



Auto Layout

Sergio Padrino Recio



¿Qué es?

- Sistema de layout por excelencia en plataformas de Apple.
- Disponible desde iOS 6
- Basado en definir “restricciones” (*constraints*) entre elementos de nuestra UI y que todo funcione “automáticamente”

¿Qué es?

Ejemplo de restricciones:

- Quiero un botón pegado a la izquierda de la pantalla.
- Quiero que ese botón tenga un ancho fijo de 200.
- Quiero una etiqueta cuyo borde izquierdo esté alineado con el borde izquierdo del botón.
- Quiero que el borde derecho de la etiqueta también esté alineado con el borde derecho del botón.

Constraints

- Son las “restricciones” que aplicamos a los elementos de nuestra UI.
- Equivalen a (in)ecuaciones con la siguiente estructura:

`element1.attribute [=|<|>] multiplier * element2.attribute + constant`

Constraints

- `element1` y `element2`: elementos de nuestra UI (botones, labels...). Pueden ser el mismo.
- `attribute`: son atributos geométricos de los elementos (`top`, `bottom`, `left`, `right`, `width`, `height`)
- `multiplier`: multiplicador para el atributo del segundo elemento.

Constraints

- `constant`: constante que siempre será sumada al resultado final.
- `[=|<|>]`: significa que podemos “pedir” a Auto Layout que `element1.attribute` sea **igual**, **menor que** o **mayor que** el resultado de la parte derecha.

Constraints

Ejemplos:

- Quiero que el label sea **al menos** el triple de ancho que el botón, más 5.

```
label.width > 3 * button.width + 5
```

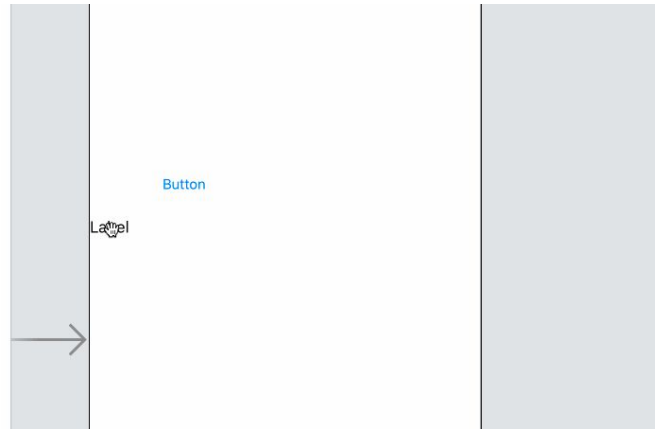
- Quiero que la imagen sea el doble de alta que de ancha:

```
image.height = 2 * image.width
```

Auto Layout desde Interface Builder

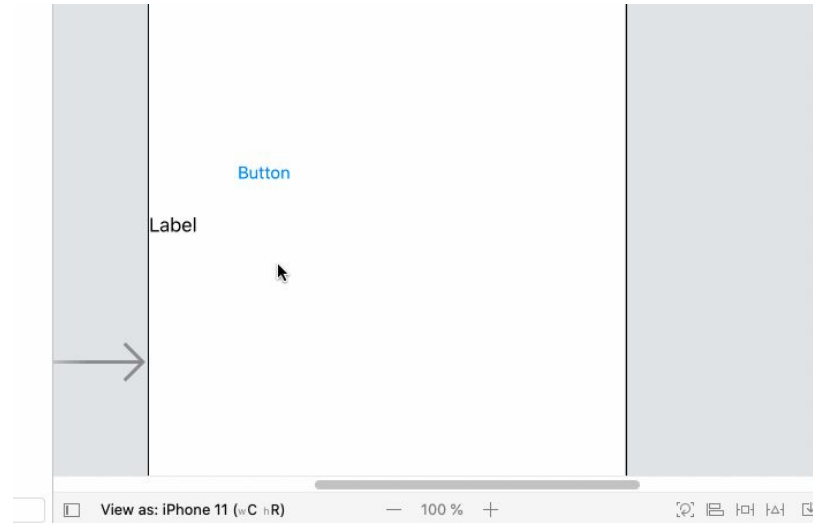
Dos maneras de añadir *constraints* a nuestra UI con Interface Builder:

- Arrastrando con el botón derecho del ratón de un elemento a otro. Se mostrará un menú con los distintos tipos de *constraints* que podemos crear.



