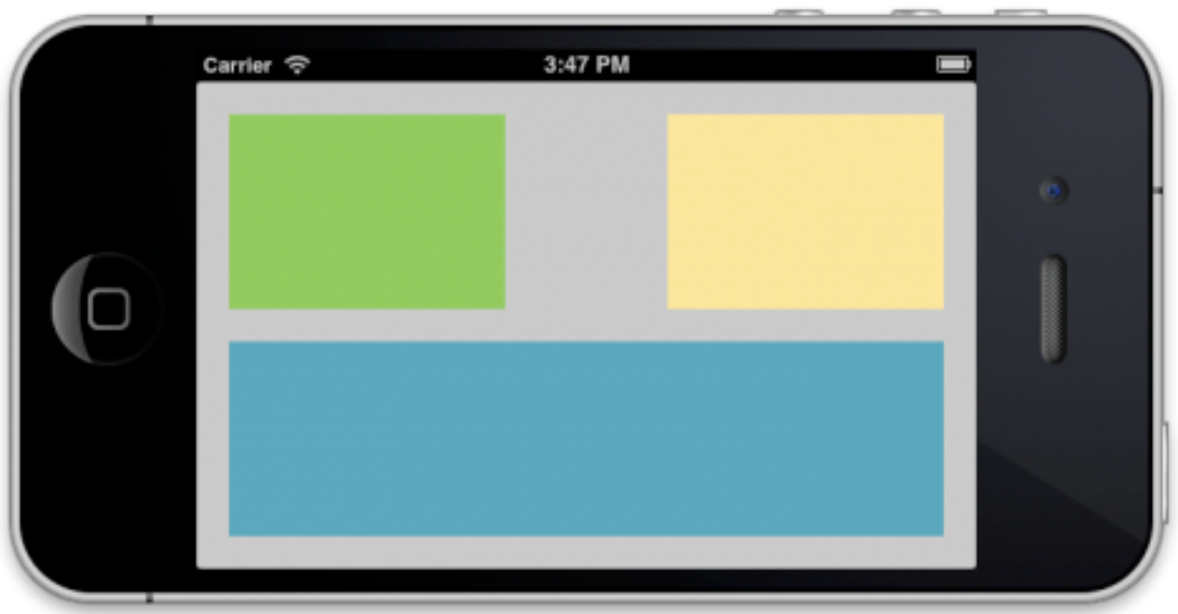
Auto Layout救星

Springs & Struts 的問題

Constraints是真實的對象（屬於 NSLayoutConstraint類），他們也擁有相應的屬性。 比如說，選中頂部兩個view間距的constraint（名為「Horizontal Space (20)」）然後切換至它的Attributes inspector。 現在你可以通過修改Constant裡的值來改變兩個view之間的距離大小。

將之設置成100並且運行程序。現在兩個view之間的距離更寬了:





