

---

# Stack Views

## Nested Stack Views



# Stack Views

---

Auto Layout allows to create flexible interfaces that scale across device types and sizes. However, auto-layouts could be very complex and it can be difficult to create dynamic interfaces due to the need to constantly add and remove constraints.

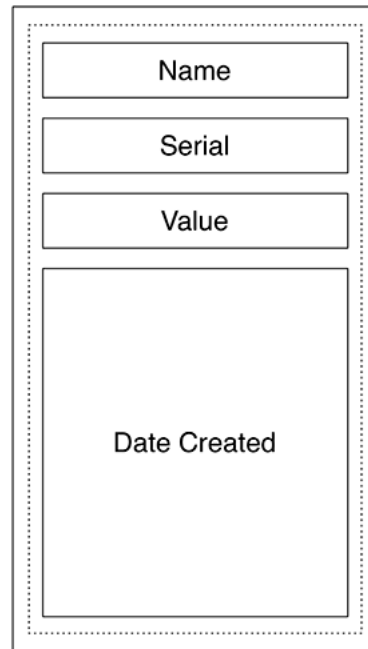
An interface can be laid out in a linear fashion (such as a vertical interface consisting of a text field and a few labels) and can be developed using a stack view.

A stack view is an instance of **UIStackView** that allows you to create a vertical or horizontal layout that is easy to lay out and manages most of the constraints that you would typically have to manage yourself.

Stack views can be nested.

# Stack Views

To create an interface for displaying the details of a specific Item in LootLogger we will use multiple nested stack views, both vertical and horizontal.



# Stack Views

---

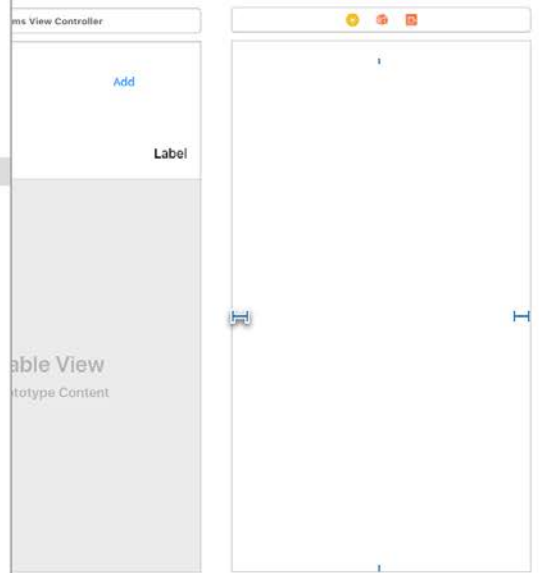
Open LootLogger project and then open Main.storyboard.

Drag a new View Controller from the object library onto the canvas.

Drag a Vertical Stack View from the object library onto the view for the View Controller.

Add 4 constraints to the stack view to pin it to the leading and trailing margins (use control key , and pin the top and bottom edges to be 8 points from the top and bottom layout guides.

- Items View Controller Scene
  - Items View Controller
    - Table View
    - First Responder
    - Exit
    - Storyboard Entry Point
- View Controller Scene
  - View Controller
    - Top Layout Guide
    - Bottom Layout Guide
    - View
      - Stack View
      - Constraints
        - Stack View.top = Top Layout Guide.bottom + 8
        - trailing = Stack View.trailing
        - Stack View.leading = leading
        - Bottom Layout Guide.top = Stack View.bottom + 8
      - First Responder
      - Exit



### Horizontal Space Constraint

First Item: Stack View.Leading  
Relation: Equal  
Second Item: Superview.Leading  
Constant: Standard  
Priority: 1000  
Multiplier: 1  
Identifier: Identifier  
Placeholder:  Remove at build time  
 Installed

led: 1  
xt: 0  
t: 1

**Horizontal Stack View** - Arranges views linearly.

**Vertical Stack View** - Arranges views linearly.

# Stack Views

Now drag four instances of UILabel from the object library onto the stack view. From top to bottom, give these labels the text “Name,” “Serial,” “Value,” and “Date Created”



# Stack Views

The screenshot displays the Xcode interface for editing a storyboard. The left sidebar shows a project named 'Homepwner' with various source files and a 'Main.storyboard' file. The central canvas shows a storyboard with two scenes: 'Items View Controller Scene' and 'View Controller Scene'. The 'View Controller Scene' contains a 'View Controller' which has a 'View' containing a 'Stack View'. The 'Stack View' contains three text labels: 'Name', 'Serial Value', and 'Date created'. The 'Date created' label is selected, and its properties are shown in the right-hand inspector. The inspector shows the following settings for the selected 'Date created' label:

- Label**
  - Text: Plain
  - Color: Default
  - Font: System 17.0
  - Dynamic Type: Automatically Adjusts Font
  - Alignment: Left
  - Lines: 1
  - Behavior: Enabled
  - Highlighted: Default
  - Shadow: Default
  - Shadow Offset: 0 (Width), -1 (Height)
- View**
  - Content Mode: Left
  - Semantic: Unspecified
  - Tag: 0
  - Interaction: User Interaction Enabled

At the bottom of the interface, the status bar shows 'View as: iPhone 8 (w C h R)' and a zoom level of 65%.

# Stack Views

---

The labels all have a red border (indicating an Auto Layout problem) and there is a warning that some views are vertically ambiguous.

There are two ways you can fix this issue:

- by using Auto Layout,
- by using a property on the stack view.

## IMPLICIT CONSTRAINTS

if you do not specify constraints that explicitly determine the width or height, the view will derive its width or height from its intrinsic content size.

It does this using implicit constraints derived from a view's content hugging priorities and its content compression resistance priorities.

- horizontal content hugging priority
- vertical content hugging priority
- horizontal content compression resistance priority
- vertical content compression resistance priority



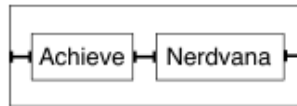


# Stack Views

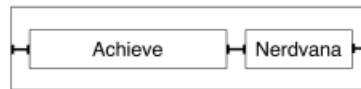
**The content hugging priority** is like a rubber band that is placed around a view. The rubber band makes the view not want to be bigger than its intrinsic content size in that dimension.

Each priority is associated with a value from 0 to 1000.

A value of 1000 means that a view cannot get bigger than its intrinsic content size on that dimension.

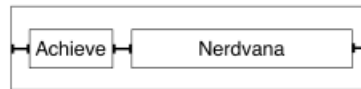


This works great until the superview becomes wider. At that point, which label should become wider? The first label, the second label, or both?



