

Introduction to Auto Layout

Hands-On Challenges

Copyright © 2015 Razeware LLC.

All rights reserved. No part of this book or corresponding materials (such as text, images, or source code) may be reproduced or distributed by any means without prior written per- mission of the copyright owner.

This book and all corresponding materials (such as source code) are provided on an "as is" basis, without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose, and noninfringement. In no event shall the authors or copyright holders be liable for any claim, damages or other liability, whether in action of contract, tort or otherwise, arising from, out of or in connec- tion with the software or the use or other dealings in the software.

All trademarks and registered trademarks appearing in this book are the property of their respective owners.

# Unconstrained Constraints

You’ve seen how to set up your auto layout constraints from interface builder. Now it’s time to apply that knowledge and try adding layout constraints from code!

Layout constraints are just objects that you can create and attach to views. You can even connect constraints in your storyboard as outlets to your code, and modify them at runtime!

In this short lab, you’ll create new views and then lay them out with constraints from code.

## Button Action

First, you’ll need a button to trigger an action method to add the new views.

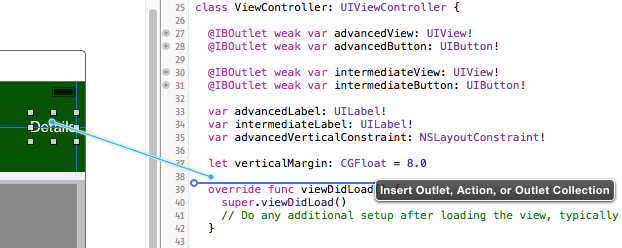
Add a new button to the top green view, and set the text to “Details” with white text color. Position it against the right margin and vertically centered.



The position is where you want it, so you can add the necessary constraints. From the menu, select **Editor\Pin\Trailing Space to Superview** to set the x-position. Next, select **Editor\Align\Vertical Center in Container** to set the y-position. You should see the joyful blue lines of happy constraints!



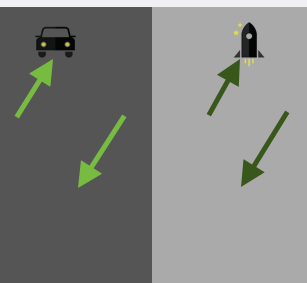
Next, you can connect the action to the button. Open the assistant editor and make sure **ViewController.swift** is in the assistant. Control-drag from the button to a spot inside the class to create the action.



Name the action method **detailsButtonTapped**. You’re all set with a place to create some views and constraints from code!

## Outlets

While you have the assistant editor open, connect the following four outlets: the advanced and intermediate buttons, and the advanced and intermediate container views.



Name the outlets as follows:

* Advanced button > advancedButton
* Advanced view > advancedView
* Intermediate button > intermediateButton
* Intermediate view > intermediateView

You’ll need these outlets to add the constraints to the containers, and to position the new views you’ll be adding relative to the buttons.

## Manual Constraint Relationships

Open **ViewController.swift** and add the following properties to the class:

var advancedLabel: UILabel!

var intermediateLabel: UILabel!

var advancedVerticalConstraint: NSLayoutConstraint!

let verticalMargin: CGFloat = 8.0

You’ll be adding two new labels to the interface, so the two UILabel properties will hold them. You’ll also want to hold a reference to one of the layout constraints, advancedVerticalConstraint, to modify it later.

Finally, verticalMargin is a constant that you’ll use to keep the spacing consistent.

