

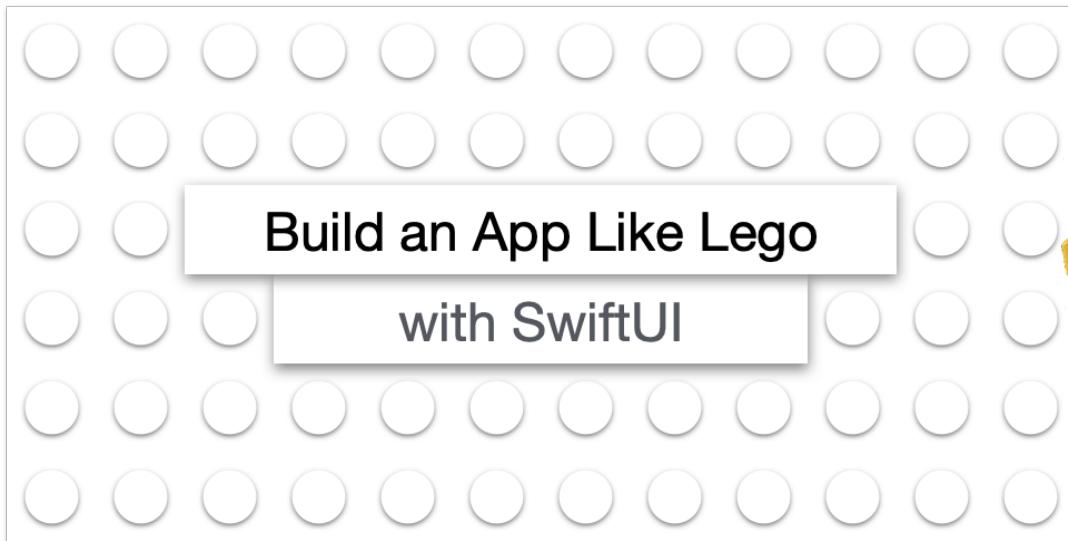
[Open in app](#)Published in Next Level Swift · [Following](#) ▾Tom Brodhurst-Hill · [Following](#)

Mar 23, 2021 · 10 min read ★

...

Create a new App from an Xcode Template

Build an App Like Lego, with SwiftUI — Tutorial 1



“Emmet” figure courtesy of [Pixy](#), under Creative Commons

1. Introduction

We all use apps every day to simplify and enrich our lives, but building them can seem like an overwhelming challenge. If you have an idea for an app — even a very simple idea — it could cost thousands of dollars to hire someone to create it. In reality, it's not that hard to get started. In this series, I'm going to demonstrate that you can create an app as easily as you can build something out of Lego.

LEGO® is a trademark of the LEGO Group of companies that do not sponsor, authorize, or endorse this tutorial series.

Together, we will:

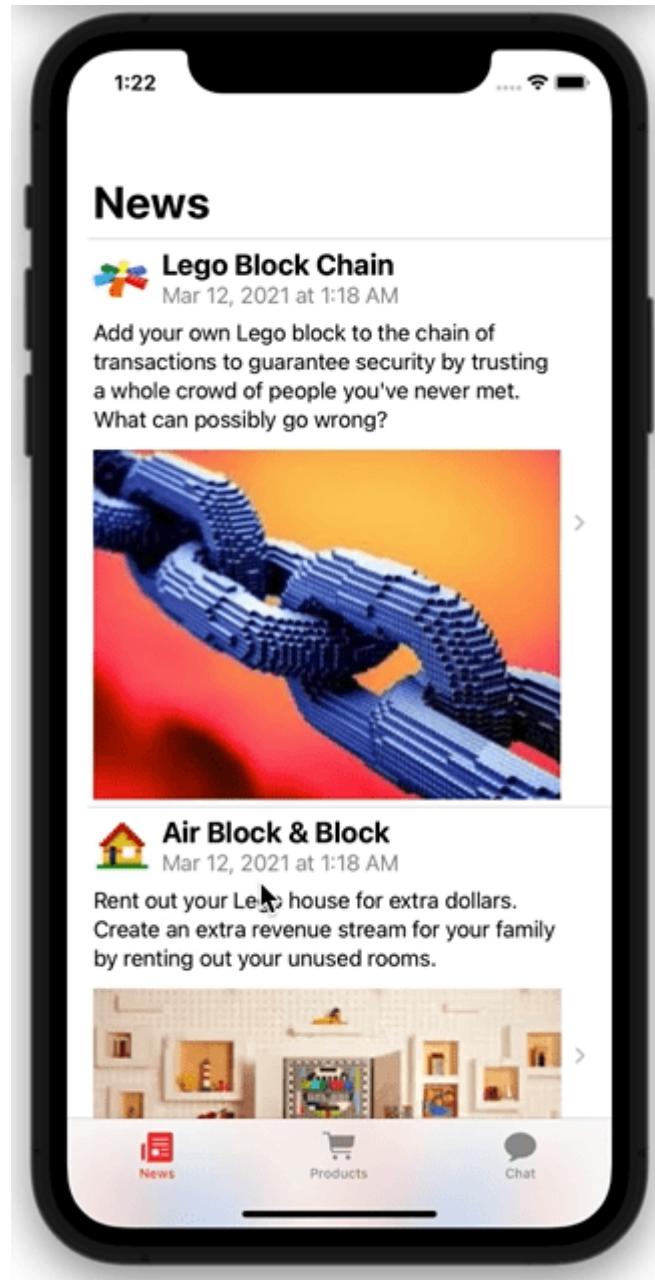
1. Create real app components, using visual tools.



[Open in app](#)

The app we build will be truly “native”. We will use Apple’s toolset to build an app that is “native” to the iOS platform. It will be a real app, not built with some prototyping tool or as a web page designed to look like an app.

The app will look something like this:



To follow along in this tutorial series, you don’t need to be an expert. You only need:

1. A Mac capable of running the latest software.

2. Good working knowledge of how to use a Mac, such as how to launch an app



[Open in app](#)

3. Xcode (which we will download together in this first tutorial).

4. A desire to learn how to build an app.

If you'd like to look ahead, you can peek at the whole tutorial series in the [table of contents](#).

2. Install Xcode

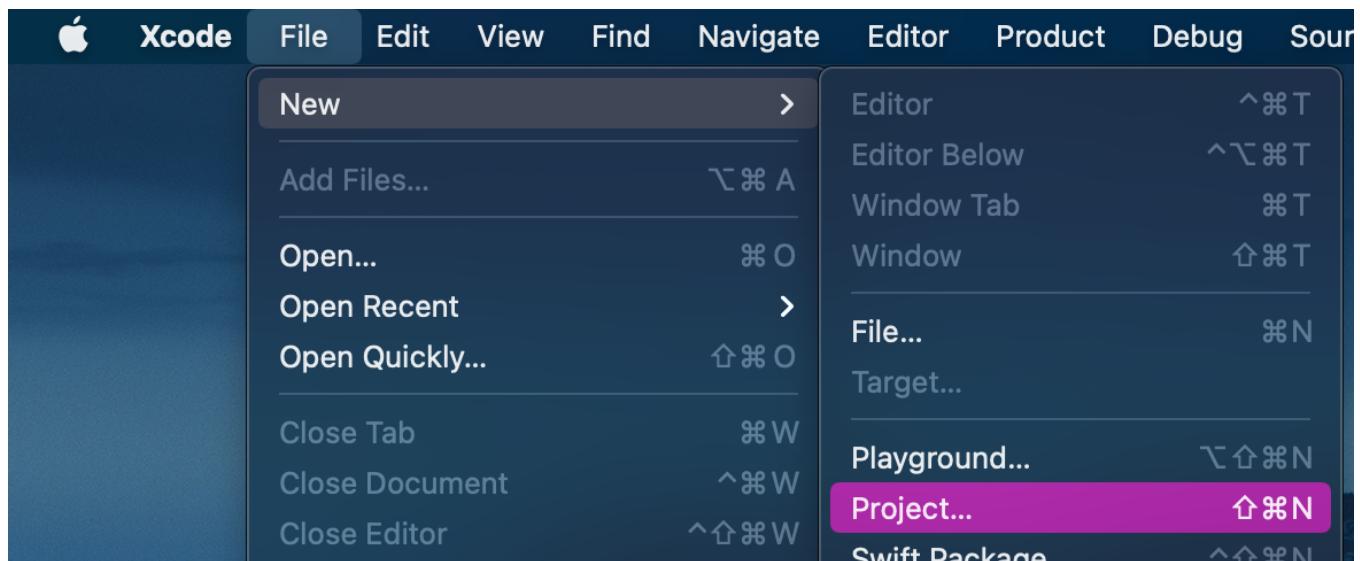
First, you will need Apple's Xcode software. It's free. I will be showing Xcode 12.5 in this tutorial.

Follow the instructions in the separate "[Install and Configure Xcode](#)" article. It is quick and easy. It shows you how to install Xcode and configure it, ready for this tutorial series.

3. Create a New Project

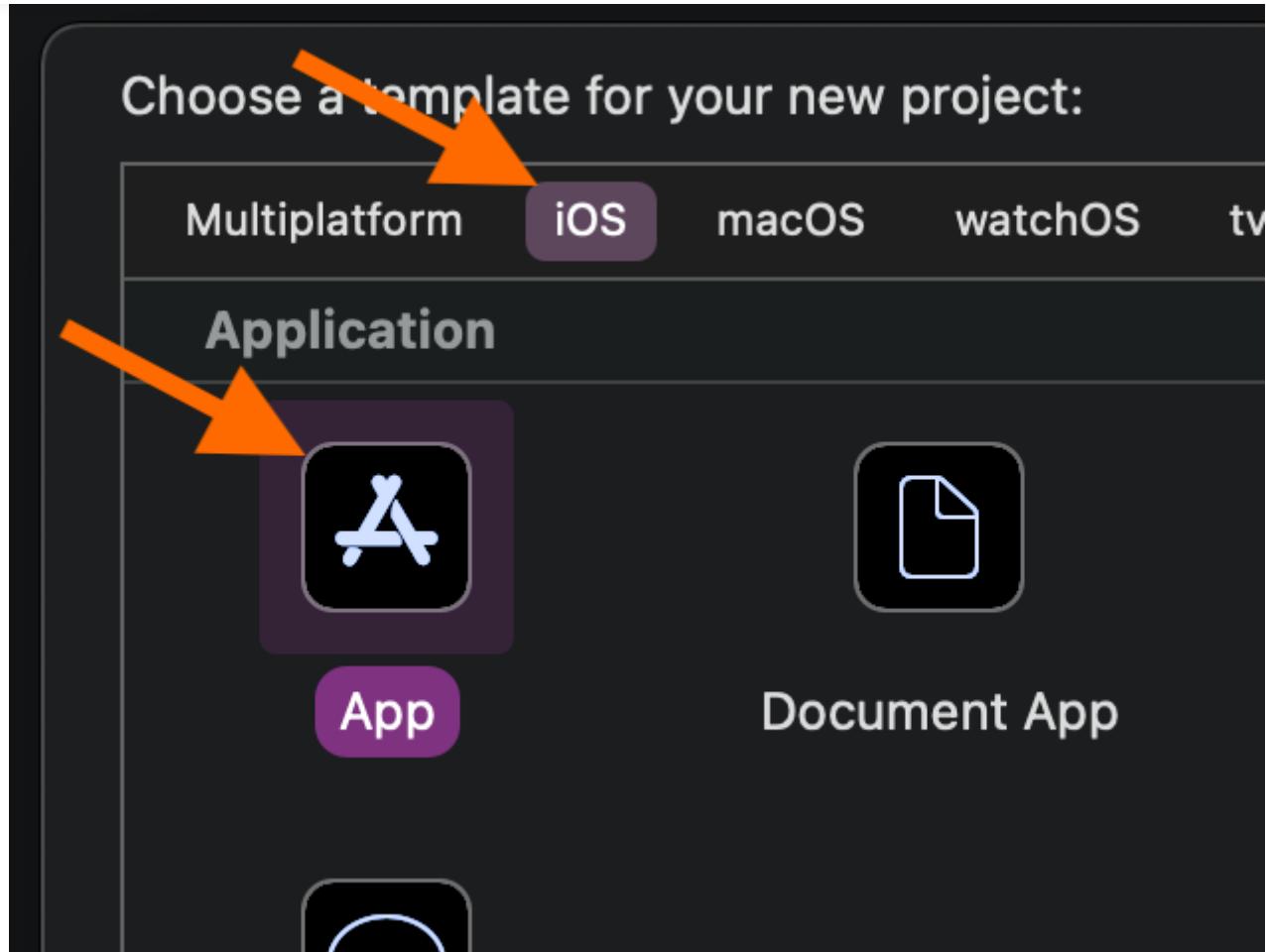
Now let's start to build an app!

👉 In Xcode's File menu, select New > Project .



👉 Xcode shows a bunch of templates. Choose iOS , then App . Make sure you choose

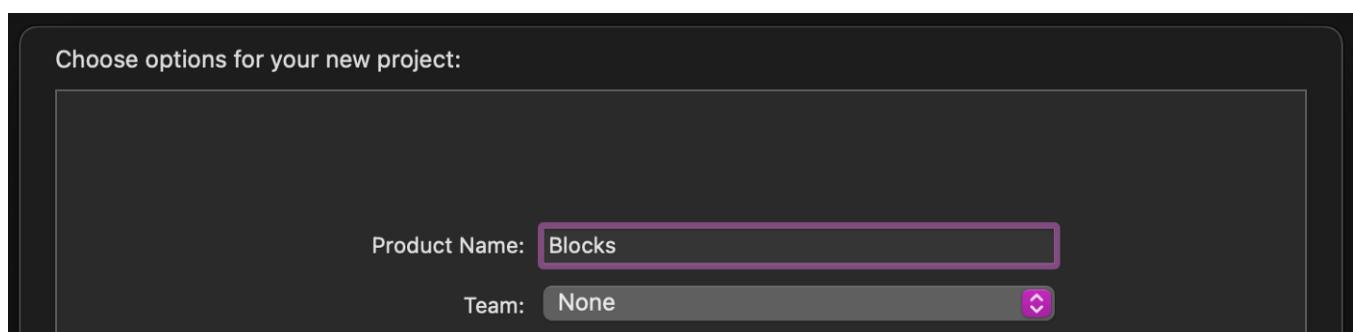


[Open in app](#)

👉 Click the `Next` button.

👀 Xcode asks you to Choose options for your new project .

👉 Enter the Product Name for your app as `Blocks` .

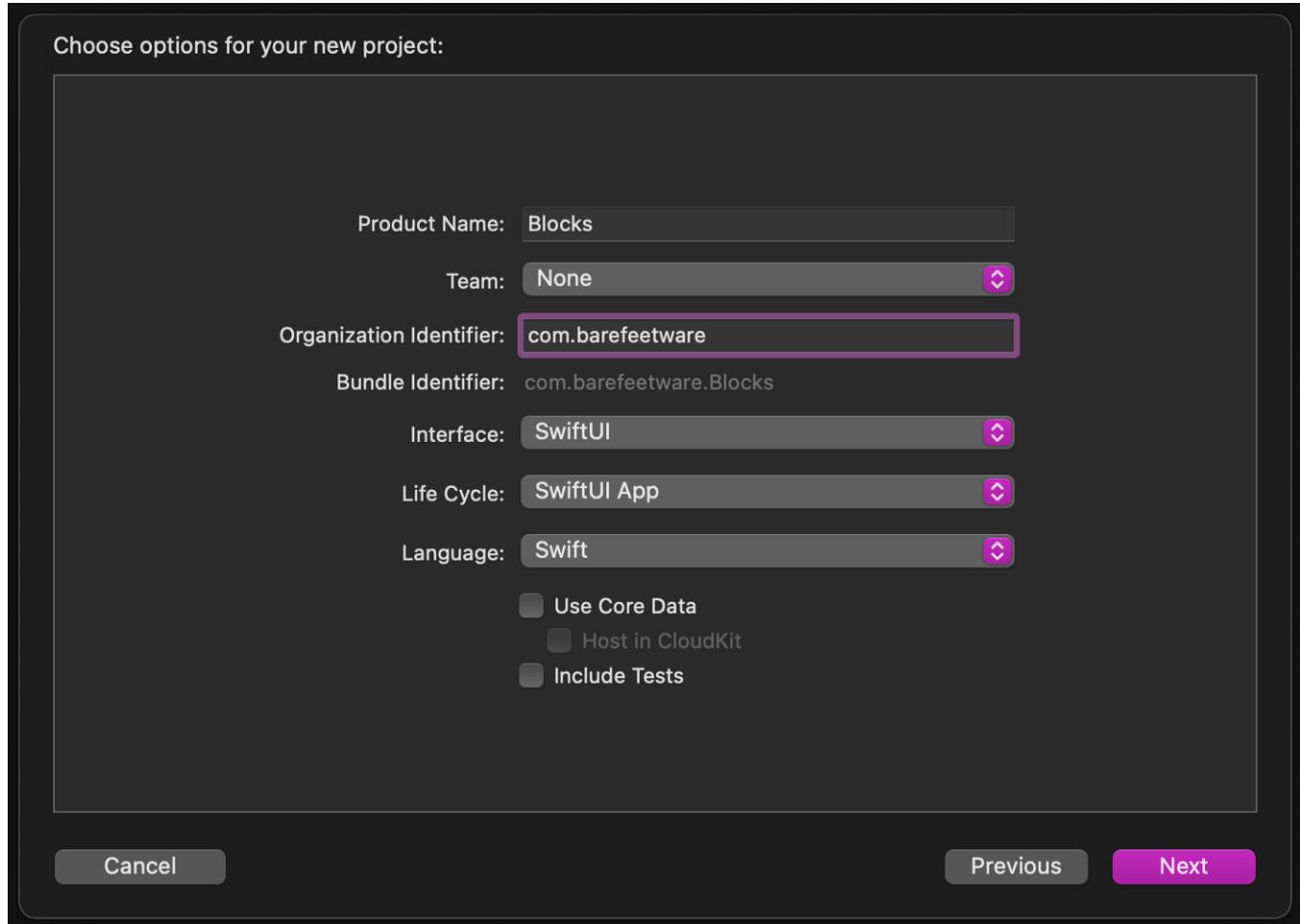


👉 You can leave the Team set to None . Or you can select a team if you have one. If it instead shows an Add Account button, then make sure that you completed the previous section “Enter Your Details”.



[Open in app](#)

identifier as `com.mycompany`. If you don't have a domain name, you can just make one up or enter anything you like. You can change it later, but you can't leave it blank here.



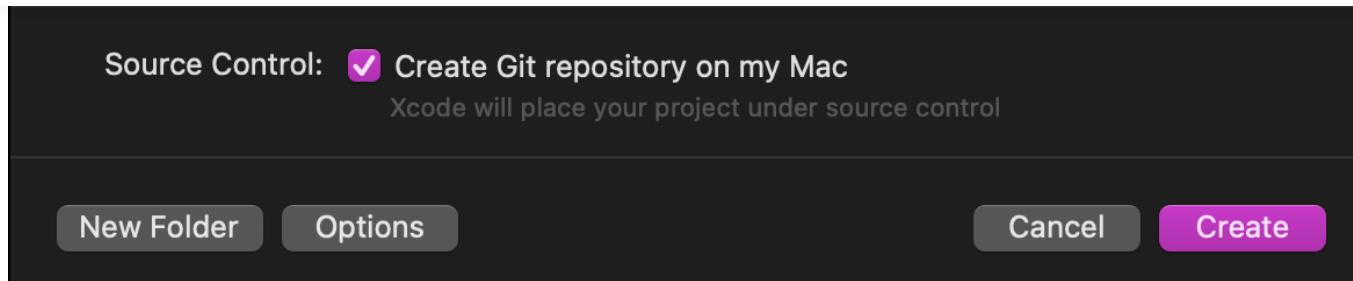
👉 Make sure that:

1. The `Interface` is set to `SwiftUI`.
2. The `Life Cycle` is set to `SwiftUI App`
3. The `Language` is set to `Swift`.
4. All the checkboxes are turned off.

👉 Click the `Next` button.

👉 Xcode asks where to save your new project. Turn on `Create Git repository`, so that we can commit changes. We'll discuss "commit" and "Git" later.



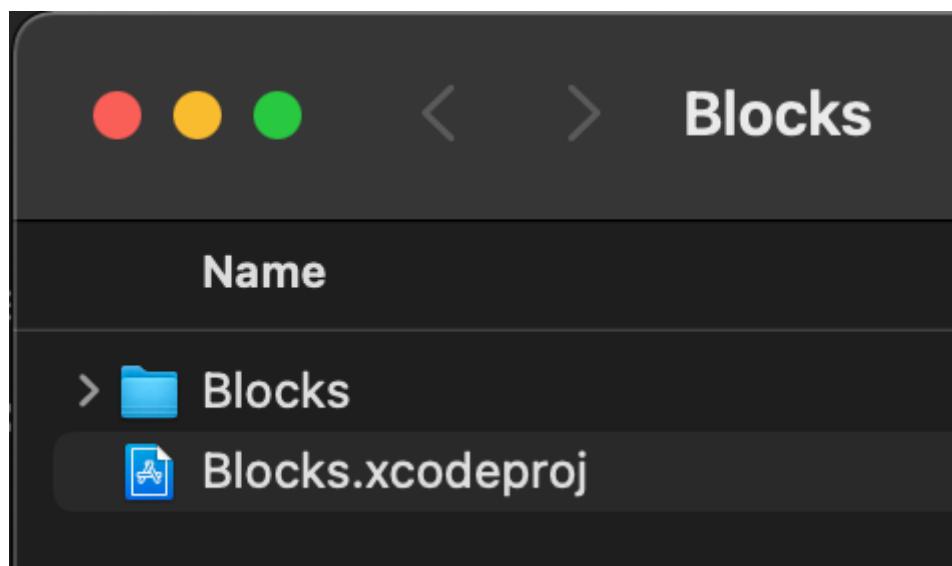
[Open in app](#)

👉 Choose to save your new project in your Documents folder or wherever makes sense to you.

👉 Click the `Create` button to save it.

Xcode will create a folder (with the same name as your chosen `Product Name`) containing all of the files needed to build our app.

In the Finder, it looks like this:



Congratulations! You've just created a native iOS app!

👁️ Xcode now shows your project containing several files. It defaults to selecting the `ContentView.swift` file.



[Open in app](#)

The screenshot shows the Xcode interface with the following details:

- Project Navigator:** Shows the project structure under the "Blocks" target. The "ContentView.swift" file is selected and highlighted with a pink background.
- Editor:** Displays the content of the "ContentView.swift" file. The code is as follows:

```
1 //  
2 // ContentView.swift  
3 // Blocks  
4 //  
5 // Created by Tom Brodhurst-Hill on 4/3/21.  
6 // Copyright © 2021 BareFeetWare. All rights reserved.  
7 //  
8  
9 import SwiftUI  
10  
11 struct ContentView: View {  
12     var body: some View {  
13         Text("Hello, world!")  
14             .padding()  
15     }  
16 }  
17  
18 struct ContentView_Previews: PreviewProvider {  
19     static var previews: some View {  
20         ContentView()  
21     }  
22 }  
23
```

Let's check that we can run the app.

👉 Click on the `Blocks >` button.



👉 In the popup menu of devices, choose the `iPhone 11` simulator.



[Open in app](#)

✓ Blocks >

Edit Scheme...

New Scheme...

Manage Schemes...

```
2 // ContentView
3 // Blocks
4 //
5 // Created by
6 //
7
8 import SwiftUI
9
10 struct ContentView {
11     var body: some View {
12         Text("Hello, world!")
13     }
14 }
15
16
17 struct ContentView_Previews {
18     static var preview: some View {
19         ContentView()
20     }
21 }
22
```

No Devices

No devices connected to 'My Mac'...