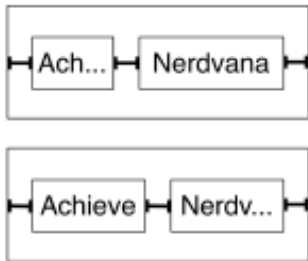
This is where **the content hugging priority** becomes relevant.

The view with the higher content hugging priority is the one that does not stretch. You can think about the priority value as the “strength” of the rubber band. The higher the priority value, the stronger the rubber band, and the more it wants to hug to its intrinsic content size.



# Stack Views

**The content compression resistance priorities** determine how much a view resists getting smaller than its intrinsic content size. Consider the same two labels. What would happen if the superview's width decreased? One of the labels would need to truncate its text. But which one?



The view with the greater content compression resistance priority is the one that will resist compression and, therefore, not truncate its text.

# Stack Views

---

To fix the problem with the stack view using AutoLayout:

Select the Date Created label and open its size inspector.

Find the Vertical Content Hugging Priority and lower it to 249.

Now the other three labels have a higher content hugging priority, so they will all hug to their intrinsic content height.

The Date Created label will stretch to fill in the remaining space.

# Stack Views

To fix the problem with the stack view:

The screenshot displays the Xcode interface for editing a storyboard. The left sidebar shows a hierarchy of views: Items View Controller Scene, Items View Controller (containing Table View, ItemCell, First Responder, and Exit), and View Controller Scene (containing View Controller, View, Safe Area, Stack View, and Constraints). The Stack View contains four labels: Name, Serial, Value, and Date created. The main canvas shows a preview of the storyboard with a stack view containing these labels. The right-hand pane shows the properties for the selected 'Date created' label. The 'Label' section shows 'Preferred Width' set to 'Automatic'. The 'View' section shows 'Show' set to 'Frame Rectangle' with dimensions of 110 width and 61.5 height. The 'Arrange' section is set to 'Position View'. The 'Layout Margins' section is set to 'Default'. The 'Constraints' section shows a diagram of the label with a grid and the text 'Ag'. Below the diagram, it says 'All This Size Class' and 'This view has no constraints'. The 'Content Hugging Priority' section shows 'Horizontal' set to 251.

Items View Controller Scene

- Items View Controller
  - Table View
    - ItemCell
  - First Responder
  - Exit
  - Storyboard Entry Point
- View Controller Scene
  - View Controller
    - View
      - Safe Area
        - Stack View
          - Name
          - Serial
          - Value
          - Date created
    - Constraints
    - First Responder
    - Exit

Label

Preferred Wid... Automatic  Explicit

View

Show Frame Rectangle

X 0 Y 61.5

Width 110 Height 569.5

Arrange Position View

Layout Margins Default

Preserve Superview Margins

Follow Readable Width

Safe Area Relative Margins

Safe Area Layout Guide

Constraints

All This Size Class

This view has no constraints

Content Hugging Priority

Horizontal 251

# Stack Views

---

Another way to fix the issue is using Stack views:

Select the stack view, either on the canvas or using the document outline.

Open its attributes inspector and find the section at the top labeled Stack View.

One of the properties that determines how the content is laid out is the Distribution property. Currently it is set to Fill, which lets the views lay out their content based on their intrinsic content size.

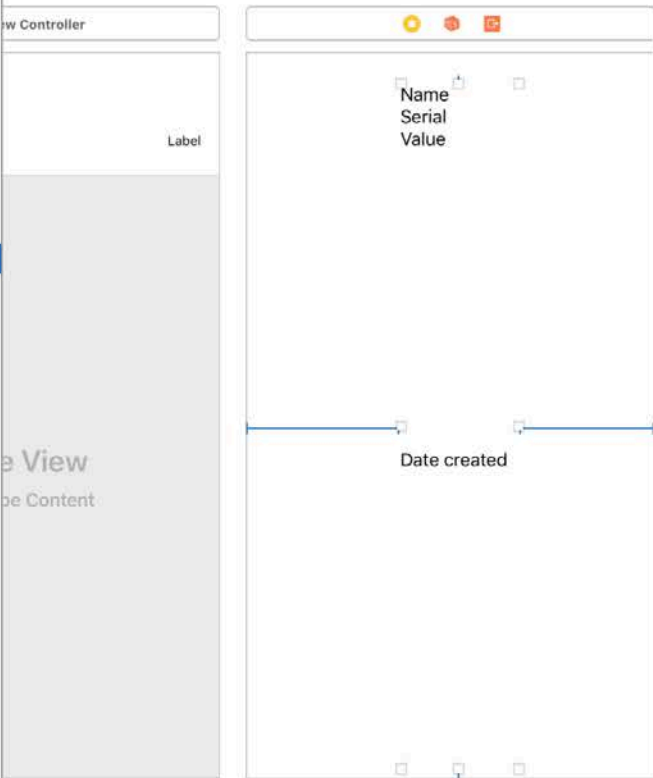
Change the value to **Fill Equally**.

This will resize the labels so that they all have the same height, ignoring the intrinsic content size.

# Stack Views

Homepwner > Homepwner > Main...yboard > Main...(Base) > View...r Scene > View...ntroller > View > Stack View

- Items View Controller Scene
  - Items View Controller
    - Table View
      - ItemCell
    - First Responder
    - Exit
    - Storyboard Entry Point
  - View Controller Scene
    - View Controller
      - View
        - Safe Area
        - Stack View**
          - Name
          - Serial
          - Value
          - Date created
        - Constraints
        - First Responder
        - Exit



### Stack View

Axis: Vertical

Alignment: Fill

Distribution: **Fill** (dropdown menu open)

- Fill
- Fill Equally**
- Fill Proportionally
- Equal Spacing
- Equal Centering

Spacing: [ ]

### View

Content Mode: Scale To Fill

Semantic: Unspecified

Tag: 0

Interaction:  User Interaction Enabled  
 Multiple Touch

Alpha: 1

Background: [Color Picker]

Tint: [Color Picker] Default

Drawing:  Opaque  
 Hidden  
 Clears Graphics Context  
 Clip to Bounds  
 Autoresize Subviews

Stretching: X: 0, Y: 0  
Width: 1, Height: 1

Installed

# Nested Stack Views

---

Change the Distribution of the stack view back to Fill to continue with this example.