Next, add the following helper method to the class:

private func addAdvancedLabel() {

// 1

advancedLabel = UILabel()

advancedLabel.setTranslatesAutoresizingMaskIntoConstraints(false)

advancedLabel.text = "Dive deep into a guided tour of more advanced topics like functional programming, Scene Kit, and more!"

advancedLabel.numberOfLines = 0

advancedView.addSubview(advancedLabel)

// 2

advancedVerticalConstraint = NSLayoutConstraint(  
 item: advancedLabel,

attribute: .Top,

relatedBy: .Equal,

toItem: advancedButton,

attribute: .Bottom,

multiplier: 1.0,

constant: verticalMargin)

advancedView.addConstraint(advancedVerticalConstraint)

// 3

advancedView.addConstraint(NSLayoutConstraint(  
 item: advancedLabel,

attribute: .Leading,

relatedBy: .Equal,

toItem: advancedView,

attribute: .LeadingMargin,

multiplier: 1.0,

constant: 0))

// 4

advancedView.addConstraint(NSLayoutConstraint(  
 item: advancedLabel,

attribute: .Trailing,

relatedBy: .Equal,

toItem: advancedView,

attribute: .TrailingMargin,

multiplier: 1.0,

constant: 0))

}

1. This method will create a new label and set up its size and position with layout constraints. Here’s what’s going on, section by section:
   1. Create a new label, set its properties, and add it to the view. Note the call to setTranslatesAutoresizingMaskIntoConstraints(false) – if you’re setting up constraints in code, you need to be sure to call this method on your view so that it doesn’t automatically create a default set of constraints. These constraints would conflict with your own, so you need to call this method to ensure you’re starting off with a clean slate.
   2. Here’s your first code constraint! You’re saving this one to the advancedVerticalConstraint property so you can adjust it later, and adding it to the view with addConstraint. Notice how the initializer parameters for NSLayoutConstraint match what you see when you edit a constraint from interface builder.  
        
      In this case, you’re saying the position of the top of the label should be equal to the position of the bottom of the button. The constant is set to verticalMargin, so you’ll get an 8-point vertical margin between the button and the label.
   3. This constraint is for the leading (left) space between the label and its superview. You’re relating the leading edge of the label to the leading margin of the superview.
   4. The final constraint is similar to the previous one; this one is for the trailing (right) space between the label and its superview. You’re relating the trailing edge of the label to the trailing margin of the superview.

That’s all you need! The label is set to 0 lines so it will automatically grow vertically to fit the amount of text. The constraints give the label its x- and y-positions, so you shouldn’t have any layout ambiguity.

Add the following lines to detailsButtonTapped:

if advancedLabel == nil {

addAdvancedLabel()  
}

This will call the helper method to set up the label.

Build and run, and tap the Details button. You should see the label appear in the correct spot. If you rotate the device or simulator, you’ll see the label resize properly to fit the constraints.

## Visual Format Language

private func addIntermediateLabel() {

intermediateLabel = UILabel()

intermediateLabel.setTranslatesAutoresizingMaskIntoConstraints(  
 false)

intermediateLabel.text = "This track is for Objective-C developers who are not yet fully up-to-speed with Swift."

intermediateLabel.numberOfLines = 0

intermediateView.addSubview(intermediateLabel)

let views = ["intermediateLabel": intermediateLabel,  
 "intermediateView": intermediateView,  
 "intermediateButton": intermediateButton]

let metrics = ["margin": 4, "verticalMargin": verticalMargin]

intermediateView.addConstraints(  
 NSLayoutConstraint.constraintsWithVisualFormat(  
 "H:|-margin-[intermediateLabel]-margin-|",  
 options: nil, metrics: metrics, views: views))

intermediateView.addConstraints(  
 NSLayoutConstraint.constraintsWithVisualFormat(  
 "V:[intermediateButton]-verticalMargin-[intermediateLabel]",  
 options: nil, metrics: metrics, views: views))

}

addIntermediateLabel()

## Modifying Constraints

private func toggleLabel() {

if intermediateLabel.alpha == 0 {

intermediateLabel.alpha = 1

advancedVerticalConstraint.constant = verticalMargin

} else {

intermediateLabel.alpha = 0

advancedVerticalConstraint.constant =  
 CGRectGetHeight(advancedView.frame)

}

}

if advancedLabel == nil {

addAdvancedLabel()

addIntermediateLabel()

} else {

toggleLabel()

}