Chapter 16
Auto Layout: Programmatic Constraints

- Visual Format Language
- Creating Constraints
- Adding Constraints
- Intrinsic Content Size
- The Other Way

Current Homeowner Applications

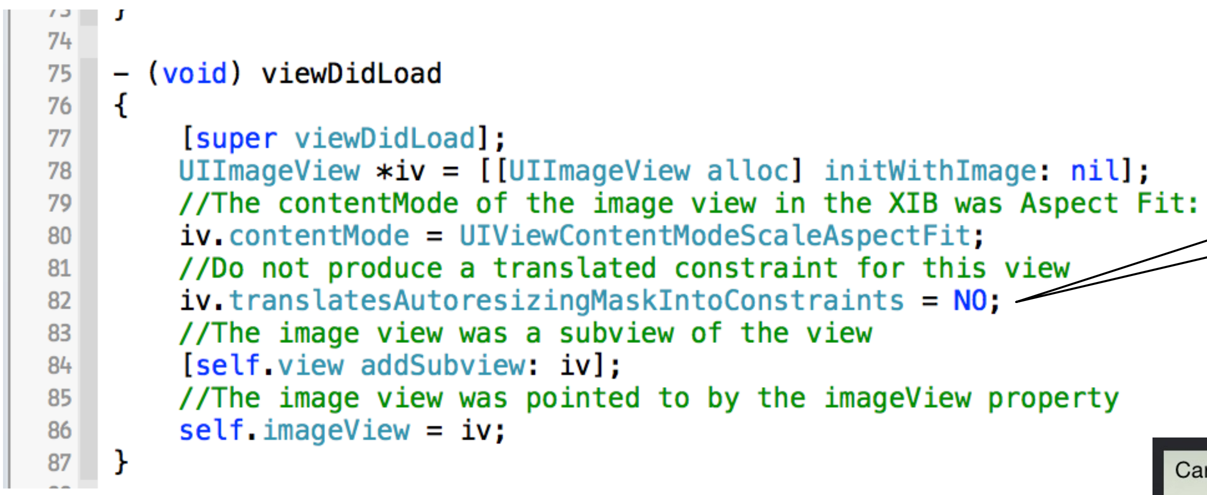
- Apple recommends that you create and constrain your views in a XIB file whenever possible.
- If your views are created in code, then you will need to constrain them programmatically.
- If you are creating and constraining an entire view hierarchy, then you override loadView.
- If you are creating and constraining an additional view to add to a view hierarchy that was created by loading a NIB file, then you override viewDidLoad instead.

Prepare Your Project

- Copy homeowner project from last chapter
- Delete imageView subview and fix any auto layout constraints issues that might popup.

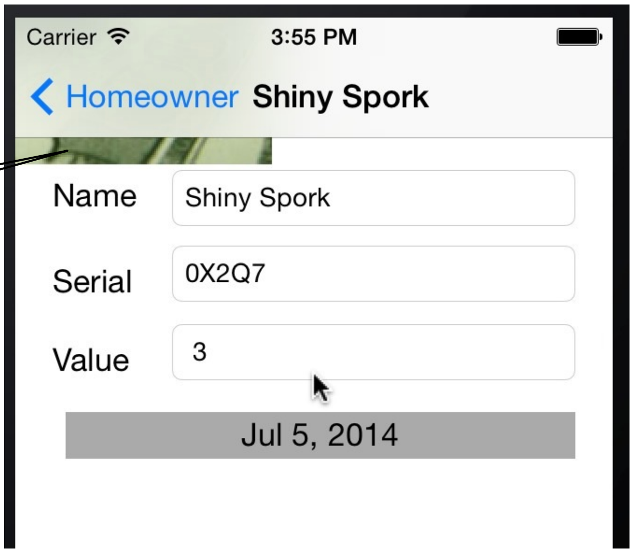
Create imageView programmatically

- Override viewDidLoad in the detailed view controller



```
- (void) viewDidLoad
{
    [super viewDidLoad];
    UIImageView *iv = [[UIImageView alloc] initWithImage: nil];
    //The contentMode of the image view in the XIB was Aspect Fit:
    iv.contentMode = UIViewContentModeScaleAspectFit;
    //Do not produce a translated constraint for this view
    iv.translatesAutoresizingMaskIntoConstraints = NO;
    //The image view was a subview of the view
    [self.view addSubview: iv];
    //The image view was pointed to by the imageView property
    self.imageView = iv;
}
```

Notice auto translation is off



Name	Shiny Spork
Serial	0X2Q7
Value	3

Jul 5, 2014

imageView is up there. It needs to be constrained

Desired imageView Constraints

- The UIImageView should span the entire width of the screen and should maintain the standard 8 point spacing between itself and the dateLabel above and the toolbar below.
- Here are the constraints for the image view spelled out:
  - Left edge is 0 points from the image view’s container
  - Right edge is 0 points from the image view’s container
  - Top edge is 8 points from the date label
  - Bottom edge is 8 points from the toolbar

Visual Format Language

- Visual Format Language is a way of describing constraints in a literal string.
- You can describe multiple constraints in one visual format string.
- A single visual format string cannot describe both vertical and horizontal constraints.
- For our image view we need two visual format strings:

  one that constrains the horizontal spacing of the image view

  one that constrains its vertical spacing.