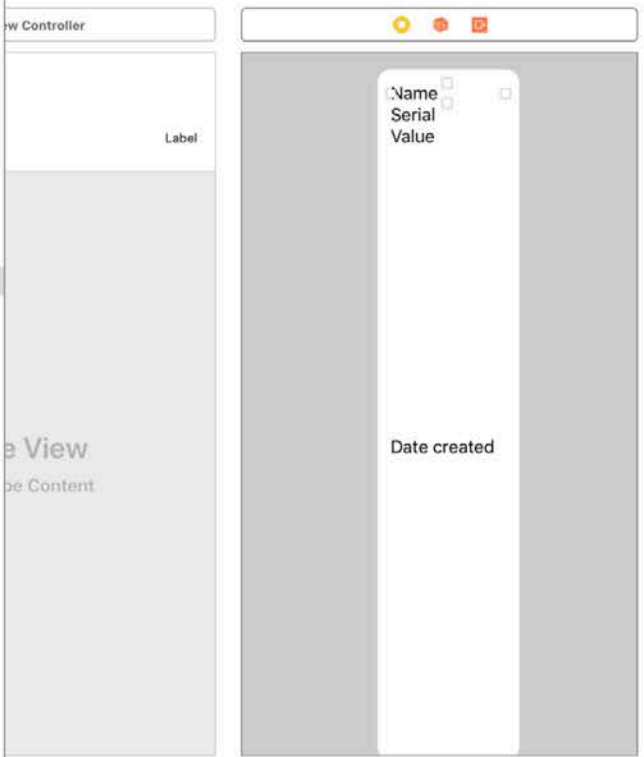
You can nest horizontal stack views within the larger vertical stack view.

Select the Name label on the canvas.



“Click the rightmost icon in the Auto Layout constraints menu () and then select Stack View from the Embed In View section. This will embed the selected view in a stack view.”

- Items View Controller Scene
  - Items View Controller
    - Table View
      - ItemCell
    - First Responder
    - Exit
    - Storyboard Entry Point
- View Controller Scene
  - View Controller
    - View
      - Safe Area
      - Stack View
        - L Name
        - L Serial
        - L Value
        - L Date created
      - Constraints
      - First Responder
      - Exit



**Label**

Text: Plain

Name

Color: Default

Font: System 17.0

Dynamic Type: Automatically Adjusts Font

Alignment: [Left, Center, Right, Justified]

Lines: 1

Behavior:  Enabled,  Highlighted

Baseline: Align Baselines

Line Break: Truncate Tail

Autoshrink: Fixed Font Size

Tighten Letter Spacing

Highlighted: Default

Shadow: Default

Shadow Offset: Width 0, Height -1

**View**

Content Mode: Left

Semantic: Unspecified

Tag: 0

Interaction:  User Interaction Enabled,  Multiple Touch

Alpha: 1

Background: [Color Picker]



# Stack Views

---

Select the new stack view and open its attributes inspector. The stack view is currently a vertical stack view, but you want it to be a horizontal stack view.

Change the Axis to Horizontal.

# Stack Views

Select the new stack view and open its attributes inspector. The stack view is currently a vertical stack view, but you want it to be a horizontal stack view.

Change the Axis to Horizontal.

The screenshot displays the Xcode interface for editing a storyboard. The top navigation bar shows the path: Homepwner > Ho...wner > Mai...oard > Mai...ase) > Vie...cene > Vie...roller > View > Stack View > Stack View. The left sidebar contains a hierarchy of scenes and views:

- Items View Controller Scene
  - Items View Controller
    - TableView
      - ItemCell
    - First Responder
    - Exit
    - Storyboard Entry Point
- View Controller Scene
  - View Controller
    - View
      - Safe Area
      - Stack View
        - Stack View
          - Serial
          - Value
          - Date created
        - Constraints
        - First Responder
        - Exit

The main canvas shows two storyboard scenes. The first scene, 'Items View Controller', contains a 'Label' and a 'View Controller' (which is currently selected). The second scene, 'View Controller', contains a 'View' with a 'Stack View' containing three items: 'Name', 'Serial', and 'Value'. Below the 'Stack View' in this scene are 'Constraints', 'First Responder', and 'Exit'. The right-hand side of the interface shows the 'Attributes Inspector' for the selected 'Stack View'.

**Stack View**

- Axis: Horizontal
- Alignment: Fill
- Distribution: Fill
- Spacing: 0
- Baseline Relative:

**View**

- Content Mode: Scale To Fill
- Semantic: Unspecified
- Tag: 0
- Interaction:  User Interaction Enabled,  Multiple Touch
- Alpha: 1
- Background: [Color Picker]
- Tint: [Color Picker] Default
- Drawing:  Opaque,  Hidden,  Clears Graphics Context

# Stack Views

---

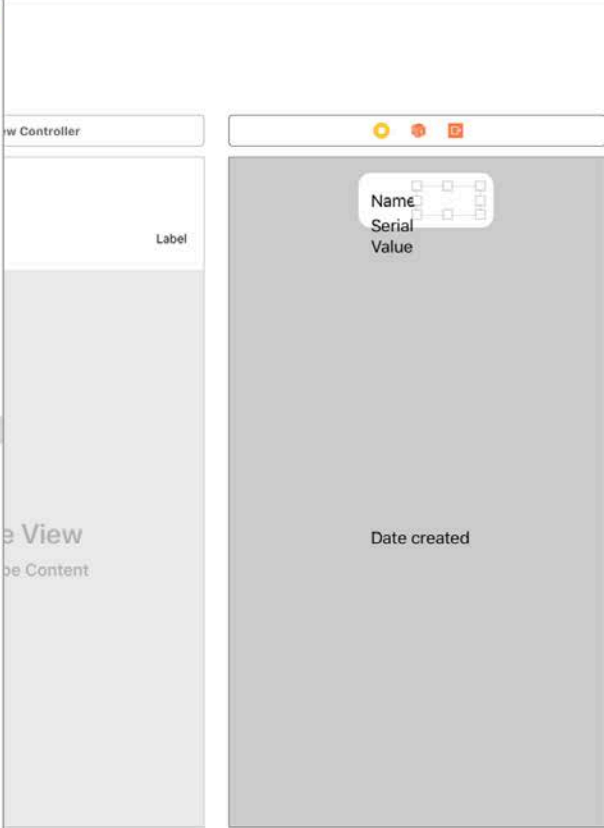
Drag a Text Field from the object library to the right of the Name label. Because labels, by default, have a greater content hugging priority than text fields, the label hugs to its intrinsic content width and the text field stretches.