Build

Any iOS Device (arm64)

iOS Simulators

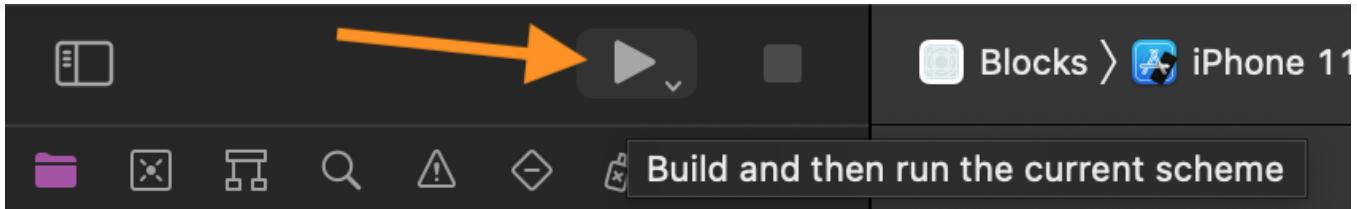
- iPad (8th generation)
- iPad Air (4th generation)
- iPad Pro (9.7-inch)
- iPad Pro (11-inch) (2nd generation)
- iPad Pro (12.9-inch) (4th generation)
- iPhone 8
- iPhone 8 Plus
- iPhone 11
- iPhone 11 Pro
- iPhone 11 Pro Max
- iPhone 12
- iPhone 12 Pro
- iPhone 12 Pro Max
- iPhone 12 mini
- iPhone SE (2nd generation)
- iPod touch (7th generation)

Add Additional Simulators...

Download Simulators...

👉 Click on the Build and Run button (it looks like the “Play” button in iTunes) in the

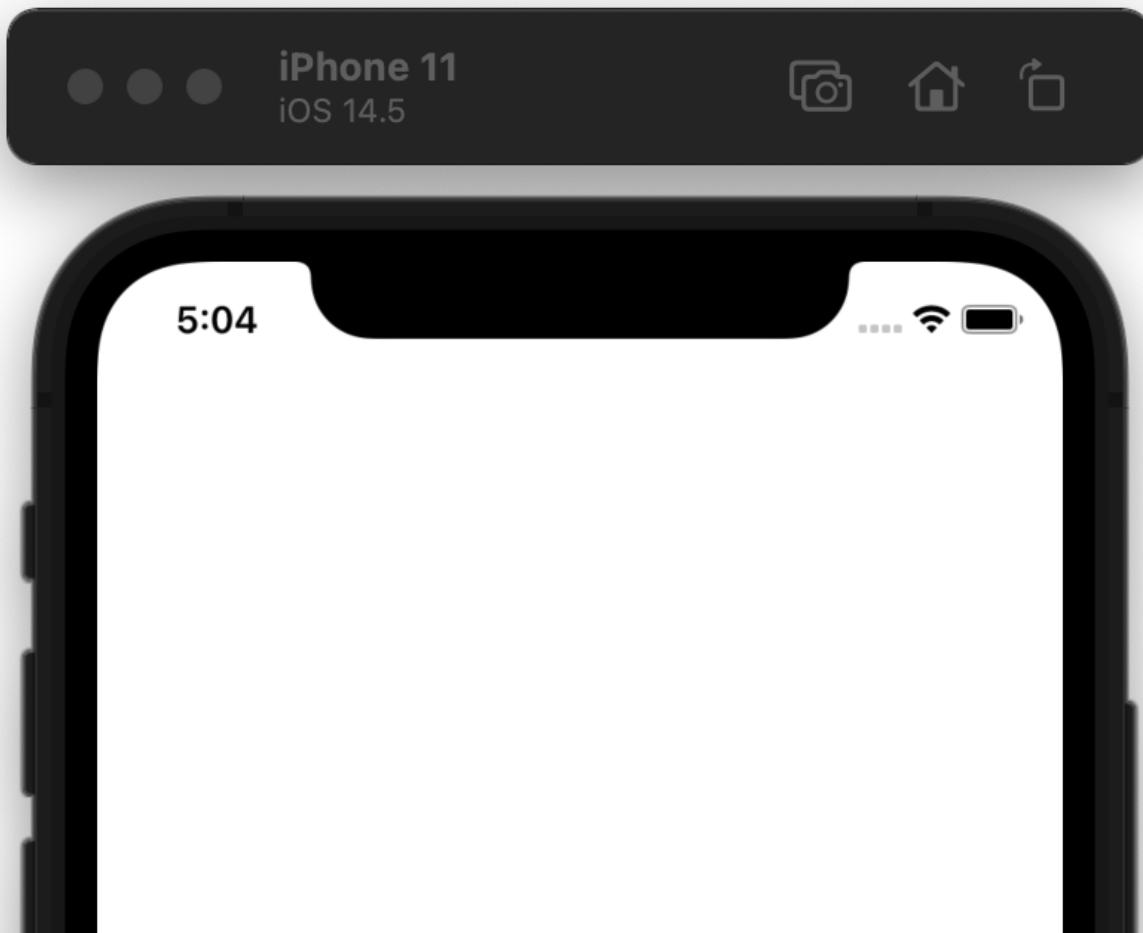


[Open in app](#)

Be patient. It will take a few minutes for Xcode to build and run the app. Xcode will launch a new Mac app called “Simulator” in a window that looks like an iPhone. It is a simulated iPhone 11 (or whatever device appeared in the top left, next to the `Build and Run` button).

🕒 At first, the iPhone screen will appear black. Eventually, you will see the app launch inside the simulated iPhone.

👉 If necessary, adjust the size of the simulator window to make it all visible. You can drag one of the corners, or select `Physical Size` from the `Window` menu.



[Open in app](#)

There — you've successfully built and run an app! It's not a prototype that tries to emulate a native app. It **is** a real native app. You could deploy this on the App Store. Well, you could try, but as you'd hope, Apple will reject it because it doesn't actually do anything useful — yet.

4. Xcode Panels

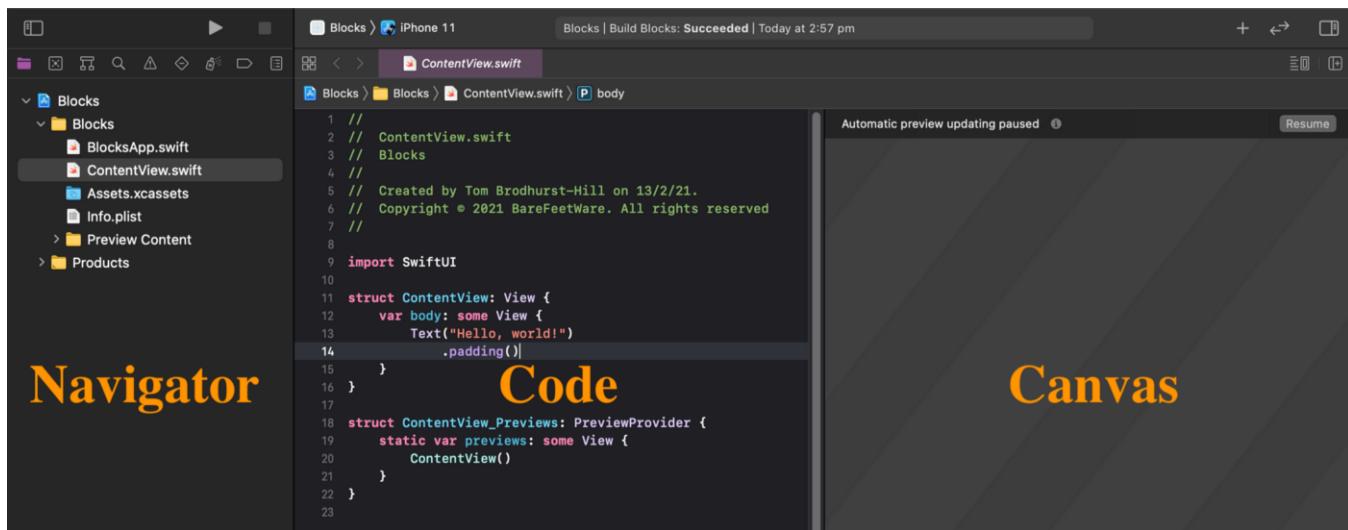
👉 Switch back to Xcode.



[Open in app](#)

The Xcode window is divided into several panels. Let's run through some basic tips on how to move around, hide and show these panels. Don't worry about the content or purpose of these panels for now. We will discuss that later.

The left panel is the “Navigator”, currently showing the “Project Navigator” (a list of files in the project). The right panel is the “Inspector”.

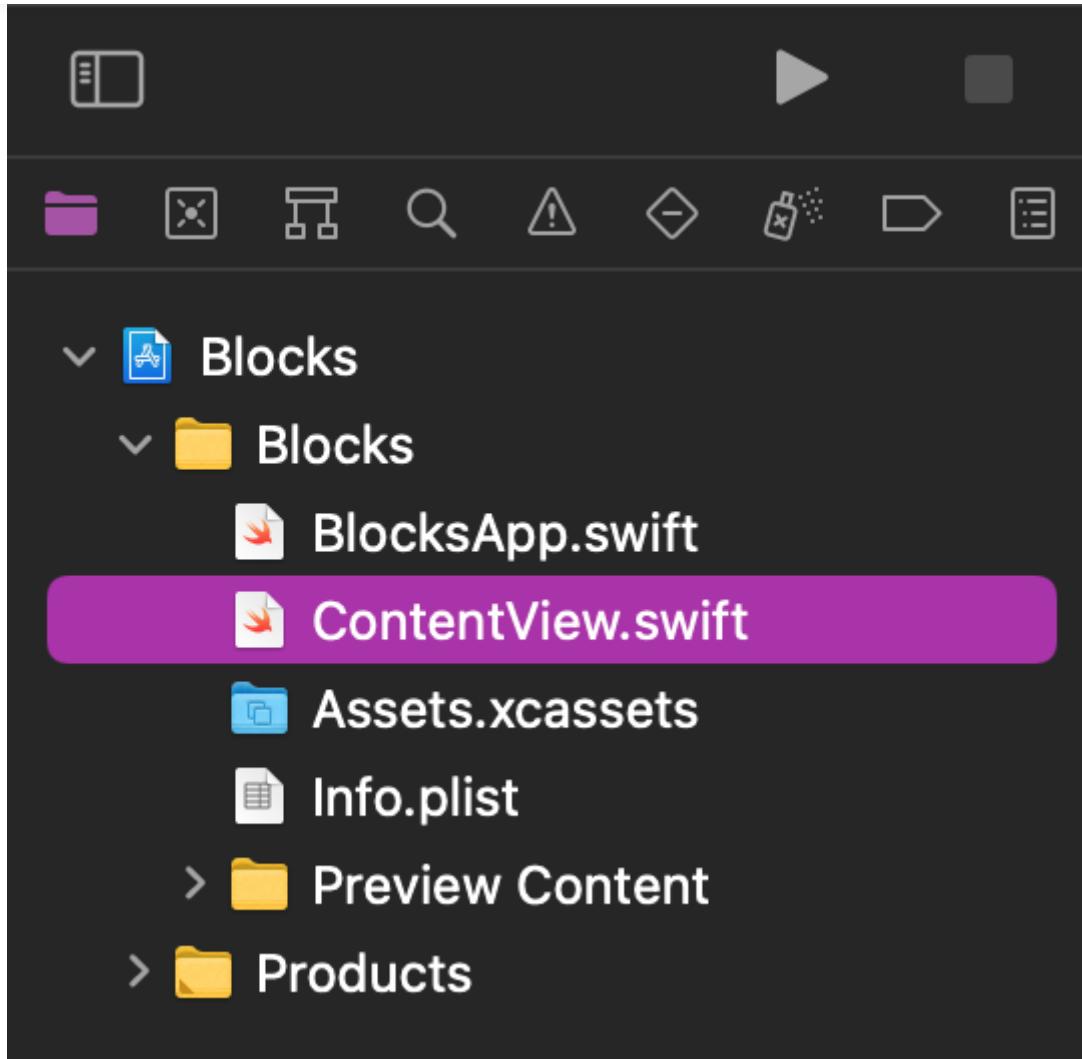


👉 Look at the Project Navigator (the left panel). The `ContentView.swift` file should be selected. If not, click **once** on it. **Don't double click**, or you'll open a new window (which you can easily close).

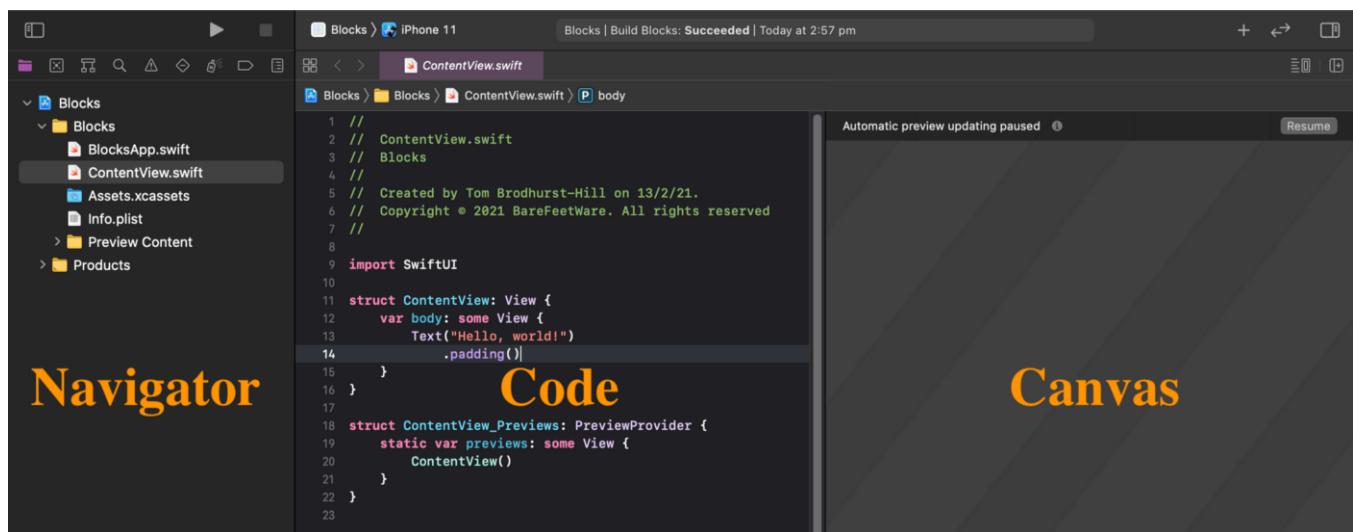




Open in app

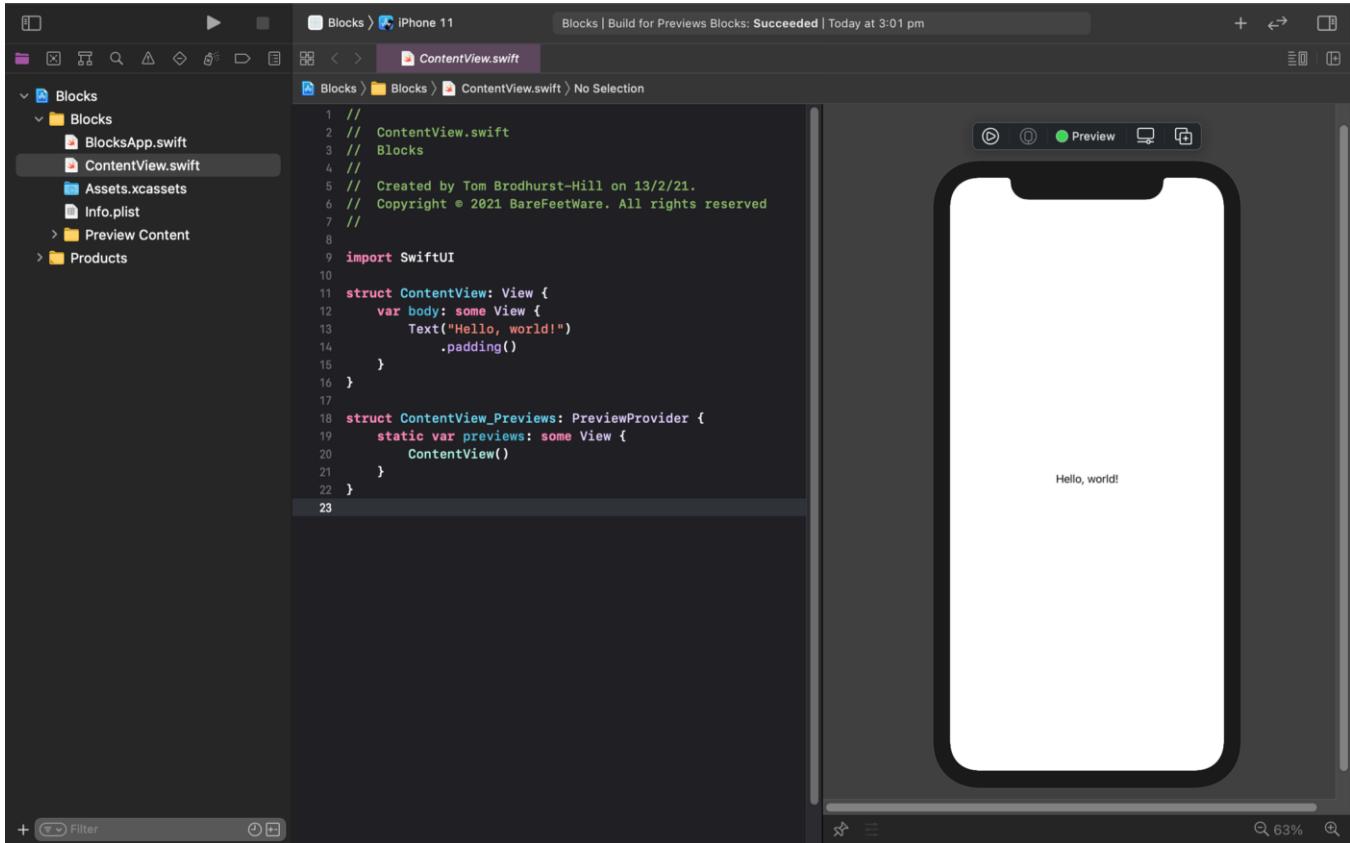


- ⌚ The middle panel contains the content of the item selected in the navigator. In this case, the `ContentView.swift` file's programming code.



[Open in app](#)

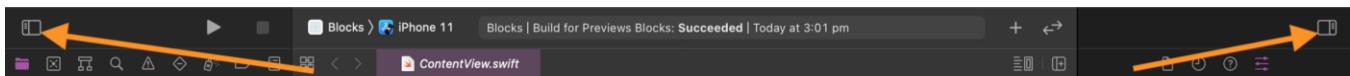
👉 Tap on the `Resume` button. After a few seconds, you should see a preview of the app:



👉 This is a preview of what the app will look like when it runs the `ContentView.swift` code on the left.

Especially on a small screen, like a laptop, you might need some extra room for the code and preview panes. Fortunately, you can hide and show the left and right panels using the `Show/Hide Navigators` and `Inspectors` buttons in the top left and top right of the window.

👉 Click on the buttons to familiarise yourself with how to hide and show the Navigator and Inspector panels.



👉 Scroll around the preview pane (vertically and horizontally) using your trackpad or mouse. You can pinch to zoom or click on the `-` and `+` button at the bottom of the



[Open in app](#)

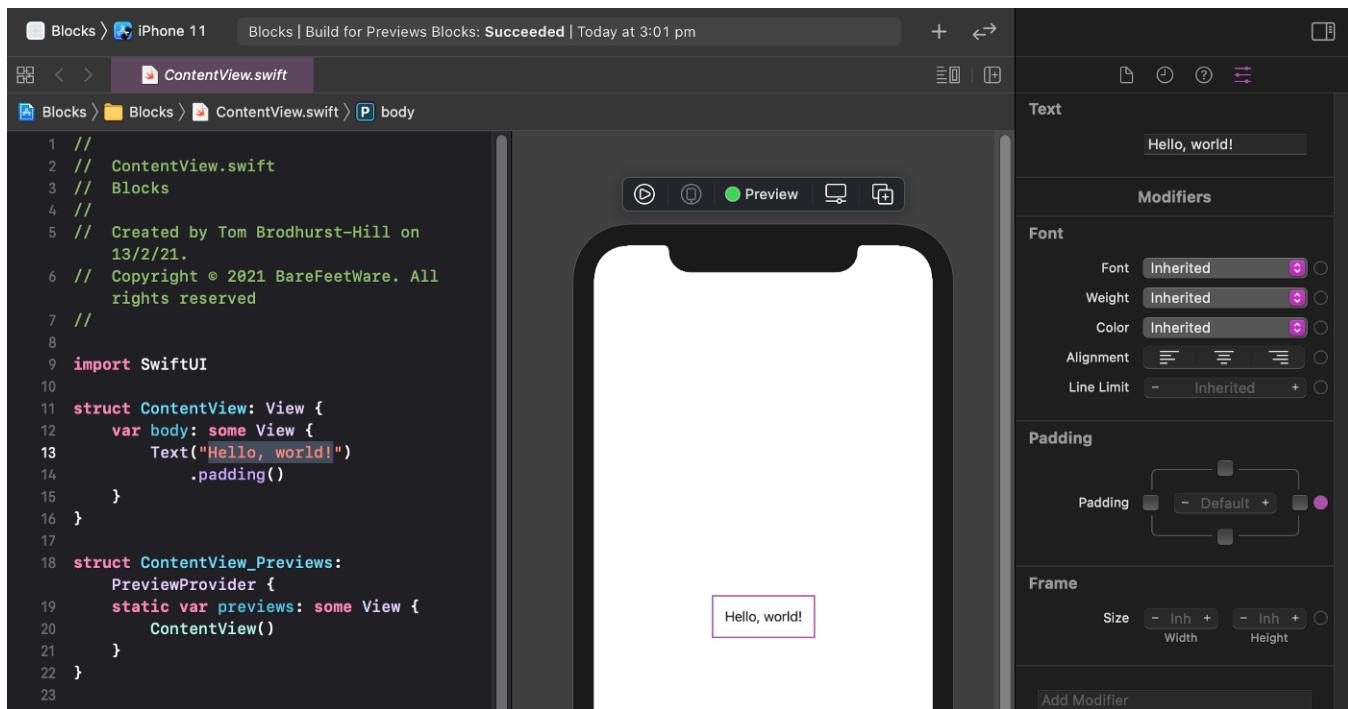
5. Edit Text

Now the fun begins!

The preview panel on the right shows what the code on the left instructs it to do.

If you're new to coding or prefer to design visually, here's the good news: In many cases, you can edit items in the preview directly, and Xcode will rewrite the code for you!

👉 Double click on the `Hello World` text in the preview, so that it shows a blue rectangle around it. Xcode will highlight the text in the code on the left and show it in the `Attributes Inspector` on the right.



👉 Now, type your own label text for this scene as `News`. Type it in the `Attributes Inspector` to be safe. Hit `Return` (on the keyboard) to make the change take effect. Alternatively, you can type it in the code, but then you must ensure that surrounding code, such as the quotation marks, remain intact, or it will cease to preview or run properly until you fix it. If you make a mistake, `Undo` is the easy way to backtrack.