Auto Layout救星

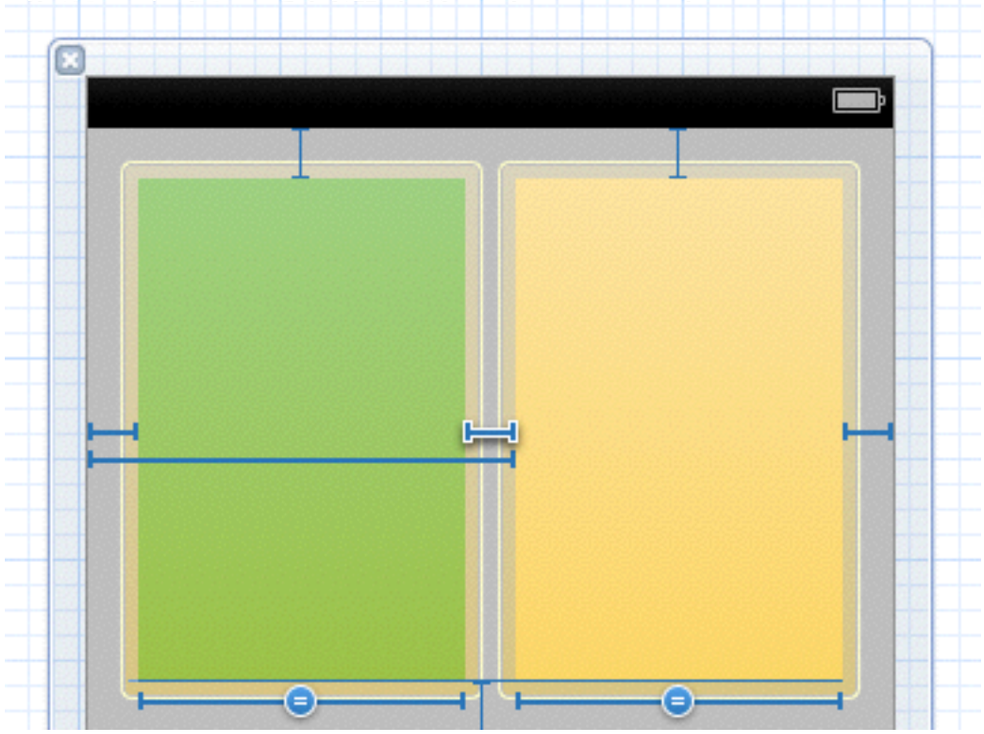
Springs & Struts 的問題

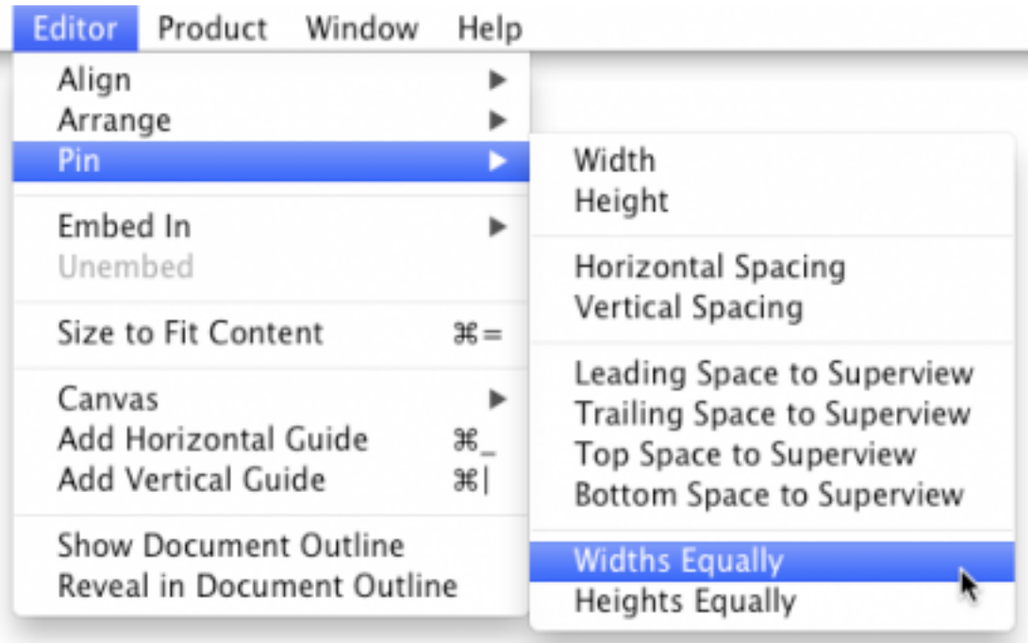
再次重新選中這兩個view並且

使用Editor>PinHorizontal Spacing操作。

(即使這兩個view看上去像被選中了當你做第一個Pin操作後。

但是請注意目前他們處在一種特殊的布局關系顯示模式中，

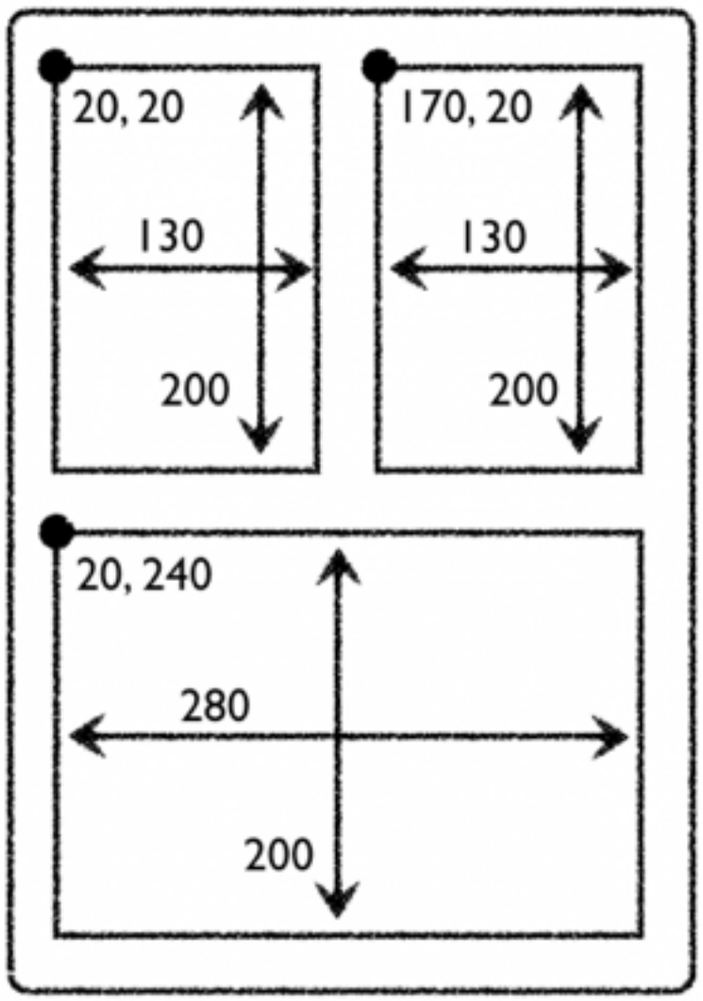


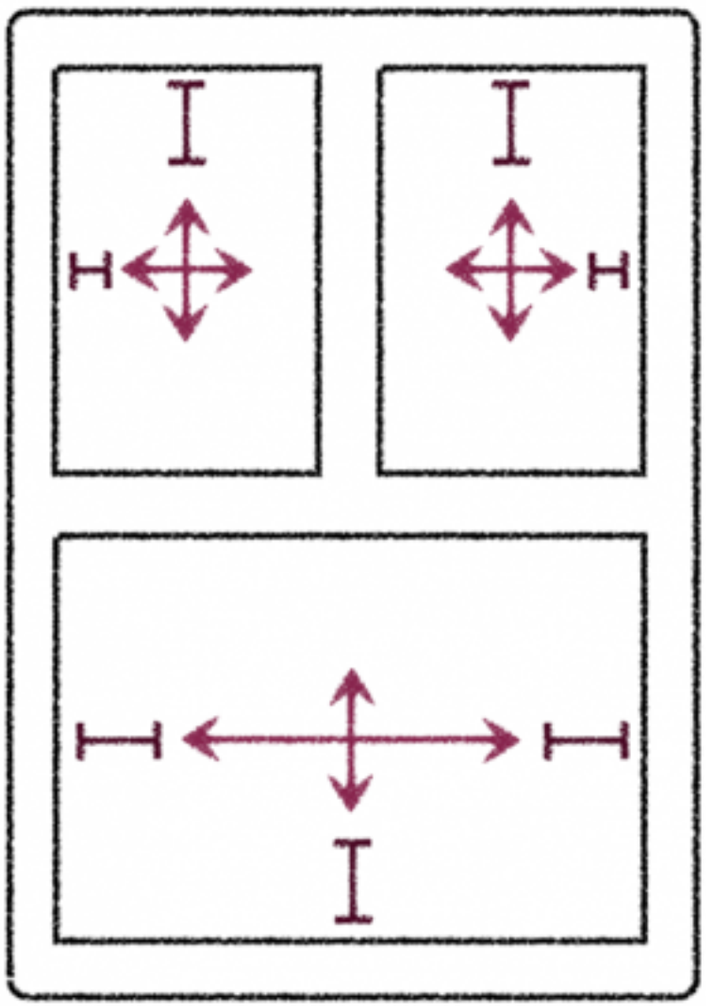


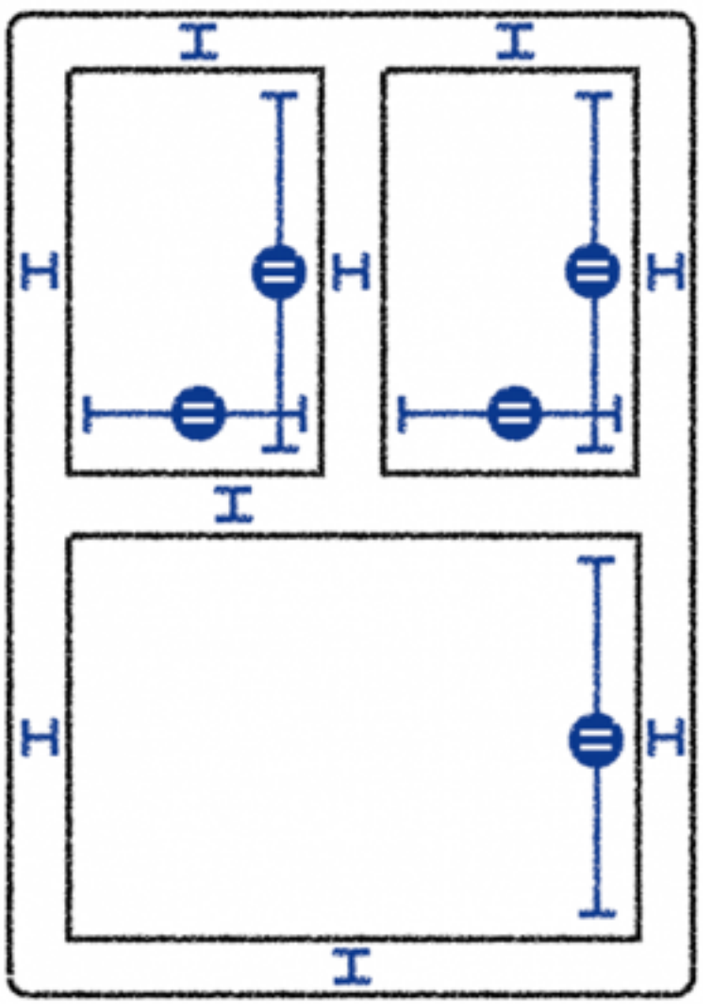
Auto Layout救星

Springs & Struts 的問題

希望狀況與兩種布局方式：`autosizing masks`、`autolayout`





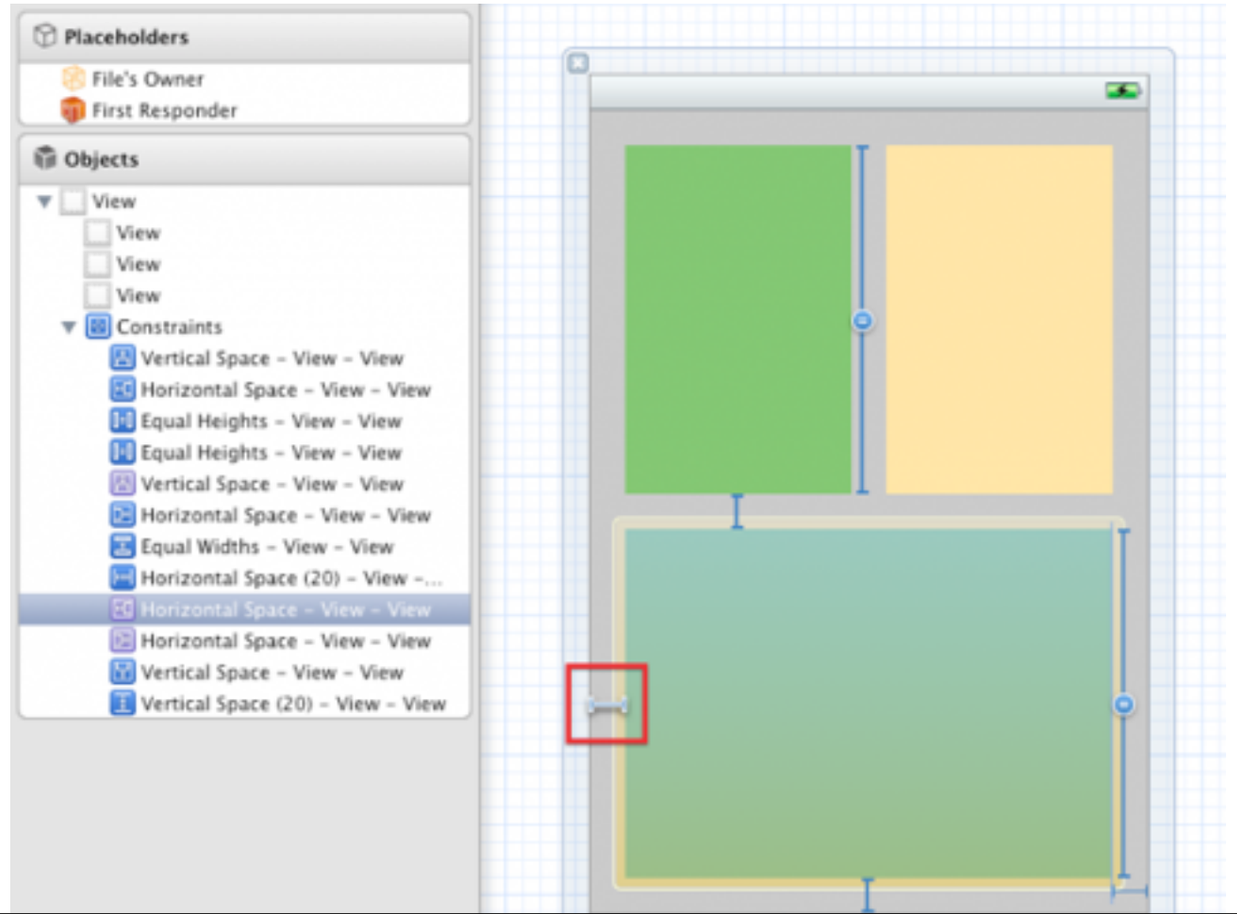


Auto Layout救星

Springs & Struts 的問題

提示： springs-and-struts布局模式也會帶來一些其他限制當你從它切換至「Use Autolayout」模式時。對於各個view和屏幕邊緣之間的邊距都基本會有一條限制，是這麼說的：「這個view總是和頂部/底部/左邊/右邊保持著20-points的距離。」

你可以看到你的所有contraints在文檔概要裡。如果你在文檔概要裡點擊一個constraint，Interface Builder會在contraint在view中所體現的地方通過畫一條白色的邊框並且對之添加一個陰影使其高亮顯示：

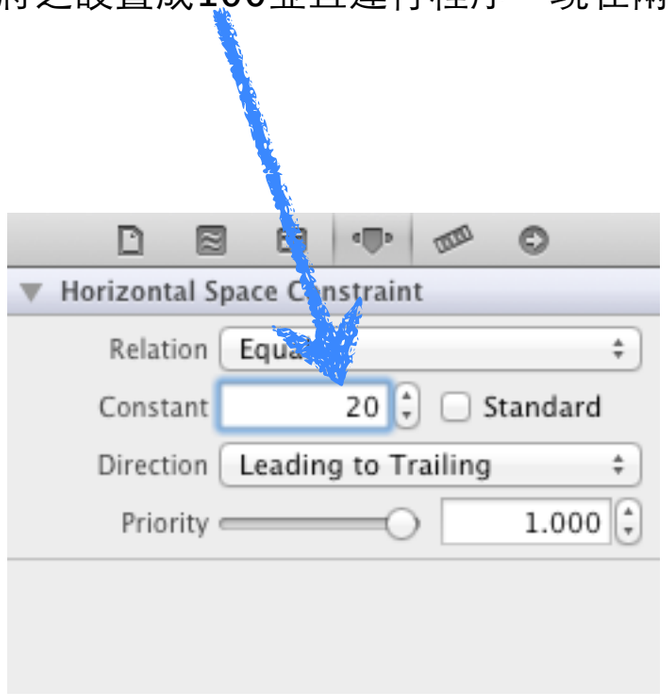


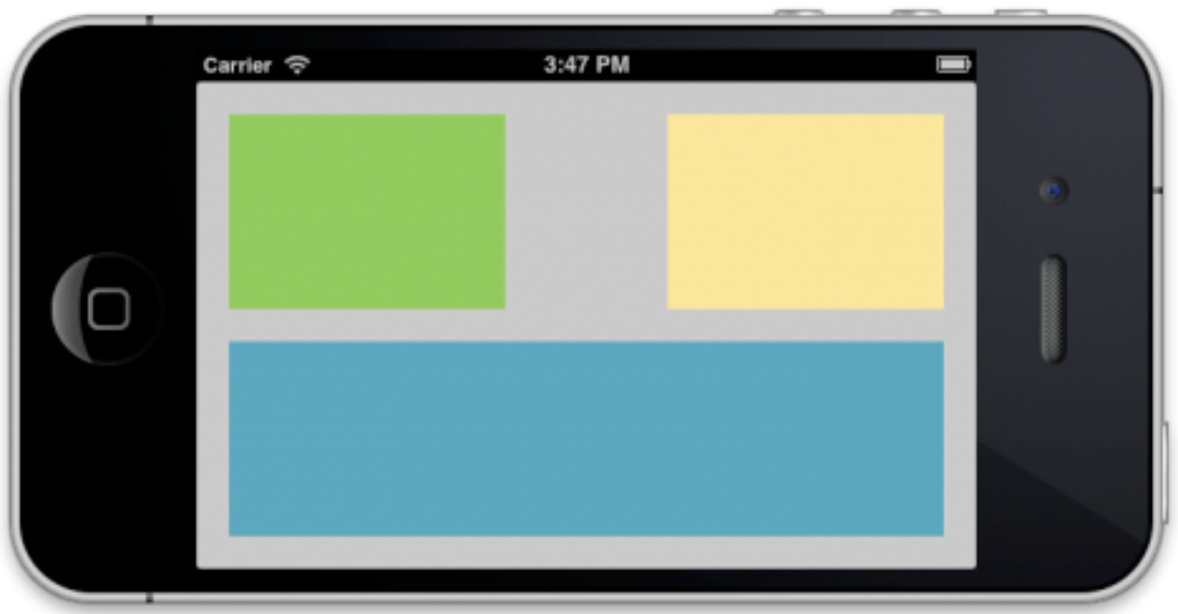
Auto Layout救星

Springs & Struts 的問題

Constraints是真實的對象（屬於 NSLayoutConstraint類），他們也擁有相應的屬性。 比如說，選中頂部兩個view間距的constraint（名為「Horizontal Space (20)」）然後切換至它的Attributes inspector。 現在你可以通過修改Constant裡的值來改變兩個view之間的距離大小。