The label and the text field currently have the same content compression resistance priorities, which would result in an ambiguous layout if the text field's text was too long.

Open the size inspector for the text field and set its **Horizontal Content Compression Resistance Priority to 749**.

This will ensure that the text field's text will be truncated if necessary, rather than the label.

- Items View Controller Scene
  - Items View Controller
    - Table View
      - ItemCell
    - First Responder
    - Exit
    - Storyboard Entry Point
- View Controller Scene
  - View Controller
    - View
      - Safe Area
      - Stack View
        - Stack View
          - Name
          - Round Style Text Field
          - Serial
          - Value
          - Date created
        - Constraints
        - First Responder
        - Exit



**View**

Show **Frame Rectangle**

X: 45 Y: 0

Width: 65 Height: 30

Arrange **Position View**

Layout Margins **Default**

- Preserve Superview Margins
- Follow Readable Width
- Safe Area Relative Margins
- Safe Area Layout Guide

**Constraints**

All **This Size Class**

This view has no constraints

**Content Hugging Priority**

Horizontal: 250

Vertical: 250

**Content Compression Resistance Priority**

Horizontal: **749**

Vertical: 750

Intrinsic Size **Default (System Defined)**

**Text** **Text Field** - Displays editable text and sends an action message to a target object when Return is tapped.

# Stack Views

The label and text field look a little squished because there is no spacing between them.

Stack views allow you to customize the spacing between items.  
Select the horizontal stack view and open its attributes inspector.  
Change the Spacing to be 8points.

Notice that the text field shrinks to accommodate the spacing, because it is less resistant to compression than the label.

The screenshot displays the Xcode interface for configuring a Stack View. On the left, the 'Items View Controller Scene' and 'View Controller Scene' are visible in the hierarchy. The 'Stack View' is selected, showing its sub-items: 'Name', 'Round Style Text Field', 'Serial', 'Value', and 'Date created'. The main canvas shows a preview of the stack view with a 'Label' and a 'Date created' text field. On the right, the 'Stack View' attributes inspector is open, showing the following settings:

- Axis: Horizontal
- Alignment: Fill
- Distribution: Fill
- Spacing: 8
- Baseline Relative:

Below the 'Stack View' section, the 'View' attributes inspector is also visible, showing settings for Content Mode (Scale To Fill), Semantic (Unspecified), Tag (0), Interaction (User Interaction Enabled checked), Multiple Touch (unchecked), Alpha (1), Background (gradient), and Tint (Default).

# Stack Views

---

Repeat these steps for the Serial and Value labels:



1. Select the label and click the icon.
2. Change the stack view to be a horizontal stack view.
3. Drag a text field onto the horizontal stack view and change its horizontal content compression resistance priority to be 749.
4. Update the stack view to have a spacing of 8 points.

# Stack Views

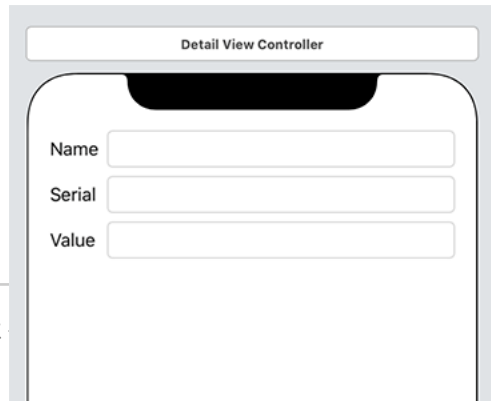
---

Select the vertical stack view, open its attributes inspector, and update the Spacing to be 8 points.

Then select the Date Created label, open its attributes inspector, and change the Alignment to be centered.

The text fields do not align on their leading edge due to the difference in the widths of the labels. To solve this, you will add leading edge constraints between the three text fields.

Control-drag from the Name text field to the Serial text field and select Leading. Then do the same for the Serial text field and the Value text field.



# Stack Views

---

Stack views allow you to create very rich interfaces in a fraction of the time it would take to configure them manually using constraints.

Stack views allow you to have very dynamic interfaces at runtime.

You can add and remove views from stack views by using **`addArrangedSubview(_:)`**, **`insertArrangedSubview(_:at:)`**, and **`removeArrangedSubview(_:)`**.

You can also toggle the `hidden` property on a view in a stack view. The stack view will automatically lay out its content to reflect that value.



# Segues

---

Most iOS applications have a number of view controllers that users navigate between.

Storyboards allow you to set up these interactions as **segues** without having to write code.

A **segue** moves another view controller's view onto the screen and is represented by an instance of **UIStoryboardSegue**.

Each segue has a **style**, an **action item**, and an **identifier**.

The style of a segue determines how the view controller will be presented.

The action item is the view object in the storyboard file that triggers the segue, like a button, a table view cell, or some other UIControl.

The identifier is used to programmatically access the segue.

This is useful when you want to trigger a segue that does not come from an action item, like a shake or some other interface element that cannot be set up in the storyboard file.

# Stack Views

---

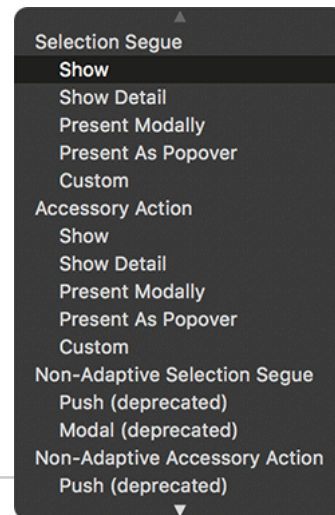
In Main.storyboard, select the ItemCell prototype cell on the Items View Controller.

Control-drag from the cell to the new view controller that you set up in the previous section.

(Make sure you are Control-dragging from the cell and not the table view!)

A black panel will appear that lists the possible styles for this segue.

Select Show from the Selection Segue section.



The image shows the Xcode interface for editing a storyboard. On the left, the storyboard canvas displays a hierarchy: Items View Controller Scene containing an Items View Controller, which contains a Table View. The Table View contains a View, which contains an ItemCell. A context menu is open over the ItemCell, listing various actions such as Selection Segue (Show, Show Detail, Present Modally, Present As Popover, Custom), Accessory Action (Show, Show Detail, Present Modally, Present As Popover, Custom), and Non-Adaptive Selection Segue (Push (deprecated), Modal (deprecated)).

The right side of the interface shows the Table View Cell inspector. The Style is set to Custom, and the Identifier is ItemCell. The Selection is set to Default, and the Accessory is set to None. The Indent While Editing checkbox is checked. The View section shows Content Mode set to Scale To Fill and Semantic set to Unspecified. The Interaction section shows User Interaction Enabled checked.