VFL for Our imageView

- Here is how we would describe the horizontal spacing constraints for the image view as a visual format string:

  @"H:|-0-[imageView]-0-|"

  The H: specifies that these constraints refer to horizontal spacing. The view is identified inside square brackets. The pipe character (|) stands for the view's container. This can be shortened as:

  @"H:|[imageView]|"
- The string for the vertical constraints looks like this:

  @"V:[dateLabel]-8-[imageView]-8-[toolbar]"
- Notice that " top" and " bottom" are mapped to " left" and " right", respectively, in this necessarily horizontal display of vertical spacing.
- The image view is 8 points from the date label at its top edge and 8 points from the toolbar at its bottom edge.
- This string may be shortened to:

  @"V:[dateLabel]-[imageView]-[toolbar]"

More VFL Examples

- Imagine you had two image views with the following horizontal constraints:
  - The horizontal spacing between the image views should be 10 points
  - The lefthand image view's left edge should be 20 points from its superview
  - The righthand image view's right edge should be 20 points from its superview
- You could describe the three constraints in one visual format string:

```
@"H:|-20-[imageViewLeft]-10-[imageViewRight]-20-|"
```

- The syntax for a fixed size constraint is simply adding an equality operator and a value in parentheses inside a view's visual format:

```
@"V:[someView(== 50)]"
```

This view's height would be constrained to 50 points.

Creating Constraints

- A constraint is an instance of the class NSLayoutConstraint.
- When creating constraints programmatically, you explicitly create one or more instances of NSLayoutConstraint and then add them to the appropriate view object.
- Creating and adding constraints is two steps when doing programmatically.
- We use the method:

```
+ ( NSArray *) constraintsWithVisualFormat:( NSString *) format
                    options:( NSLayoutFormatOptions) opts
                    metrics:( NSDictionary *) metrics
                    views:( NSDictionary *) views
```

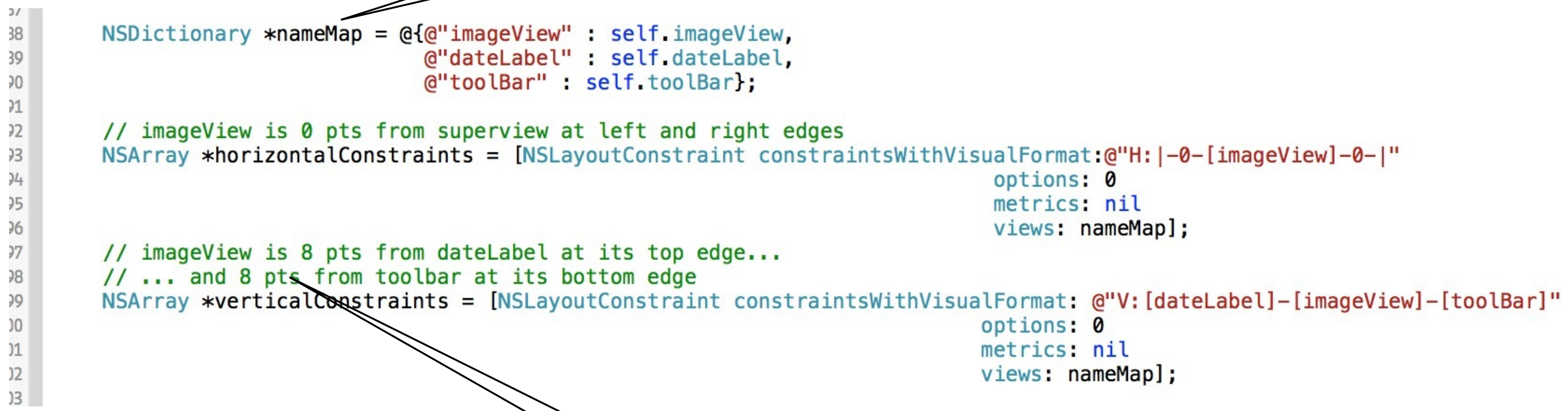
- The first argument is the visual format string.
- The fourth argument is an NSDictionary that maps the names in the visual format string to view objects in the view hierarchy.
- Notice that this method returns an array of constraints (e.g. left edge constraints and right edge constraints are two distinct constraints)

Implementing Creating Constraints

- The two visual format strings that you will use to constrain the image view refer to view objects by the names of the variables that point to them (imageView and toolBar).
- A visual format string is just a string which means that putting the name of a variable inside it means nothing unless you explicitly make the association between the string name and the actual view.
- The association between the variable name string (e.g. imageView) and the actual view (imageView) is accomplished by using a dictionary.
- The only exception when this association is not necessary is when the view name is the | character, which is a reserved name for the superview (container) of the views being referenced in the string.

Sample Constraints Creation

Dictionary associating variables names with views



```
NSDictionary *nameMap = @{@"imageView" : self.imageView,
                          @"dateLabel" : self.dateLabel,
                          @"toolBar" : self.toolBar};

// imageView is 0 pts from superview at left and right edges
NSArray *horizontalConstraints = [NSLayoutConstraint constraintsWithVisualFormat:@"H:|-0-[imageView]-0-|"
                                                                           options: 0
                                                                           metrics: nil
                                                                           views: nameMap];

// imageView is 8 pts from dateLabel at its top edge...
// ... and 8 pts from toolbar at its bottom edge
NSArray *verticalConstraints = [NSLayoutConstraint constraintsWithVisualFormat: @"V:[dateLabel]-[imageView]-[toolBar]"
                                                                         options: 0
                                                                         metrics: nil
                                                                         views: nameMap];
```

Creating the array of vertical constraints

Adding Constraints

- We now have two arrays of NSLayoutConstraint objects.
- These constraints will have no effect on the layout until we explicitly add them using the UIView method

  - (void) addConstraints:(NSArray *) constraints
- The question now is to which view should I add these constraints
- The closest common ancestor of the views that are affected by the constraint.
- List of rules you can follow to determine which view you should add constraints to:

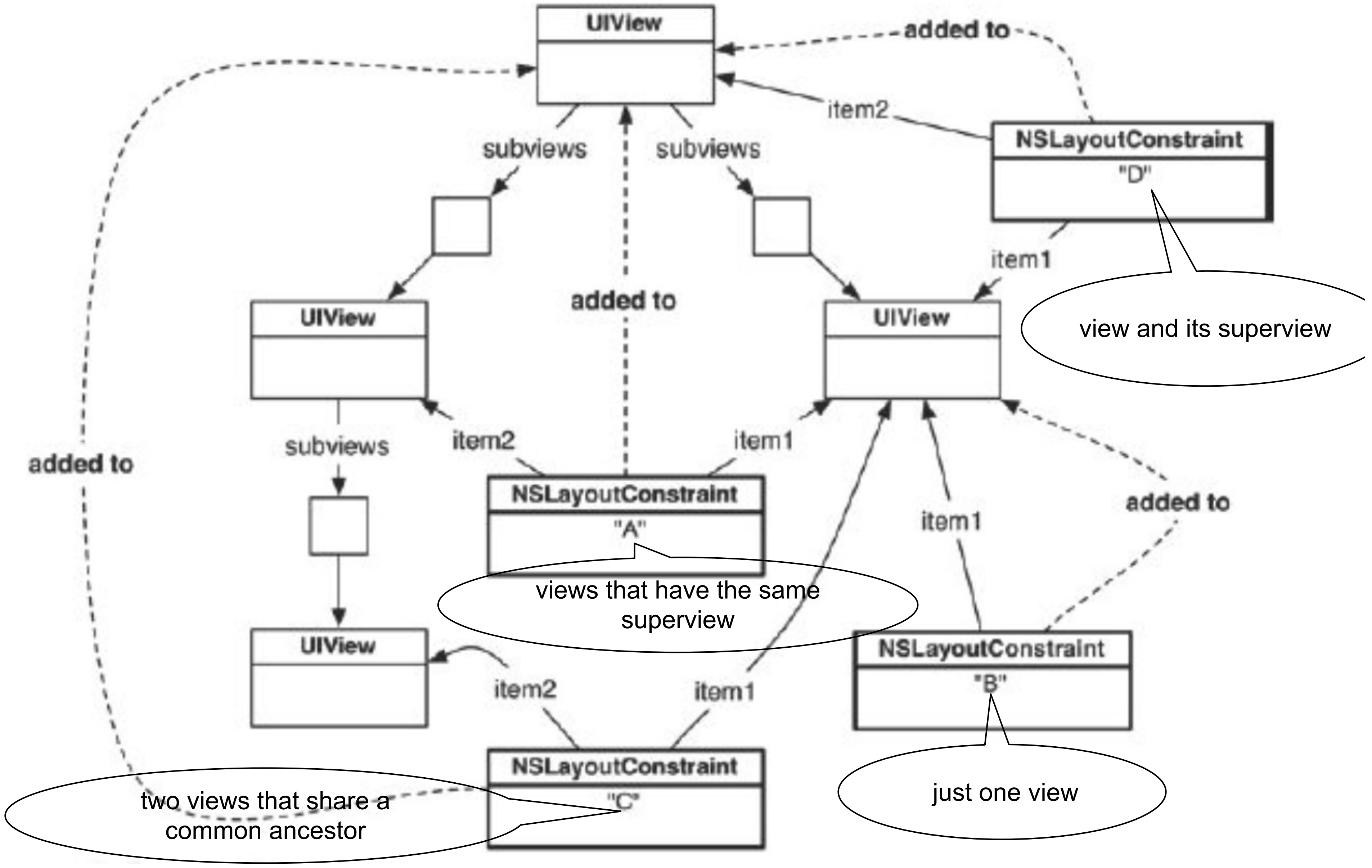
  If a constraint affects two views that have the same superview, then the constraint should be added to their superview.

  If a constraint affects just one view, then the constraint should be added to the view being affected.

  If a constraint affects two views that do not have the same superview but do share a common ancestor much higher up on the view hierarchy, then the first common ancestor gets the constraint.

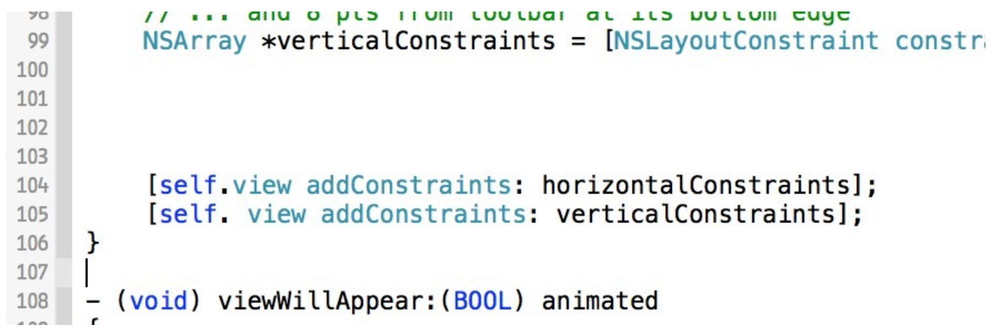
  If a constraint affects a view and its superview, then this constraint will be added to the superview.

Constraint Hierarchy

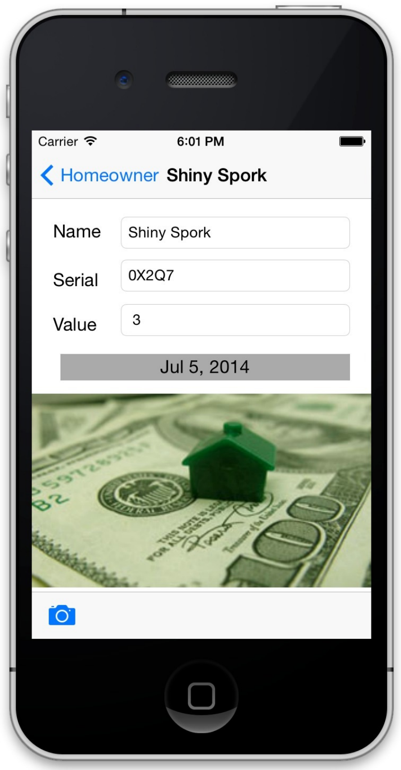


Implementing Adding Constraints

- Since the image view's horizontal constraints affect only the imageView and its superview, so you add them to the superview namely INIDetailViewController.
- For the vertical constraints, the imageView, dateLabel, and toolbar are the affected views. They all share the same superview INIDetailViewController, so you also add these constraints to the superview.



```
    NSArray *verticalConstraints = [NSLayoutConstraint constr



    [self.view addConstraints: horizontalConstraints];
    [self. view addConstraints: verticalConstraints];
}

- (void) viewWillAppear:(BOOL) animated
```