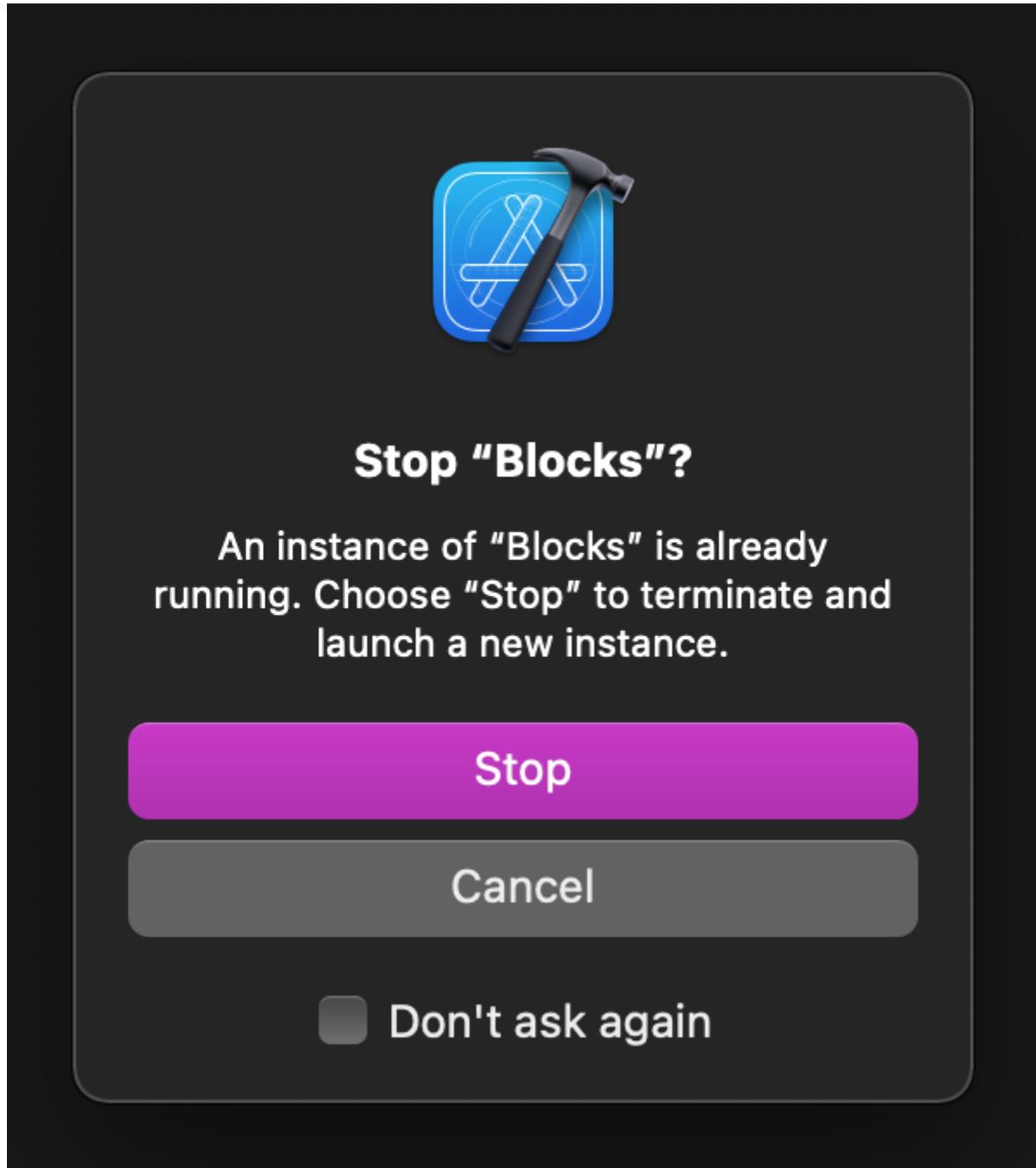
[Open in app](#)

The screenshot shows the Xcode interface with the following details:

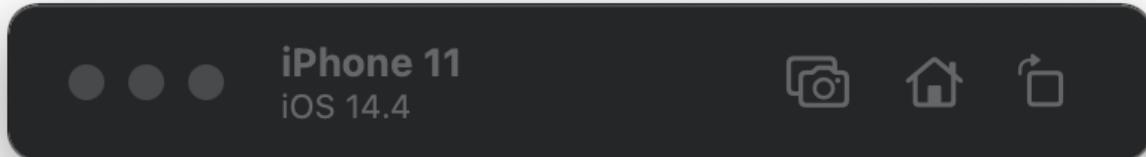
- Project Navigator:** Shows "Blocks" and "ContentView.swift".
- Editor:** Displays the code for `ContentView`. The line `Text("News")` is selected.
- Preview:** An iPhone 11 simulator is shown with the word "News" centered on the screen.
- Attributes Inspector:** On the right, it shows the following settings:
 - Text:** A purple box labeled "News".
 - Modifiers:** Font (Inherited), Weight (Inherited), Color (Inherited), Alignment (Inherited), Line Limit (Inherited).
 - Padding:** Padding (Default).
 - Frame:** Size (Width Inherited, Height Inherited).

👉 Run the app again by clicking the Build and Run button in the top left of the Xcode window. If you already have the app running, Xcode will ask if you're happy for it to quit the old one in order to run the new one.



[Open in app](#)

- 👉 Click `Stop`, so the new app can run.
- 👁️ The app should now show the altered `News` text.



[Open in app](#)

News





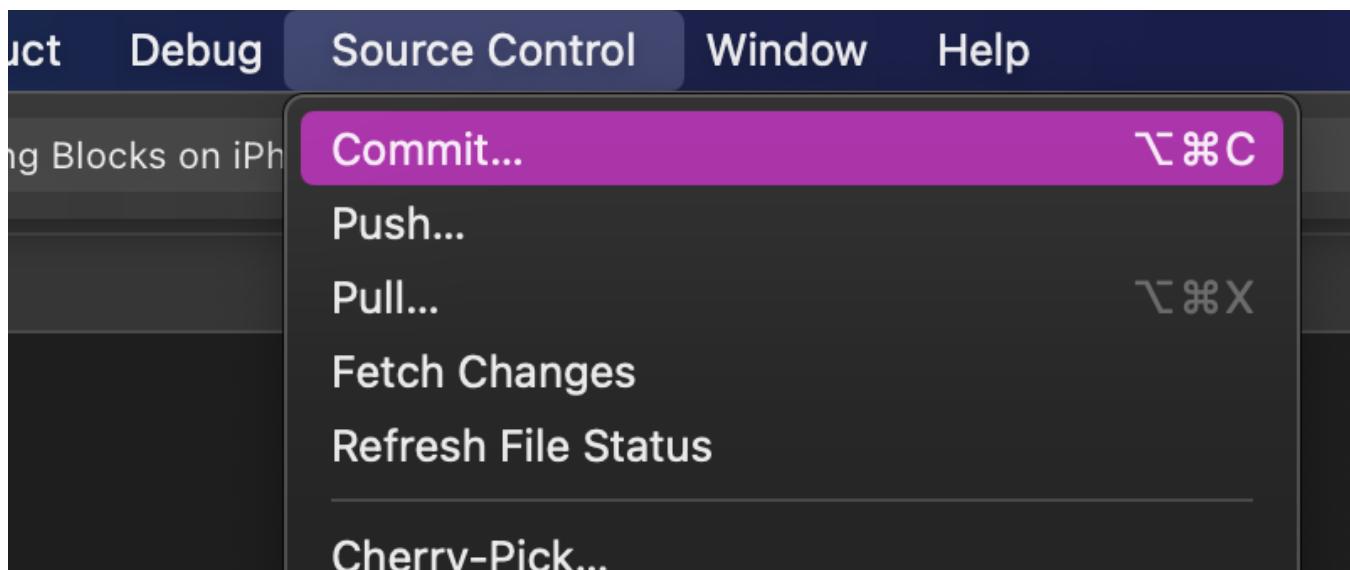
6. Commit the Changes

👉 Switch back to Xcode.

Xcode has already saved our changes, so we could quit Xcode, relaunch and pick up where we left off. However, along the way, we'll be making lots of changes and might like to experiment with different ideas or go back to a version of the project that we created a few days ago. To enable us to save particular versions or milestones, we can use the "Git" version control system. Git is widely used in all programming disciplines and document control in general. Xcode provides a few menu items to access some Git commands easily.

Let's commit the changes that we've done so far, with a description of what we changed. Later, we will be able to see this in the history of our app development and revert back if desired.

👉 In Xcode, in the Source Control menu (in the menu bar at the very top of the screen), select Commit .

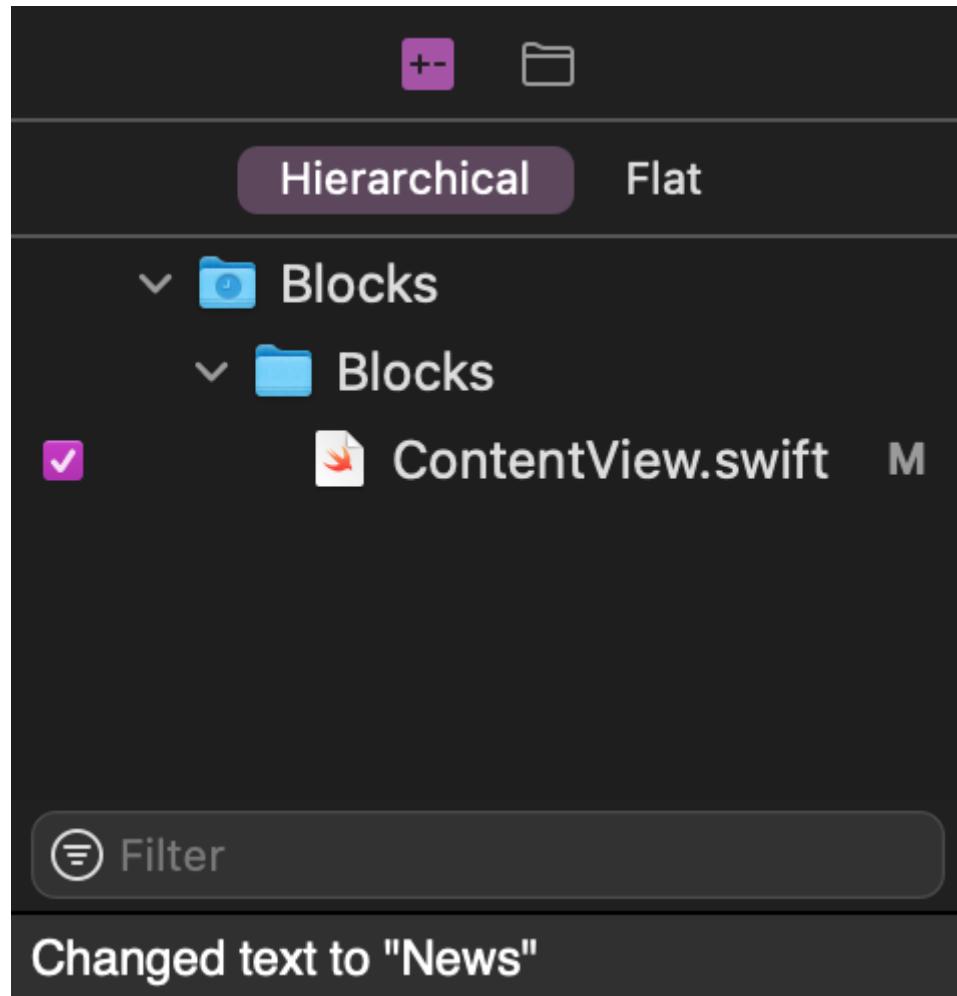


👁️ Xcode shows the files that we've changed. In this case, we've only changed the ContentView.swift file. Make sure that the checkbox next to the file is ticked.



[Open in app](#)

- 👉 At the bottom of the window, where the cursor is flashing, type in a description of your changes, such as: Changed text to "News".



- 👉 Click the `commit` button.

7. Recap Tutorial 1

That brings us to the end of Tutorial 1. So far, we have:

1. Installed the Apple iPhone developer toolset (Xcode).
2. Created a native iOS app.
3. Modified just a bit of it.
4. Run the app on an iPhone simulator.

[Open in app](#)

Now, move on to [Tutorial 2](#), where we'll start to customize the navigation flow of our app.

This series is released via [Next Level Swift](#). Subscribe to keep updated and never miss a new Tutorial of this series!

Next Level Swift

Next Level Swift

Next Level Swift aims at sharing knowledge and insights into better programming for iOS and is dedicated to help...

[medium.com](https://medium.com/@nextlevels)

We are always looking for talented and passionate Swift developers! Feel free to check out our writer's section and find out how you can share your knowledge with the Next Level Swift Community!

Sign up for Next Level Swift Newsletter

By Next Level Swift

Get all the latest articles, posts and news straight to your mailbox! [Take a look.](#)

[Get this newsletter](#)

Emails will be sent to research2learn@yahoo.co.uk.
[Not you?](#)



[Open in app](#)

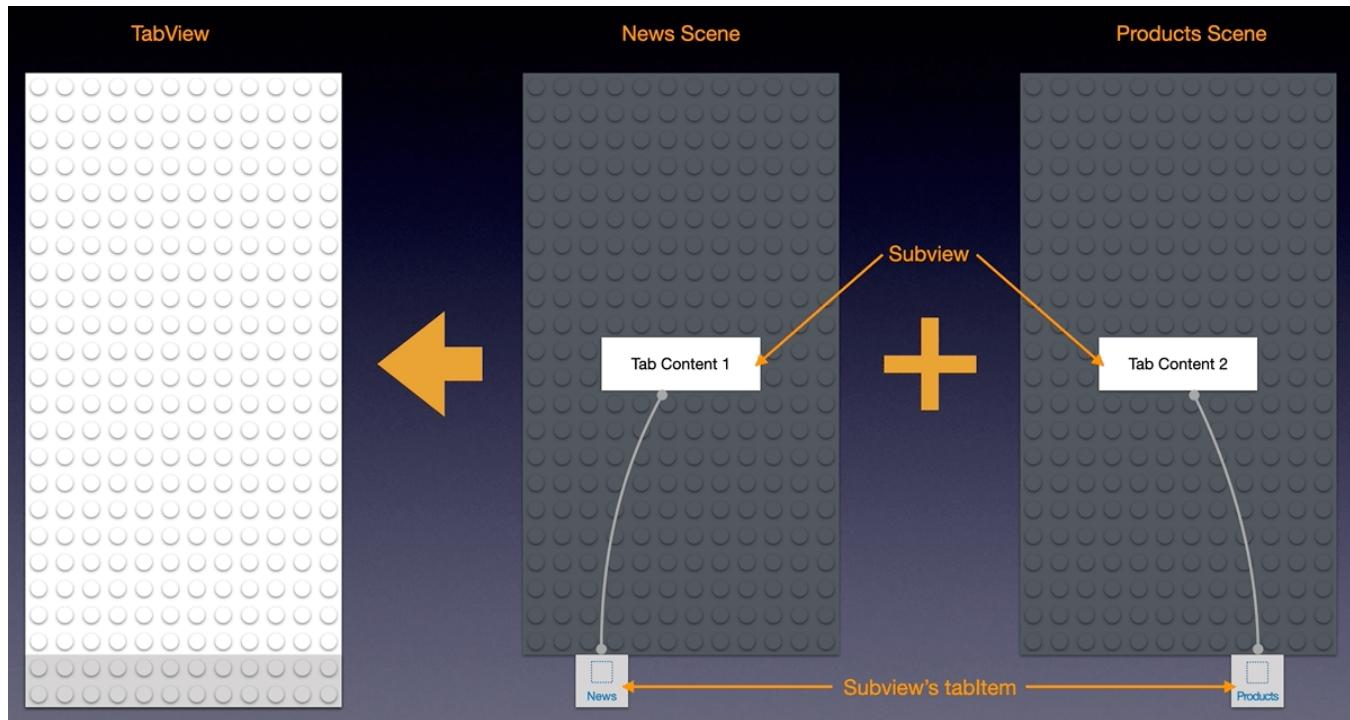
[Open in app](#)Published in Next Level Swift · [Following](#) ▾Tom Brodhurst-Hill · [Following](#)

Mar 20, 2021 · 7 min read ★

...

Scenes, Subviews, and Tab Views

Build an App Like Lego, with SwiftUI — Tutorial 2



1. Introduction

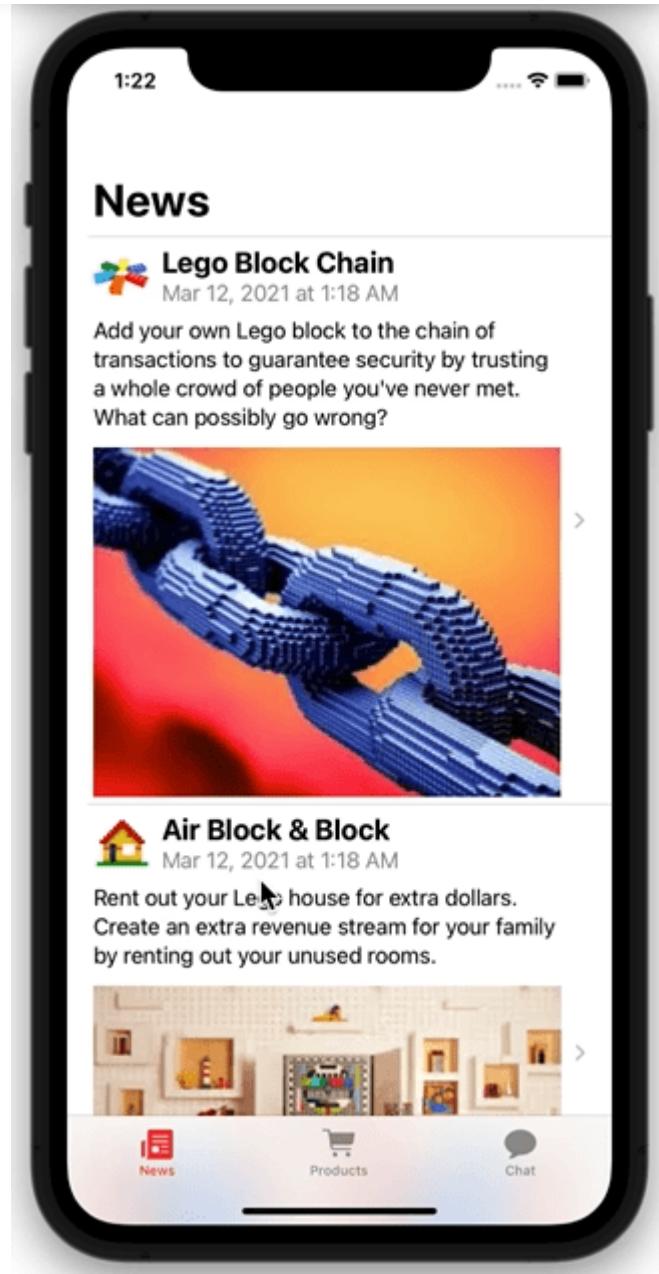
Navigating a mobile app is fundamentally different from using a web browser.

Customers love the simplicity and predictability of a mobile UI (user interface). They can move from one app to another while using the same navigation actions to switch between features, scroll through a list, drill down to something more specific, and so on.

In this Tutorial 2 of the series, we will learn how visual content is managed within other content, using “views”. We will introduce the “tab view”.

Recall that we’re aiming to build an app that looks something like this:



[Open in app](#)

In [Tutorial 1](#) we created a template app, with just one view that showed the text “News”. As shown above, we need to instead show a tab bar with three tab items (News, Products, and Chat), each of which has its own view content. We also need a navigation bar for each. Each scene needs a scrolling list.

Make sure that you have completed [Tutorial 1](#) in this series. We’ll pick up here where the previous tutorial left off. Or, you can [download](#) the prepared project, ready to start this tutorial.

2. Scene and View



[Open in app](#)

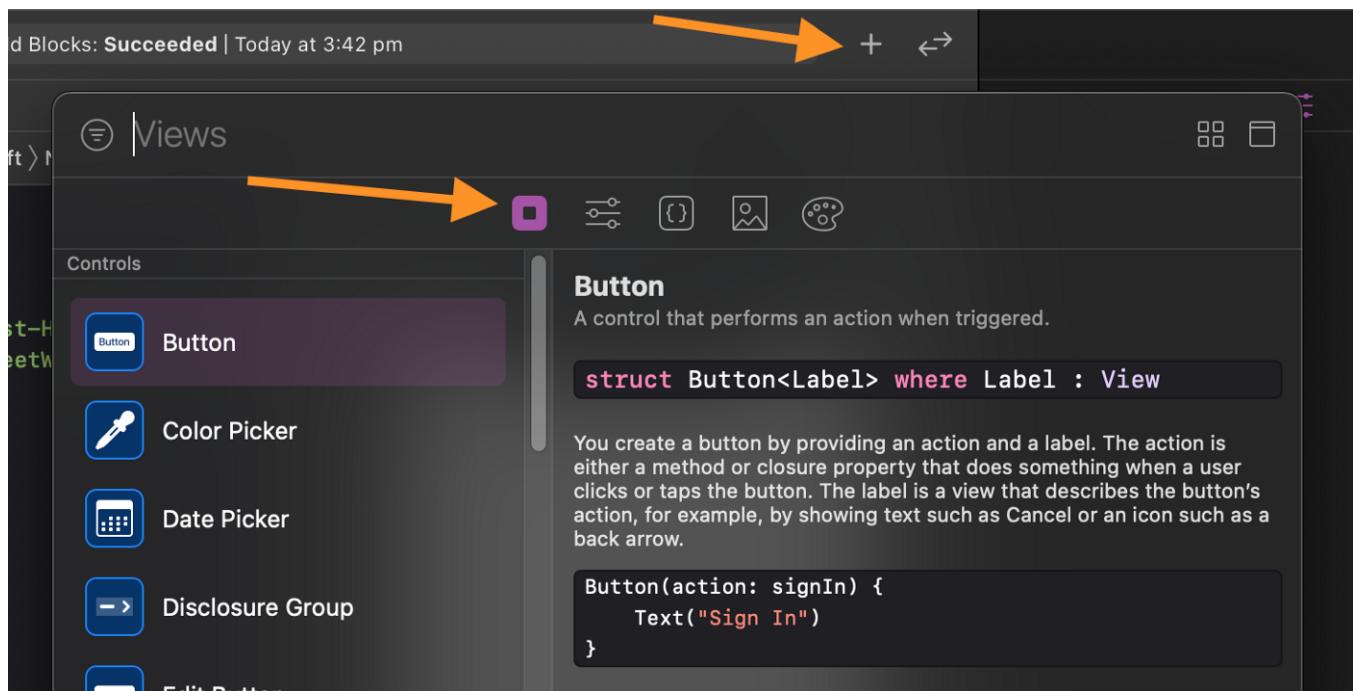
1. A **view** is a visual object, usually in a rectangular shape. It can be something small like a `Text` object, a `Switch` or `Slider`, or something large like a `ScrollView` or `TabView`.
2. A **scene** is a view that occupies the whole screen or a panel in the screen (such as in a split view).

Informally, you may also hear people refer to a scene as a “page”, “window”, or “screen”. Technically, however, the iPhone has just had one physical “screen” and each app usually exists inside just one “window”.

3. Views Library

- 👉 In the Project Navigator, click on the `ContentView.swift` file to select it, if it's not already selected.
- 👉 Near the top right of the Xcode window, click on the “Library” button which has a + icon.

The `Library` panel appears. Ensure that the first icon in it is selected to show you the `Views` library.



[Open in app](#)

👉 In the search field at the top, type `tab`. Xcode shows just the views related to “tab”:

The screenshot shows the Xcode interface with the 'Views' library open. At the top, there's a search bar containing the text 'tab'. Below the search bar, there are several icons representing different UI components: a square for View, a gear for Control, a curly brace for Collection, a grid for Image, and a circular arrow for Animation. On the left, there are two sections: 'Controls' and 'Other'. Under 'Controls', the 'List' and 'Tab View' items are listed; 'Tab View' is currently selected and has a purple background. Under 'Other', 'Text Field' and 'Capsule' are listed. To the right of the library, there's a detailed description of 'Tab View' and some sample code for creating it.

Tab View
A view that switches between multiple child views using interactive user interface elements.

```
struct TabView<SelectionValue, Content> where SelectionValue : Hashable, Content : View
```

To create a user interface with tabs, place views in a TabView and apply the `tabItem(_:)` modifier to the contents of each tab. The following creates a tab view with three tabs:

```
TabView {
    Text("The First Tab")
    .tabItem {
        Image(systemName: "1.square.fill")
        Text("First")
    }
}
```

👉 Drag the Tab View from the Views Library into the code, between the `var` body line and the `Text` line. When Xcode makes a new empty line between, then release the dragged object.

The screenshot shows the Xcode code editor with the file 'ContentView.swift' open. The code contains the following lines:

```
1 // ContentView.swift
2 // Blocks
3 // ContentView.swift
4 // Blocks
```

Below the code editor is the Xcode Views Library. A cursor is positioned above the 'List' component in the 'Controls' section. The 'Tab View' component is also visible in the library.

List
A container that presents rows of data arranged in a single column, optionally providing the ability to select one or more members.

```
struct List<SelectionValue, Content> where SelectionValue : Hashable, Content : View
```

In its simplest form, a List creates its contents statically, as shown in the following example:





Open in app

```

 9 import SwiftUI
10
11 struct ContentView: View {
12     var body: some View {
13         TabView(selection: Selection) {
14             Text("Tab Content 1").tabItem { Text("Tab Label 1") }.tag(1)
15             Text("Tab Content 2").tabItem { Text("Tab Label 2") }.tag(2)
16         }
17         Text("News")
18             .padding()
19     }
20 }
```

👉 If your Mac's screen is small, the code might soft wrap over more lines, making it harder to read. In that case, widen the code panel by dragging its vertical dividers, so the code looks similar to the above screenshot.