At the bottom of the interface, there are two sections: View Controller - A controller that manages a view, and Storyboard Reference - Provides a placeholder for a view controller in an external storyboard.

# Stack Views

---

Notice the arrow that goes from the table view controller to the new view controller.

This is a segue. The icon in the circle tells you that this segue is a show segue – each segue has a unique icon.

Build and run the application.

Tap a cell and the new view controller will slide up from the bottom of the screen.

Use Option key to simulate tap.

(Sliding up from the bottom is the default behavior when presenting a view controller modally.)

Sta

Notic

This i  
a uni

Build  
Tap a  
(Slidi  
mod

So fa  
displ  
view



- Items View Controller Scene
  - Items View Controller
    - Table View
      - View
        - ItemCell**
          - Content View
          - First Responder
          - Exit
          - Storyboard Entry Point
          - Show segue to "View Controller"
- View Controller Scene
  - View Controller
    - Top Layout Guide
    - Bottom Layout Guide
    - View
      - Stack View
        - Stack View
          - L Name
          - F Round Style Text Field
        - Stack View
          - L Serial
          - F Round Style Text Field
        - Stack View
          - L Value
          - F Round Style Text Field
        - L Date Created
      - Constraints
      - Constraints
      - First Responder
      - Exit

Items View Controller Scene

Edit Add

Prototype Cells

Label Label

Table View  
Prototype Content

View Controller

Name

Serial

Value

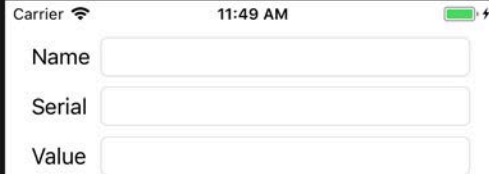
Date Created

# Stack Views

---

So far, so good! But there are two problems at the moment:

1. The view controller is not displaying the information for the Item that was selected,
2. there is no way to dismiss the view controller to return to the ItemsViewController.



Carrier  11:49 AM 

Name

Serial

Value

Date Created



# Stack Views

---

To display the information for the selected Item, you will need to create a new UIViewController subclass.

Create a new Swift file and name it DetailViewController.

Open DetailViewController.swift and declare a new UIViewController subclass named DetailViewController.

```
import UIKit
```

```
class DetailViewController: UIViewController {
```

```
}
```



# Stack Views

---

Because you need to be able to access the subviews you created during runtime, **DetailViewController** needs outlets for them.

The plan is to add four new outlets to **DetailViewController** and then make the connections.





# Stack Views

The screenshot displays the Xcode IDE with three main panels:

- Left Panel (Project Navigator):** Shows the project structure with files like AppDelegate.swift, Main.storyboard, DetailViewController.swift, and ItemCell.
- Center Panel (Code Editor):** Shows the Swift code for `DetailViewController`. The code includes comments and an import statement for `UIKit`.

```
1 //  
2 // DetailViewController.swift  
3 // Homepwner  
4 //  
5 // Created by daria tsoupikova on  
6 // Copyright © 2019 Big Nerd Ranch. All  
7 // rights reserved.  
8  
9 import UIKit  
10  
11 class DetailViewController:  
12     UIViewController {  
13  
14 }  
15  
16
```
- Right Panel (Storyboard Editor):** Shows the storyboard for `ItemCell`. The hierarchy includes `Items View Controller Scene`, `Items View Controller`, `Table View`, `View`, and `ItemCell`. The `ItemCell` contains a `Content View` with a `Label`. A `Text Field` is visible in the bottom right corner of the storyboard, with a tooltip that reads: "Text Field - Displays editable text and sends an action message to a...".



# Stack Views

---

Your window has become a little cluttered.

To make some temporary space:

Hide the navigator area by clicking the left button in the View control at the top of the workspace (the shortcut for this is Command-0).

Then, hide the document outline in Interface Builder by clicking the toggle button in the lower-left corner of the editor.



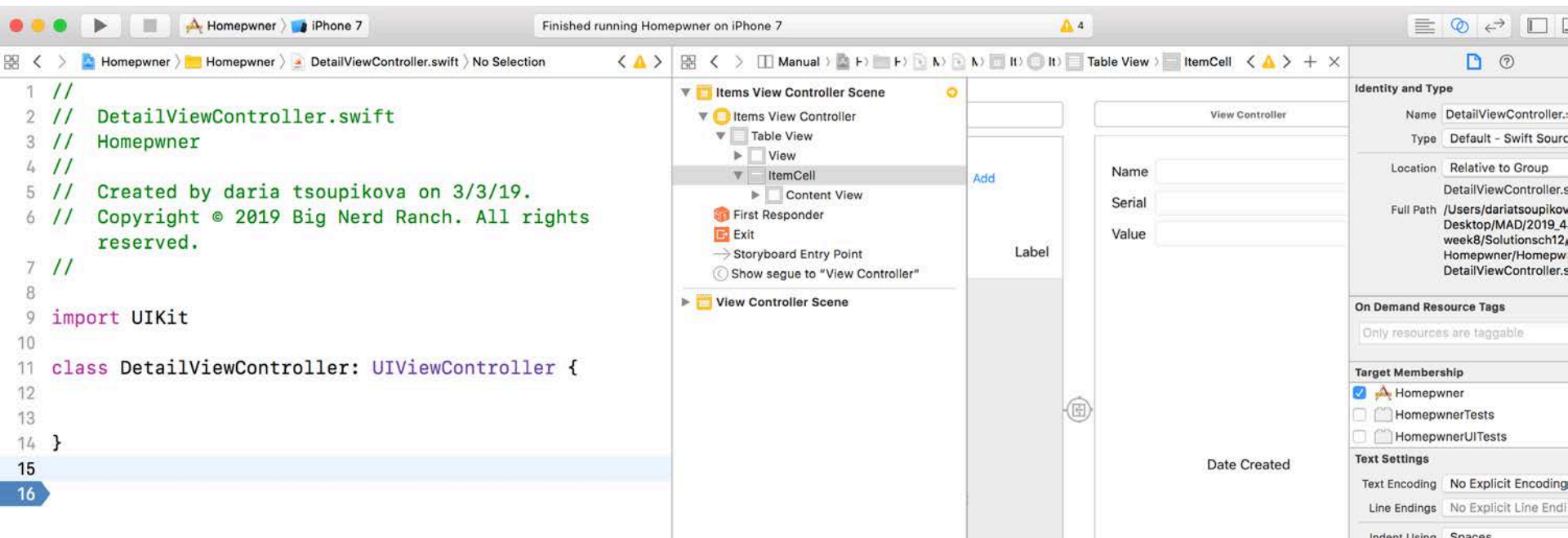
# Stack Views

Your window has become a little cluttered.