Intrinsic Content Size

- Intrinsic content size is information that a view has about how big it should be based on what it’s content.
- A label’s intrinsic content size is based on how much text it is displaying.
- Our image view’s intrinsic content size is the size of the image that we selected.
- Auto Layout takes the intrinsic content size information into consideration by creating intrinsic content size constraints for each view.
- Unlike other constraints, these constraints have two priorities:

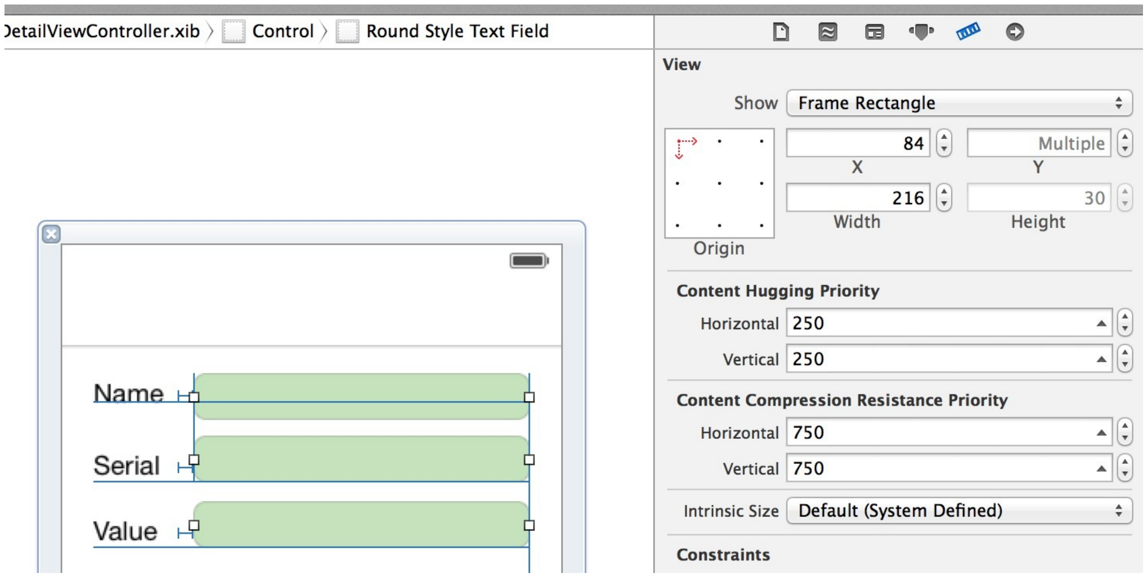
  a content hugging priority and

  a content compression resistance priority.

  both priorities have separate horizontal and vertical values so that you can set different priorities for a view’s height and width. This makes a total of four intrinsic content size priority values per view.

Content Hugging Priority

- Content hugging priority tells Auto Layout how important it is that the view’s size stay close to, or “ hug”, its intrinsic content.
- A value of 1000 means that the view should never be allowed to grow larger than its intrinsic content size.
- A value less than 1000 means Auto Layout may increase the view’s size when necessary.


DetailViewController.xib 〉 Control 〉 Round Style Text Field

View

Show: Frame Rectangle

X: 84
Y: Multiple
Width: 216
Height: 30

Content Hugging Priority
Horizontal: 250
Vertical: 250

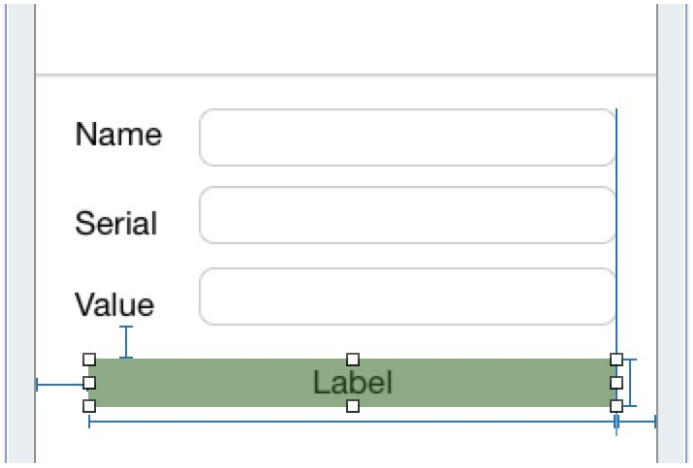
Content Compression Resistance Priority
Horizontal: 750
Vertical: 750

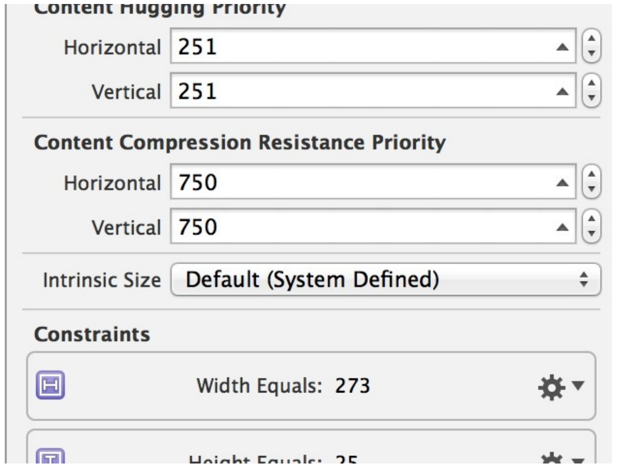
Intrinsic Size: Default (System Defined)

Constraints

Content Compression Resistance Priority

- Content compression resistance priority tells Auto Layout how important it is that the view avoid shrinking, or " resist compressing", its intrinsic content.
- A value of 1000 means that the view should never be allowed to be smaller than its intrinsic content size.
- A value less than 1000 means that Auto Layout may shrink the view when necessary.