將之設置成100並且運行程序。現在兩個view之間的距離更寬了:

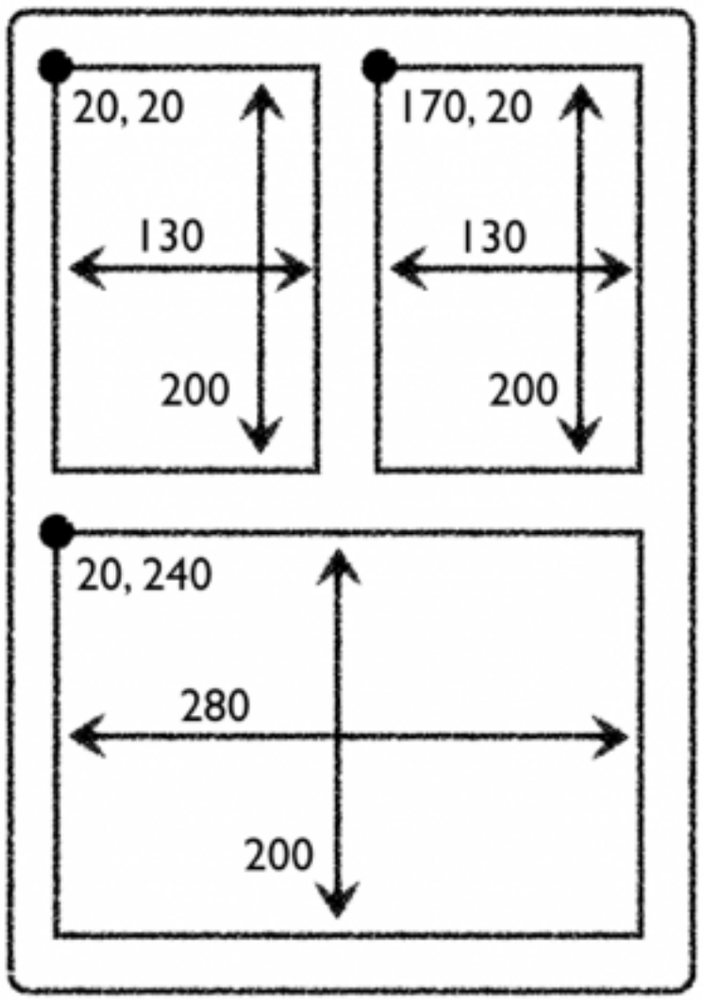


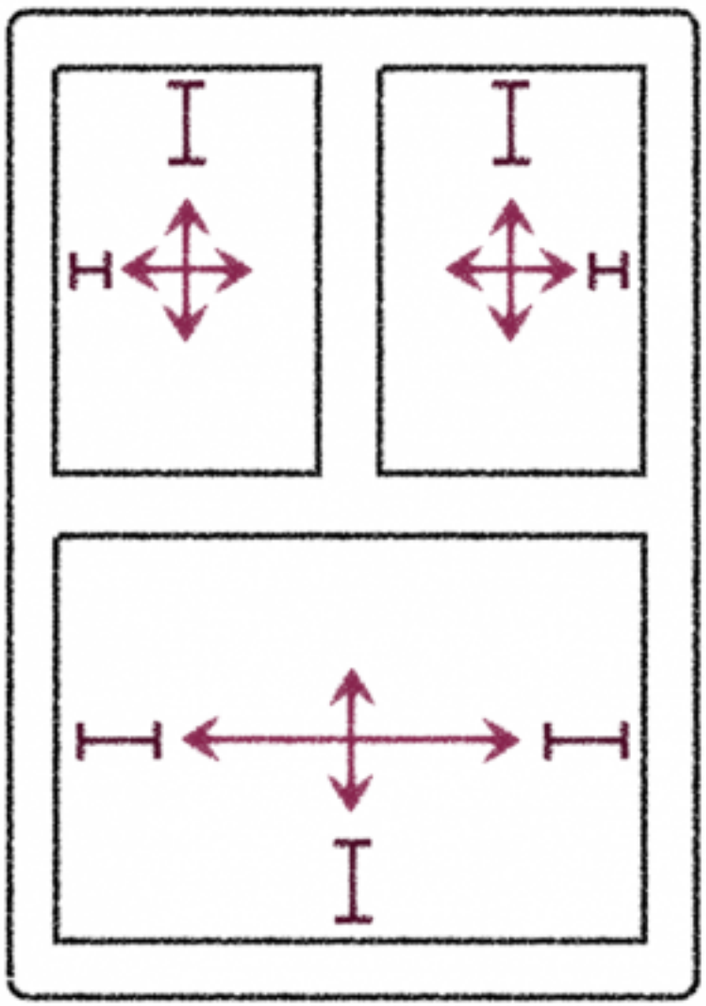


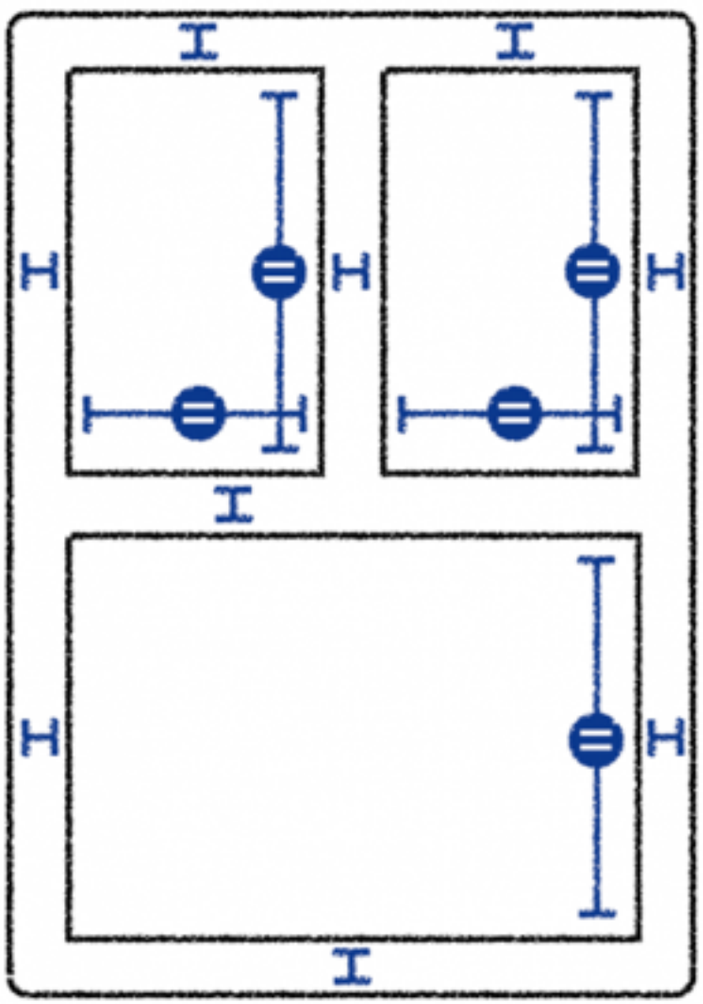
Auto Layout救星

Springs & Struts 的問題

希望狀況與兩種布局方式：`autosizing masks`、`autolayout`







愛上constraints

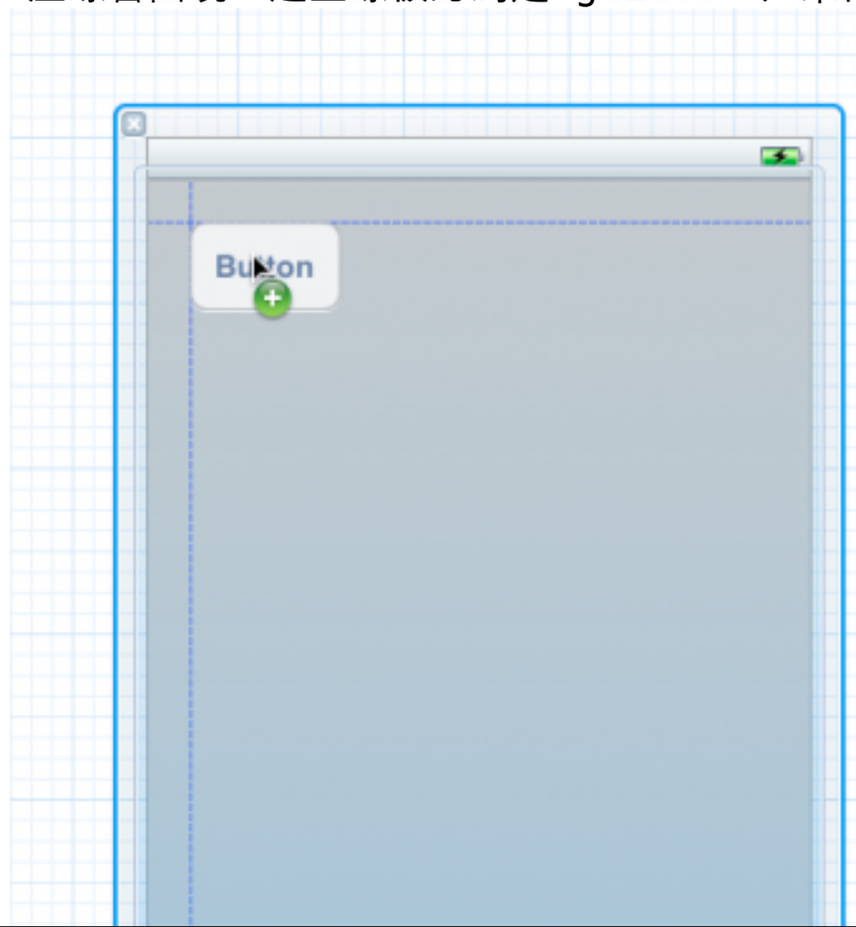
關掉你目前的工程然後創建一個新的項目使用Single View Application樣板。