To make some temporary space:

Hide the navigator area by clicking the left button in the View control at the top of the workspace (the shortcut for this is Command-0).

Then, hide the document outline in Interface Builder by clicking the toggle button in the lower-left corner of the editor.



# Stack Views

Before you connect the outlets, you need to tell the detail interface that it should be associated with the DetailViewController.

Select the View Controller on the canvas and open its identity inspector. Change the Class to be DetailViewController .

The screenshot shows the Xcode interface with the Identity Inspector open for a Detail View Controller. The breadcrumb at the top reads: View Controller Scene > Detail View Controller. The left sidebar shows a project structure with two scenes: 'Items View Controller Scene' and 'Detail View Controller Scene'. The 'Detail View Controller Scene' is expanded to show 'Detail View Controller'. The central canvas displays a storyboard with a detail view controller containing a stack view with three items: a yellow circle, a red hexagon, and a red square. The Identity Inspector on the right has the following sections:

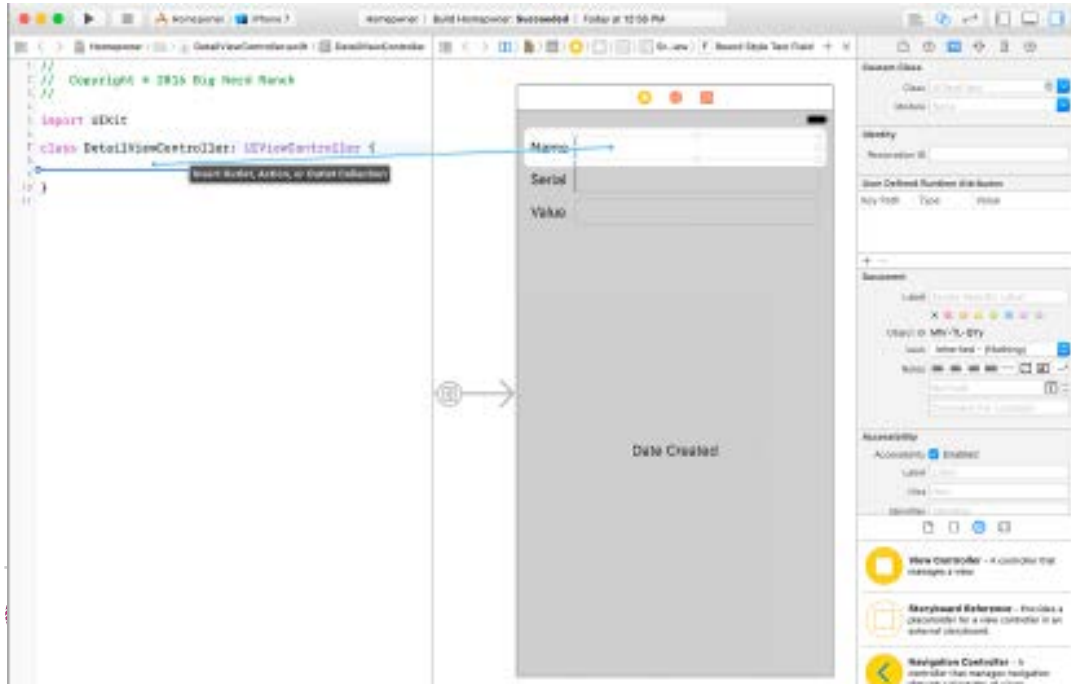
- Custom Class**: Class is set to 'DetailViewController' and Module is 'Homepwner'. The 'Inherit Module From Target' checkbox is checked.
- Identity**: Storyboard ID and Restoration ID fields are empty. The 'Use Storyboard ID' checkbox is unchecked.
- User Defined Runtime Attributes**: A table with columns for Key Path, Type, and Value.

Key Path	Type	Value
----------	------	-------

# Stack Views

The three instances of UITextField and bottom instance of UILabel will be outlets in DetailViewController.

Control-drag from the UITextField next to the Name label to the top of DetailViewController.swift.



# Stack Views

Homeowner > Homeowner > DetailViewController.swift > No Selection

```
1 //  
2 // DetailViewController.swift  
3 // Homeowner  
4 //  
5 // Created by daria tsoupikova on 3/3/19.  
6 // Copyright © 2019 Big Nerd Ranch. All rights reserved.  
7  
8  
9  
10  
11  
12  
13  
14 }  
15  
16
```

Connection: Outlet  
Object: Detail View Controller  
Name: nameField  
Type: UITextField  
Storage: Strong  
Cancel Connect

er: UIViewController {

Round Style Text Field

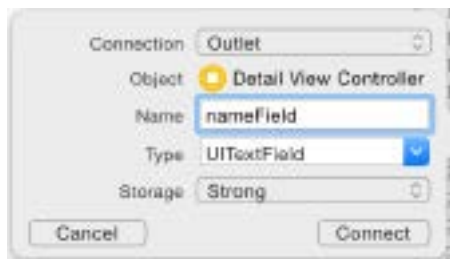
- Items View Controller Scene
  - Items View Controller
    - Table View
      - View
        - ItemCell
          - Content View
  - First Responder
  - Exit
  - Storyboard Entry Point
  - Show segue to "View Controller"
- Detail View Controller Scene
  - Detail View Controller
    - Top Layout Guide
    - Bottom Layout Guide
    - View
      - Stack View
        - Stack View
          - L Name
          - F Round Style Text Field
        - Stack View
        - Stack View
          - L Date Created
        - Constraints
      - Constraints
    - First Responder
    - Exit

