

Collection View Custom Layouts

Lab Instructions

Copyright © 2014 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 non-infringement. 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 connection 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.

# No Sense of Time

The calendar view may look great, but aesthetics aren’t enough; it currently lacks any visual indication of when each session starts, or how long it’s likely to last. You’ll fix this now.

In this short challenge, you’ll add some headers that display the hours of the day using the custom calendar layout.

## Lab: The Session Starts When?

Since collection views can have pretty much any style of layout, the concept of headers and footers doesn’t really apply here, especially when you’re subclassing UICollectionViewLayout directly, instead of UICollectionViewFlowLayout. So in their place, Apple has provided *supplementary views*, which can be associated with each cell.

A collection view can contain many different types of supplementary views, and you provide each one with a unique string identifier, known as its *kind*, so that you can infer which type you’re dealing with when in the data source, delegate, and layout methods.

Each cell can have many supplementary views associated with it, as long as its *kind* is unique within the scope of that cell – that is, you can’t have multiple supplementary views of the same kind associated with a single cell.

Even though you can do far more than just headers with supplementary views, in the case of this layout you are going to use supplementary views to create headers. Specifically, you’ll create a series of supplementary view headers that display the hours of the day, and have the custom layout position them horizontally across the top of the view.

Open **ScheduleLayout.swift** and add the following to the bottom of layoutAttributesForElementsInRect(\_:), just before the return statement:

// 1

let hourHeaderViewIndexPaths = dataSource.indexPathsOfHourHeaderViews()

// 2

for indexPath in hourHeaderViewIndexPaths {

// 3

let hourHeaderViewAttributes =

layoutAttributesForSupplementaryViewOfKind("HourHeader", atIndexPath:

indexPath as NSIndexPath)

// 4

attributes.addObject(hourHeaderViewAttributes)

}

1. Here’s what’s happening:
   1. Ask the data source for an array containing the index paths for the header views. Supplementary views have an index path just like cells do, and it’s used as the mechanism to associate the two. This is also why a cell can’t have multiple supplementary views of the same kind.
   2. Iterate of each index path in the array.
   3. Request a set of layout attributes for the header corresponding to the current index path by calling layoutAttributesForSupplementaryViewOfKind(\_:atIndexPath:). You’ll add this method shortly; it’s responsible for determining what layout attributes the supplementary view needs.
   4. The attributes returned are then added to the attributes array, which contains the attributes for all the elements you want to lay out.

**Note**: To glean maximum performance from UICollectionView you would normally want only the index paths for the headers that fall within the bounds of the rect passed to layoutAttributesForElementsInRect(\_:). But since the data source you’re using is small and static, indexPathsOfHourHeaderViews() simply returns all index paths. The performance hit is negligible and the code is much simpler, which in this case is a reasonable trade-off.

Next, add this new method, which is responsible for creating the layout attributes for supplementary views. Remember these are specifically the layout-related attributes such as frame, transform, zIndex etc.:

override func layoutAttributesForSupplementaryViewOfKind(elementKind: String, atIndexPath indexPath: NSIndexPath) -> UICollectionViewLayoutAttributes! {

// 1

let attributes =

UICollectionViewLayoutAttributes(forSupplementaryViewOfKind:

elementKind, withIndexPath: indexPath)

// 2

attributes.frame = frameForHourHeaderViewAtIndexPath(indexPath)

// 3

attributes.zIndex = -1

return attributes

}

1. Here you:
   1. Create a new set of collection view layout attributes using one of the initializers of UICollectionViewLayoutAttributes
   2. Set the frame of the attributes using the convenience method frameForHourHeaderViewAtIndexPath(\_:) which calculates where the header will be displayed on screen. You’ll add this method in just a minute.
   3. Set the zIndex to -1 so that the header view is guaranteed to appear *underneath* the cells, which is necessary for this layout, as you’ll see later.