Dragging code from the Library has the same effect as just manually typing it. But dragging from the Library is obviously faster and more accurate.

👉 We don't want the old `Text("News")` anymore, so delete it and the `.padding()`.

```

11 struct ContentView: View {
12     var body: some View {
13         TabView(selection: Selection) {
14             Text("Tab Content 1").tabItem { Text("Tab Label 1") }.tag(1)
15             Text("Tab Content 2").tabItem { Text("Tab Label 2") }.tag(2)
16         }
17         Text("News")
18             .padding() Delete
19     }
20 }
```

👁️ After deleting, check that your code looks like this:

```

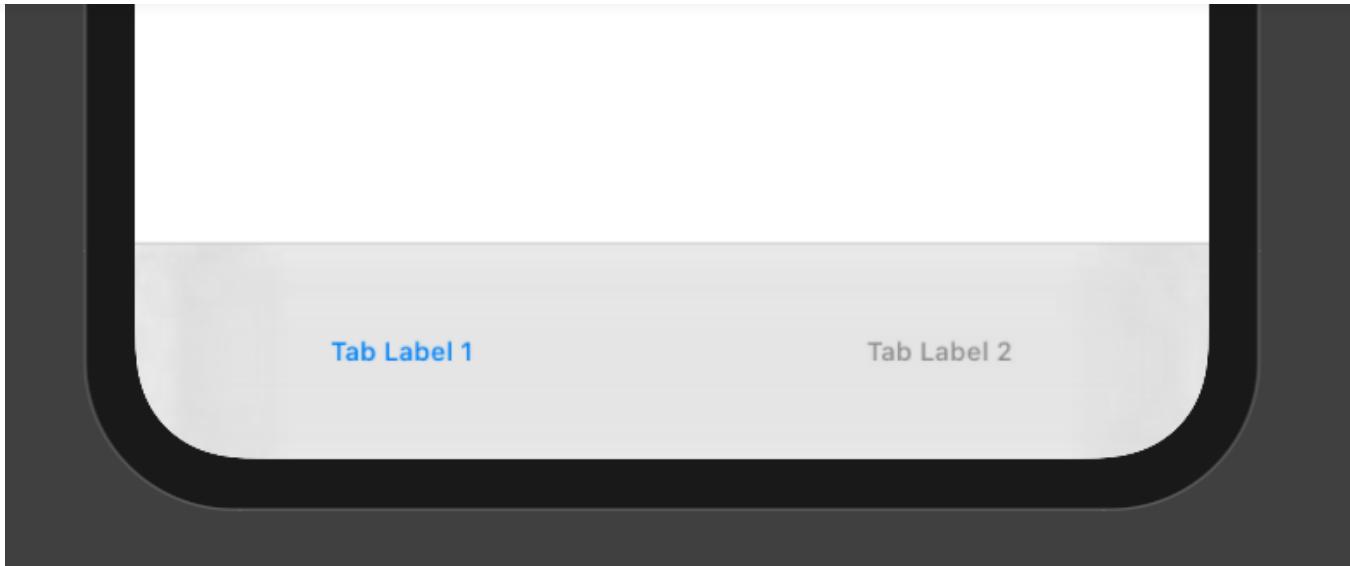
11 struct ContentView: View {
12     var body: some View {
13         TabView(selection: Selection) {
14             Text("Tab Content 1").tabItem { Text("Tab Label 1") }.tag(1)
15             Text("Tab Content 2").tabItem { Text("Tab Label 2") }.tag(2)
16         }
17     }
18 }
```



[Open in app](#)

Tab Content 1



[Open in app](#)

- 👉 In the code, change the quoted text of the two `tabItem`s to `News` and `Products` respectively. You might need to double-click in the code to edit inside the quotes.

```

8
9 import SwiftUI
10
11 struct ContentView: View {
12     var body: some View {
13         TabView(selection: $selection) {
14             Text("Tab Content 1").tabItem { Text("News") }.tag(1)
15             Text("Tab Content 2").tabItem { Text("Products") }.tag(2)
16         }
17     }
18 }
19
20 struct ContentView_Previews: PreviewProvider {
21     static var previews: some View {
22         ContentView()
23     }
24 }
25

```

Tab Content 1

News Products

Text 0x0 75%

- ⌚ Confirm that the preview shows the two tab items with the new text. If necessary, click on the `Resume` button to refresh it. If it shows any error, check that the code is correct.

3. Superview and Subviews

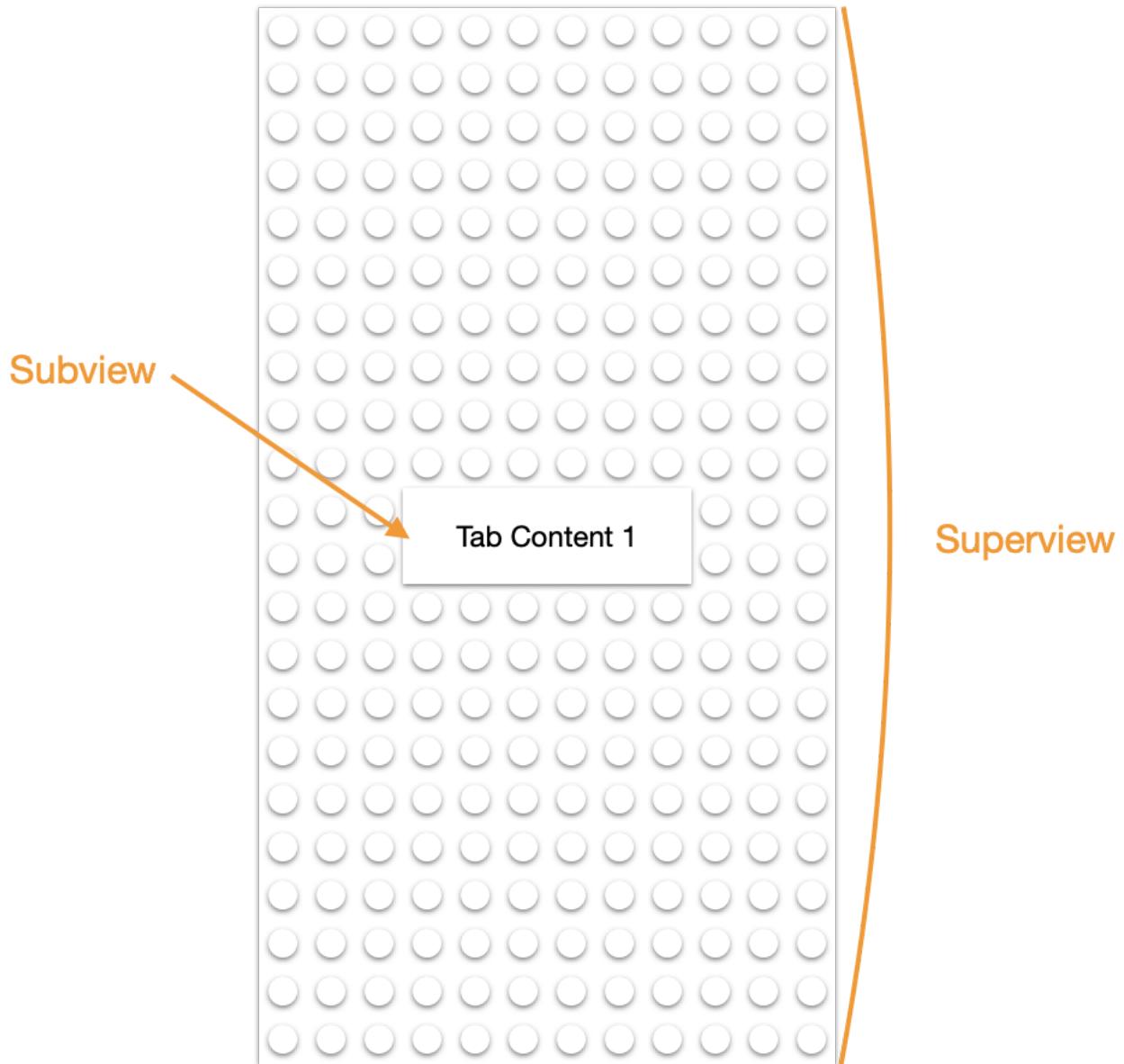
A view has a rectangular shape. A view may contain other views. The containing view is called the “superview” and the views within it are called its “subviews”.



[Open in app](#)

You can think of a superview as a large rectangular Lego block containing smaller Lego blocks as subviews. The subview blocks can only be placed within the rectangular bounds of the superview.

In the News scene, the `Text` subview is like a Lego block placed on top of the large rectangular scene's view.

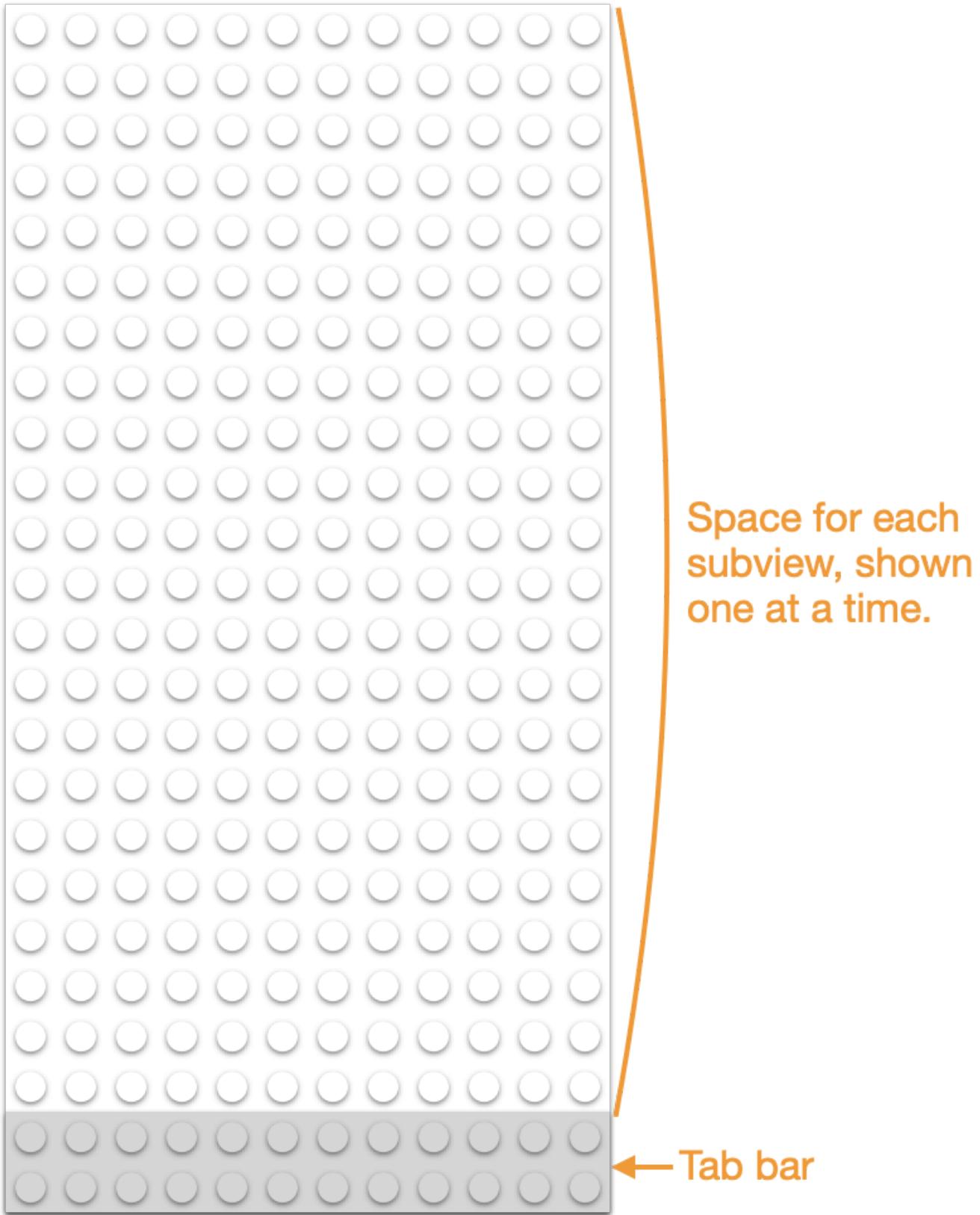


Some types of views don't have subviews. For instance, you don't add a subview in a `Text` view. In the Lego analogy, the `Text` views don't have connecting studs on top of them, as you can see in the above diagram since they don't accept subviews.



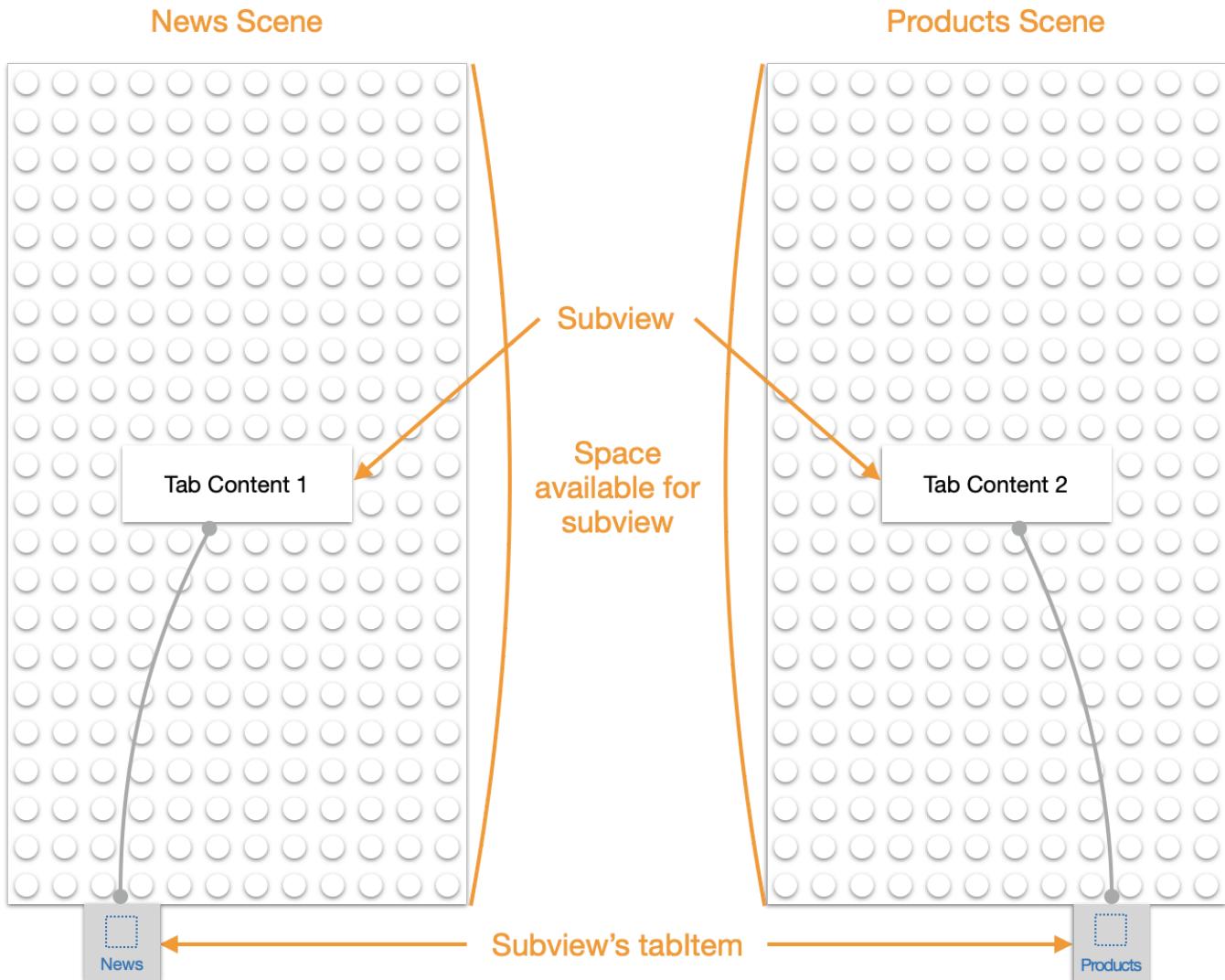
[Open in app](#)

We can think of the `TabView` as a Lego base plate that provides a space for each subview and a tab bar at the bottom, which contains the `tabItem` label of each subview.



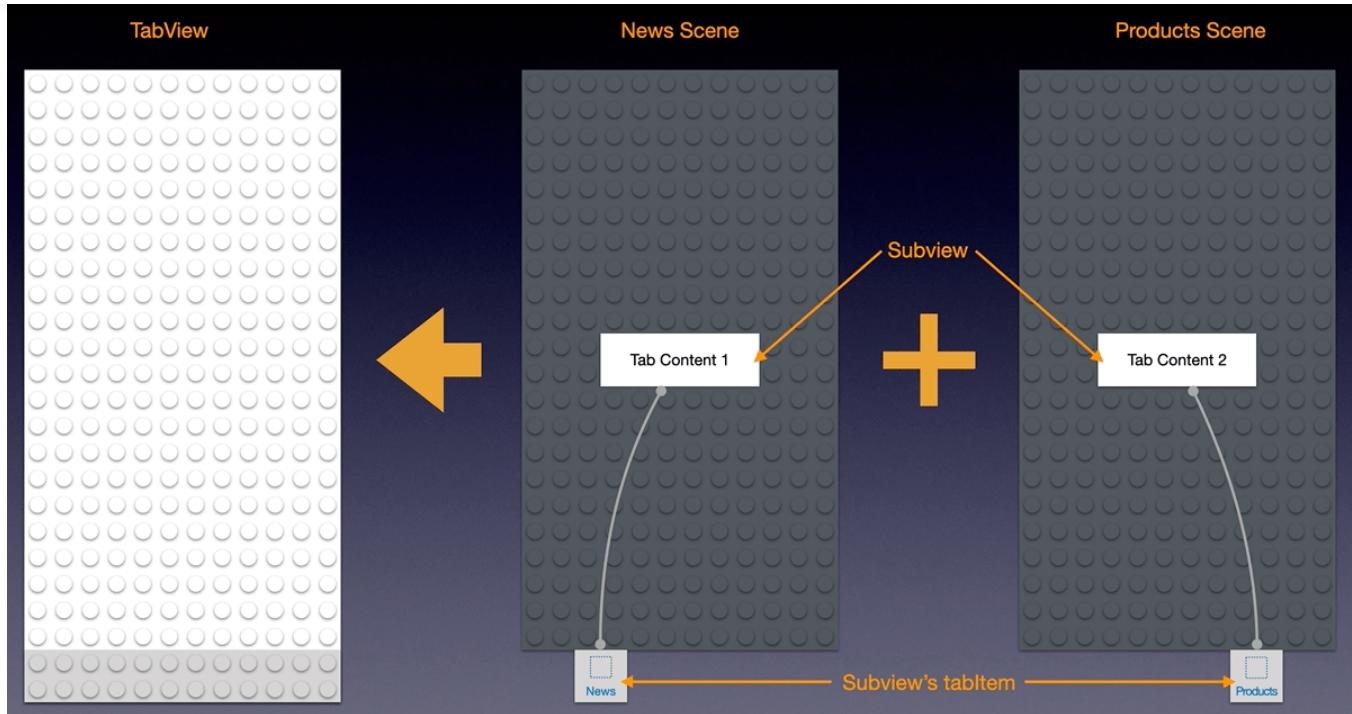
[Open in app](#)

We can picture the News scene and Product scene as these Lego constructions, each with a view and a `tabItem`:



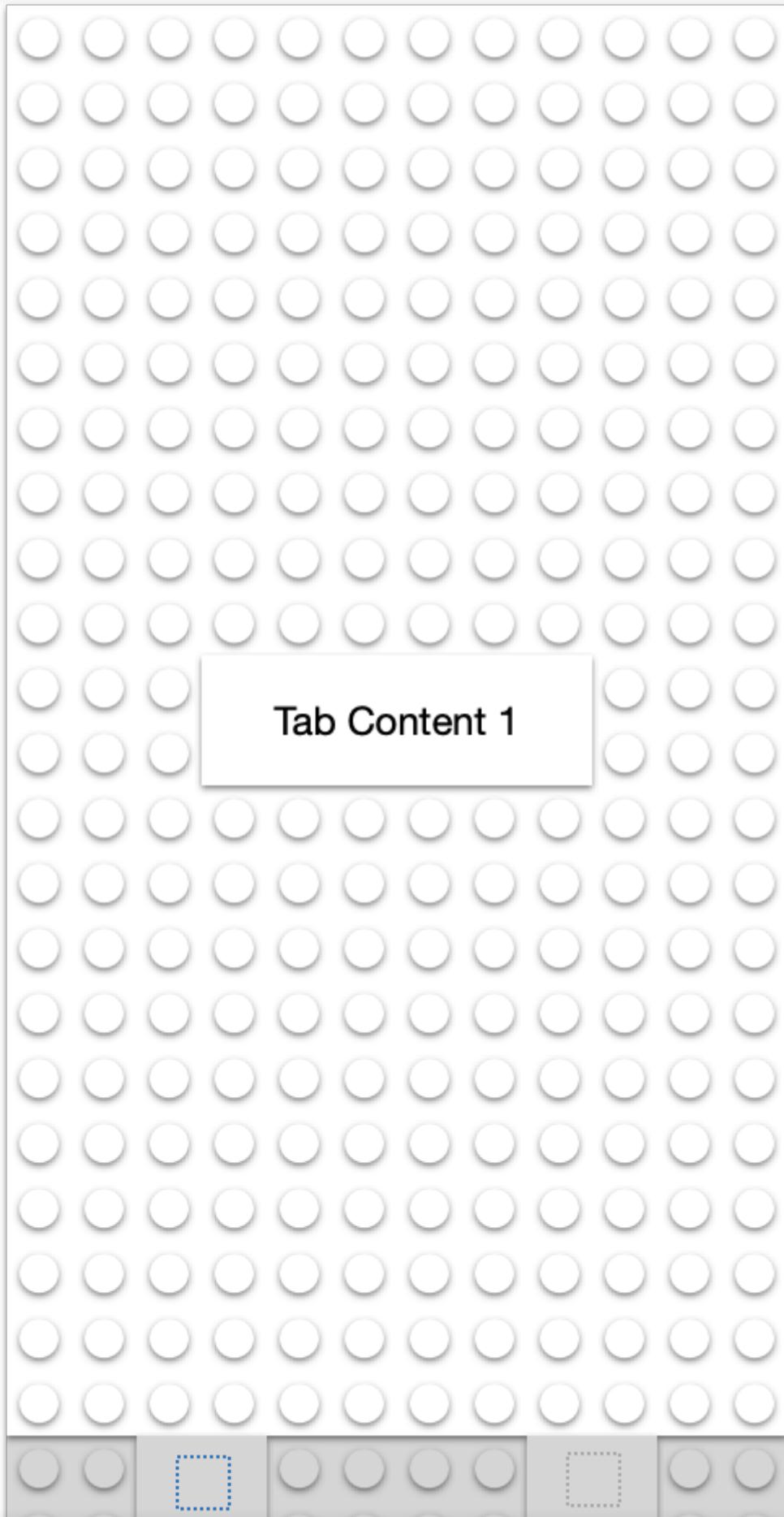
The `TabView` loads each subview when needed but shows the `tabItem` of all of the subviews at once.



[Open in app](#)

For example, when the News tab item is selected, we can picture the composed tab view like this:



[Open in app](#)

[Open in app](#)

In summary, there are three parts to the tab view structure:

1. The `TabView`. It is responsible for showing the current scene's view inside it and the tab bar (at the bottom).
2. The **tab bar** sits at the bottom of the screen. It contains all of the tab items and usually changes the color of the currently selected one.
3. A `tabItem` (containing text and/or icon) for each scene.

You can see that the News scene contains a tab item with the text `News`. The Products scene contains a tab item with the text `Products`. The `TabView` shows its tab bar containing all of the tab items.

5. Add a New Scene

Let's add a new scene for "Chat". This will provide a third view in the `TabView` with its own `tabItem`.