Content Hugging Priority

Horizontal	251
Vertical	251

Content Compression Resistance Priority

Horizontal	750
Vertical	750
Intrinsic Size	Default (System Defined)

Constraints

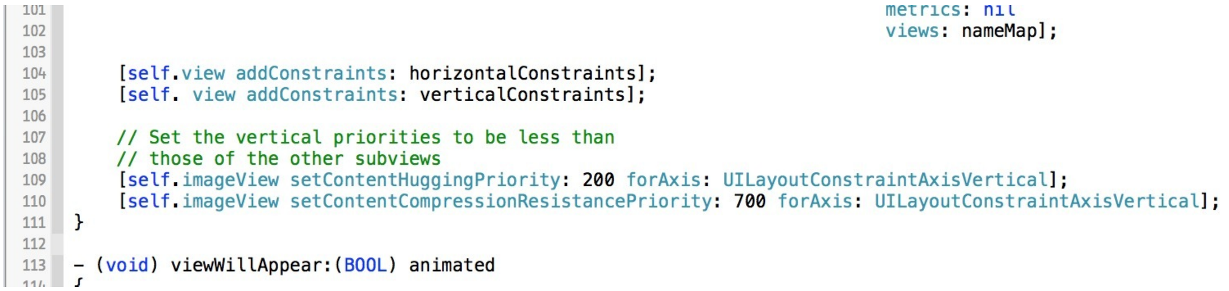
Width Equals: 273

Height Equals: 25

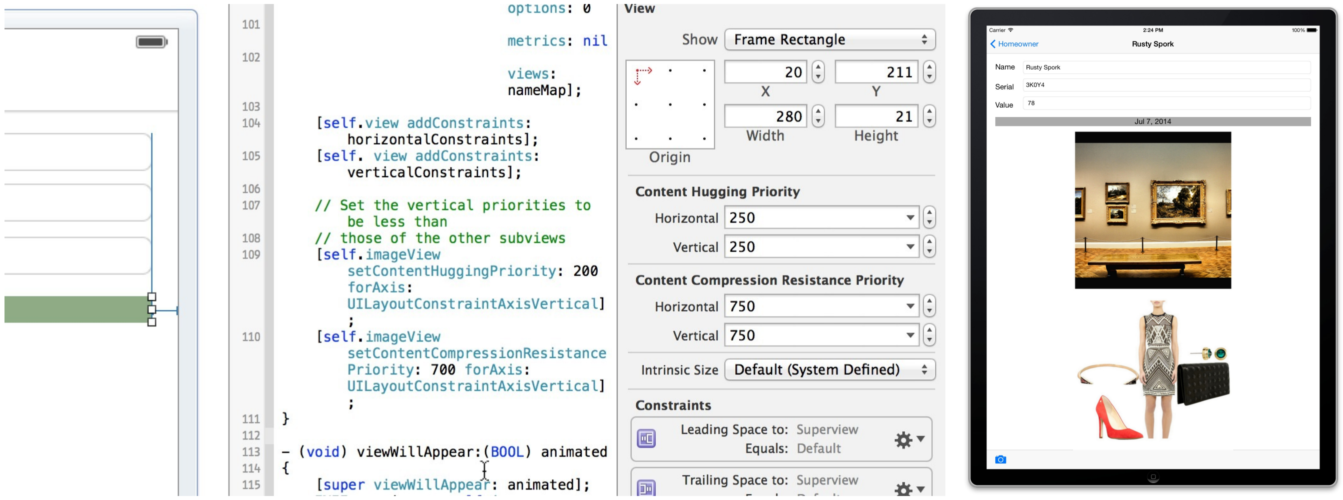
Priorities Example

- If all priorities are 1000 then the view should never be allowed to grow larger or be smaller than its intrinsic content size. Thus it should be exactly the same dimension as its intrinsic content size which could conflict with constraints that we impose on the view.
- Since the values of these priorities of our text fields and labels are not 1000, then they will never conflict with the constraints that you have added so far.
- Consider the case where the vertical content hugging priority of the image view is higher (251) than that of the date label (250) and that we introduce a very small image. Then in order to keep the constraint about the vertical separation valid, the Auto Layout will shrink the image and expand the hight of the label. The image view is, in effect, is pulling down on the date label’s bottom edge making it taller.
- If the same situation was true for content compression resistance and we had a large image, then it will compress the label.
- It would be better if the image view had a smaller vertical content hugging and compression resistance priority than the other subviews.

Specifying Intrinsic Content Size Programmatically



```
                                                        metrics: nil
                                                        views: nameMap];

    [self.view addConstraints: horizontalConstraints];
    [self. view addConstraints: verticalConstraints];

    // Set the vertical priorities to be less than
    // those of the other subviews
    [self.imageView setContentHuggingPriority: 200 forAxis: UILayoutConstraintAxisVertical];
    [self.imageView setContentCompressionResistancePriority: 700 forAxis: UILayoutConstraintAxisVertical];
}

- (void) viewWillAppear:(BOOL) animated
{
```



```
                        options: 0
                        metrics: nil
                        views:
                        nameMap];
    [self.view addConstraints:
        horizontalConstraints];
    [self. view addConstraints:
        verticalConstraints];

    // Set the vertical priorities to
        be less than
    // those of the other subviews
    [self.imageView
        setContentHuggingPriority: 200
        forAxis:
        UILayoutConstraintAxisVertical]
        ;
    [self.imageView
        setContentCompressionResistance
        Priority: 700 forAxis:
        UILayoutConstraintAxisVertical]
        ;
}

- (void) viewWillAppear:(BOOL) animated
{
    [super viewWillAppear: animated];
```