Now add the following:

private func frameForHourHeaderViewAtIndexPath(indexPath: NSIndexPath) -> CGRect {

let frame = CGRectMake((dataSource.widthPerHour \*

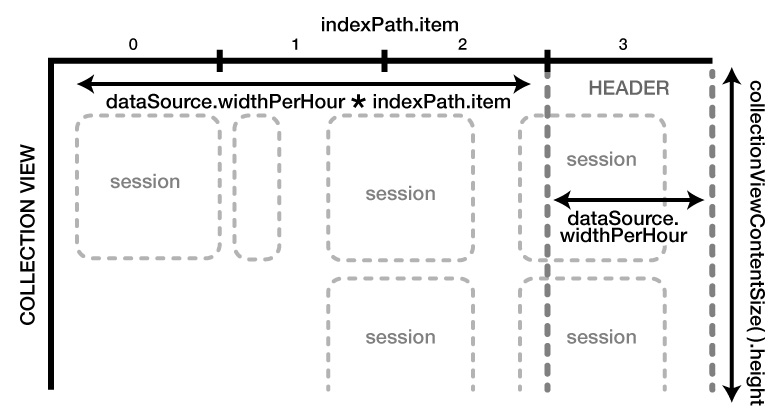
CGFloat(indexPath.item)), 0, dataSource.widthPerHour,

collectionViewContentSize().height)

return frame

}

This gets a references to the data source, and then uses some of it’s properties to calculate the frame of the header for the given index path. The math looks more complicated that it actually is; see the diagram below:



It takes a predetermined value for the width of the header and multiplies it by the item of the index path to work out the x coordinate. The y coordinate is 0, since you want the header fixed to the top of the view. The width is the predetermined value, and the height is the same as the height of the collection view. The last value may strike you as being a little odd, but all will become clear shortly.

Finally, make the following change to **ScheduleLayout.swift**. Locate this line in frameForSession(\_:atIndexPath:):

let y = heightPerTrack \* CGFloat(indexPath.item)

And replace it with this one:

let y = dataSource.hourHeaderHeight + (heightPerTrack \*

CGFloat(indexPath.item))

This makes sure that the height of the header is now taken into account when laying out the cells of the collection view.

Build and run. Wait…what’s that? Oh, it’s crashed? Take a look at the error message printed to the console:

'could not dequeue a view of kind: HourHeader with identifier ScheduleHeader - must register a nib or a class for the identifier or connect a prototype cell in a storyboard'

Ah, of course! You’ve told the collection view to use a supplementary view with the kind HourHeader for the header, but you’ve not actually registered a matching class or nib for it to use as the UI.

Open **ScheduleViewController.swift** and add the following to viewDidLoad() just below the call to super:

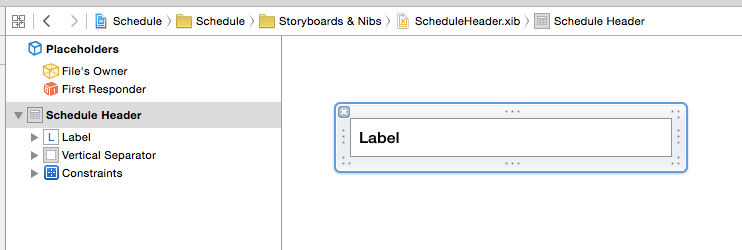
let headerNib = UINib(nibName: "ScheduleHeader", bundle: nil)