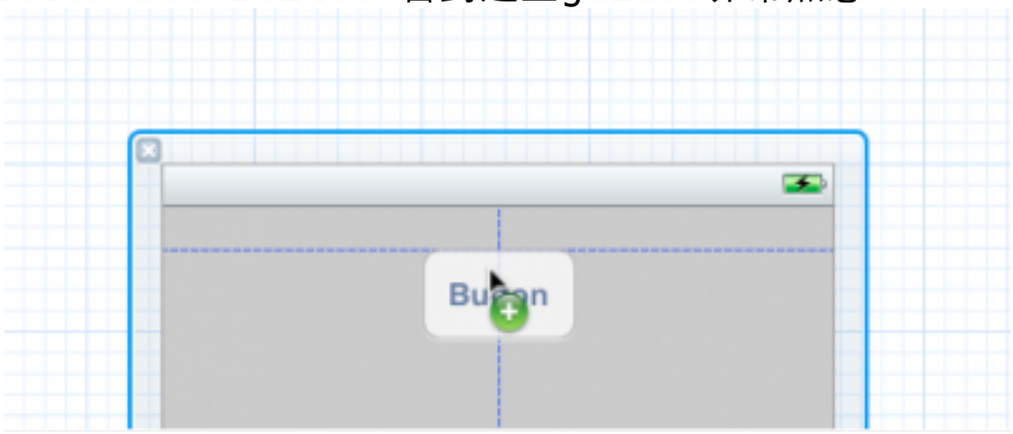
命名項目為「Constraints」。

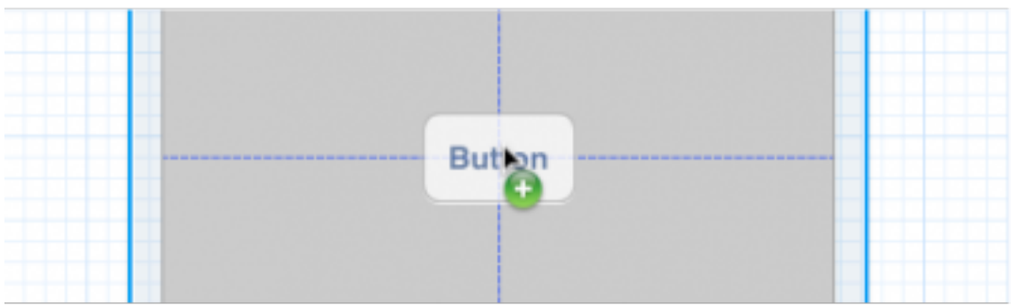
然後選擇為iPhone project並且不使用storyboards，但是我們需要用到ARC。

一個使用Xcode4.5創建的新項目會預設為你選擇啟動Auto Layout，所以你必要做任何特別的事情來啟用它。

點擊ViewController.xib 來打開Interface Builder。把一個新的圓角按鈕拖進canvas。 注意當你拖拽的時候，藍色虛線會出現。這些線被認為是 guides，如果你之前就常用Interface Builder會對這些guides非常熟悉





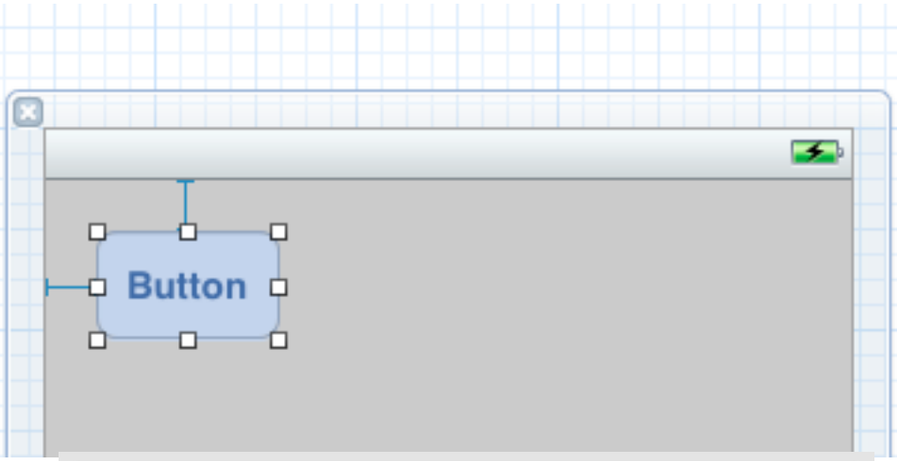


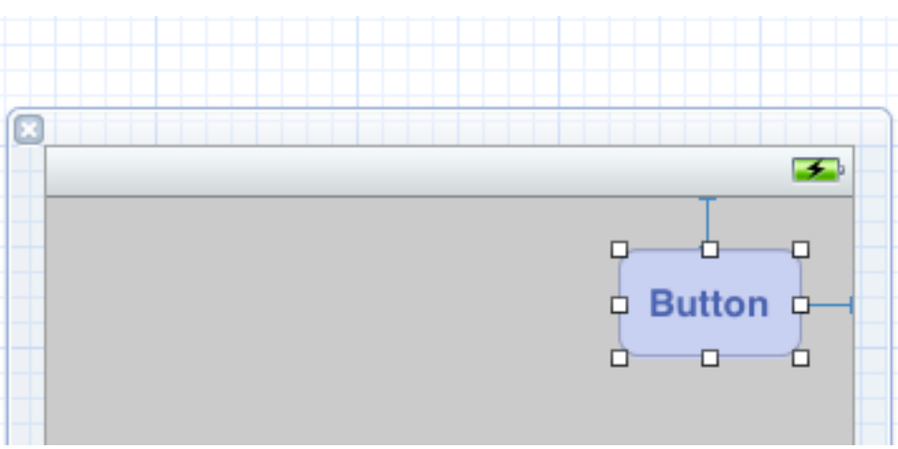
愛上constraints

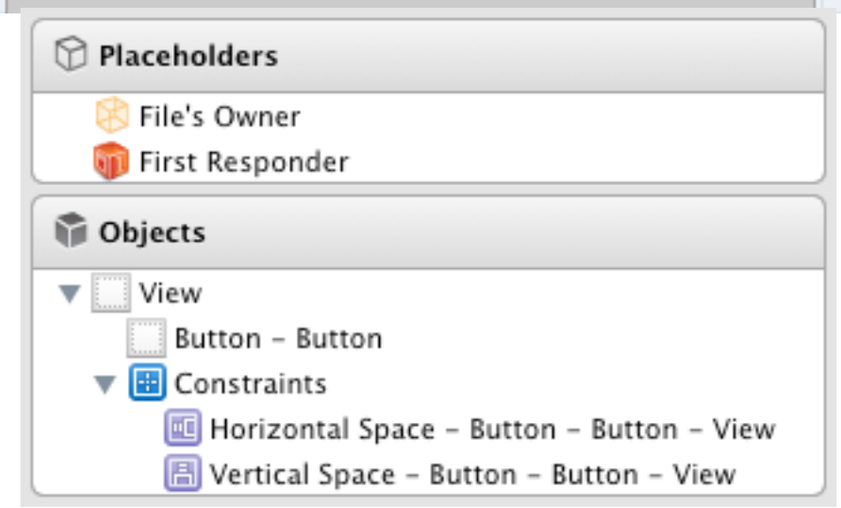
Interface Builder會自動產生constraint

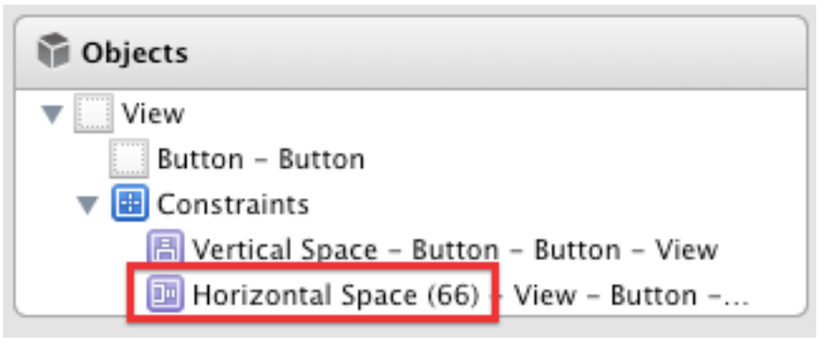
並且隨著你更動它的位置，

Interface Builder也會自動更新它的contraints









愛上constraints

接下來我們看尺寸面板在有auto layout時會多了一些選項。

當Auto Layout 禁用時，

X，Y，Width或者Height裡的值會改變所選中view的位置以及尺寸。

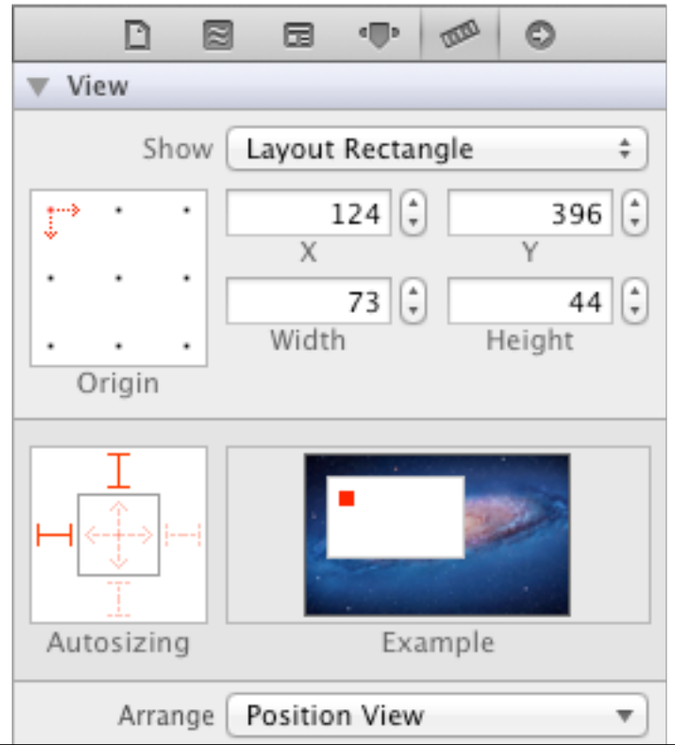
當Auto Layout啟用時，

你還是能夠在這些框裡面輸入新的值，但通常結果不會你是想要的效果。

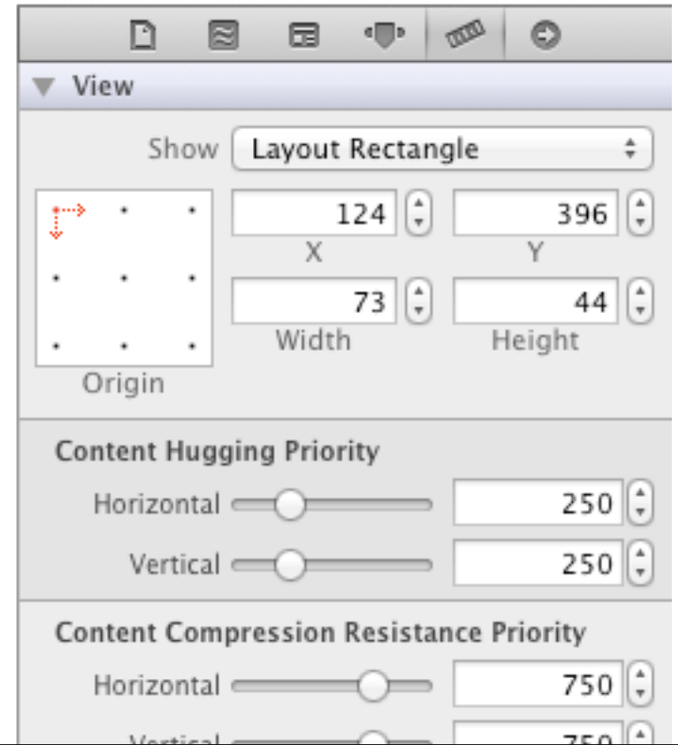
那個view 會移動，

但是Interface Builder也會基於你的新值來計算出新的constraints。

Without Auto Layout:



With Auto Layout:

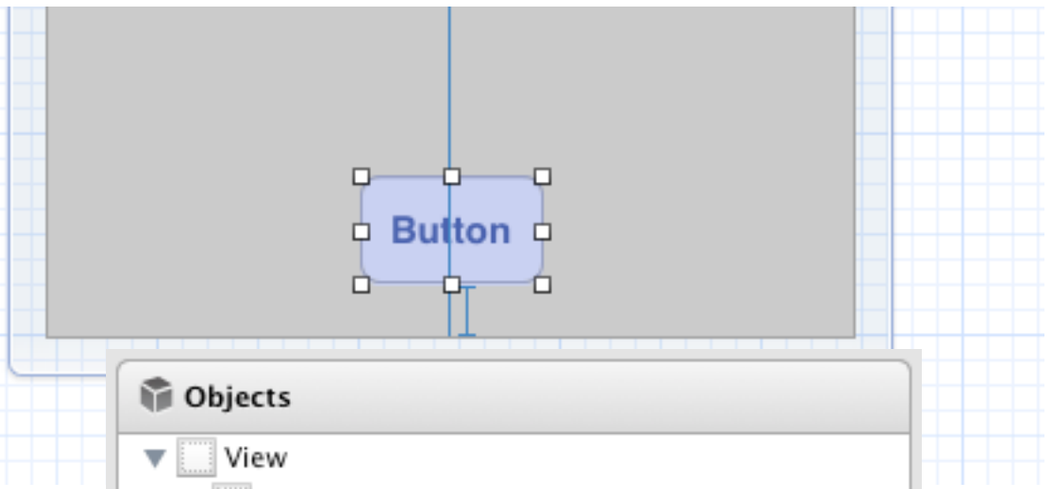


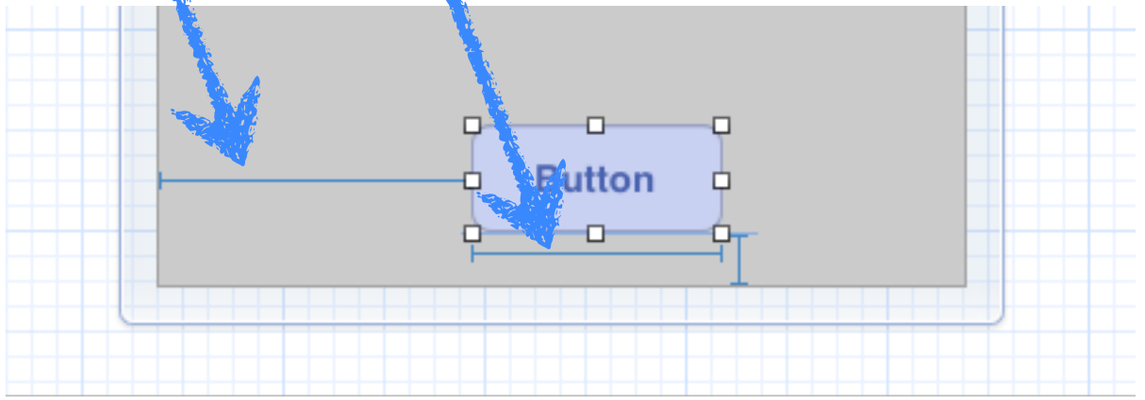
愛上constraints

比如說，把Width的值改到100，canvas裡面的按鈕會變成下圖這個樣子：

原本「center」的constraint消失了，改為距左邊的Horizontal Space

這個按鈕同時也會產生了一個新的constraint，它強制使按鈕的寬度固定在100 points（可以看到按鈕下方的藍色欄）。





愛上constraints

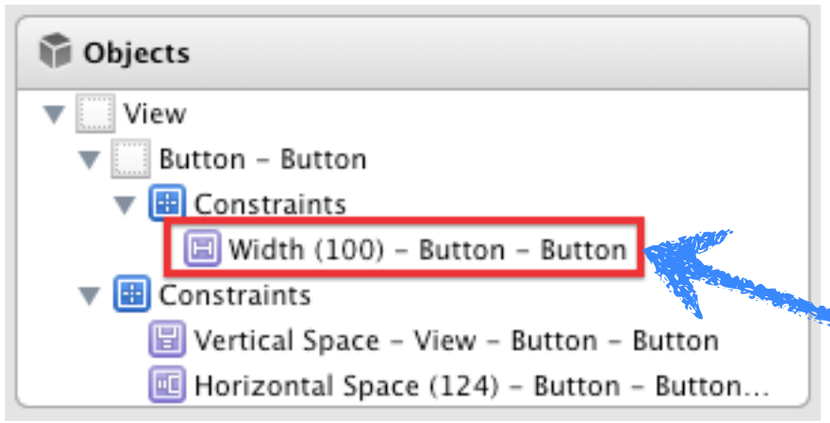
現在 Center X Alignment constraint的值消失了，

取而代之的是一個把按鈕連在屏幕左邊緣的Horizontal Space，

這個按鈕同時也會產生了一個新的constraint，

它強制使按鈕的寬度固定在100 points（可以看到按鈕下方的藍色欄）。

你在文檔概要圖的左邊看到現在有了一個新的Width constraint：



不像其他的constraint，那些是在按鈕和它的superview之間，這個寬度constraint只能應用於按鈕本身。你可以認為它是一個按鈕和按鈕之間的constraint。

拖動按鈕使它再次卡在 Center X Alignment constraint 上。

注意:

因為通過Size inspector來改變位置和大小可能會搞亂你的constraints，

我建議盡量不要這麼做，如果你非要改動布局，請更改constraints。

關於Width constraint

你現在可能想知道為什麼按鈕之前沒有一個Width constraint。

在沒有的情況下，Auto Layout是如何知道要改變按鈕的長度的呢?

可以這麼解釋:

這個按鈕自身知道他的寬度應該是多少，
它通過在它裡面的標題文字外加上一些圓角的邊距填充，可以計算出來寬度。
如果你設置了一個按鈕的背景圖片，它也會把這一點計算在內的。
這個現象被認為是固有內容尺寸。
不是所有的空間都會這樣，但是大部分是這樣的（UILable不在內）。

如果一個view能夠計算出它自己的尺寸，

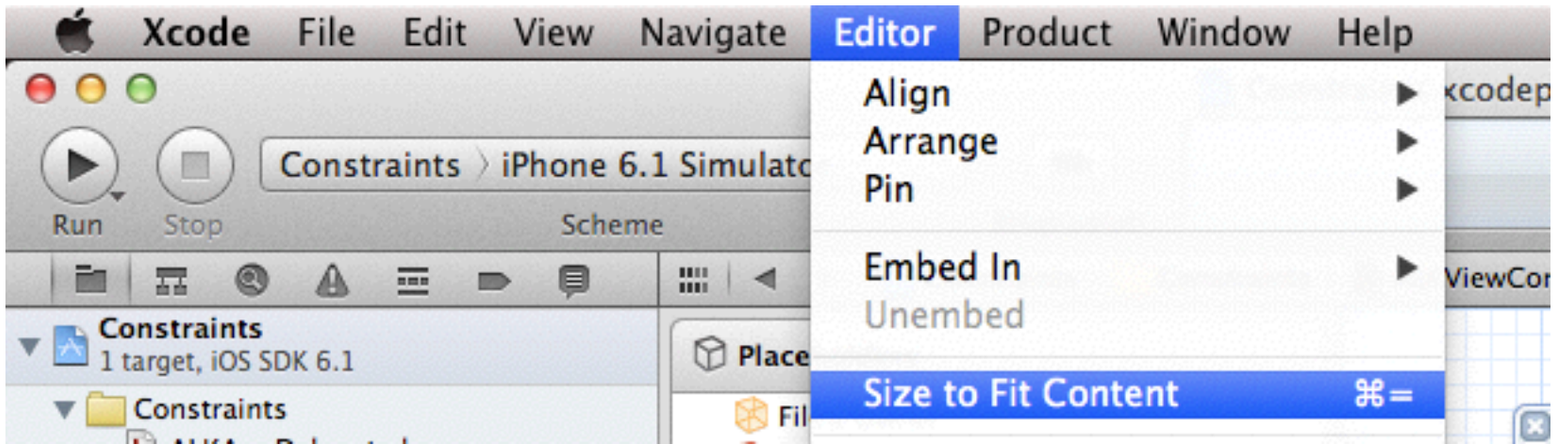
那麼你就沒有必要對其專門設置Width or Height constraints 了。

關於這個以後你就看得多了。

關於Width constraint

為了得到按鈕的最佳尺寸，選中它並且在Editor菜單裡將至設置為Size to Fit Content。

這步操作會使按鈕擺脫明確的Width constraint 並且將之恢復為按鈕的固有內容尺寸模式。



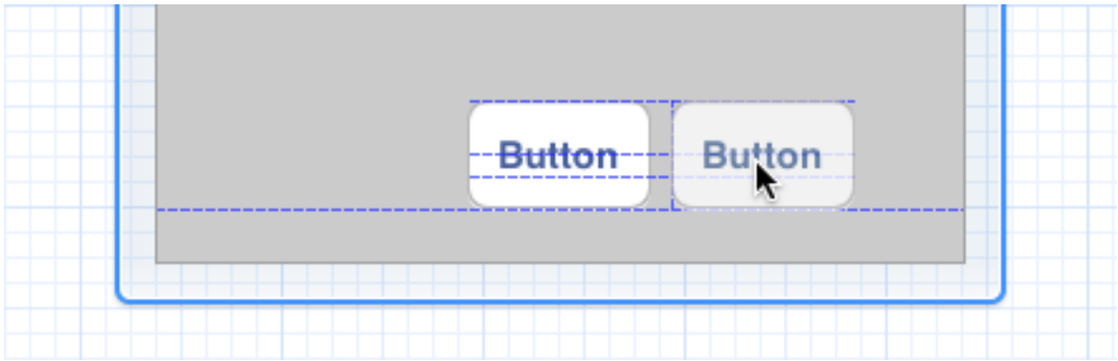
兩個按鈕

Guides 不僅可以出現在view與superview的，

也可以出現在同一階層的多個view之間。

為了證明這點，現在請在canvas裡面拖進一個新的圓角矩形按鈕。

如果你把這個按鈕放得離另外一個比較遠，這個按鈕回得到自己的constraints。然而，如果你把兩個按鈕放的足夠近，那麼這兩個按鈕的constraint會開始互相作用。



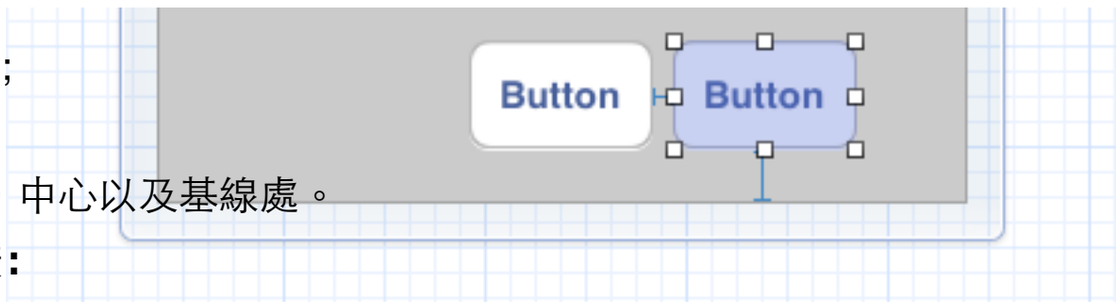
把新的按鈕捕捉到原來按鈕的旁邊:

現在這裡會出現一些點狀guidelines，

但Interface Builder不會把他們全部轉換成constraint；

因為這有一點多了。

但這基本會認出這兩個按鈕能在各個方位對齊 – 在他們的頂部，中心以及基線處。

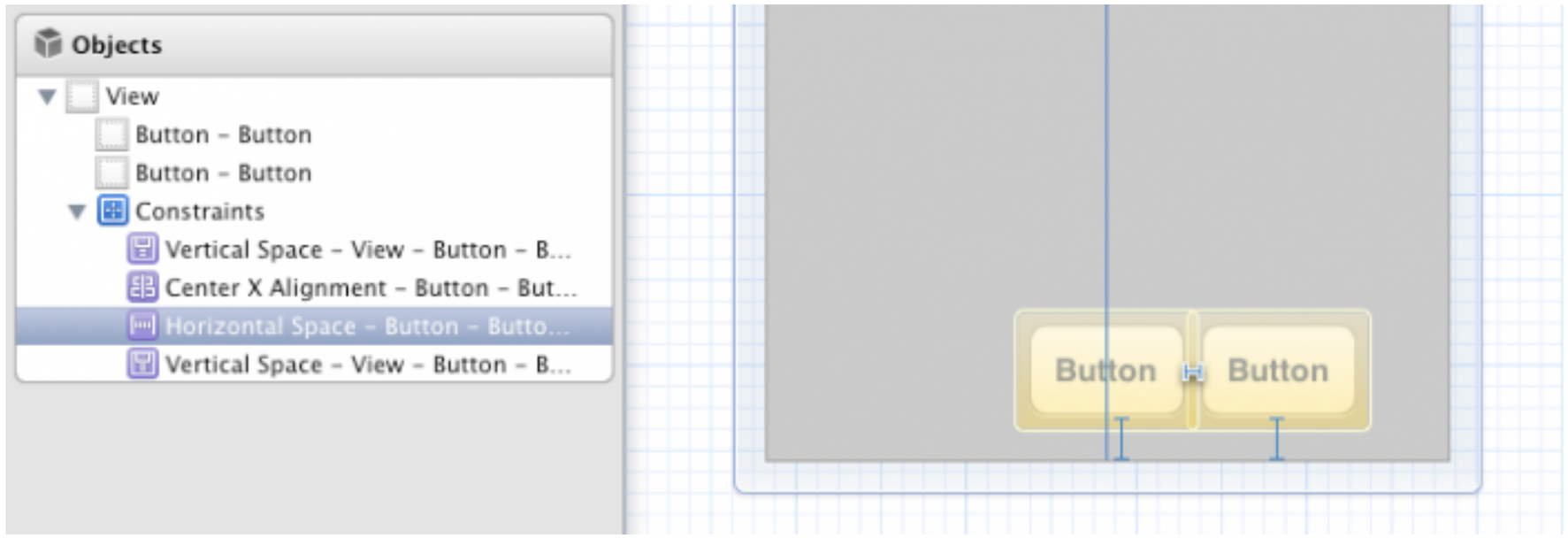


在把新按鈕放下後，這個按鈕的constraints會看上去像這樣:

兩個按鈕

如圖所示，新的按鈕有一個Vertical Space對於屏幕的底部，也有一個Horizontal Space對於另外一個按鈕。但是這個space是非常小的（只有8points），T型狀對象可能很難看到，但一定是存在於那裡的。

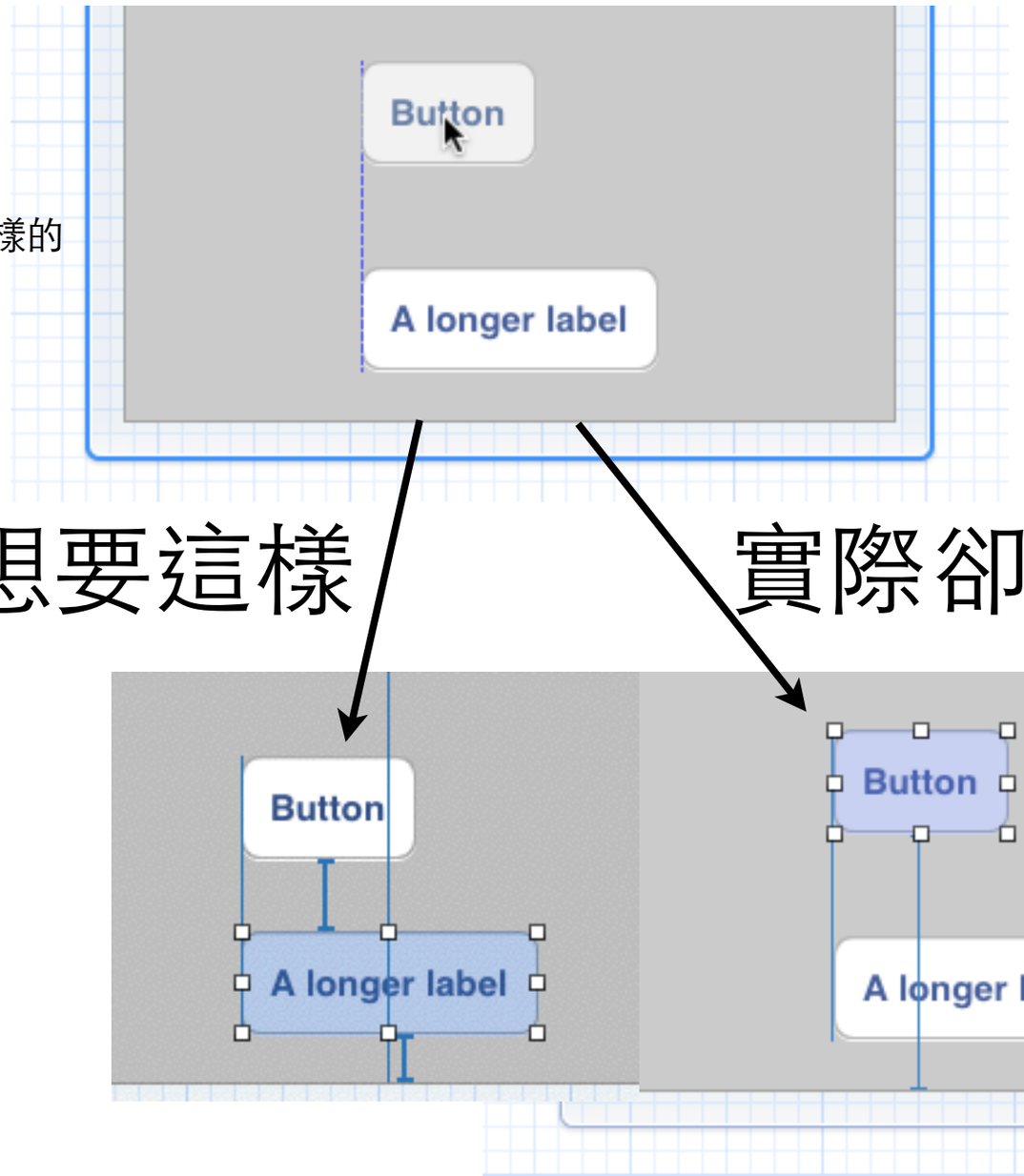
在文檔概要圖中選中Horizontal Space constraint ：



兩個按鈕

當你透過拖曳重新設定constraint時

你會發現，它有時候會跟你想的不一樣，這時的心情應該是這樣的



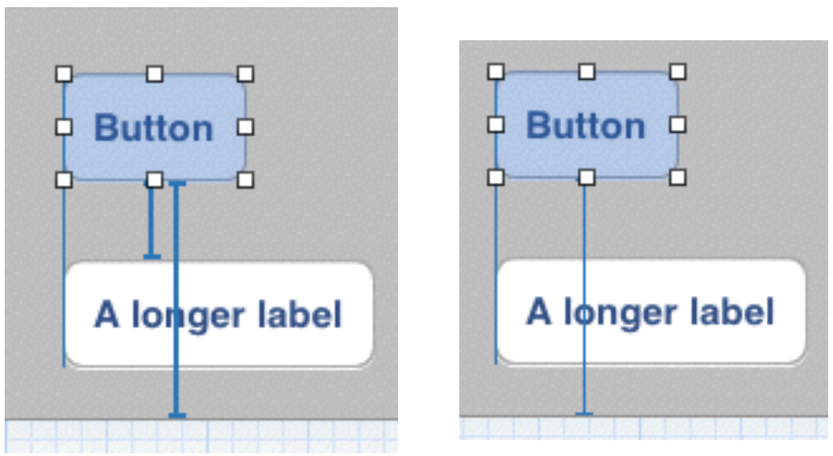
User Constraints

也許你已經注意在canvas裡面到有些T型狀對象看上去比其他的要粗一點。

這些加粗的，我們稱之為user constraints，你刪除它們後的效果和刪除細的是不同的。

當你刪除一個user constraints，Interface Builder經常會自動在刪除的地方放置一個不可刪除的constraint來代替之。我馬上就會講到為什麼會這樣。

在文檔概要圖中，user constraint有一個藍色按鈕：



User Constraints

在文檔概要圖中，user constraint有一個藍色按鈕：

新的constraint有一個紫色的按鈕，並且圖中它的線沒有被加粗，這也就代表著它是不可刪除的。

現在這兩個不再在垂直方向連接在一起了。

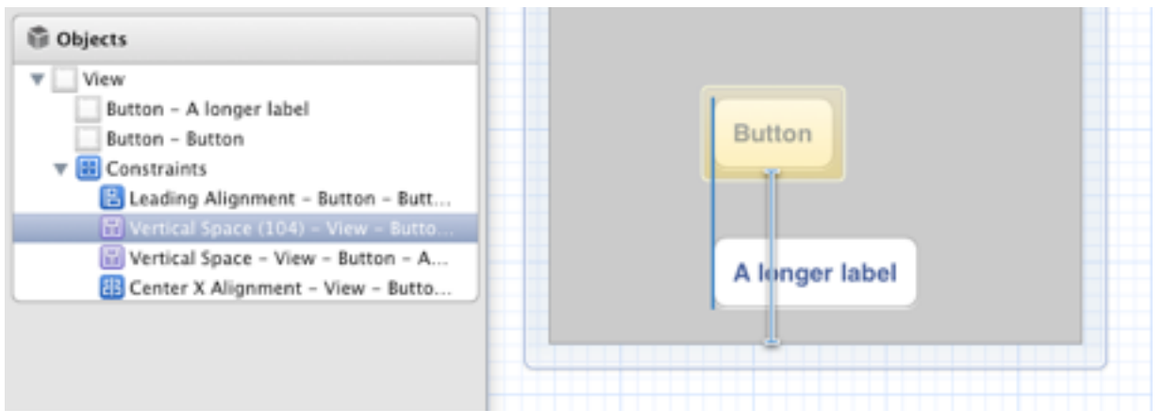
盡管由於Leading Alignment constraint 而依然左邊對齊著。