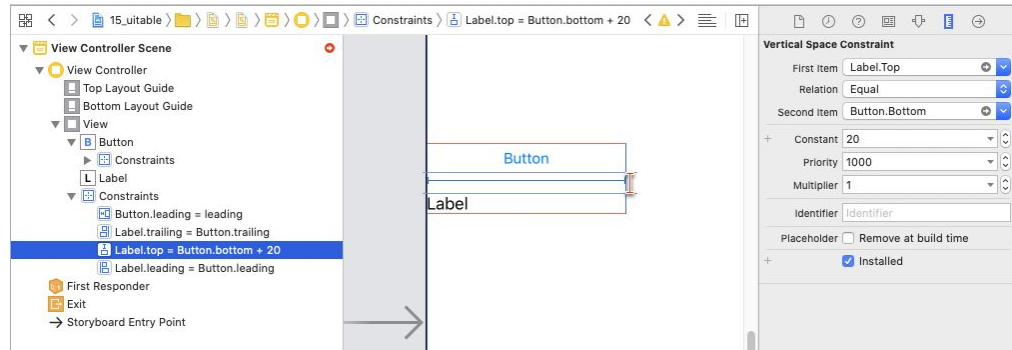
Auto Layout desde Interface Builder

- Seleccionando un elemento y usando el botón “Add New Constraints”. Se mostrará un menú con las distintas *constraints* que podemos crear.



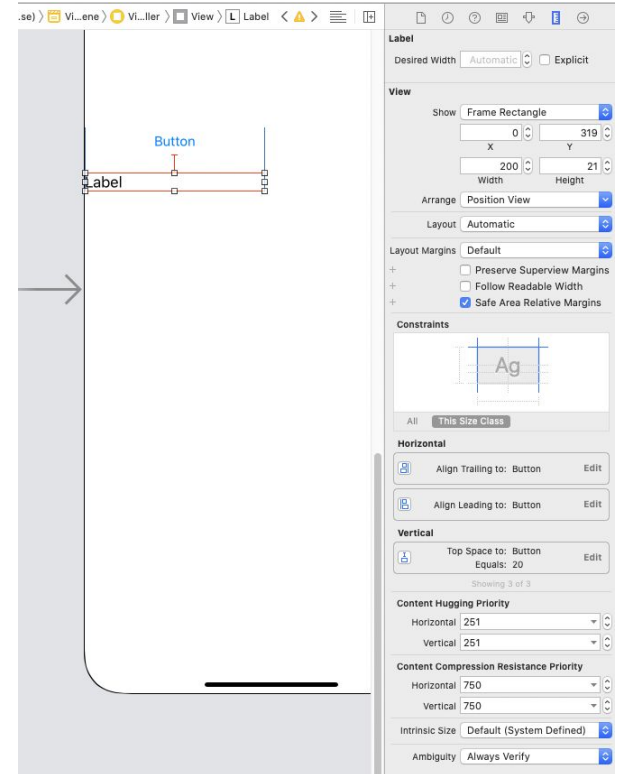
Auto Layout desde Interface Builder

- Podemos seleccionar *constraints* existentes directamente desde el “canvas”, o también desde la jerarquía de vistas a la izquierda.
- En el panel de la derecha podremos acceder a sus propiedades (elementos, atributos, relación, constante, multiplicador...)



Auto Layout desde Interface Builder

- Seleccionando el elemento, a la derecha veremos todas sus *constraints*.
- Podemos hacer doble click sobre ellas o editarlas directamente pulsando en "Edit"



¿Cómo funciona?

- Internamente, se trata de resolver un sistema de (in)ecuaciones de múltiples variables.

$$\begin{cases} 5x + 6y = 20 \\ 3x + 8y = 34 \end{cases}$$

- Cada *constraint* que añadamos es una (in)ecuación.
- El motor de layout usará esas ecuaciones para calcular el resto de parámetros de la geometría de nuestros elementos de UI.

¿Cómo funciona?

Ejemplo (simplificado):

- Un botón colocado a la izquierda:
`button.left = 0`
- Ancho fijo 200 para el botón:
`button.width = 200`
- Anclamos un label a su lado izquierdo:
`label.left = button.left`
- Anclamos el mismo label a su lado derecho:
`label.right = button.right`

¿Cómo funciona?

¿Cuál es el ancho final del label? Resolvamos el sistema:

- Sabemos que `label.width = label.right - label.left`
- `label.left = button.left = 0`
- `label.right = button.right =`
`= button.left + button.width = 0 + 200 = 200`
- Por tanto `label.width = 0 + 200 = 200`

¿Cómo funciona?

- Si aplicamos este mismo proceso a todas nuestras *constraints* podremos averiguar la geometría (coordenadas y tamaño) de todos los elementos de nuestra UI...
- ...en teoría 🤖

Layouts ambiguos

- Cuando añadimos *constraints*, tenemos en mente un resultado concreto para nuestra UI.
- Sin embargo, los sistemas de (in)ecuaciones a veces tienen **varios resultados**.