collectionView!.registerNib(headerNib,

forSupplementaryViewOfKind: "HourHeader",

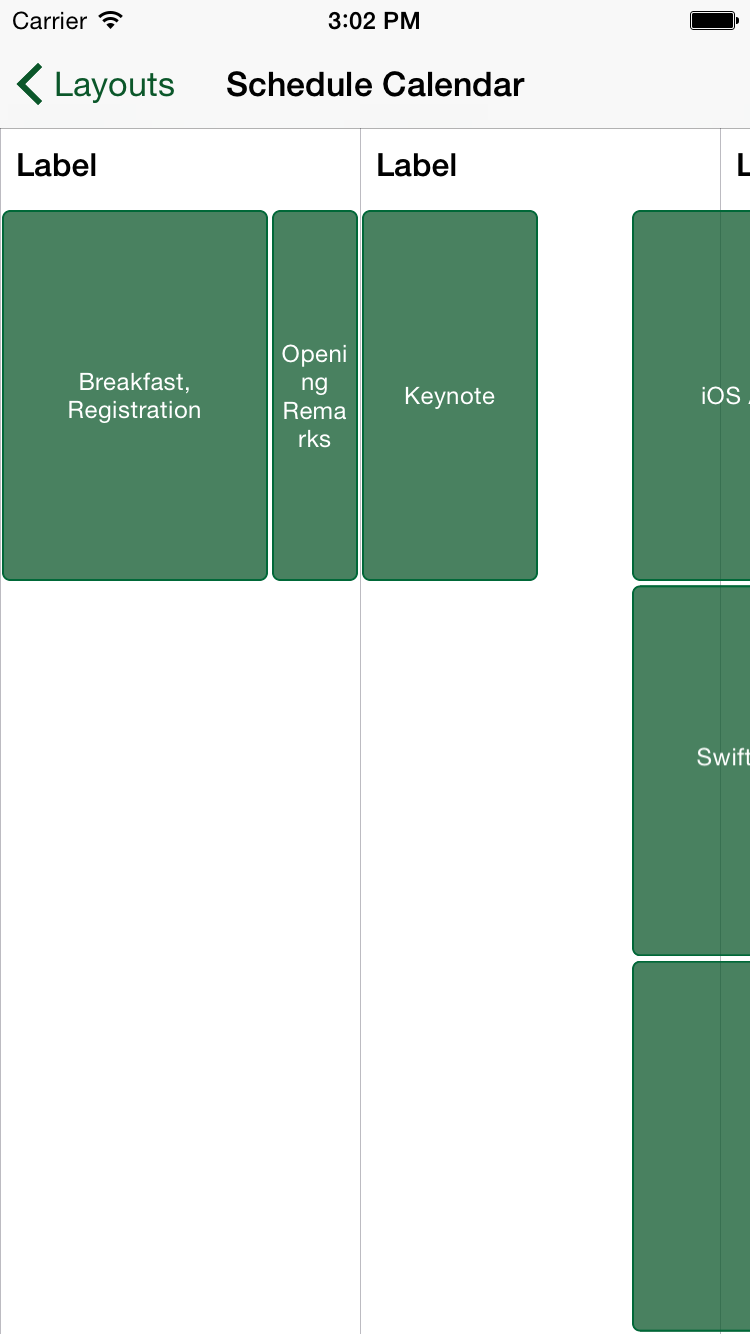
withReuseIdentifier: "ScheduleHeader")

This tells the collection view to use the nib named **ScheduleHeader** for any supplementary view with the kind HourHeader. The nib is already part of the project so you don’t need to worry about creating that yourself.



The nib is pretty basic, containing a label and a thin gray view used as a vertical separator. The Custom Class is set to **ScheduleHeader**, which is a subclass of **UICollectionReusableView**; all supplementary views must descend from this class so they can be properly managed by the collection view.

Build and run. Fantastic! No more crashing, and the calendar now has headers.



You can see the headers are laid out properly across the top of the collection view. You’ll also notice the faint hour line that runs from the top to the bottom of the view; this is the reason why the height of the header was set to match the height of the collection view, and why the zIndex was set to -1, because it needed to run underneath the cells.

The final task is to give the headers a proper title, in this case the hour each section represents.