👉 Select the second `tabItem` line of code (such as by triple-clicking in that line). Copy it and paste it into a new line below it. In the new line, change both of the `2`s to `3` and change `Products` to `Chat`.

The screenshot shows the Xcode editor with the following code:

```

8
9 import SwiftUI
10
11 struct ContentView: View {
12     var body: some View {
13         TabView(selection: Selection) {
14             Text("Tab Content 1").tabItem { Text("News") }.tag(1)
15             Text("Tab Content 2").tabItem { Text("Products") }.tag(2)
16             Text("Tab Content 3").tabItem { Text("Chat") }.tag(3)
17         }
18     }
19 }
20
21 struct ContentView_Previews: PreviewProvider {
22     static var previews: some View {
23         ContentView()
24     }
25 }
26

```

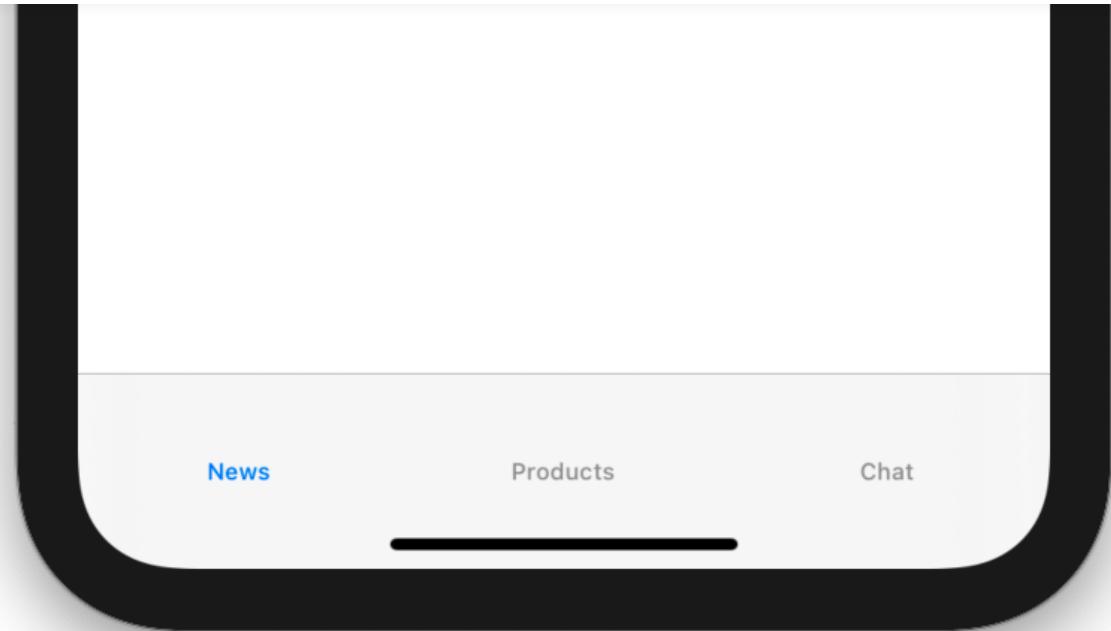
To the right, a preview window shows a tab bar with three tabs: "News", "Products", and "Chat". The "News" tab is selected, and the text "Tab Content 1" is displayed above the tabs. The status bar at the bottom indicates "Text 0x0", "75%", and a magnifying glass icon.



[Open in app](#)

💡 Verify that the running app looks similar to the preview, showing three tab items. Use the mouse to tap/click on the Products and Chat tab items to switch the selected tab item and content in the tab view.



[Open in app](#)

6. Commit Changes

Switch back to Xcode.

As we did in [Tutorial 1](#):

1. Choose Commit from the Source Control menu.
2. Enter a description such as: Added TabView with three tabItems
3. Click on the Commit button.

7. Recap

In this Tutorial 2 we have:

1. Discussed subviews in a superview.
2. Added a new “Chat” scene to the tab view.

In [Tutorial 3](#) we will continue modifying the scene flow using navigation views.

If you have any questions or comments, please add a response below.



[Open in app](#)

Next Level Swift

Next Level Swift

Next Level Swift aims at sharing knowledge and insights into better programming for iOS and is dedicated to help...

[medium.com](https://medium.com/@nextlevelsdk)

We are always looking for talented and passionate Swift developers! Feel free to check out our writer's section and find out how you can share your knowledge with the Next Level Swift Community!

Sign up for Next Level Swift Newsletter

By Next Level Swift

Get all the latest articles, posts and news straight to your mailbox! [Take a look.](#)

[Get this newsletter](#)

Emails will be sent to research2learn@yahoo.co.uk.

[Not you?](#)



[Open in app](#)Published in Next Level Swift · [Following](#) ▾Tom Brodhurst-Hill · [Following](#)

Mar 25, 2021 · 7 min read ★

...

Navigation Views, Inside a Tab View

Build an App Like Lego, with SwiftUI — Tutorial 3



Photo by [Michael Bader](#) on [Unsplash](#)

1. Introduction

At the end of [Tutorial 2](#) the News, Products and Chat scenes appeared as tabs in a tab view. In this Tutorial 3, we will embed each of those scenes in their own navigation view, and add navigation titles.



[Open in app](#)

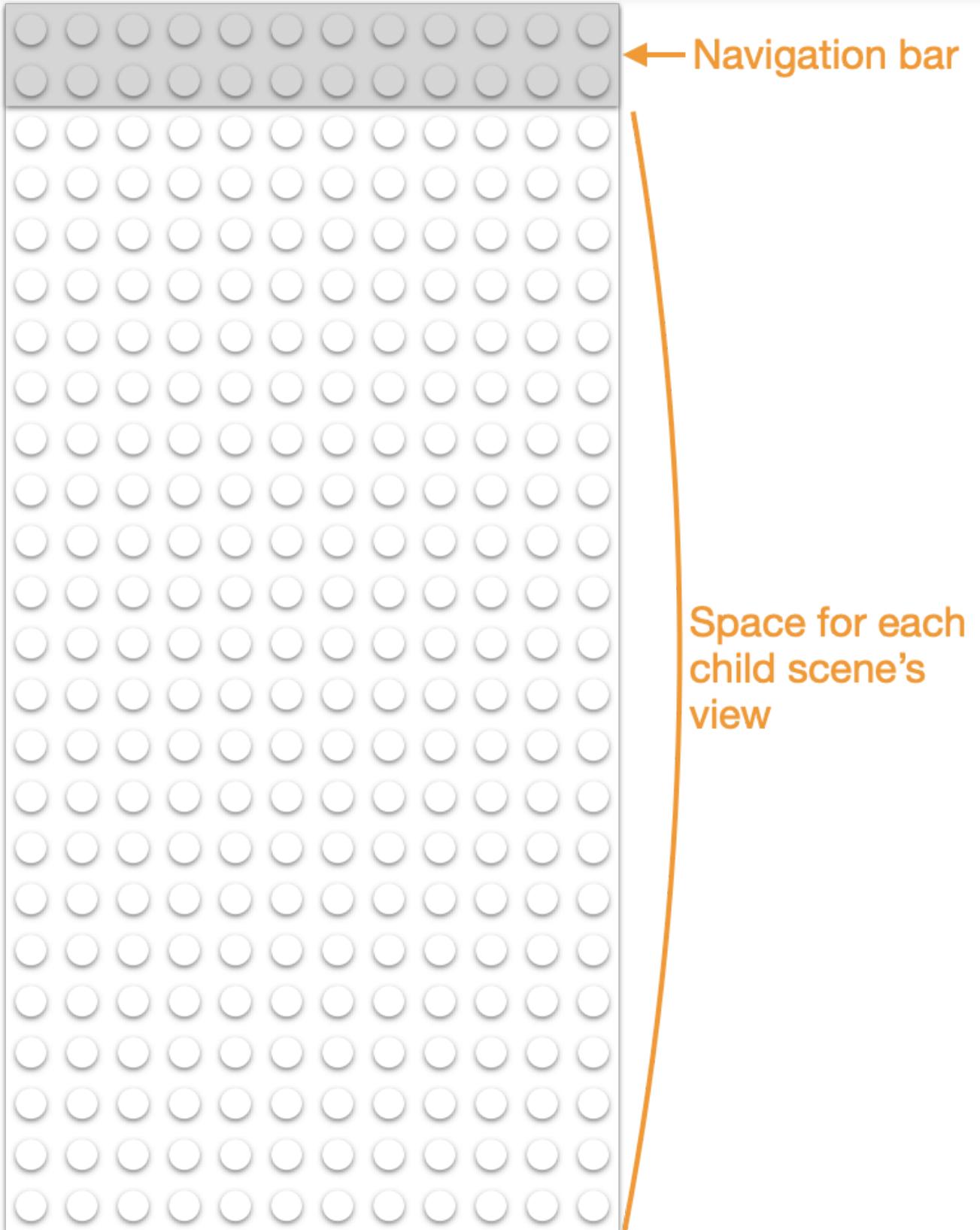
Make sure that you have completed the [previous tutorials](#) in this series. We'll pick up here where the last tutorial left off. Or, you can [download](#) the prepared project, ready to start this tutorial.

2. Navigation View

A navigation view presents a series of scenes, where tapping some button, cell, or link in the first scene navigates to the corresponding second scene, and so on. The second scene usually has a back button that returns to the first scene.

A navigation view is similar to the tab view that we discussed in [Tutorial 2](#). It has a large space in which it shows the current child scene and a navigation bar (at the top) that shows the title of each child scene. Imagined as a Lego baseplate, it would look like this:



[Open in app](#)

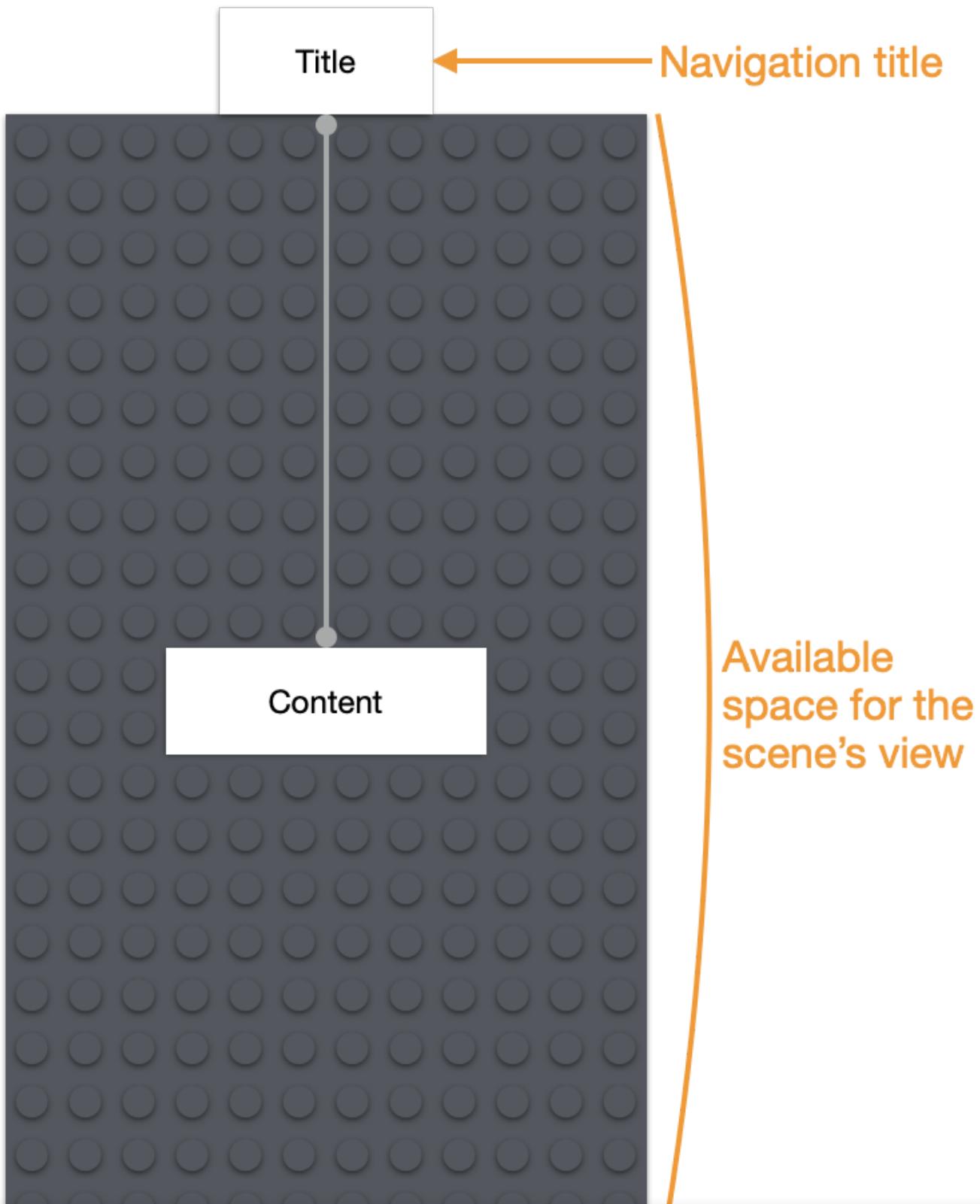
Like the tab view, the navigation view shows one child scene at a time. Each child scene can provide its own “navigation title” to appear in the navigation bar. Unlike the tab bar, the navigation bar only shows one navigation title at a time — the one belonging to



[Open in app](#)

on the left and right, such as buttons, which we'll discuss later. For now, we'll just deal with the navigation title.

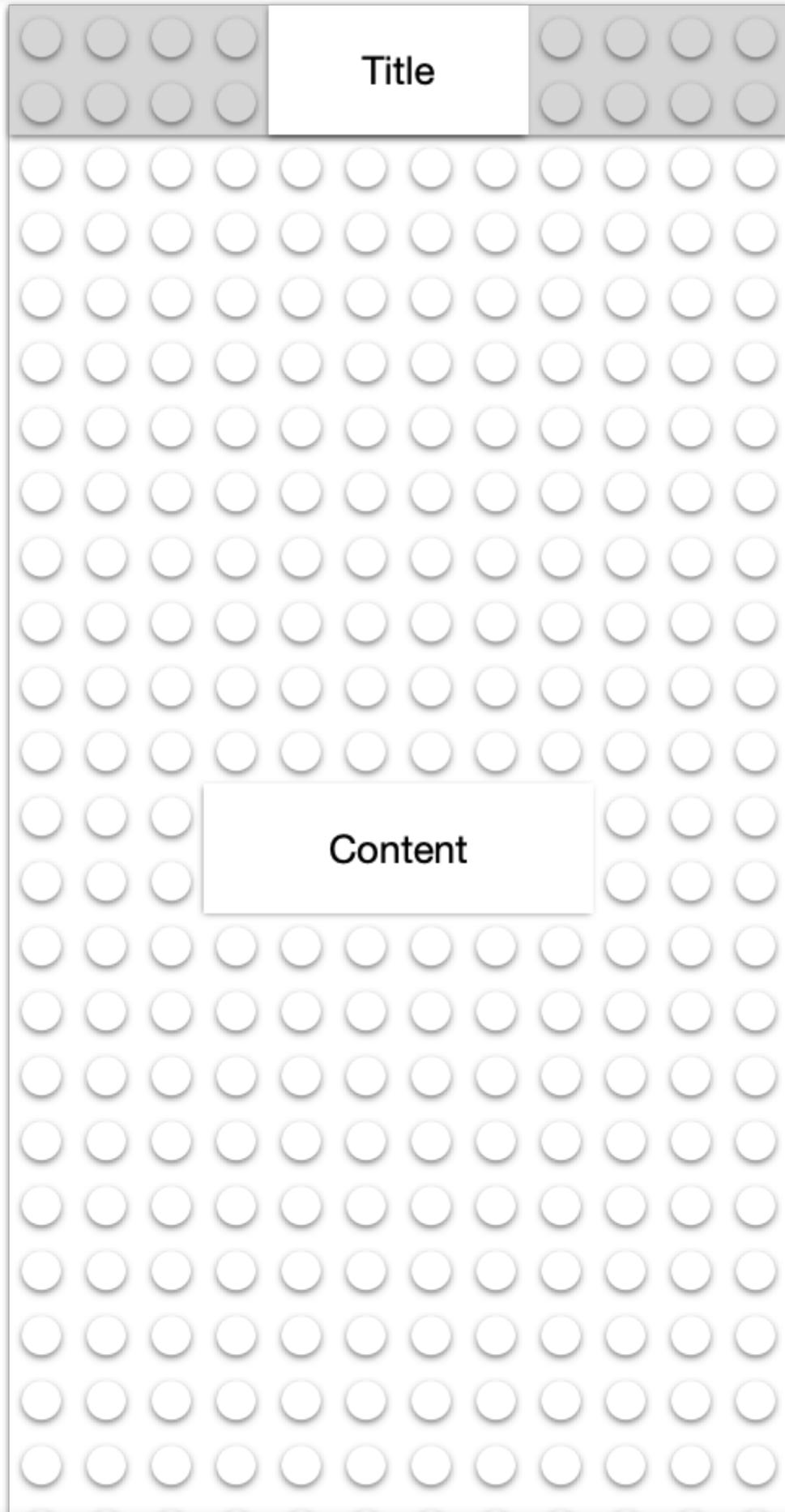
We can imagine a scene's view and navigation title as Lego blocks:



[Open in app](#)

When we put the child scene in the navigation view, it is like adding the scene's Lego blocks (the content and title) onto the navigation view base plate. The navigation title appears on the navigation bar and the scene's content occupies as much of the rest of the space as it needs.



[Open in app](#)

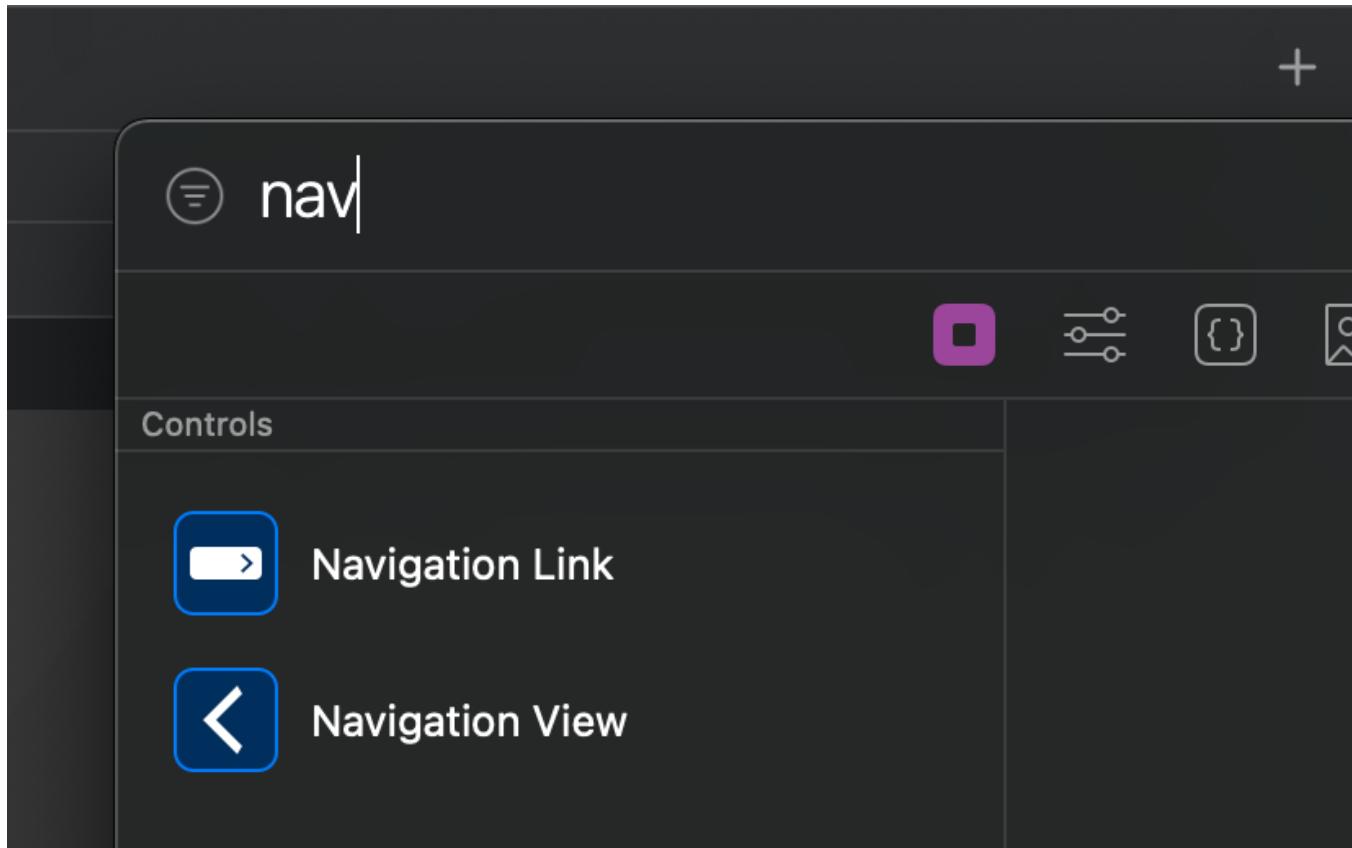
[Open in app](#)

1. The **navigation view**. It is responsible for showing the navigation bar (at the top) and the current child scene inside it.
2. The **navigation bar**, which sits at the top of the screen. It contains the navigation title of the currently displayed scene within it.
3. A **navigation title** for each scene, which appears inside the navigation bar when that scene is active.

3. Add a Navigation View

Our News scene will eventually contain a list of news article summaries, where the user can tap a summary to navigate to a detailed scene for that article. For that, we will need to embed the News scene in a navigation view.

👉 Click on the + icon near the top right of the Xcode window, to show the Library again. Ensure that `Views` is selected and type `nav` to filter the list.



👉 Drag the `Navigation View` from the Library to be inserted just after the `TabView` line



[Open in app](#)

```

1 // ContentView.swift
2 // Blocks
3 //
4 //
5 // Created by Tom Brodhurst-Hill on 4/3/21.
6 // Copyright © 2021 BareFetWare. All rights reserved.
7 //
8
9 import SwiftUI
10
11 struct ContentView: View {
12     var body: some View {
13         TabView(selection: $selection) {
14             NavigationView {
15                 NavigationLink(destination: Destination) { Content }
16             }
17             Text("Tab Content 1").tabItem { Text("News") }.tag(1)
18             Text("Tab Content 2").tabItem { Text("Products") }.tag(2)
19             Text("Tab Content 3").tabItem { Text("Chat") }.tag(3)
20         }
21     }
22 }
23
24 struct ContentView_Previews: PreviewProvider {
25     static var previews: some View {
26         ContentView()
27     }
28 }
```



💡 Note in the preview that there are now four tab items, with the first one before News having no label. The new NavigationView is currently the first subview of the TabView, so it has become the first tab item.

We need to move the first Text item into the NavigationView, replacing the default NavigationLink that came with the template code.

👉 Select the Text("Tab Content 1") code (not the .tabItem after it).

👉 Cut it (such as by selecting Cut from the Edit menu).

👉 Select the whole NavigationLink line of code. Paste over it.