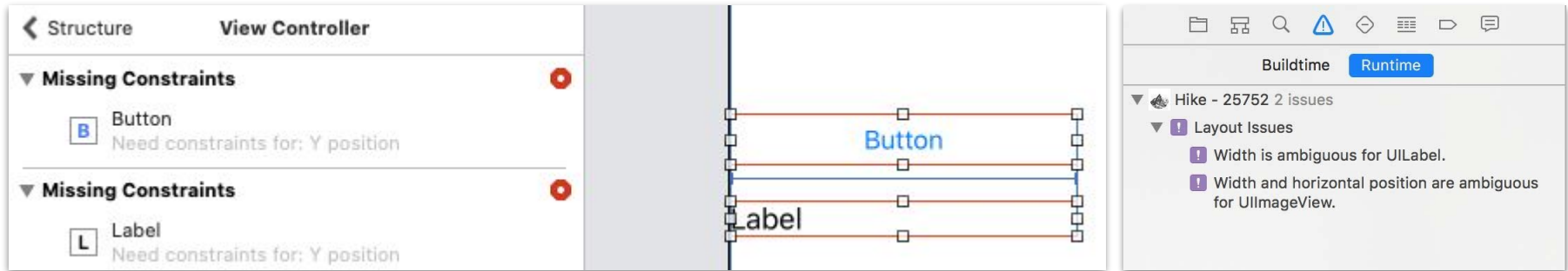
$$x + y = 5$$

$$x - y > 0$$

x	y	x+y	x-y
5	0	5	5
4	1	5	3
3	2	5	1
10	-5	5	15

Layouts ambiguos

- iOS no sabrá dónde o de qué tamaño pintar nuestros componentes...
- ...y Xcode nos lo hará saber.



Layouts ambiguos: motivos

Existen varias maneras de llegar a esta situación donde hay múltiples soluciones:

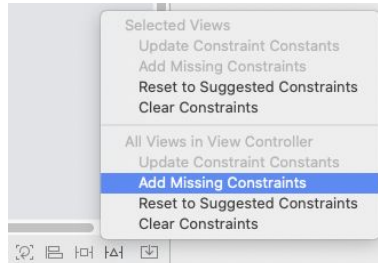
- Uso de inecuaciones (*menor que* o *mayor que*) en nuestras *constraints*.
- Faltan *constraints* para definir alguno de los atributos. En nuestro ejemplo con el botón y la etiqueta, definimos el ancho y las coordenadas horizontales, pero no la altura ni las coordenadas verticales.

Layouts ambiguos: motivos

- Que **aparentemente** estén todas las *constraints* necesarias pero los elementos “compitan” por el espacio.
- Que algún elemento no defina bien el tamaño de su contenido.
- Que varias *constraints* sean contradictorias (como fijar el ancho de un elemento a dos valores distintos).

Layouts ambiguos

- Casi todos estos problemas se pueden arreglar añadiendo más *constraints*.
- Por eso Xcode nos da un botón que arreglará todos esos problemas:



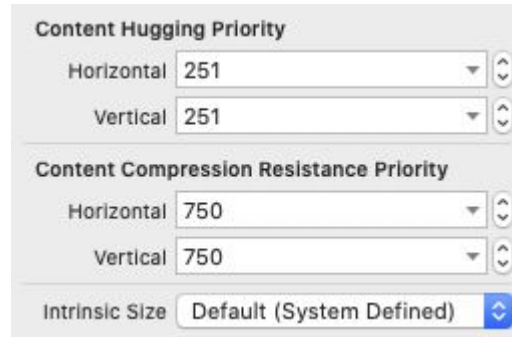
- Sin embargo, añadir *constraints* también puede añadir otros problemas (rendimiento, complejidad) y ocultar los problemas originales.

Tamaño del contenido

- Algunos elementos poseen un tamaño **intrínseco** por su contenido: imágenes, textos, botones...
- En estos casos, no necesitaremos definir *constraints* para el tamaño de los mismos si no queremos.
- Sí podremos definir cómo queremos que se comporte Auto Layout con ese contenido cuando varios elementos “compitan” por el espacio.

Tamaño del contenido

- Para ello haremos uso del Content Hugging y el Compression Resistance



The image shows a settings panel with three sections. The first section, 'Content Hugging Priority', has 'Horizontal' and 'Vertical' both set to '251'. The second section, 'Content Compression Resistance Priority', has 'Horizontal' and 'Vertical' both set to '750'. The third section, 'Intrinsic Size', is set to 'Default (System Defined)'.

Content Hugging Priority	
Horizontal	251
Vertical	251

Content Compression Resistance Priority	
Horizontal	750
Vertical	750

Intrinsic Size	
	Default (System Defined)

Content Hugging Priority

- Indica cómo de importante es que el tamaño del elemento se ajuste al tamaño de su contenido (lo “abrace”).
- Cuanto más alto sea el valor, más probabilidades hay de que el elemento se ajuste a su contenido.
- Cuanto más bajo sea el valor, más probabilidades hay de que el elemento emplee más espacio del que realmente necesita.