View

Show: Frame Rectangle

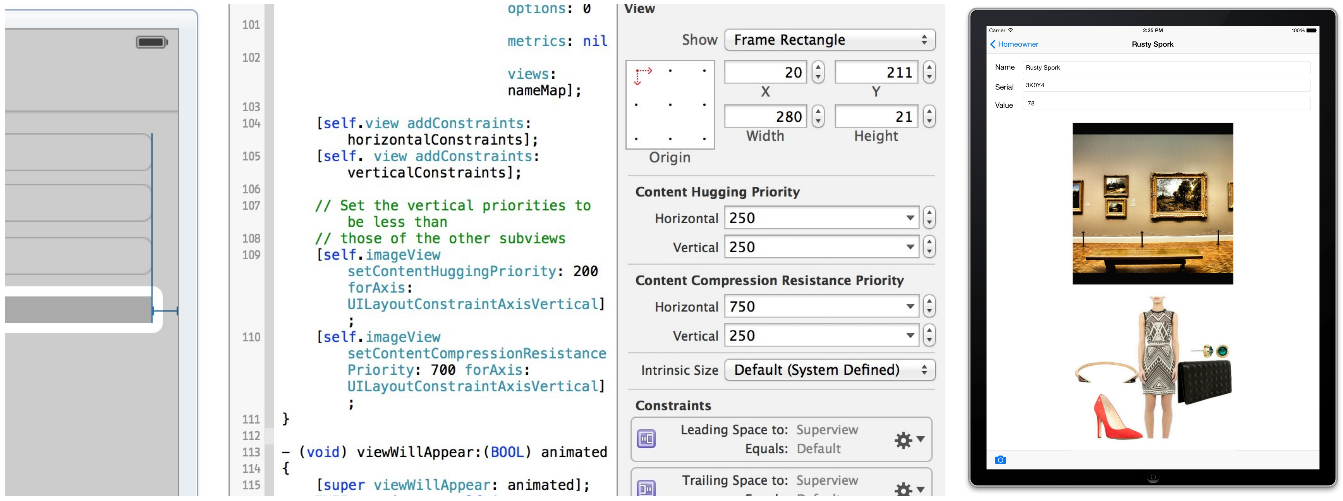
Origin	X: 20	Y: 211
	Width: 280	Height: 21

Content Hugging Priority
- Horizontal: 250
- Vertical: 250

Content Compression Resistance Priority
- Horizontal: 750
- Vertical: 750

Intrinsic Size: Default (System Defined)

Constraints
- Leading Space to: Superview, Equals: Default
- Trailing Space to: Superview



```
                        options: 0
                        metrics: nil
                        views:
                        nameMap];
    [self.view addConstraints:
        horizontalConstraints];
    [self. view addConstraints:
        verticalConstraints];

    // Set the vertical priorities to
        be less than
    // those of the other subviews
    [self.imageView
        setContentHuggingPriority: 200
        forAxis:
        UILayoutConstraintAxisVertical]
        ;
    [self.imageView
        setContentCompressionResistance
        Priority: 700 forAxis:
        UILayoutConstraintAxisVertical]
        ;
}

- (void) viewWillAppear:(BOOL) animated
{
    [super viewWillAppear: animated];
```

View

Show: Frame Rectangle

Origin	X: 20	Y: 211
	Width: 280	Height: 21

Content Hugging Priority
- Horizontal: 250
- Vertical: 250

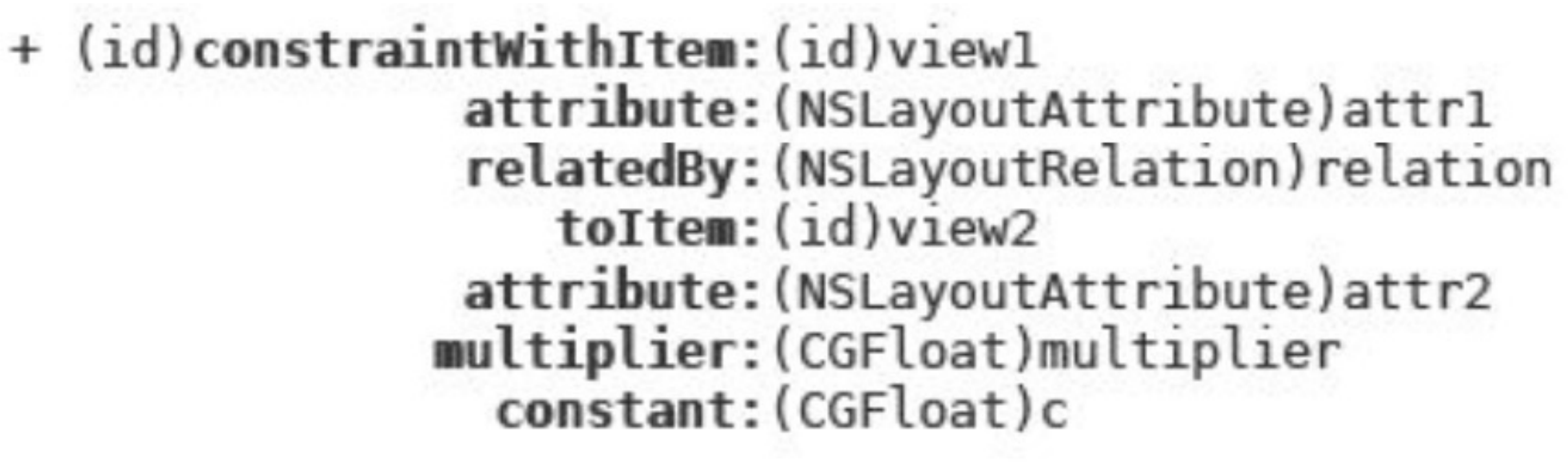
Content Compression Resistance Priority
- Horizontal: 750
- Vertical: 250

Intrinsic Size: Default (System Defined)

Constraints
- Leading Space to: Superview, Equals: Default
- Trailing Space to: Superview