Name  
Serial  
Value  
Date Created

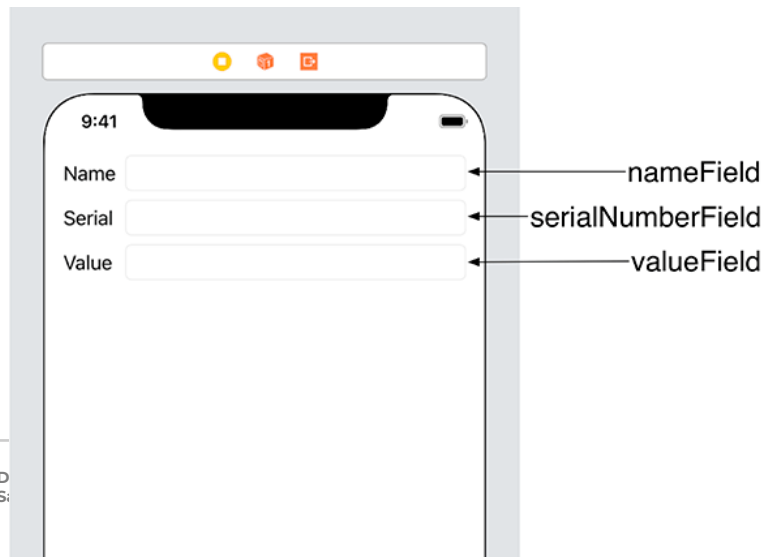
# Stack Views

Let go and a pop-up window will appear. Enter nameField into the Name field, make sure the Storage is set to Strong, and click Connect .

This will create an @IBOutlet property of type UITextField named nameField in DetailViewController.



Create the other three outlets the same way and name them:

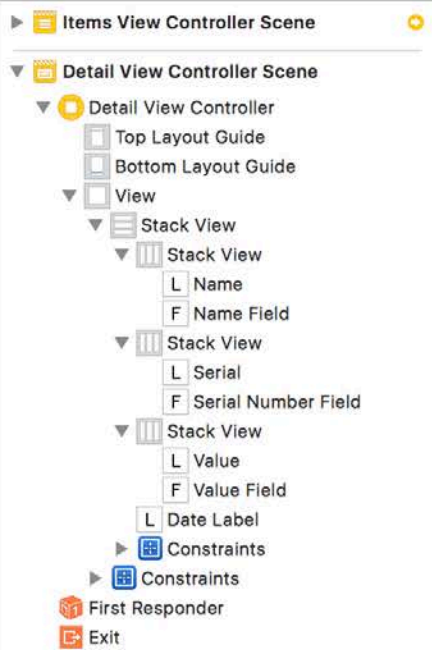


# Stack Views

Homepwner > Homepwner > DetailViewController.swift > No Selection

```
1 //
2 // DetailViewController.swift
3 // Homepwner
4 //
5 // Created by daria tsoupikova on 3/3/19.
6 // Copyright © 2019 Big Nerd Ranch. All rights
  reserved.
7 //
8
9 import UIKit
10
11 class DetailViewController: UIViewController {
12
13     @IBOutlet var nameField: UITextField!
14     @IBOutlet var valueField: UITextField!
15     @IBOutlet var serialNumberField: UITextField!
16     @IBOutlet var dateLabel: UILabel!
17 }
18
19
```

Manual > H > H > M > Main.storyboard (Base) > No Selection





# Stack Views

---

“After you make the connections, DetailViewController.swift should look like below. “If your file looks different, then your outlets are not connected correctly.

```
import UIKit
```

```
class DetailViewController: UIViewController {  
  
    @IBOutlet var nameField: UITextField!  
    @IBOutlet var serialNumberField: UITextField!  
    @IBOutlet var valueField: UITextField!  
    @IBOutlet var dateLabel: UILabel!  
  
}
```

# Stack Views

---

No bad connections – debugger could be used to fix them.

A bad connection typically happens when you change the name of a property but do not update the connection in the interface file or when you completely remove a property but do not remove it from the interface file.

A bad connection will cause your application to crash when the interface file is loaded.



# Stack Views

---

In `DetailViewController.swift`, add a property for an `Item` instance:

```
class DetailViewController: UIViewController {  
  
    @IBOutlet var nameField: UITextField!  
    @IBOutlet var serialNumberField: UITextField!  
    @IBOutlet var valueField: UITextField!    @IBOutlet var dateLabel: UILabel!  
  
    var item: Item!  
    override func viewWillAppear(_ animated: Bool) {  
    super.viewWillAppear(animated)  
    nameField.text = item.name  
    serialNumberField.text = item.serialNumber  
    valueField.text = "\(item.valueInDollars)"  
    dateLabel.text = "\(item.dateCreated)"  
    }  
    }
```

```
8
9 import UIKit
10
11 class DetailViewController: UIViewController {
12
13     @IBOutlet var nameField: UITextField!
14     @IBOutlet var valueField: UITextField!
15     @IBOutlet var serialNumberField: UITextField!
16     @IBOutlet var dateLabel: UILabel!
17
18     var item: Item!
19     override func viewWillAppear(_ animated: Bool) {
20         super.viewWillAppear(animated)
21         nameField.text = item.name
22         serialNumberField.text = item.serialNumber
23         valueField.text = "\(item.valueInDollars)"
24         dateLabel.text = "\(item.dateCreated)"
25     }
26 }
27
```

# Stack Views

---

instead of using string interpolation to print out the `valueInDollars` and `dateCreated`, it would be better to use a formatter. Add an instance of `NumberFormatter` and `DateFormatter` to the `DetailViewController`. Use these formatters in `viewWillAppear(_:)` to format the `valueInDollars` and `dateCreated`.

```
var item: Item!
```

```
let numberFormatter: NumberFormatter = {  
let formatter = NumberFormatter()  
formatter.numberStyle = .decimal  
formatter.minimumFractionDigits = 2  
formatter.maximumFractionDigits = 2  
return formatter  
}
```

```
let dateFormatter: DateFormatter = {  
let formatter = DateFormatter()  
formatter.dateStyle = .medium  
formatter.timeStyle = .none  
return formatter  
}
```



# Stack Views

---

```
let dateFormatter: DateFormatter = {  
    let formatter = DateFormatter()  
    formatter.dateStyle = .medium  
    formatter.timeStyle = .none  
    return formatter  
}()
```

```
valueField.text = "\(item.valueInDollars)"  
dateLabel.text = "\(item.dateCreated)"
```

```
valueField.text =  
numberFormatter.string(from: NSNumber(value: item.valueInDollars))    dateLabel.text =  
dateFormatter.string(from: item.dateCreated)
```

```
10
11 class DetailViewController: UIViewController {
12
13     @IBOutlet var nameField: UITextField!
14     @IBOutlet var valueField: UITextField!
15     @IBOutlet var serialNumberField: UITextField!
16     @IBOutlet var dateLabel: UILabel!
17
18     var item: Item!
19
20     let numberFormatter: NumberFormatter = {
21         let formatter = NumberFormatter()
22         formatter.numberStyle = .decimal
23         formatter.minimumFractionDigits = 2
24         formatter.maximumFractionDigits = 2
25         return formatter
26     }()
27
28     let dateFormatter: DateFormatter = {
29         let formatter = DateFormatter()
30         formatter.dateStyle = .medium
31         formatter.timeStyle = .none
32         return formatter
33     }()
34
35
```