Content Hugging Priority



Compression Resistance Priority

- Indica cómo de importante es que el tamaño del elemento no sea menor que el de su contenido (resistencia a ser comprimido).
- Cuanto más alto sea el valor, más difícil es que el elemento sea más pequeño que su contenido y haya que truncarlo.
- Cuanto más bajo sea el valor, más fácil es que el elemento se comprima y sea necesario truncar el contenido.

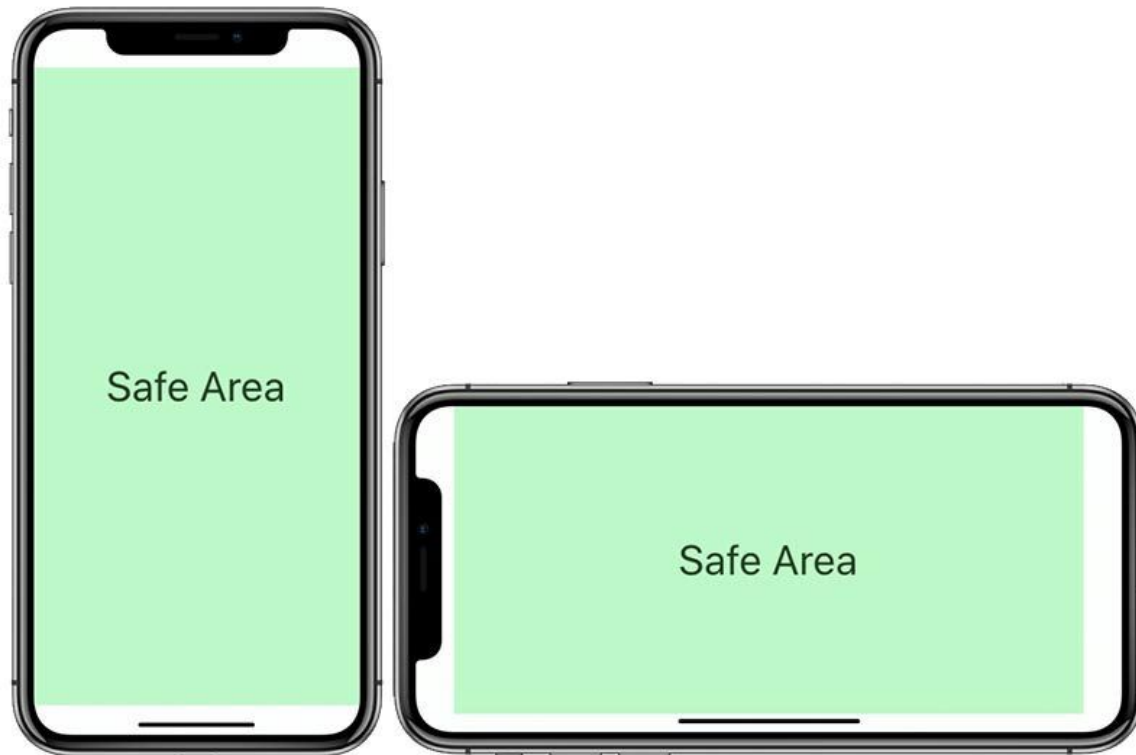
Compression Resistance Priority



Safe Area

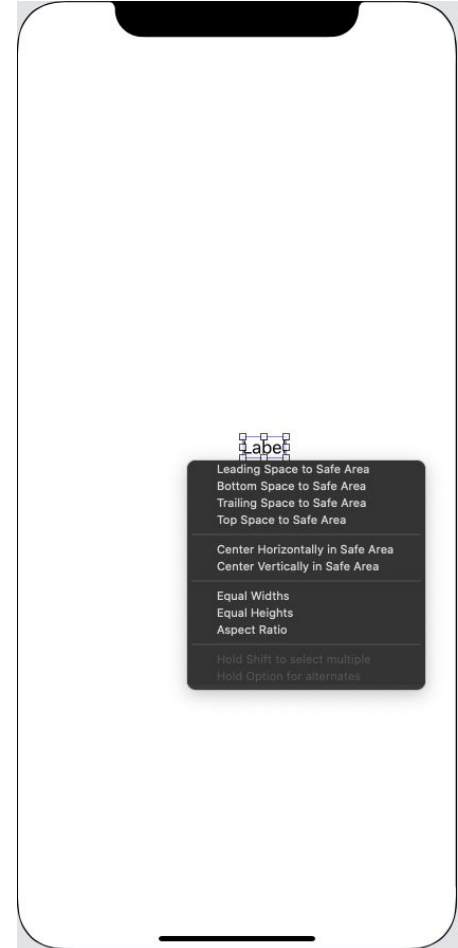
- Con la introducción de iPhone X, llegaron nuevos “retos” a nuestras UIs.
- Ahora nuestro espacio disponible para pintar UI está recortado:
 - Para redondear las esquinas.
 - Por la muesca o *notch* de la cámara y el auricular
 - En la parte de abajo debemos dejar espacio para el reemplazo del antiguo botón *Home*.