The Other Way

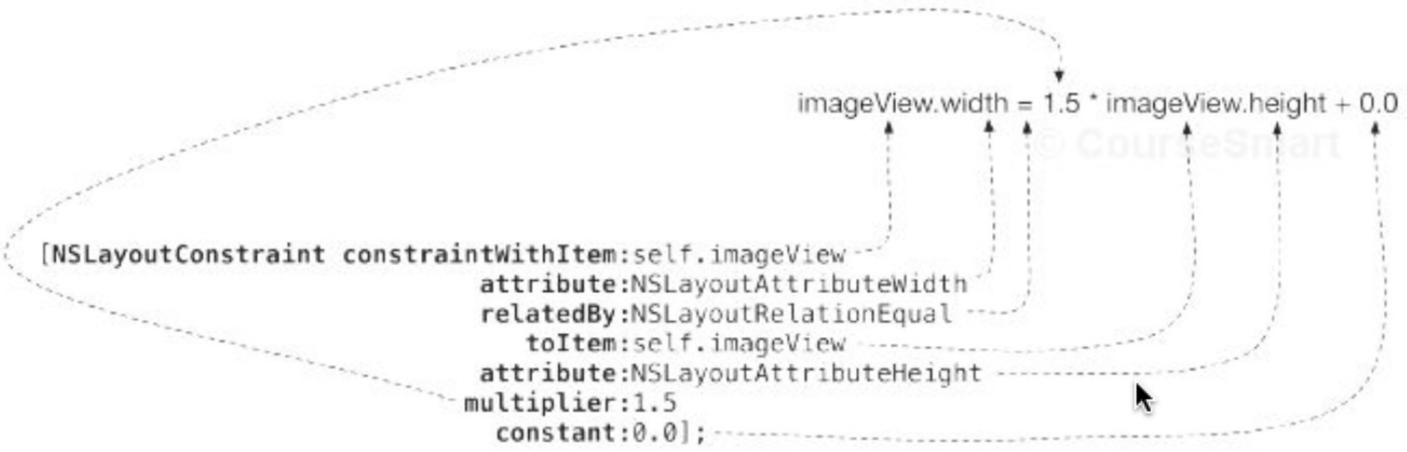
- There are times when a constraint cannot be created with a visual format string.
- As an example: you cannot use VFL to create a constraint based on a ratio
  - If you wanted the date label to be twice as tall as the name label
  - If you wanted the image view to always be 1.5 times as wide as it is tall.
- For this situation we use the NSLayoutConstraint method:



```
+ (id)constraintWithItem:(id)view1
               attribute:(NSLayoutAttribute)attr1
               relatedBy:(NSLayoutRelation)relation
                  toItem:(id)view2
               attribute:(NSLayoutAttribute)attr2
              multiplier:(CGFloat)multiplier
                constant:(CGFloat)c
```

Aspect Ratio Constraint Example

```
NSLayoutConstraint *aspectConstraint =
    [NSLayoutConstraint constraintWithItem:self.imageView
                                 attribute:NSLayoutAttributeWidth
                                 relatedBy:NSLayoutRelationEqual
                                    toItem:self.imageView
                                 attribute:NSLayoutAttributeHeight
                                multiplier:1.5
                                  constant:0.0];
```



```
[NSLayoutConstraint constraintWithItem:self.imageView
                             attribute:NSLayoutAttributeWidth
                             relatedBy:NSLayoutRelationEqual
                                toItem:self.imageView
                             attribute:NSLayoutAttributeHeight
                            multiplier:1.5
                              constant:0.0];
```

NSLayoutAttribute

Layout attributes are used to specify the part of the object's visual representation that should be used to get the value for the constraint.

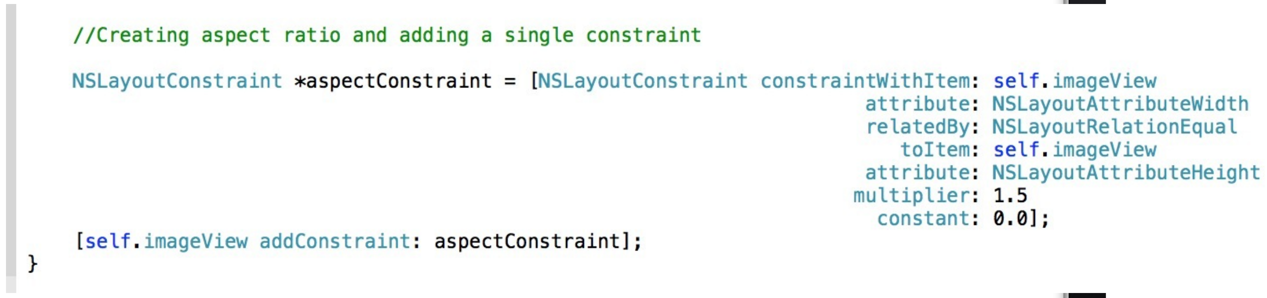
```
enum {
   NSLayoutAttributeLeft = 1,
   NSLayoutAttributeRight,
   NSLayoutAttributeTop,
   NSLayoutAttributeBottom,
   NSLayoutAttributeLeading,
   NSLayoutAttributeTrailing,
   NSLayoutAttributeWidth,
   NSLayoutAttributeHeight,
   NSLayoutAttributeCenterX,
   NSLayoutAttributeCenterY,
   NSLayoutAttributeBaseline,

   NSLayoutAttributeNotAnAttribute = 0
};
typedef NSInteger NSLayoutAttribute;
```