👉 Edit if necessary so your code looks like this:

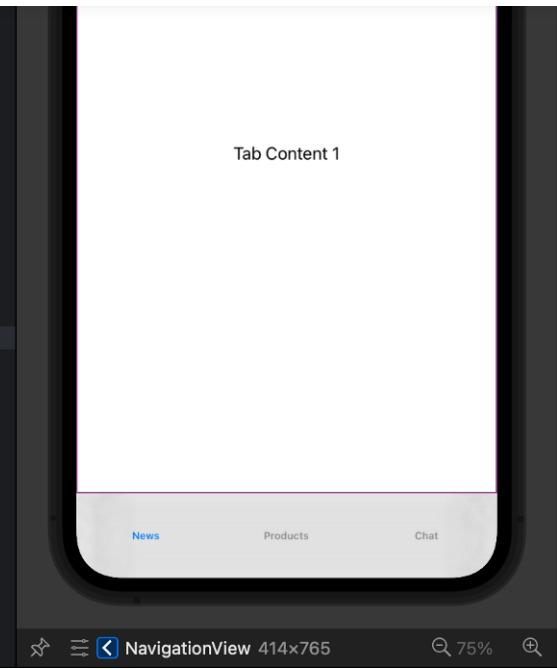



[Open in app](#)

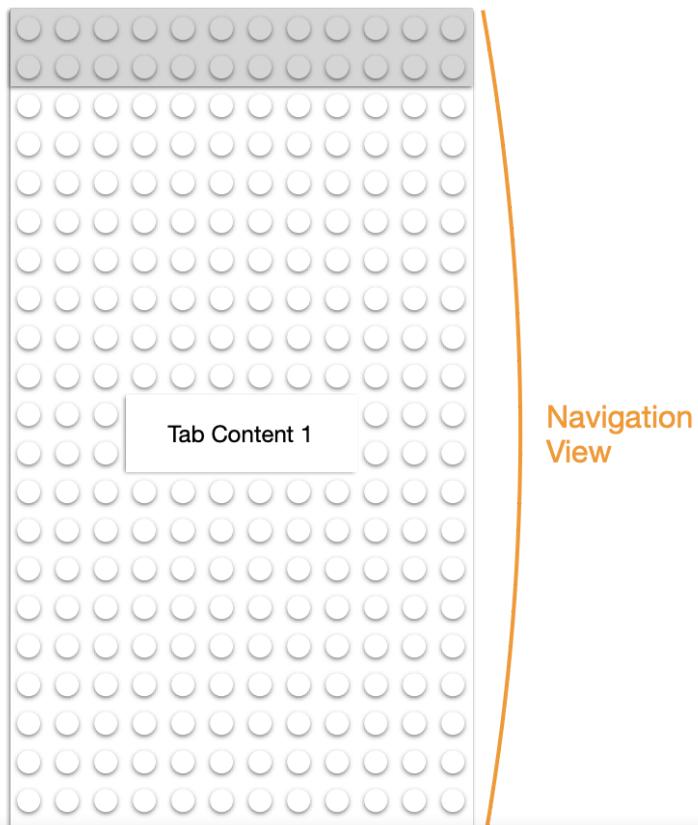
```

1 // ContentView.swift
2 // Blocks
3 //
4 //
5 // Created by Tom Brodhurst-Hill on 4/3/21.
6 // Copyright © 2021 BareFeetWare. All rights reserved.
7 //
8
9 import SwiftUI
10
11 struct ContentView: View {
12     var body: some View {
13         TabView(selection: Selection) {
14             NavigationView {
15                 Text("Tab Content 1")
16             }
17             .tabItem { Text("News") }.tag(1)
18             Text("Tab Content 2").tabItem { Text("Products") }.tag(2)
19             Text("Tab Content 3").tabItem { Text("Chat") }.tag(3)
20         }
21     }
22 }
23
24 struct ContentView_Previews: PreviewProvider {
25     static var previews: some View {
26         ContentView()
27     }
28 }
29

```



- ⌚ Confirm that the preview now shows the original three tab items again. If necessary, refresh the preview by clicking the Resume button.
- ⌚ Note that the `.tabItem("News")` is now attached to the `NavigationView`, instead of directly to the `Text("Tab Content 1")`. In Lego, this change looks like this:



[Open in app](#)

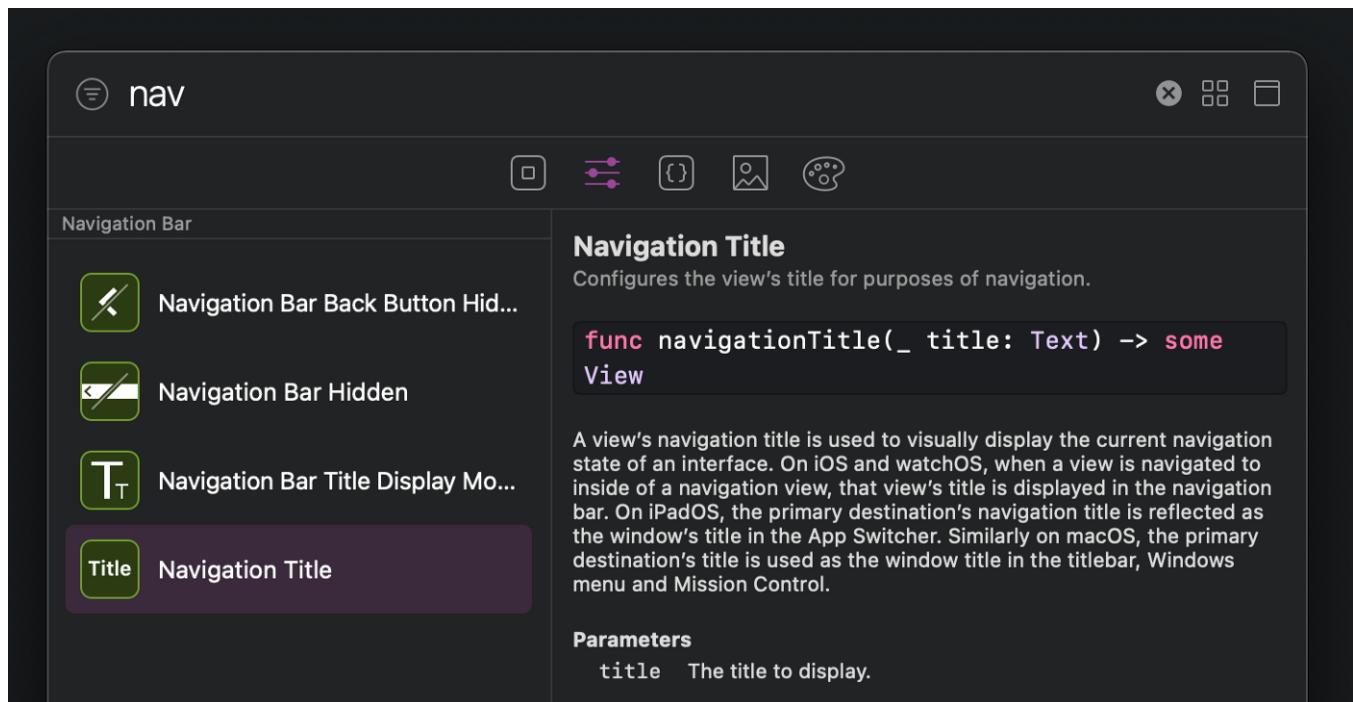
Now we again have three scenes in the `TabView`. The first scene is a `NavigationView`, which itself contains the `Text("Tab Content 1")`.

The preview doesn't yet look any different to how it was before we added the `NavigationView`. That's because there's not yet any navigation title or navigation links in the scene. Let's add a `navigationTitle` for the News scene.

In the same way that the `tabItem`s must be inside the `TabView` that displays them, the `navigationTitle` must be inside the `NavigationView` that displays it.

`tabItem` and `navigationTitle` are “modifiers” of views. They modify an existing view to give it additional features.

👉 Show the Library and select the `Modifiers` tab (the second icon that looks like slider controls). Type `nav` into the search bar and select the `Navigation Title` item in the list.

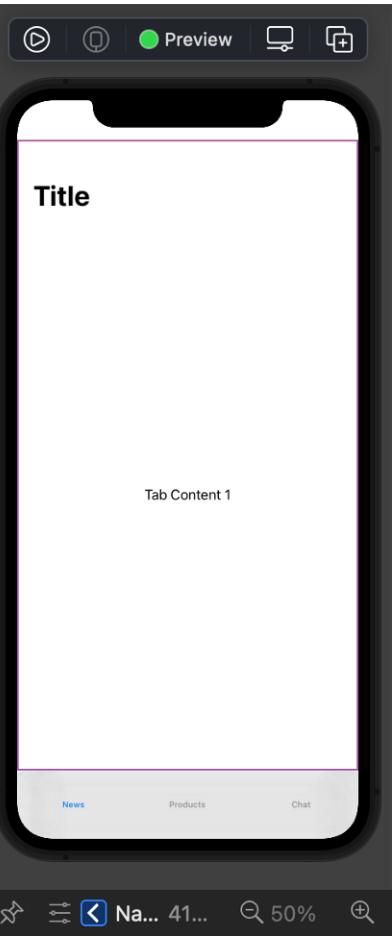


👉 Drag the `Navigation Title` into the code to insert a new line after the first `Text` line.



[Open in app](#)

```
8 import SwiftUI
9
10 struct ContentView: View {
11     var body: some View {
12         TabView(selection: $selection) {
13             NavigationView {
14                 Text("Tab Content 1")
15                     .navigationTitle("Title")
16             }
17             .tabItem { Text("News") }.tag(1)
18             Text("Tab Content 2").tabItem { Text("Products") }.tag(2)
19             Text("Tab Content 3").tabItem { Text("Chat") }.tag(3)
20         }
21     }
22 }
23
24 struct ContentView_Previews: PreviewProvider {
25     static var previews: some View {
26         ContentView()
27     }
28 }
29
30
```



⌚ The preview should show the `Title` at the top of the News scene, in the navigation bar.

👉 Change the "Title" to "News" and verify that it updates in the preview.



The screenshot shows the Xcode interface with the code editor on the left and a simulator preview on the right. The code editor contains the following SwiftUI code:

```

8 import SwiftUI
9
10 struct ContentView: View {
11     var body: some View {
12         TabView(selection: $selection) {
13             NavigationView {
14                 Text("Tab Content 1")
15                     .navigationTitle("News")
16             }
17             .tabItem { Text("News") }.tag(1)
18             Text("Tab Content 2").tabItem { Text("Products") }.tag(2)
19             Text("Tab Content 3").tabItem { Text("Chat") }.tag(3)
20         }
21     }
22 }
23
24 struct ContentView_Previews: PreviewProvider {
25     static var previews: some View {
26         ContentView()
27     }
28 }
29
30

```

The simulator preview shows an iPhone screen with a tab bar at the bottom labeled "News", "Products", and "Chat". The "News" tab is selected, displaying the text "Tab Content 1". The status bar at the top of the simulator shows "News" and "Tab Content 1".

👉 Repeat the above process to add a `NavigationView` and `navigationTitle` to the Products and Chat scenes. You can drag from the library or copy and paste or type code — whatever you find easiest.

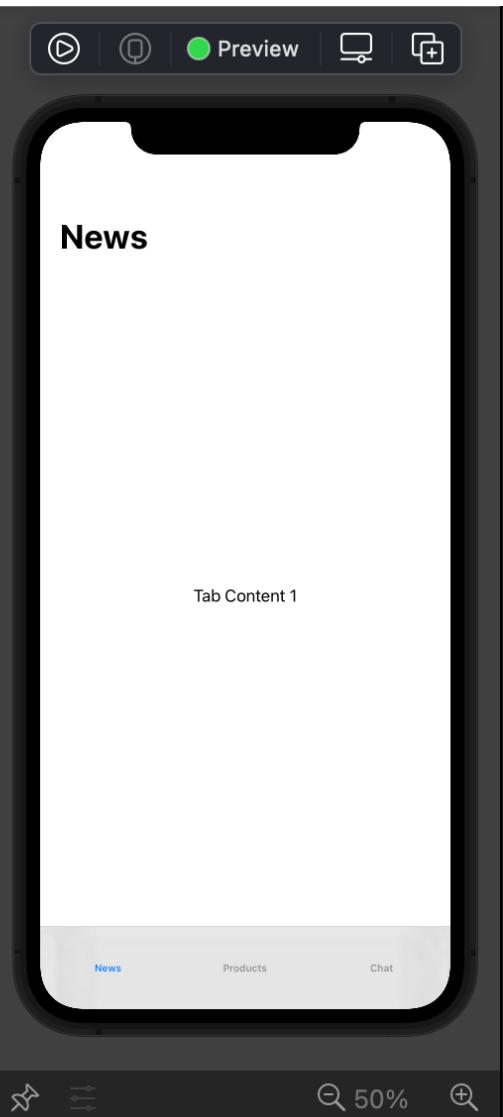
Notice that Xcode likes to indent the code a particular way. It is a handy way to check your code for any missing lines. You can apply Xcode's indenting at any time by:

1. In the `Edit` menu, select `Select All`. Or use `command - a` on the keyboard.
 2. In the `Editor` menu, in the `Structure` submenu, select `Re-Indent`. Or use `control - i` on the keyboard.
- ⌚ When you're done, the code should look like below and the preview should refresh without any errors. If not, then meticulously check your code.



[Open in app](#)

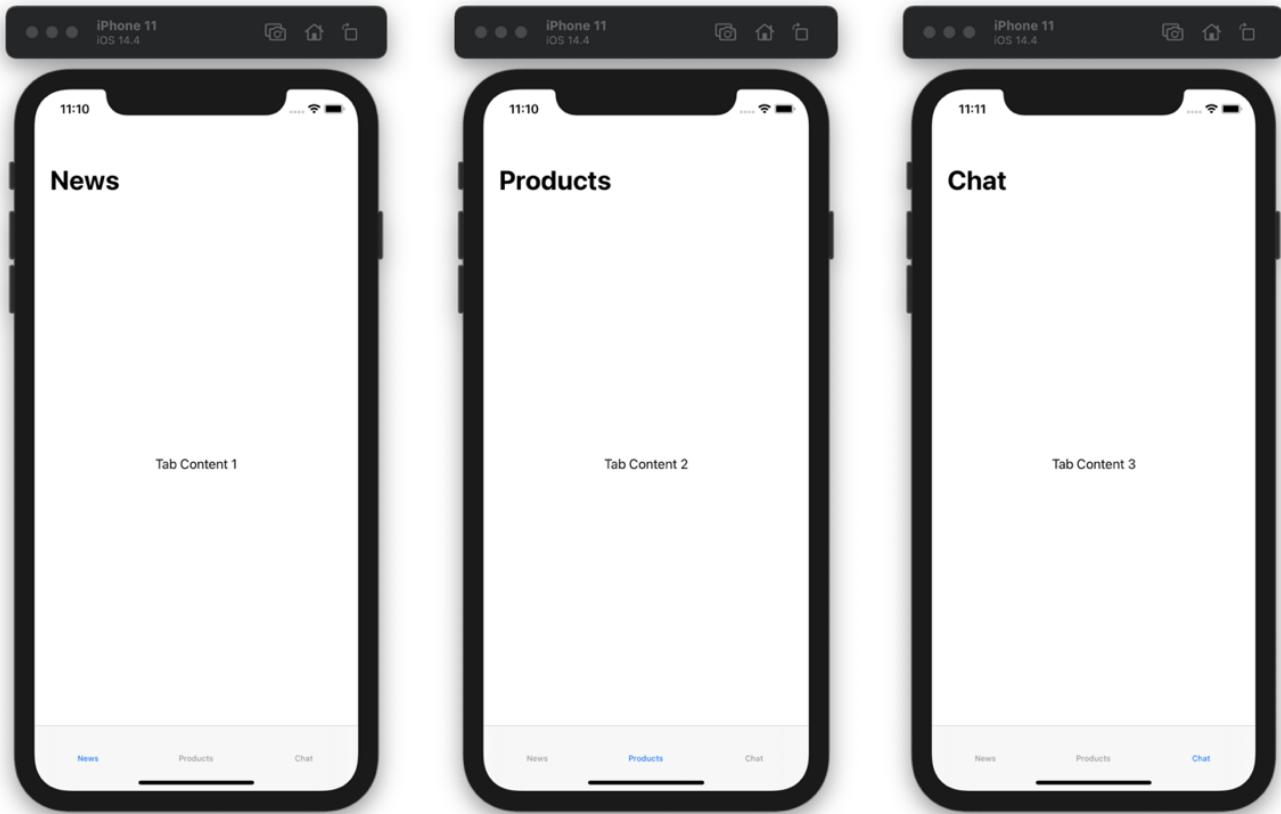
```
8  
9 import SwiftUI  
10  
11 struct ContentView: View {  
12     var body: some View {  
13         TabView(selection: $selection) {  
14             NavigationView {  
15                 Text("Tab Content 1")  
16                     .navigationTitle("News")  
17             }  
18             .tabItem { Text("News") }.tag(1)  
19             NavigationView {  
20                 Text("Tab Content 2")  
21                     .navigationTitle("Products")  
22             }  
23             .tabItem { Text("Products") }.tag(2)  
24             NavigationView {  
25                 Text("Tab Content 3")  
26                     .navigationTitle("Chat")  
27             }  
28             .tabItem { Text("Chat") }.tag(3)  
29         }  
30     }  
31 }  
32  
33 struct ContentView_Previews: PreviewProvider {  
34     static var previews: some View {  
35         ContentView()  
36     }  
37 }
```



👉 When the code and preview show no errors, click the `Run` button to run the app in the simulator.

👉 Tap on each tab item and confirm that the selected tab switches to show the correct navigation title.



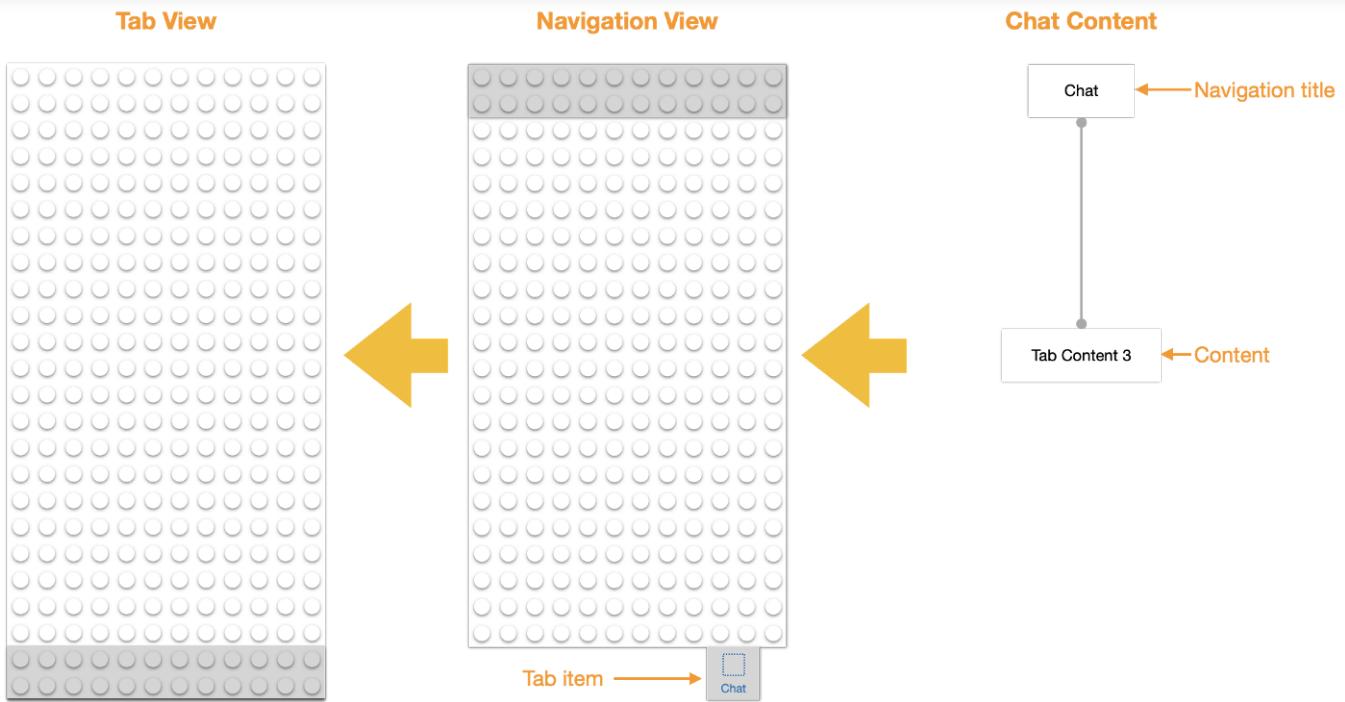
[Open in app](#)

⌚ Each scene is showing:

1. The **tab item** label (News, Products, Chat) in the tab bar, at the bottom. They have no icons yet.
2. The **navigation bar**, at the top. Each belongs to a navigation view.
3. A **navigation title** for each scene, which appears in the navigation bar when that scene is active.
4. A text view inside each tab item, showing placeholder “Tab Content” titles. We will soon replace each of those with a vertically scrolling list of content.

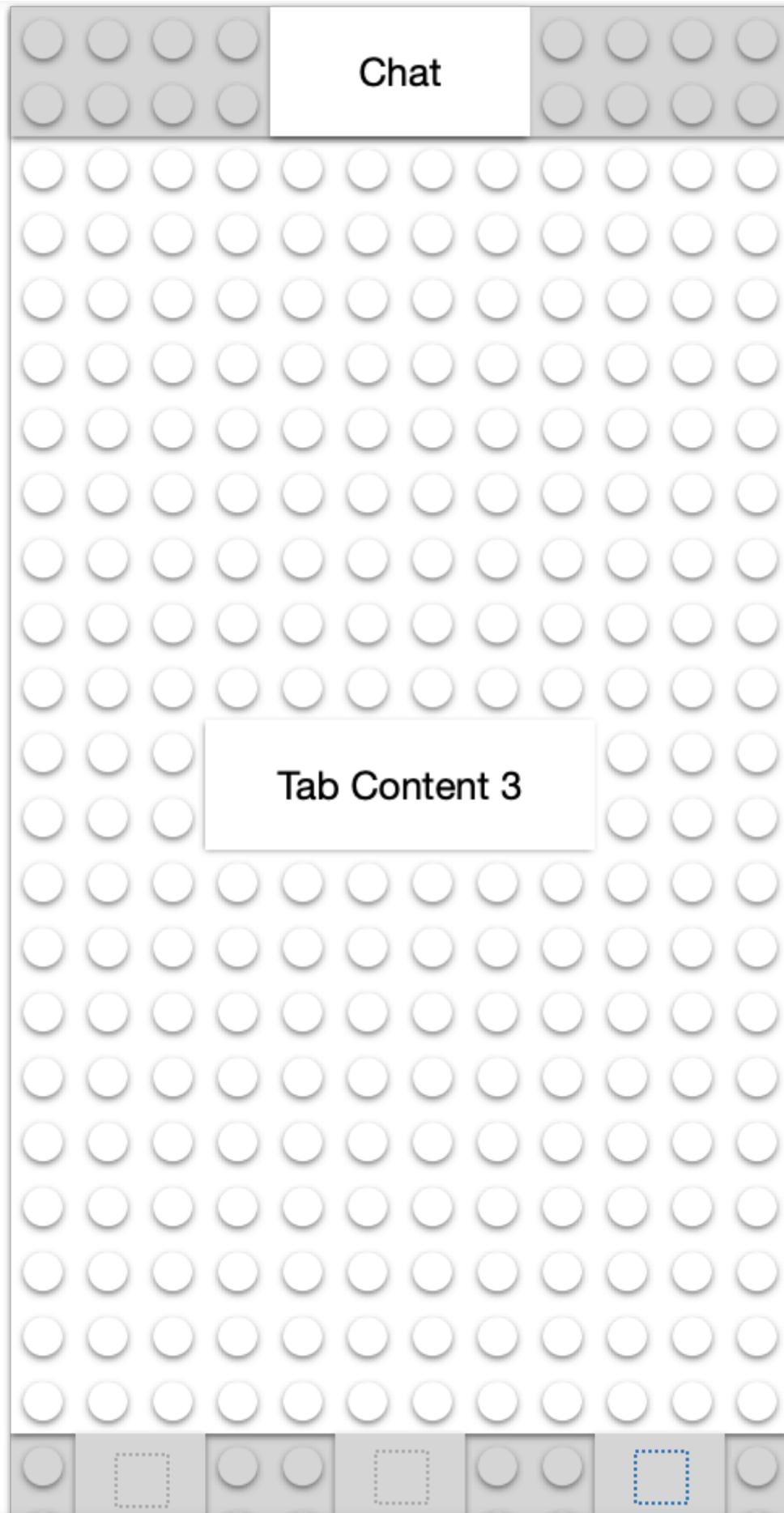
Notice that the tab bar and the navigation bar are both displayed. The tab view contains the navigation views, which each contain a navigation bar and one text view. We can picture it as these layers of Lego being added on top of each other. For example, when the Chat tab item is selected, the visible structure is like this:



[Open in app](#)

Added together, with the tab items of the other scenes, we get the composite of all of the scenes as:



[Open in app](#)

[Open in app](#)

Switch back to Xcode.