Safe Area



Safe Area

- Debemos evitar que haya componentes interactivos en esas zonas.
- Para ello podemos anclar nuestras *constraints* al Safe Area en lugar de a los lados o márgenes de la pantalla.



¿Leading y Trailing?

- El uso de Auto Layout también gestiona cierta semántica por nosotros.
- Una de las características de iOS es su soporte para idiomas RTL (Right-to-Left).
- En estos casos, iOS automáticamente invertirá toda la UI (gestos incluidos) horizontalmente.



¿Leading y Trailing?

- Teniendo esto en mente anclar vistas a la izquierda o a la derecha de otro elemento rompería la compatibilidad con idiomas RTL.
- En su lugar, Auto Layout ofrece los conceptos de **leading** (inicio) y **trailing** (final) que equivalen a:
 - Izquierda y derecha, respectivamente, en idiomas LTR.
 - Derecha e izquierda, respectivamente, en idiomas RTL.
- Solo debemos usar *left* y *right* cuando sea exactamente lo que buscamos.

Auto Layout desde código

- Antes de configurar *constraints*, hay que desactivar el mecanismo “antiguo” de layout: las *autoresizing masks*.
- Para ello hay que decirle a iOS que no convierta dichas *autoresizing masks* a *constraints* para evitar conflictos con nuestras propias *constraints*:

```
myView.translatesAutoresizingMaskIntoConstraints = false
```


Auto Layout desde código

- Se pueden crear constraints utilizando el constructor de `NSLayoutConstraint...` pero no es lo más sencillo.
- En su lugar, a partir de iOS 9 las vistas tienen una serie de *anchors* que podemos utilizar para anclar distintos atributos entre elementos.

Auto Layout desde código

```
extension UIView {  
  
    /* Constraint creation conveniences. See NSLayoutAnchor.h for details.  
    */  
    @available(iOS 9.0, *)  
    open var leadingAnchor: NSLayoutXAxisAnchor { get }  
  
    @available(iOS 9.0, *)  
    open var trailingAnchor: NSLayoutXAxisAnchor { get }  
  
    @available(iOS 9.0, *)  
    open var leftAnchor: NSLayoutXAxisAnchor { get }  
  
    @available(iOS 9.0, *)  
    open var rightAnchor: NSLayoutXAxisAnchor { get }  
  
    @available(iOS 9.0, *)  
    open var topAnchor: NSLayoutYAxisAnchor { get }  
  
    @available(iOS 9.0, *)  
    open var bottomAnchor: NSLayoutYAxisAnchor { get }  
  
    @available(iOS 9.0, *)  
    open var widthAnchor: NSLayoutDimension { get }  
  
    @available(iOS 9.0, *)  
    open var heightAnchor: NSLayoutDimension { get }  
  
    @available(iOS 9.0, *)  
    open var centerXAnchor: NSLayoutXAxisAnchor { get }  
  
    @available(iOS 9.0, *)  
    open var centerYAnchor: NSLayoutYAxisAnchor { get }  
  
    @available(iOS 9.0, *)  
    open var firstBaselineAnchor: NSLayoutYAxisAnchor { get }  
  
    @available(iOS 9.0, *)  
    open var lastBaselineAnchor: NSLayoutYAxisAnchor { get }  
}
```

Auto Layout desde código

- Estos *anchors* tienen múltiples métodos que empiezan por *constraint* y que nos permitirán anclarlos a otros *anchors*.

```
button.leadingAnchor.constraint|
[M] NSLayoutConstraint constraintEqualToSystemSpacingAfter(anchor: NSLayoutXAxisAnchor, multiplier: CGFloat)
[M] NSLayoutConstraint constraintLessThanOrEqualToSystemSpacingAfter(anchor: NSLayoutXAxisAnchor, multiplier: CGFloat)
[M] NSLayoutConstraint constraintGreaterThanOrEqualToSystemSpacingAfter(anchor: NSLayoutXAxisAnchor, multiplier: CGFloat)
[M] NSLayoutConstraint constraint(equalTo: NSLayoutAnchor<NSLayoutXAxisAnchor>)
[M] NSLayoutConstraint constraint(equalTo: NSLayoutAnchor<NSLayoutXAxisAnchor>, constant: CGFloat)
[M] NSLayoutConstraint constraint(lessThanOrEqualTo: NSLayoutAnchor<NSLayoutXAxisAnchor>)
[M] NSLayoutConstraint constraint(greaterThanOrEqualTo: NSLayoutAnchor<NSLayoutXAxisAnchor>)
[M] NSLayoutConstraint constraint(lessThanOrEqualTo: NSLayoutAnchor<NSLayoutXAxisAnchor>, constant: CGFloat)
Returns a constraint that defines by how much the current anchor trails the specified anchor.
```