為什麼會發生這種事情？

為什麼Interface Builder對按鈕發出了一個新的 Vertical Constraint，

甚至你還沒有告訴它要刪除這麼一個constraint？答案是：

對於每一個view都必須要有足夠的constraints來對其進行位置以及大小的控制。



足夠的constraints

當用到 Auto Layout時，這是一條最重要的規則。

如果對於一個view沒有足夠的constraints，

它的Auto Layout將不能決定它的位置以及大小。

這種布局是被認為無效的。待會你會看到幾個無效布局的例子。

Interface Builder會盡量幫你避免布局無效。

這兩個按鈕的尺寸是可以知道的因為他們根據他們包含的文本，背景圖以及其他的一些東西–固定尺寸內容是可以確定下來他們的尺寸，還記得不？所以這不會是一個問題，

上方按鈕的X–位置也可以通過它的左邊界與下方按鈕的左邊界對齊來獲取，而下方按鈕又一直保持著底部居中。

現在唯一不確定的就是它的Y–位置。

之前的話，這兩個按鈕用一個 Vertical Space相連。

這個條件足夠能推導出上方按鈕的Y–位置。

但如果你刪除了Vertical Space，上方按鈕就沒有依據來定位它在view裡的垂直位置了。

因為Auto Layout不知道如何決定它的Y位置，所以它不能在屏幕中顯示出來。

為了避免這種情況發生，Interface Builder需要重新在view中找一個離按鈕底部邊界最近的地方「pin」它。

動態測試

現在你知道的一點基礎知識有：如何使用guides來放置控件，如何使他們相連對齊，如何在空間之間設置空白空間。在這篇教程後，你也會了解到Align and Pin 菜單的其他選項。

使用Interface Builder來主導constraints是非常不錯的, 但現在讓我們來看看，在程序運行的時候這個是如何工作的。

在?ViewController.m 中加入如下方法:

```
- (IBAction)buttonTapped:(UIButton *)sender
{
    if ([[sender titleForState:UIControlStateNormal] isEqualToString:@"X"])
        [sender setTitle:@"A very long title for this button"
                forState:UIControlStateNormal];
    else
        [sender setTitle:@"X" forState:UIControlStateNormal];
}
```