NSLayoutRelation

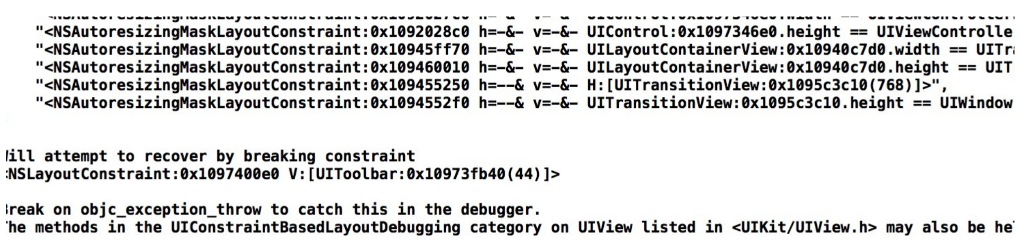
Describes the relation between the first attribute and the modified second attribute in a constraint.

```
enum {
   NSLayoutRelationLessThanOrEqual = -1,
   NSLayoutRelationEqual = 0,
   NSLayoutRelationGreaterThanOrEqual = 1,
};
typedef NSInteger NSLayoutRelation;
```

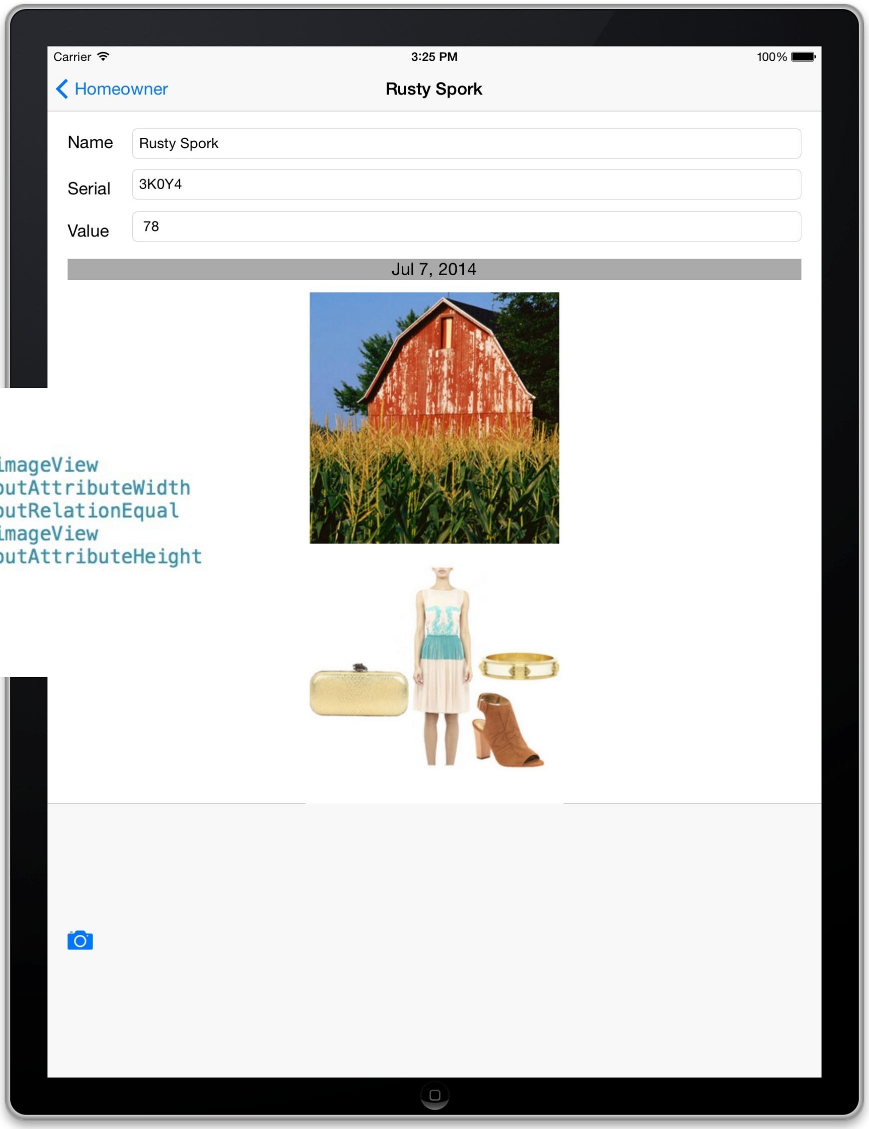
Implementing Aspect Ratio Constraint



```
    //Creating aspect ratio and adding a single constraint

    NSLayoutConstraint *aspectConstraint = [NSLayoutConstraint constraintWithItem: self.imageView
                                                                         attribute: NSLayoutAttributeWidth
                                                                         relatedBy: NSLayoutRelationEqual
                                                                            toItem: self.imageView
                                                                         attribute: NSLayoutAttributeHeight
                                                                        multiplier: 1.5
                                                                          constant: 0.0];


    [self.imageView addConstraint: aspectConstraint];
}
```



```
  "<NSAutoresizingMaskLayoutConstraint:0x1092028c0 h=-&- v=-&- UIControl:0x1097346e0.height == UIViewControlle
  "<NSAutoresizingMaskLayoutConstraint:0x10945ff70 h=-&- v=-&- UILayoutContainerView:0x10940c7d0.width == UITr
  "<NSAutoresizingMaskLayoutConstraint:0x109460010 h=-&- v=-&- UILayoutContainerView:0x10940c7d0.height == UIT
  "<NSAutoresizingMaskLayoutConstraint:0x109455250 h=--& v=-&- H:[UITransitionView:0x1095c3c10(768)]>",
  "<NSAutoresizingMaskLayoutConstraint:0x1094552f0 h=--& v=-&- UITransitionView:0x1095c3c10.height == UIWindow

ill attempt to recover by breaking constraint
NSLayoutConstraint:0x1097400e0 V:[UIToolbar:0x10973fb40(44)]>

reak on objc_exception_throw to catch this in the debugger.
he methods in the UIConstraintBasedLayoutDebugging category on UIView listed in <UIKit/UIView.h> may also be he
```


Carrier
3:25 PM
100%
Homeowner
Rusty Spork
Name
Rusty Spork
Serial
3K0Y4
Value
78
Jul 7, 2014