Auto Layout desde código

- Dichos métodos devolverán una constraint que, después, tendremos que activar para que sea efectiva.

```
let constraint = label.leadingAnchor.constraint(equalTo: button.leadingAnchor)  
constraint.isActive = true
```

- También podemos activar varias constraints de una vez:

```
NSLayoutConstraint.activate([ constraint1, constraint2, ... ])
```

Auto Layout desde código

- Una vez la *constraint* ha sido creada, podemos modificar su *constant*:

```
constraint.constant = 5.0
```

- Sin embargo, **no** podemos modificar su *multiplier*:

```
constraint.multiplier = 4.0 // ERROR
```

Auto Layout desde código

- Podemos cambiar el Content Hugging y Compression Resistance de las vistas:

```
label.setContentHuggingPriority(.required, for: .horizontal)
label.setContentHuggingPriority(UILayoutPriority(rawValue: 500), for: .vertical)
label.setContentCompressionResistancePriority(.defaultLow, for: .horizontal)
label.setContentCompressionResistancePriority(.defaultHigh, for: .vertical)
```

Auto Layout desde código

- Podemos cambiar el Content Hugging y Compression Resistance de las vistas

```
label.setContentHuggingPriority(.required, for: .horizontal)
label.setContentHuggingPriority(UILayoutPriority(rawValue: 500), for: .vertical)
label.setContentCompressionResistancePriority(.defaultLow, for: .horizontal)
label.setContentCompressionResistancePriority(.defaultHigh, for: .vertical)
```

Animaciones con Auto Layout

Dos sencillos pasos:

1. Se modifican las *constraints* como se desee para el resultado final (añadir, borrar, activar, desactivar, cambiar constantes...).
2. Se invoca el método `layoutIfNeeded` de la vista dentro de un bloque de animaciones.

Animaciones con Auto Layout

```
leadingConstraint.constant += 5.0
topConstraint.isActive = false
let newConstraint = someView.leftAnchor.constraint...
newConstraint.isActive = true

UIView.animate(withDuration: 2.0) {
    self.layoutIfNeeded()
}
```

Auto Layout en custom views

- Crearemos subvistas y añadiremos *constraints* con normalidad.
- La principal diferencia: debemos definir el tamaño *intrínseco* del contenido, si lo hay.
- Esto se hace sobrescribiendo la propiedad `intrinsicContentSize`, que devolverá un `CGSize` con el tamaño del contenido.

Auto Layout en custom views

```
class CustomLabel: UIView
{
    override var intrinsicContentSize: CGSize
    {
        get
        {
            let textSize: CGSize =
                calculateSize(forText: self.text, font: self.font) // Made-up function
            return textSize
        }
    }
}
```

Auto Layout en custom views

- Si el contenido no tiene un tamaño *intrínseco* (en una o ambas dimensiones), podemos simplemente devolver un `CGSize` con `UIViewNoIntrinsicMetric` en la anchura y/o altura:

```
override var intrinsicContentSize: CGSize
{
    get
    {
        return CGSize(width: UIViewNoIntrinsicMetric, height: UIViewNoIntrinsicMetric)
    }
}
```

Auto Layout en custom views

- Cuando el contenido cambie, debemos informar al motor de layout.
- Para ello, llamaremos al método `invalidateIntrinsicContentSize`:

```
public var text: String
{
    didSet {
        self.invalidateIntrinsicContentSize()
    }
}
```

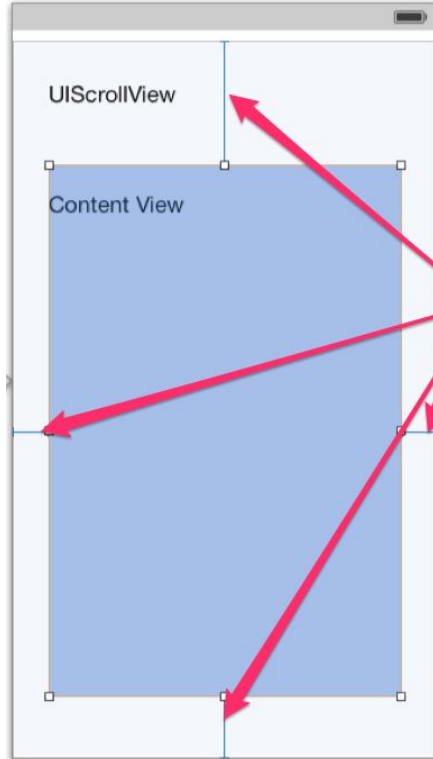
Auto Layout en UIScrollView

- Los scroll views son un caso especial con Auto Layout.
- Las subvistas del scroll view deben tener *constraints* suficientes para que:
 - El layout no sea ambiguo, obviamente.
 - Las aristas (leading, trailing, top, bottom) o las dimensiones (width, height) del scroll view estén ancladas a dichas subvistas.