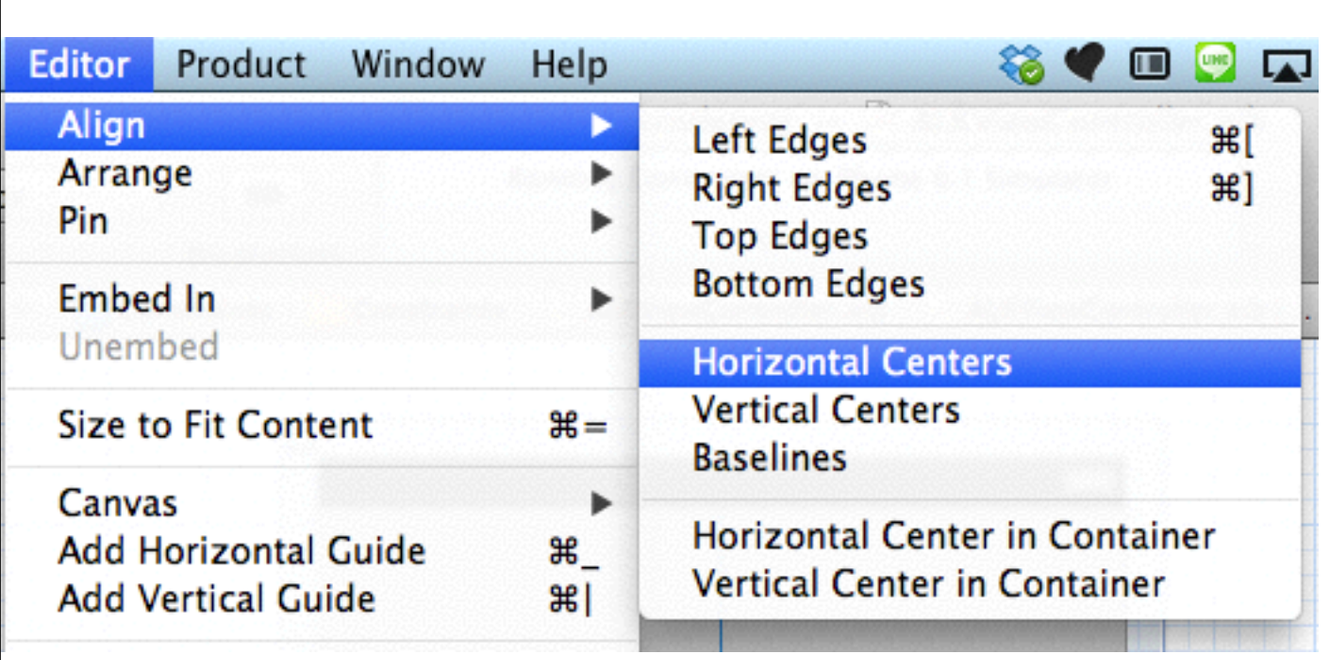
動態測試

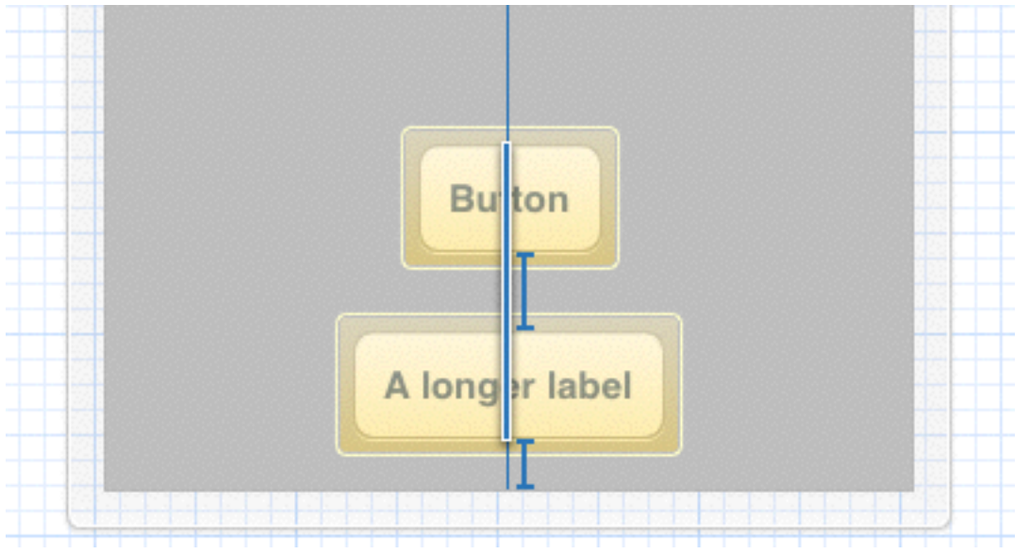
不論你觸碰哪個按鈕，現在皆以下方原則排版

- 在水平方向，下方的按鈕總是在窗口中水平居中。
- 下方的按鈕總是和窗口的底部保持20 points距離。
- 上方的按鈕總是和下方的保持左邊界對齊。

動態測試

現在同時選擇兩個按鈕，Editor > Align > Horizontal Centers





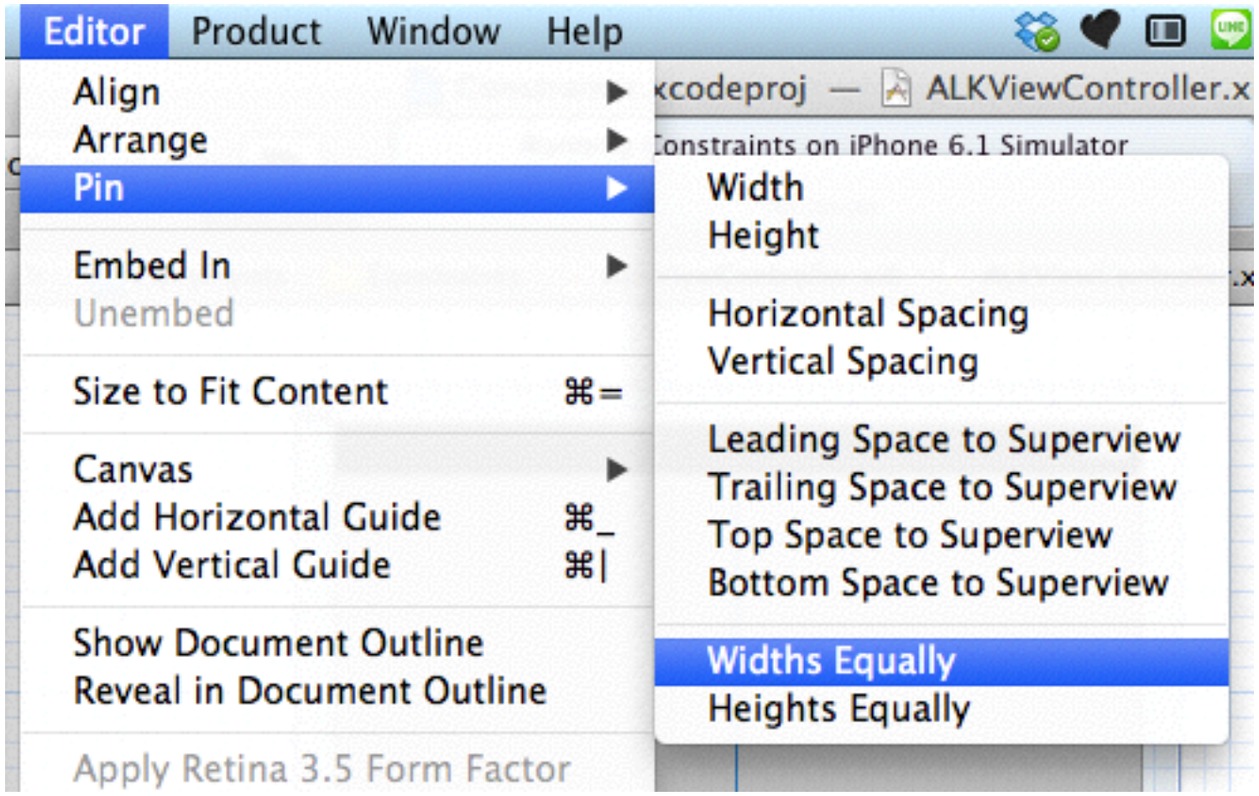
動態測試

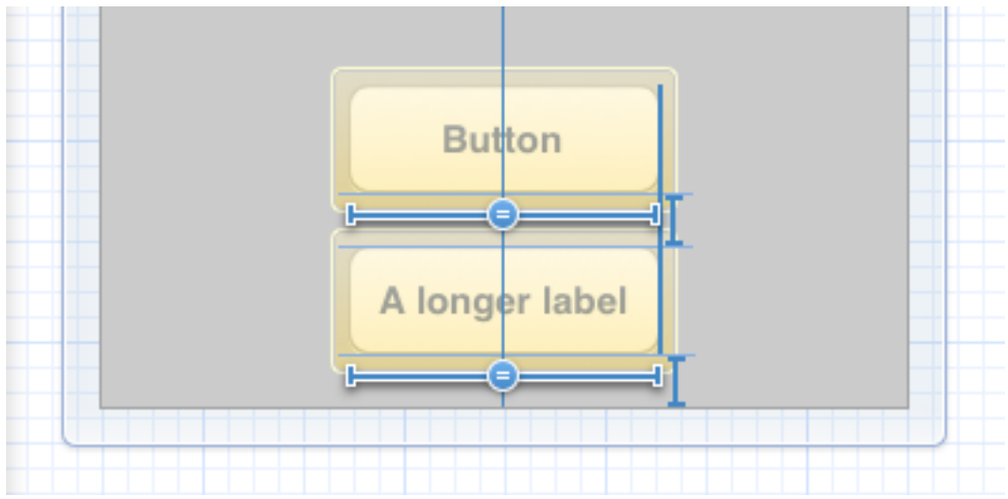
現在兩個都會居中對齊

修復寬度

如果我們想要兩個寬度都動態的一樣

試試看`Editor > Pin > Widths Equally`

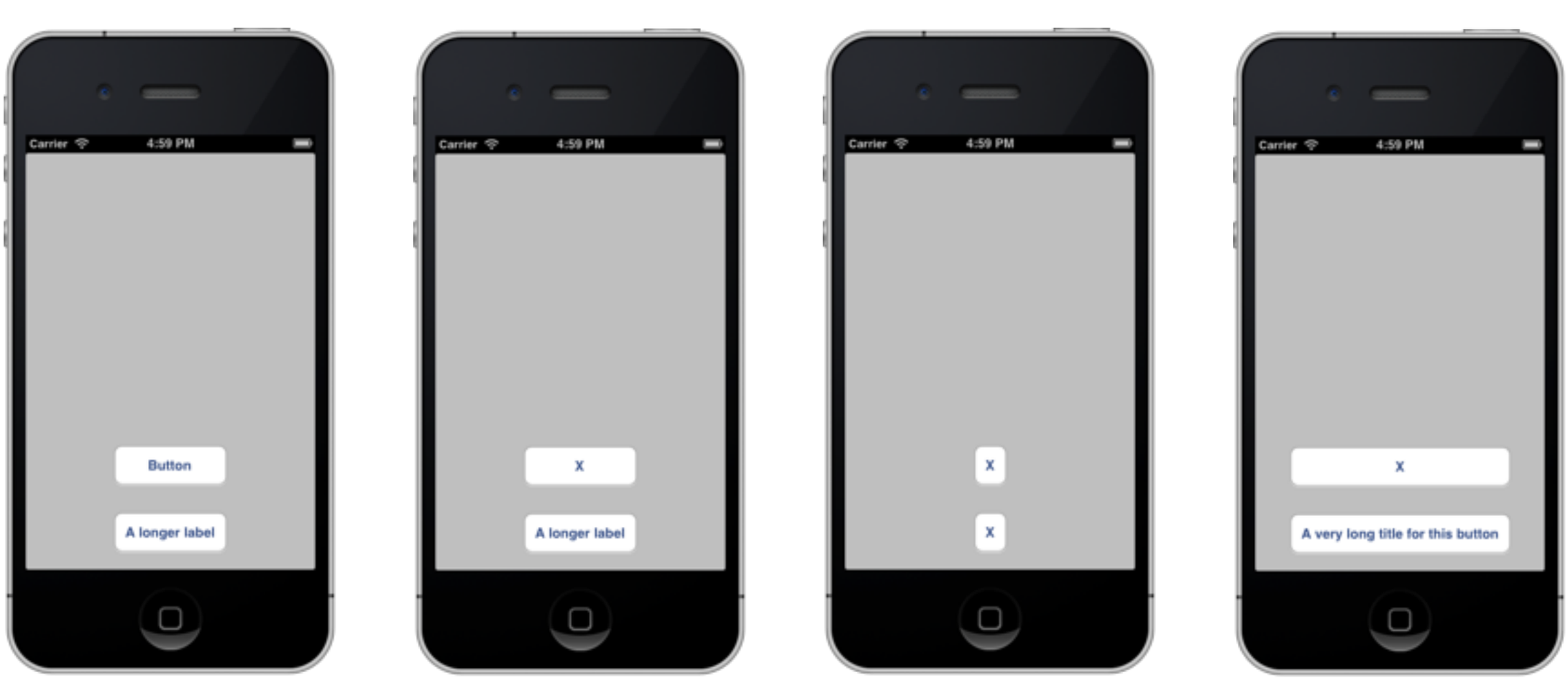




修復寬度

如果我們想要兩個寬度都動態的一樣

試試看`Editor > Pin > Widths Equally`



固有尺寸內容

固有內容尺寸

在Auto Layout功能出現之前，你通常會設置你的各種控件該有多大，或者在通過定制他們的frame、bounds屬性來改變大小，或者直接在Interface Builder來改變他們的大小。

但目前的情況是大部分的控件完全有能力針對它們的內容來計算出自身所要佔的空間大小。

一個label能夠通過設置在它上面的文本長度以及文本字體來計算出自己的寬和高。類似地，button,可以通過在它帶有背景的文本以及一些圓角的填充來計算出適合自己的寬和高。

這也適用於很多分段控件，如一些progress bars，還有許多的其他控件，盡管有一些是有一個預定義的高，但是它寬還是可以設置的。

這就是所謂的 固有尺寸內容, 在Auto Layout這是一個重要的概念。

在按鈕的操作中你已經見識到了吧。Auto Layout會先問你的空間他們有多大，然後在基於空間給出的信息將他們展示到屏幕上去。

你也可以不使用它，但你要明確設置這個空間的Width 或者 Height constraint。如果你這麼做了，那麼 Interface Builder 為自動生成一個你設置的constraint。

如果想要再次恢復固有內容尺寸的話，你只需重新使用**Size to Fit Content**操作，另外之前你設置的 Width or Height constraints會自動消失。通常來說，你使用固有內容尺寸功能就夠了，但有些情況下這功能還是不盡如人意的。想象一下，當你需要在UIImageView上設置一個image的時候，如果那個image要比屏幕大的多，你通常會給image設置一個合適的寬度以及高度來適應UIImageView的內容尺寸，除非你想讓imageview來自動幫你重新設置image的dimensions。

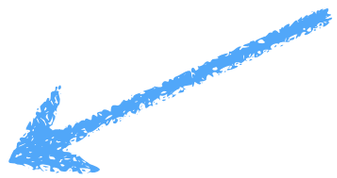
Intermediate Topics

Apple Document

https://developer.apple.com/library/ios/documentation/userexperience/conceptual/AutolayoutPG/Introduction/Introduction.html

Organization of This Document

- “Auto Layout Concepts”
- “Working with Constraints in Interface Builder”
- “Working with Auto Layout Programmatically”
- “Resolving Auto Layout Issues”
- “Auto Layout by Example
- “Implementing a Custom View to Work with Auto Layout”
- “Adopting Auto Layout”
- “Visual Format Language”



https://developer.apple.com/library/ios/documentation/userexperience/conceptual/AutolayoutPG/Introduction/Introduction.html

Advanced Topics

Best Practice

Article

10 Things You Need To Know About Cocoa Auto Layout

Turn auto layout flexible by manually code

Ole Begemann

iOS Development

10 Things You Need To Know About Cocoa Auto Layout

- 1. Rule of Thumb: At Least Two Constraints in Each Dimension
- 2. Embrace Intrinsic Size
- 3. IB Will Never Let You Create Ambiguous Layouts
- 4. Before You Can Delete An Existing Constraint, You Must Create Another One
- 5. Don’t Resize Controls Explicitly
- 6. Avoid Premature Optimization
- 7. Forget frame
- 8. Don’t Forget To Disable Autoresizing Masks
- 9. The Debugger Console Is Your Friend
- 10. Animate the Constraints, Not the Frame

Article

How I Learned to Stop Worrying and Love Cocoa Auto Layout

Turn auto layout flexible by manually code

Ole Begemann

iOS Development

How I Learned to Stop Worrying and Love Cocoa Auto Layout

Turn auto layout flexible by manually code

- Seems to create more problems than it solves

- Mix Auto Layout With Manual Layout Code

- Just Another Step In layoutSubviews

- Override layoutSubviews

Seems to create more problems than it solves

And while Auto Layout clearly is the future and I strongly encourage you to use i
wherever practical, there are some cases where Auto Layout seems to create mo
problems than it solves

- Animating views under Auto Layout requires you to animate the constraints directly, which can be pain.
- Auto Layout does not play nicely with view transforms.
- Auto Layout can be too slow.

Mix Auto Layout With Manual Layout Code

- Consider this example: You are using Auto Layout to position all but one of them.

- Xcode 5 tries to adds missing constraints at build time. So you should…

- set translatesAutoresizingMaskIntoConstraints = NO

Just Another Step In layoutSubviews

- Auto Layout as just an additional step that runs automatically in your view's layoutSubviews method

- What you do to your subviews' frames after Auto Layout has done its job, doesn't matter.

Override layoutSubviews

Turn auto layout flexible
by manually code

- All you now have to do is override layoutSubviews (or viewDidLayoutSubviews if you want to do this in a view controller)

```
- (void)layoutSubviews
{
    // Important. This lets the Auto Layout engine do its job.
    [super layoutSubviews];

    // Now all subview frames are set according to their constraints.
    // We can freely reposition and/or resize any view here, even one that has already been
    // positioned with Auto Layout (that is not recommended, though).

    // Here, we position and size a view we want to lay out manually.
    // Don't modify the frame directly because the view has a transform applied.
    self.manualLayoutView.bounds = CGRectMake(...);
    self.manualLayoutView.center = CGPointMake(...);
}
```

Article

Autolayout vs. View Transforms

Auto Layout does not play nicely with view transforms

http://stackoverflow.com/questions/12943107/how-do-i-adjust-the-anchor-point-of-a-calayer-when-auto-layout-is-being-used/14105757#14105757

Autolayout vs. View Transforms

- Solution 1: No Constraints
- Solution 2: Use Only Appropriate Constraints
- Solution 3: Use a Subview
- Solution 4: Use Layer Transforms Instead

http://stackoverflow.com/questions/12943107/how-do-i-adjust-the-anchor-point-of-a-calayer-when-auto-layout-is-being-used/14105757#14105757

Debugging

Issues With Achieving Auto Layout Zen

- Debugging is calling [self.view _autolayoutTrace] on LLDB

- UIConstraintBasedLayoutDebugging has three methods for trying to debug constraints

Commets

- I think they are a fundamental aspect of development going forward that will be difficult to avoid using.

carpeaqua

Guru Topics

Background Theory

it solves a system of linear equations

Cassowary, Cocoa Autolayout, and enaml constraints

it solves a system of linear equations

http://stacks.11craft.com/cassowary-cocoa-autolayout-and-enaml-constraints.html

Efficiency

Auto Layout Performance on iOS

http://floriankugler.com/blog/2013/4/21/auto-layout-performance-on-ios

Resources

基本觀念

- Furnace iOS
    - http://furnacedigital.blogspot.tw/2013/10/auto-layout.html
    - 簡介與中文說明

- Beginning Auto Layout Tutorial in iOS 7: Part 1
    - http://www.raywenderlich.com/50317/beginning-auto-layout-tutorial-in-ios-7-part-1
    - 程式實作教學

- Autolayout guide
    - Apple
    - https://developer.apple.com/library/ios/documentation/userexperience/conceptual/AutolayoutPG/Introduction/Introduction.html
    - 官方文件