Open **ScheduleViewController.swift** and add the following to viewDidLoad() just below the cell configuration block:

dataSource.headerConfigurationBlock = {(header: ScheduleHeader,

indexPath: NSIndexPath, group: NSDictionary, kind: String) in

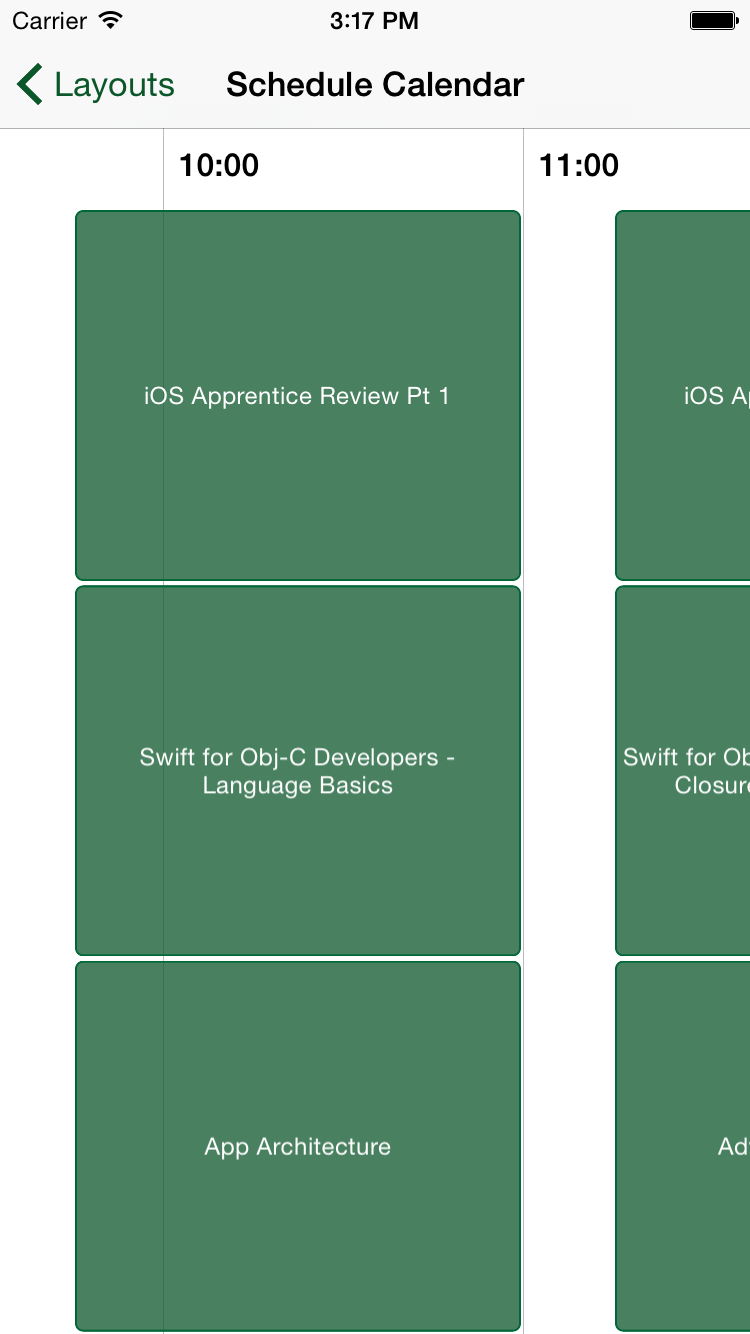
header.titleLabel.text =

dataSource.titleForHourHeaderViewAtIndexPath(indexPath)

}

Here you add a header configuration block to the data source. The data source calls this block once for each header that is to be displayed in the collection view. You use it to set the titleLabel of the header to the title for the hour at the given index path, using a convenience method on the data source.

Build and run. Your headers now have a proper title. Nice work!



Congratulations, the core functionality of your calendar-like custom layout is complete! You’re ready to continue on to the challenges, where you’ll take what you’ve learnt here and add a second set of headers; the Track headers!