Auto Layout en UIScrollView

- Esto último es muy importante, porque aunque las subvistas estén ancladas mediante *constraints* al scroll view, iOS internamente ignorará esas *constraints* y simplemente las utilizará para calcular el `contentSize` del scroll.
- Es decir, esas *constraints* realmente se estarán anclando no al scroll view, sino a los límites de su contenido scrolleable.

Auto Layout en UIScrollView



Estas son *constraints* especiales que le dirán al UIScrollView cuál es el tamaño del contenido.

No tienen ningún efecto en el tamaño o la posición de *content view*.

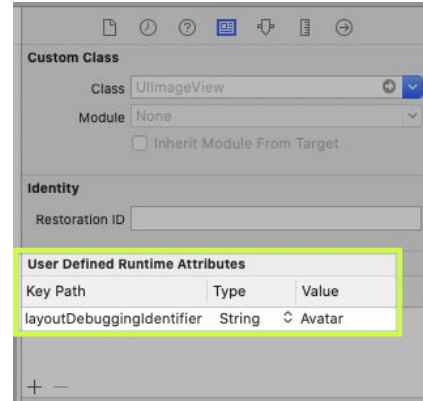
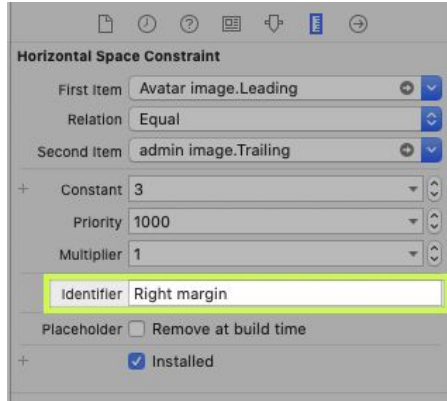
Depurando Auto Layout

- En general, depurar problemas con Auto Layout puede ser muy complejo.
- No disponemos muchas herramientas para ayudarnos en esta tarea...
- ...salvo que el problema sean layouts ambiguos.

Depurando Auto Layout: consejos generales

- Dar identificadores a *constraints* y vistas para hacer los logs más legibles:

```
<NSLayoutConstraint:0x600003bee080 H:[UIImageView:0x7feccf5db670]-(3)-[UIImageView:0x7feccf5da350]  
(active)>
```



```
<NSLayoutConstraint:0x600000ac8be0 'Right margin' H:[Badge]-(3)-[Avatar]  
(active, names: Avatar:0x7ffb4d7b6740, Badge:0x7ffb4fa9b000 )>
```

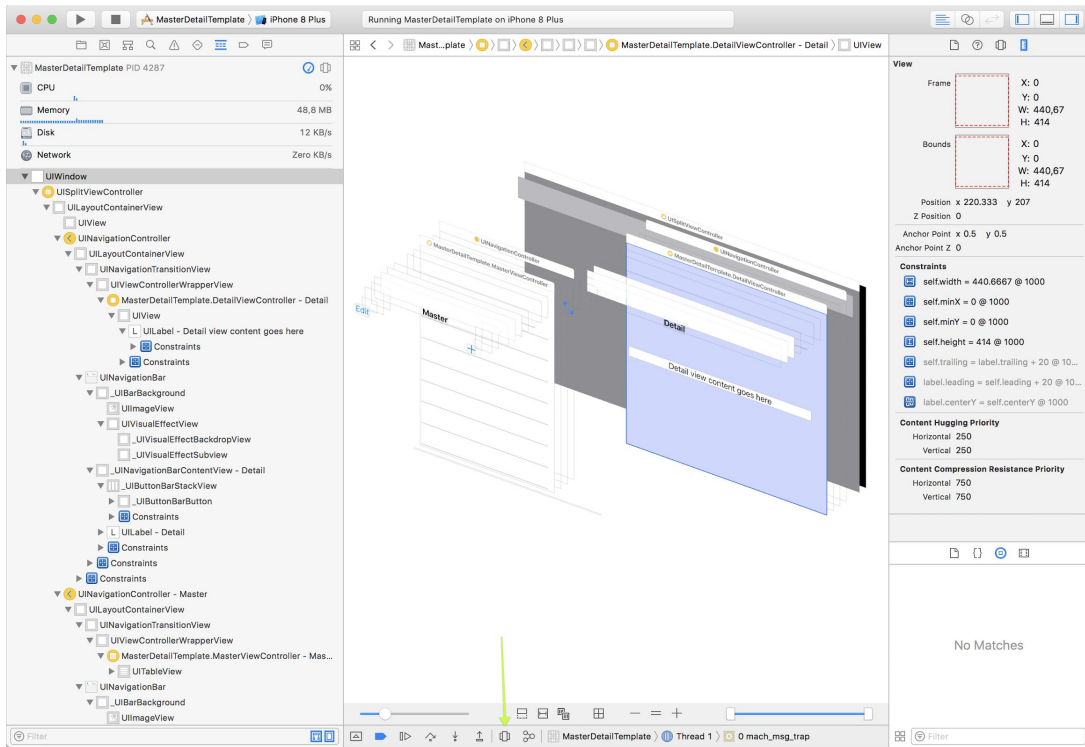
Depurando Auto Layout: consejos generales

- No añadir más *constraints* de las estrictamente necesarias. Más *constraints* = más complejidad = más dificultad para resolver problemas.
- En caso de encontrar problemas, hacer pequeñas modificaciones una a una y probar. No modificar varias cosas a la vez.