As we did in Tutorial 1:

1. Choose Commit from the Source Control menu.
2. Enter a description such as: Added navigation views and titles
3. Click on the Commit button.

5. Next...

We've covered a lot in a short time. In this tutorial we have:

1. Created three navigation views, each with a navigation bar.
2. Set the navigation title of each navigation view.
3. Embedded the navigation views in a tab view.
4. Moved each tab item to be connected to a navigation view.

In [Tutorial 4](#), we will introduce the Attributes inspector to modify content and color in the app.

!? If you have any questions or comments, please add a response below.

This series is released via [Next Level Swift](#). Subscribe to keep updated and never miss a new Tutorial of this series!

[Next Level Swift](#)

Next Level Swift

Next Level Swift aims at sharing knowledge and insights into better programming for iOS and is dedicated to help...

[medium.com](https://medium.com/nextlevelswift)



[Open in app](#)

Swift Community!

Sign up for Next Level Swift Newsletter

By Next Level Swift

Get all the latest articles, posts and news straight to your mailbox! [Take a look.](#)

[Get this newsletter](#)

Emails will be sent to research2learn@yahoo.co.uk.

[Not you?](#)



[Open in app](#)Published in Next Level Swift · [Following](#) ▾Tom Brodhurst-Hill · [Following](#)

Apr 1, 2021 · 6 min read ★

...

Attributes Inspector and Accent Color

Build an App Like Lego, with SwiftUI — Tutorial 4



1. Introduction

In [Tutorial 3](#), we embedded the News, Products, and Chat scenes inside their own navigation views, as subviews in tab view, each with a tab item. Each scene contained just one view: a text view with the words `Tab Content` and a number.

In this tutorial, we will add “modifiers” to views to change their appearances, such as font and weight. We will also change the whole app’s “accent color”.



[Open in app](#)

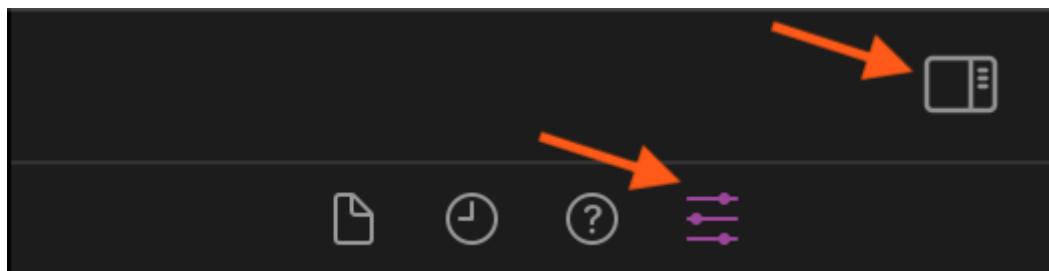
2. Attributes Inspector

In our Xcode project, the code is the source of truth. Every change we make affects the code. The preview and the running app read that code.

But code can be mysterious and tedious for a human to type correctly. So, Xcode provides several visual tools that we can use to write the code for us. For example, in earlier tutorials, we have already dragged some components into our code from the Views Library.

Xcode provides another tool called the “Attributes inspector”, which we can use to modify the content and properties of views. We can type words in a text field or choose from popup menus to make Xcode write the required code for us.

👉 If the Inspector panel (on the right-hand side) is not currently visible, make it appear by clicking on the top-right icon in the Xcode window.



👉 Click on the Attributes icon to show the Attributes inspector.

3. Text and Font

👉 You should see the preview showing the News tab item selected with the contents showing Tab Content 1 .



[Open in app](#)

The screenshot shows the Xcode interface with the ContentView.swift file open. The code defines a ContentView struct using SwiftUI's TabView and NavigationView components. The preview window on the right shows an iPhone 11 displaying the app's interface with three tabs at the bottom: News, Products, and Chat. The News tab is selected, showing the text "Tab Content 1".

```
1 //  
2 // ContentView.swift  
3 // Blocks  
4 //  
5 // Created by Tom Brodhurst-Hill on 4/3/21.  
6 // Copyright © 2021 BareFeetWare. All rights reserved.  
7 //  
8  
9 import SwiftUI  
10  
11 struct ContentView: View {  
12     var body: some View {  
13         TabView(selection: $selection) {  
14             NavigationView {  
15                 Text("Tab Content 1")  
16                     .navigationTitle("News")  
17             }  
18             .tabItem { Text("News") }.tag(1)  
19             NavigationView {  
20                 Text("Tab Content 2")  
21                     .navigationTitle("Products")  
22             }  
23             .tabItem { Text("Products") }.tag(2)  
24             NavigationView {  
25                 Text("Tab Content 3")  
26                     .navigationTitle("Chat")  
27             }  
28             .tabItem { Text("Chat") }.tag(3)  
29         }  
30     }  
31 }  
32  
33 struct ContentView_Previews: PreviewProvider {  
34     static var previews: some View {  
35         ContentView()  
36     }  
37 }
```

👉 Select the content text by double-clicking on the Tab Content 1 text in the preview. If you only click once, it will only select the navigation view. Double-clicking selects the next inner view.



[Open in app](#)

The screenshot shows the Xcode interface with the following details:

- Code Editor:** Displays `ContentView.swift` code. The line `Text("Tab Content 1") .navigationTitle("News")` is selected.
- Preview View:** Shows a mobile device simulation with a tab bar at the bottom labeled "News", "Products", and "Chat". The main screen displays the text "News" under the heading "Tab Content 1".
- Attributes Inspector:** On the right, the "Text" section is active, showing "Tab Content 1" in the preview. Below it, the "Font" section has "Inherited" selected. Other sections like "Modifiers", "Padding", "Frame", and "Navigation Title" are also visible.

💡 In the code, you should now see the `Text` view and its `navigationTitle` modifier selected. In the Attributes inspector, you should see the `Text` with its attributes (content and modifiers) listed below it.

This text will eventually be the title of each news article in our list. For now, let's change its wording to just be the word “Text”.

👉 In the Attributes inspector, change the words from `Tab Content 1` to `Text`. Hit the return key to trigger the preview to update.

👉 Click on the `Font` popup menu and change it from `Inherited` to `Title2`.

🐞 If `Title2` isn't listed in the popup menu, select `Title`, then try again to select `Title2`.



[Open in app](#)

The screenshot shows the Attributes Inspector for a Text view. The 'Font' section is currently active, displaying a list of font styles. The 'Title2' style is selected, indicated by a pink highlight and a checkmark icon. Other options include Inherited, Large Title, Title, Title3, Headline, Subheadline, Body, Callout, Footnote, Caption, and Caption2. The 'Weight' section is partially visible on the left. At the bottom, there are buttons for 'Padding' (with minus and plus signs), 'Inherited', and a search bar.

👉 Similarly, change the Weight to Bold .



[Open in app](#)

The screenshot shows the Xcode Attributes Inspector with the following settings:

- Font:** Title2
- Weight:** ✓ Inherited (selected)
- Color:** Purple (selected)
- Modifiers:** None
- Font Size:** 14pt
- Font Style:** Regular
- Font Weight Options:** Semibold, Regular, Heavy, Light, Thin, Medium
- Font Color Options:** Black, Ultra Light, Black

- ⌚ Check that the code, preview, and Attributes inspector appear as shown below.





Open in app

The screenshot shows the Xcode interface with the following details:

- Left Panel (Code View):** Displays the `ContentView.swift` file. The code defines a `ContentView` struct that contains a `body` variable and a `TabView` with three `tabItem`s labeled "News", "Products", and "Chat".
- Middle Panel (Preview View):** Shows a preview of an iPhone 11 screen displaying a navigation view. The title bar says "News". Below it is a tab bar with three items: "News" (blue), "Products" (gray), and "Chat" (gray). The main content area is labeled "Text".
- Right Panel (Attributes Inspector):** Shows settings for the "Text" element selected in the preview. It includes sections for **Font** (Title2, Bold, Inherited), **Modifiers**, **Padding**, **Frame**, and **Navigation Title** (set to "News").

4. Accent Color and Affordance

The user should be able to tell, just by looking at a scene, which objects they can touch. This visual identification is called “affordance”.

One way that an app can provide affordance is by using a consistent “accent color” for the objects (subviews) that the user can touch. This includes buttons, tab item labels, and popup menus. The accent color is sometimes also called the “tappable color”.

We can change the accent color of individual scenes or views, but the best user interface will consistently use one accent color throughout the whole app. The app should avoid using that accent color for anything that is not tappable, to not confuse the user with “false affordance”.

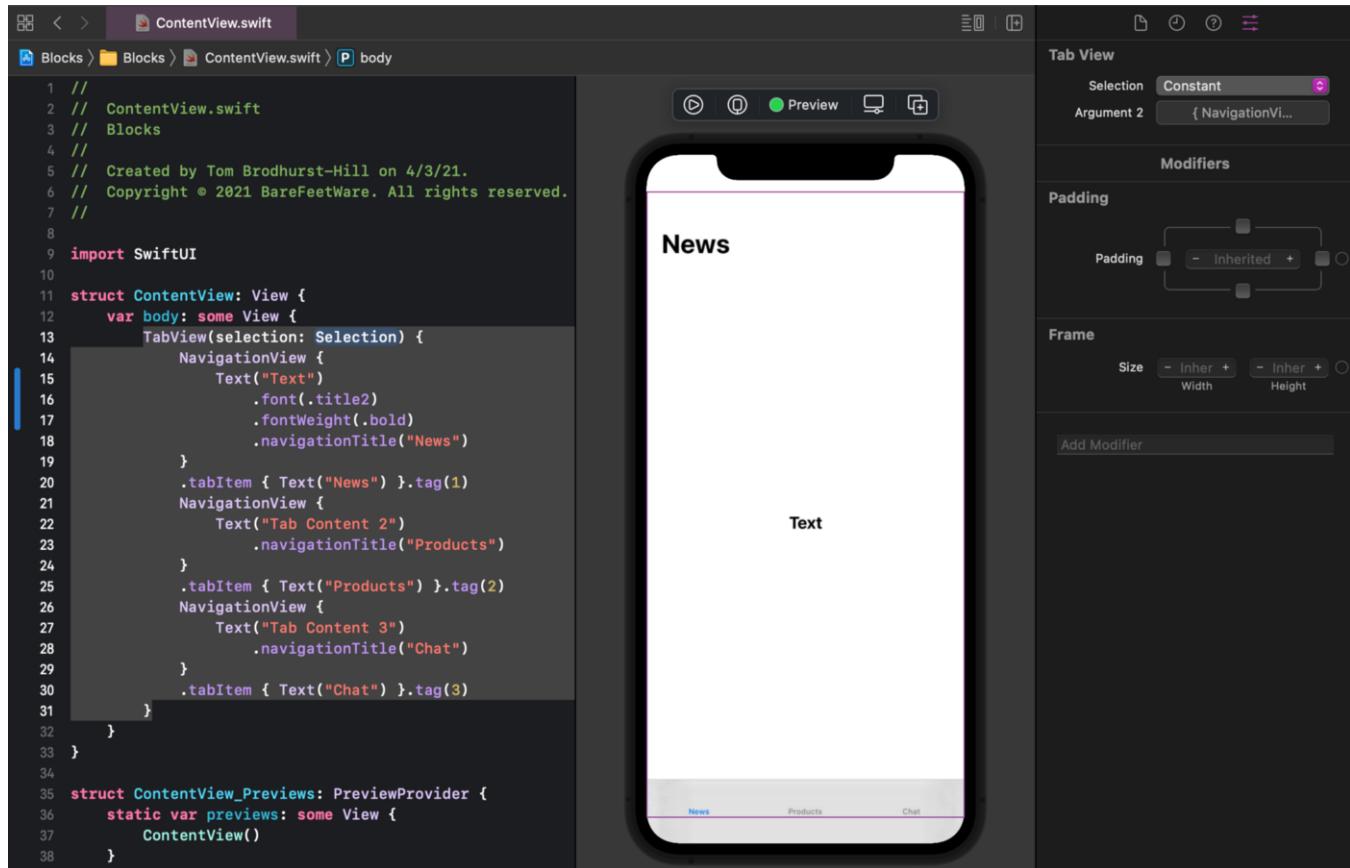
💡 As you can see in our app so far, the three-tab items at the bottom are all shown in blue.



[Open in app](#)

Previously, we added new modifiers by clicking the + Library button and dragging the modifier into our code from the list. Now we'll instead add the accent color modifier via the Attributes inspector.

👉 In the preview, click once on the tab bar at the bottom.



🐞 If you mistakenly double click on the tab bar, the `TabView` at the top of the code will change from the `Selection` placeholder to `.constant(1)`. You can just `Undo` to go back to how it was and then try again. But if it changes to `.constant(1)`, that is also OK.

👀 As shown above, you should see the `TabView` highlighted in the code, along with all of its content (subviews).

👉 In the Attributes inspector, click in the bottom field that has the `Add Modifier` placeholder text. You should see a popup menu of all the available modifiers. If the text cursor is flashing in the field but the popup menu doesn't appear, hit the down arrow on the keyboard to make the menu appear.



[Open in app](#)

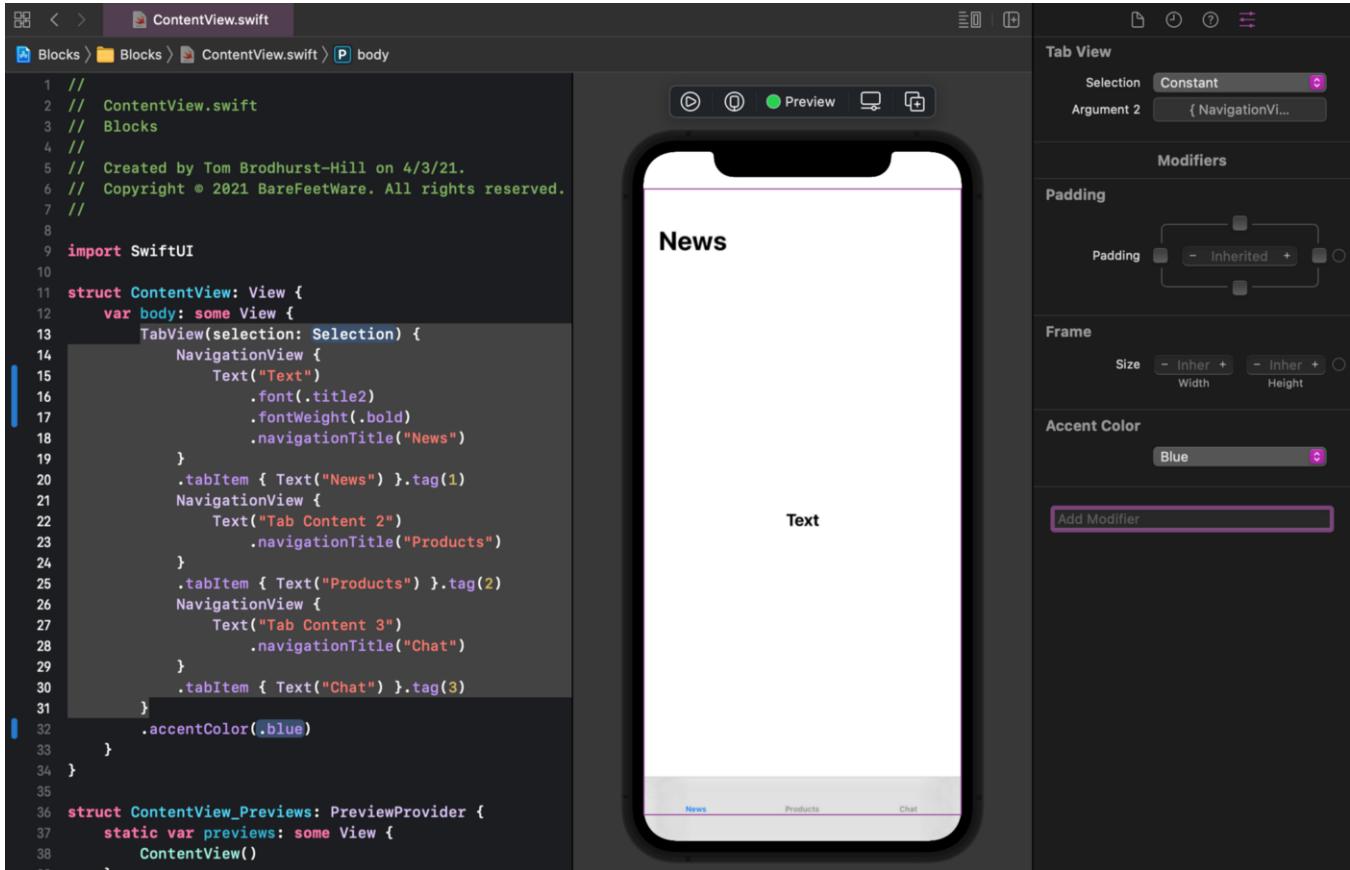
👉 In the list of modifiers, click on Accent Color to select it.



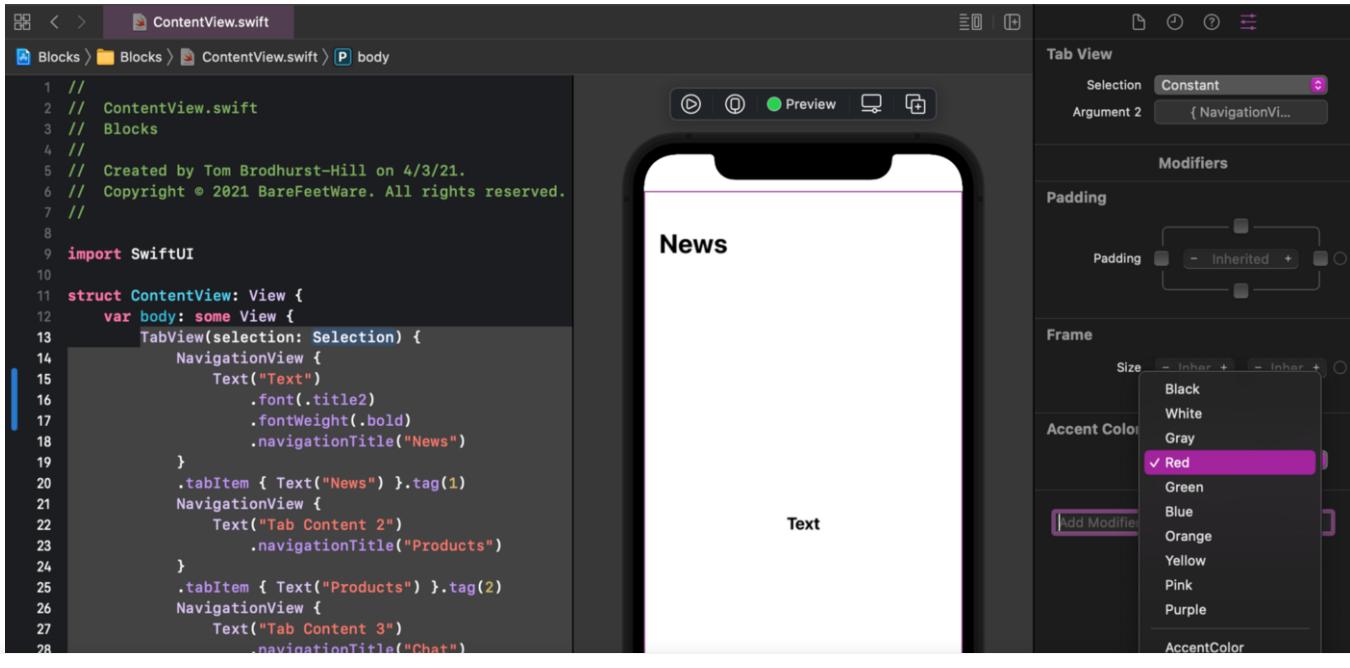


Open in app

blue .



👉 In the Attributes inspector, click on the Accent Color pop up menu and change it from blue to red .



[Open in app](#)

- 🕒 As shown above, the code and preview update to show the red accent color.

5. Commit Changes

As you've done before:

1. ➡ Choose Commit from the Source Control menu.
2. ➡ Enter a description such as: Modified text attributes and app accent color
3. ➡ Click on the Commit button.

6. Next...

In this Tutorial 4, we used the Attributes inspector to customize some text and the app's accent color.

In [Tutorial 5](#) we will add a list, with cells that navigate to a linked destination scene.

❗ If you have any questions or comments, please add a response below.

This series is released via [Next Level Swift](#). Subscribe to keep updated and never miss a new Tutorial of this series!

[Next Level Swift](#)

Next Level Swift

Next Level Swift aims at sharing knowledge and insights into better programming for iOS and is dedicated to help...

[medium.com](https://medium.com/next-level-swift)

We are always looking for talented and passionate Swift developers! Check out our writer's section and find out how you can share your knowledge with the Next Level Swift Community!



[Open in app](#)

Sign up for Next Level Swift Newsletter

By Next Level Swift

Get all the latest articles, posts and news straight to your mailbox! [Take a look.](#)

[Get this newsletter](#)

Emails will be sent to research2learn@yahoo.co.uk.
[Not you?](#)



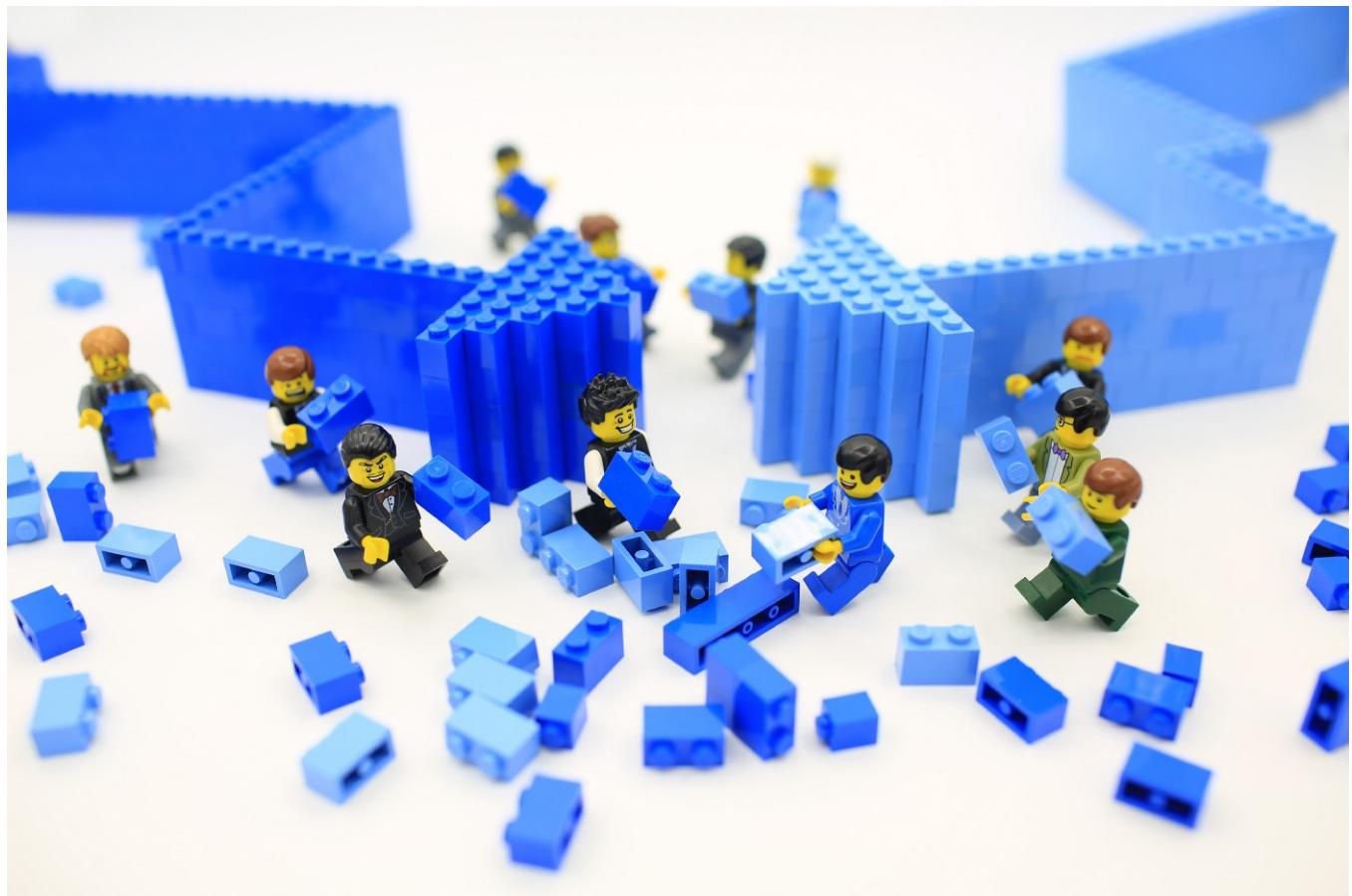
[Open in app](#)Published in Next Level Swift · [Following](#) ▾Tom Brodhurst-Hill · [Following](#)

Apr 6, 2021 · 7 min read ★

...