# Stack Views

36

```
37     override fun viewWillAppear(_ animated: Bool) {
38         super.viewWillAppear(animated)
39         nameField.text = item.name
40         serialNumberField.text = item.serialNumber
41         /* valueField.text = "\(item.valueInDollars)"
42         dateLabel.text = "\(item.dateCreated)"*/
43         valueField.text =
44             numberFormatter.string(from: NSNumber(value:
45                 item.valueInDollars))
46         dateLabel.text = dateFormatter.string(from: item.dateCreated)
47     }
48 }
49
```



# Passing Data Around

---

When a row in the table view is tapped, you need a way of telling the DetailViewController which item was selected.

Whenever a segue is triggered, the `prepare(for:sender:)` method is called on the view controller initiating the segue.

This method has two arguments: the `UIStoryboardSegue`, which gives you information about which segue is happening, and the sender, which is the object that triggered the segue (a `UITableViewCell` or a `UIButton`, for example).

The `UIStoryboardSegue` gives you three pieces of information:

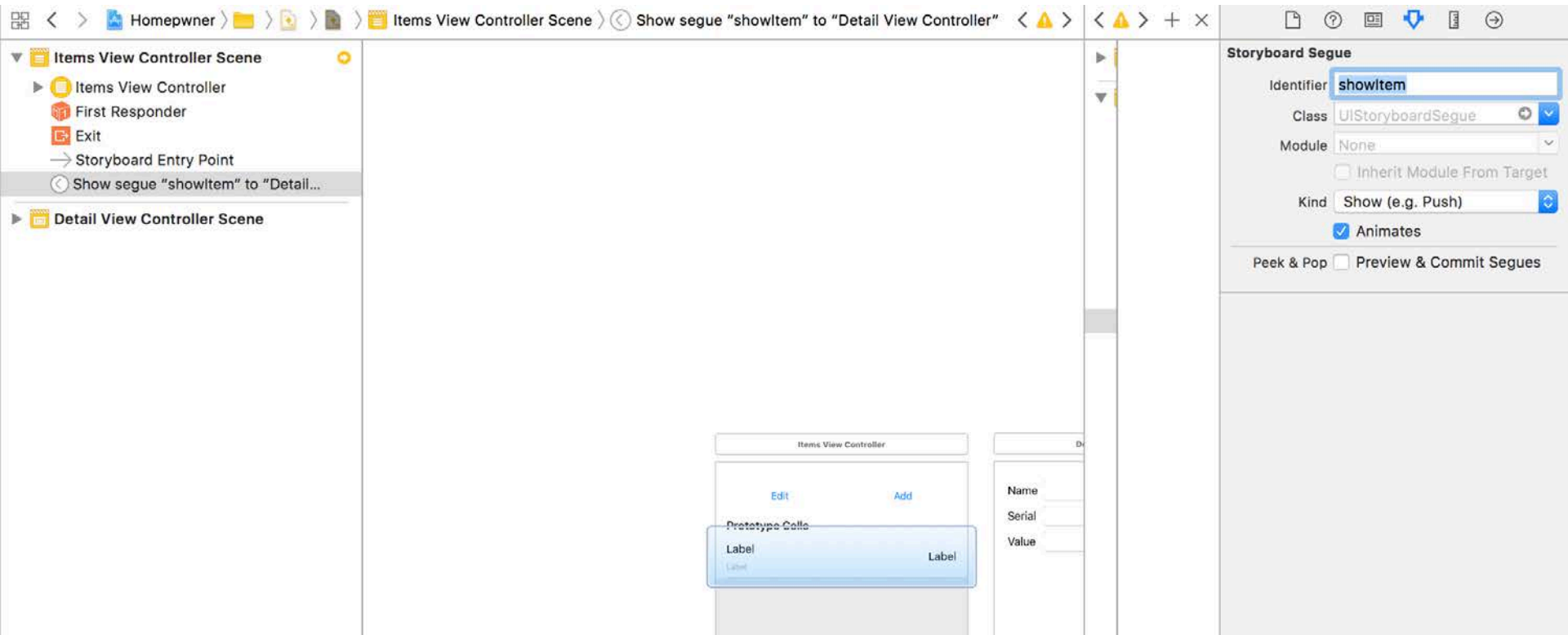
1. the source view controller (where the segue originates),
2. the destination view controller (where the segue ends),
3. the identifier of the segue. The identifier lets you differentiate segues.

The identifier is missing now

# Stack Views

Open Main.storyboard again.

Select the show segue by clicking on the arrow between the two view controllers and open the attributes inspector. For the identifier, enter showItem.



# Stack Views

---

Open `ItemsViewController.swift` and implement `prepare(for:sender:)`.

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {  
// If the triggered segue is the "showItem" segue  
switch segue.identifier {  
case "showItem"?:  
  
// Figure out which row was just tapped  
if let row = tableView.indexPathForSelectedRow?.row {  
  
// Get the item associated with this row and pass it along  
let item = itemStore.allItems[row]  
let detailViewController  
= segue.destination as! DetailViewController  
detailViewController.item = item  
{  
default:  
preconditionFailure("Unexpected segue identifier.")  
}  
}
```



```
118
119 override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
120     // If the triggered segue is the "showItem" segue
121     switch segue.identifier {
122     case "showItem"?:
123
124         // Figure out which row was just tapped
125         if let row = tableView.indexPathForSelectedRow?.row {
126
127             // Get the item associated with this row and pass it
128             // along
129             let item = itemStore.allItems[row]
130             let detailViewController
131                 = segue.destination as! DetailViewController
132             detailViewController.item = item
133         }
134         default:
135             preconditionFailure("Unexpected segue
136                 identifier.")
137     }
138 }
```

# Stack Views

---

Build and run the application. Tap a row and the `DetailViewController` will slide onscreen, displaying the details for that item. And you can dismiss the detail screen by swiping down on the interface – that makes two points in the plus column.

But there is still work to be done. One issue is that any changes that you make to the item's details will not persist. And, in terms of style, there is a more conventional way to present and dismiss detail screens.



# Stack Views

Build and run the application.

Tap on a row and the DetailViewController will slide onscreen, displaying the details for that item.