Depurando Auto Layout: consejos generales

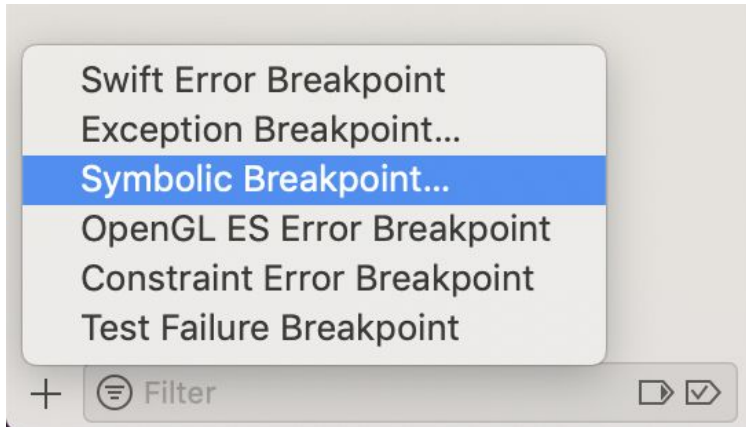
- Siempre que sea posible, es mejor utilizar `UIStackView` en vez de añadir `constraints` directamente.
- El inspector de la jerarquía de vistas nos permite ver en tiempo de ejecución qué *constraints* hay activas en nuestra UI.

Depurando Auto Layout: consejos generales



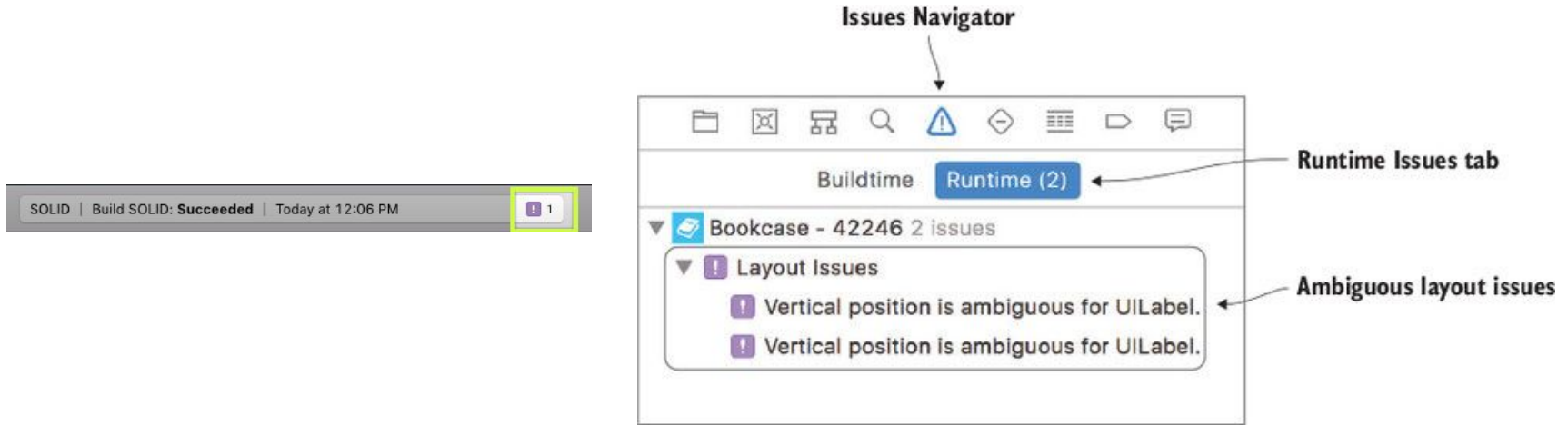
Depurando Auto Layout: layouts ambiguos

- Podemos añadir un *breakpoint* simbólico en `UIViewAlertForUnsatisfiableConstraints` para para el depurador cuando se produzca un layout ambiguo.



Depurando Auto Layout: layouts ambiguos

- Durante la ejecución de nuestra app, recibiremos *Runtime Issues* que avisarán, entre otras cosas, de problemas en nuestros *layouts*.



Depurando Auto Layout: layouts ambiguos

- También el inspector de la jerarquía de vistas nos dará más detalles en estos casos.