Create a List of Navigation Links

Build an App Like Lego, with SwiftUI — Tutorial 5



1. Introduction

At the end of [Tutorial 4](#), we had navigation views, but they didn't facilitate navigation to anything. In this Tutorial 5, we will add a list of tappable cells that navigate to a destination scene.



[Open in app](#)

2. List

The News scene contains one text view with the word `Text`. That text doesn't require much space, so the navigation view containing it positions the text in its center.

Instead of just one, we want a list of news titles. So, we need to embed the text in a list. We could drag in a list view from the library, then move the text into it. But there's an easier way. We can ask Xcode to embed the text in a list, in one step.

👉 While holding down the `Command` key on the keyboard, click on the `Text("Text")`.



 Open in app

```
9 import SwiftUI
10
11 struct ContentView: View {
12     var body: some View {
13         TabView(selection: $selection) {
14             NavigationView {
15                 Text("Text")
16                     .font(.title2)
17
18             }
19
20             .tag(1)
21
22             .tag(2)
23
24             .tag(3)
25
26             .tag(4)
27
28             .tag(5)
29
30             .tag(6)
31
32             .tag(7)
33
34         }
35
36         $selection
37     }
38 }
```

A screenshot of Xcode showing a code editor with the following Swift code:

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        TabView(selection: $selection) {
            NavigationView {
                Text("Text")
                    .font(.title2)
            }
            .tag(1)
            .tag(2)
            .tag(3)
            .tag(4)
            .tag(5)
            .tag(6)
            .tag(7)
        }
        $selection
    }
}
```

The cursor is positioned over the line `Text("Text")`. A context menu titled "Actions" is open, listing various code refactoring options. The option `Embed in List` is highlighted with a pink rectangle.

💡 Xcode shows the Actions popup menu, listing the actions you can perform on the selected code.

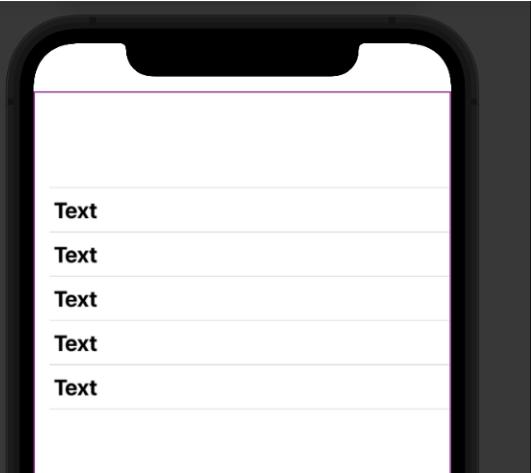
👉 In the Actions menu, select `Embed in List`.



[Open in app](#)

```

8 import SwiftUI
9
10
11 struct ContentView: View {
12     var body: some View {
13         TabView(selection: $selection) {
14             NavigationView {
15                 List(0 ..< 5) { item in
16                     Text("Text")
17                         .font(.title2)
18                         .fontWeight(.bold)
19                         .navigationTitle("News")
20                 }
21             }
22             .tabItem { Text("News") }.tag(1)
23             NavigationView {
24                 Text("Tab Content 2")
25                     .navigationTitle("Products")
26             }
27         }
28     }
29 }
```



- ⌚ Xcode inserts a new `List` view between the `NavigationView` and the `Text` view. The `Text` is “embedded” in the `List`. The list defaults to showing five rows, currently with each row as a copy of the `Text("Text")`.
- ⌚ All of the modifiers that were attached to the `Text` have stayed with it, and have been embedded in the `List`. For `font` and `fontWeight`, that is what we want. But, notice that the navigation title “News” has disappeared from the preview. We need to move the `.navigationTitle` modifier to again be immediately inside the `NavigationView`, so that it can again display the title in the bar.

👉 Triple-click (click the mouse button three times quickly) on the line of code `.navigationTitle("News")`. That should select the entire line, including empty space.

```

9 import SwiftUI
10
11 struct ContentView: View {
12     var body: some View {
13         TabView(selection: $selection) {
14             NavigationView {
15                 List(0 ..< 5) { item in
16                     Text("Text")
17                         .font(.title2)
18                         .fontWeight(.bold)
19                         .navigationTitle("News")
20                 }
21             }
22         }
23     }
24 }
```



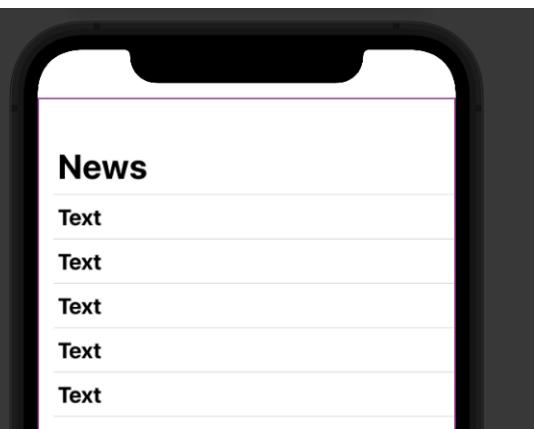
[Open in app](#)

- 👉 Drag the selected line down one line, between the closing } of the List and the NavigationView , as shown below.

```

8
9 import SwiftUI
10
11 struct ContentView: View {
12     var body: some View {
13         TabView(selection: $selection) {
14             NavigationView {
15                 List(0 ..< 5) { item in
16                     Text("Text")
17                         .font(.title2)
18                         .fontWeight(.bold)
19                 }
20                 .navigationTitle("News")
21             }
22             .tabItem { Text("News") }.tag(1)
23         NavigationView {

```



- 👁️ The News navigation title should now reappear in the preview.

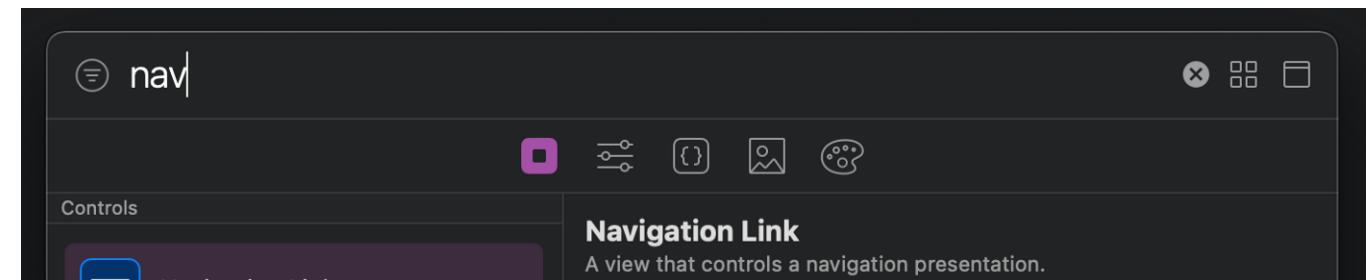
3. Navigation Link

Now we have a list of text views, each of which will eventually show the title of a different news article. The list gives each item a full horizontal row and inserts divider lines between them. Each row in a list is often referred to as a “cell”.

We want each cell in the list to be a link that navigates to a new scene containing the detail of that news item. SwiftUI provides a `NavigationLink` view for this purpose.

We need to embed the `Text` view inside a `NavigationLink` , with a similar end result to how we embedded it in a `List` . Unfortunately, the Action menu doesn’t provide an “Embed in `NavigationLink` ” action. So, we need to grab it from the view library and move the `Text` into it.

- 👉 Click the + button to show the Library. Click the Views icon. Type `nav` in the filter.



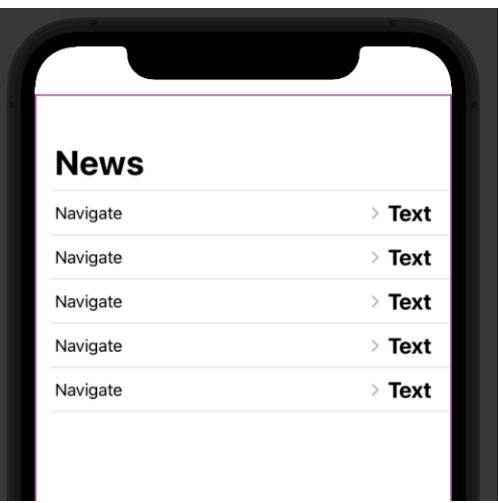
[Open in app](#)

- 👉 Drag the `NavLink` from the Library into the code, to insert a new line immediately before the `Text("Text")`.

```

8 import SwiftUI
9
10 struct ContentView: View {
11     var body: some View {
12         TabView(selection: $selection) {
13             NavigationView {
14                 List(0 ..< 5) { item in
15                     NavLink(destination: Destination) {
16                         Label Content
17                     }
18                     Text("Text")
19                         .font(.title2)
20                         .fontWeight(.bold)
21                 }
22                 .navigationTitle("News")
23             }
24             .tabItem { Text("News") }.tag(1)
25         }
26     }

```



- 👁️ As you can see, each cell in the `List` now contains two subviews: the `NavLink` and the `Text`. It displays them side by side in the preview.

We instead want the `Text` embedded inside the `NavLink`.

- 👉 Select the `Text("Text")` line of code and the two subsequent lines of modifiers (`font` and `fontWeight`). Move those three lines of code up one line to before the closing `}` of the `NavLink`. Delete the `Label Content` placeholder line.

```

8 import SwiftUI
9
10 struct ContentView: View {
11     var body: some View {
12         TabView(selection: $selection) {
13             NavigationView {
14                 List(0 ..< 5) { item in
15                     NavLink(destination: Destination) {
16                         Text("Text")
17                             .font(.title2)
18                             .fontWeight(.bold)
19                     }
20                 }
21             }
22             .navigationTitle("News")
23         }
24         .tabItem { Text("News") }.tag(1)
25     }

```



- 👁️ In the preview (refresh it if needed), you can see that each row in the list now has a “chevron” (the head of an arrow) on the far right, pointing to the right.



[Open in app](#)

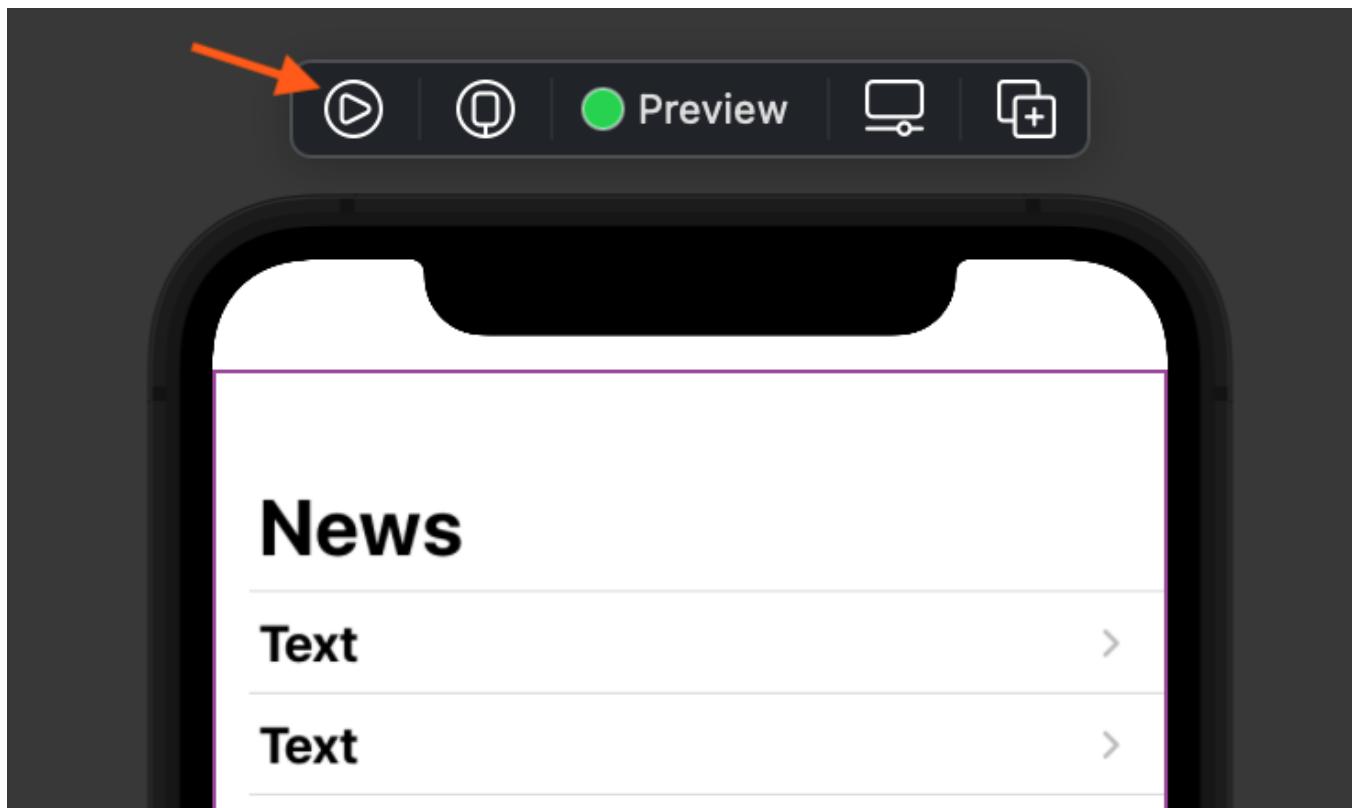
disclosure indicator is gray, not the red accent color because you don't have to tap on it directly — just anywhere in that cell.

The `NavLink` code includes a `Destination`. For now, it's just a placeholder. Later, we will replace it with a scene containing detail of each news article.

4. Live Preview

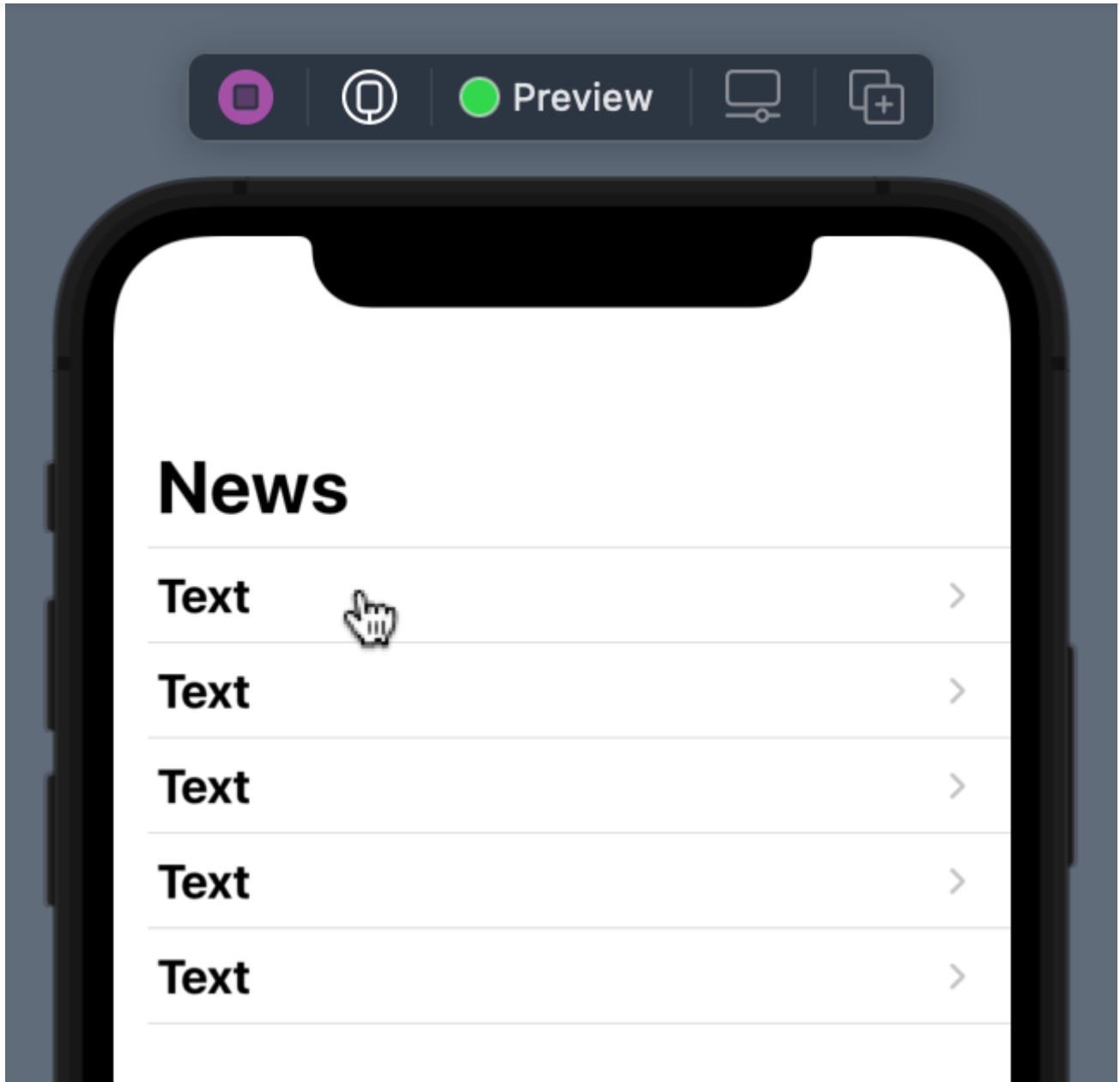
Now we have a navigation link that, when tapped, will navigate to a new destination scene. We can't see this dynamic interaction in the static preview. We will need to either run the app or use the "Live Preview" that Xcode provides for this purpose.

👉 In the preview, click on the `Live Preview` button, which looks like a play triangle in a circle.



👁️ After a few seconds, the button changes to a stop icon, and the background of the preview pane changes color. Move the mouse pointer into the iPhone preview and see that it changes from an arrow to a hand and finger. These all indicate that the preview is now live and accepting interaction.



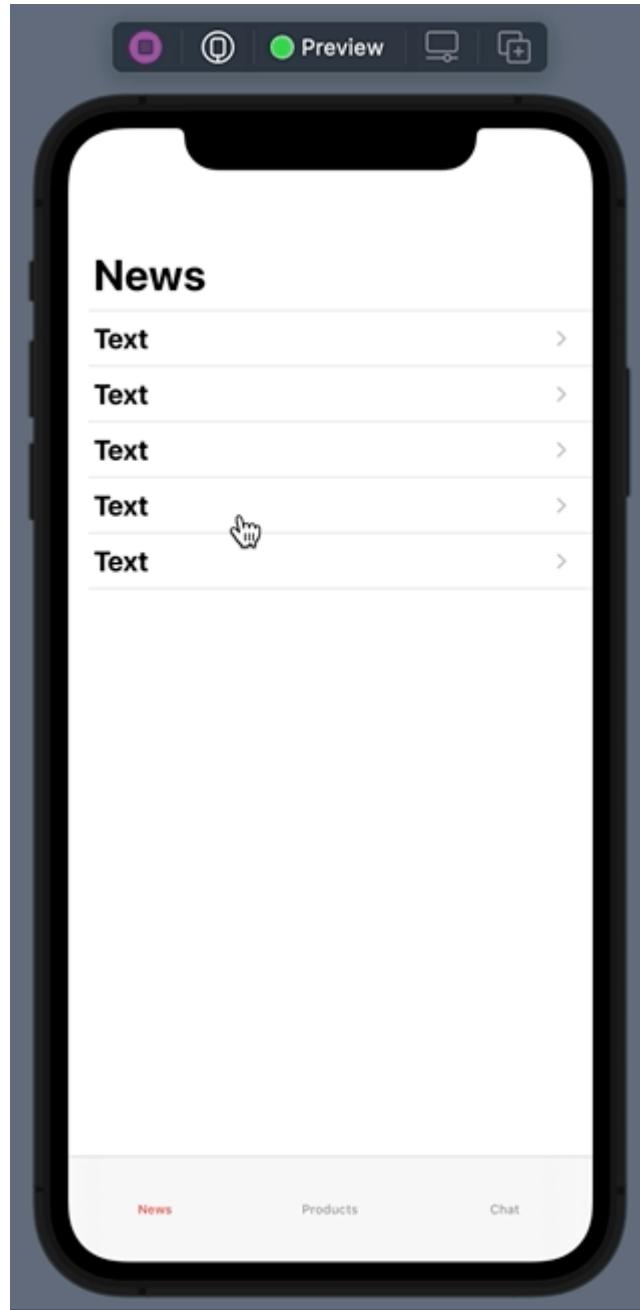
[Open in app](#)

- 👉 Click on one of the `Text >` cells. You can tap anywhere in a cell, bounded by the divider lines and the left and right edges of the screen. You don't have to tap on the disclosure indicator specifically.
- 👀 The app preview accepts the click as a finger tap and navigates to the linked next scene, which is currently just placeholder text that says `Destination`. Note the subtle detail of this transition. The destination scene slides in from the right, pushing the first scene off the screen to the left. The navigation bar and tab bar stay put, but the navigation bar shrinks in height. The large `News` title in the first scene animates to become the title of the back button in the second scene, which is in the app's accent



[Open in app](#)

💡 Note how the animation now runs in reverse, sliding the first scene in from the left.



The navigation link in the navigation view takes care of a lot of UI for us. As another example, instead of tapping the back button, you can swipe from the left edge to the right in the destination scene. If you do it slowly, as you can see in the video above, you can watch the transition in slow motion.

5. Run the app

👉 Run the app.



[Open in app](#)

👉 Switch back to Xcode.

As you've done before:

1. 👉 Choose Commit from the Source Control menu.
2. 👉 Enter a description such as: News: Added list with navigation links
3. 👉 Click on the Commit button.

7. Next...

Our app now has the basic bones of the navigation flow. It has three scenes (News, Products, and Chat) in a tab view. Each scene has its own navigation view. The News scene now contains a list of news titles, each linked to a destination scene, with a back button. We have a smooth animation between scenes, which follows [Apple's Human Interface Guidelines](#).

The tab bar items could use some custom icons. The cells need images. We'll add those next, starting in [Tutorial 6](#).

❗ If you have any questions or comments, please add a response below.

This series is released via [Next Level Swift](#). Subscribe to keep updated and never miss a new Tutorial of this series!

[Next Level Swift](#)

Next Level Swift

Next Level Swift aims at sharing knowledge and insights into better programming for iOS and is dedicated to help...

[medium.com](https://medium.com/nextlevelsnextlevelswift)

We are always looking for talented and passionate Swift developers! Feel free to check out



[Open in app](#)

Sign up for Next Level Swift Newsletter

By Next Level Swift

Get all the latest articles, posts and news straight to your mailbox! [Take a look.](#)

[Get this newsletter](#)

Emails will be sent to research2learn@yahoo.co.uk.

[Not you?](#)



[Open in app](#)Published in Next Level Swift · [Following](#) ▾Tom Brodhurst-Hill · [Following](#)

May 9, 2021 · 8 min read ★

...

Images and SF Symbol Icons

Build an App Like Lego, with SwiftUI — Tutorial 6



Photo by [James Pond](#) on [Unsplash](#)

1. Introduction

We can add two different types of images to an app built in Xcode:

1. “SF Symbol” icons.
2. Or image files.



[Open in app](#)

An image file is usually a “bitmap” graphic, such as PNG (from a drawing program) or JPEG (from photos). It just contains a grid of colored pixels that will stretch or shrink if the image is resized. It can look “pixelated” if it doesn’t have enough pixels to fill its assigned area.

In [Tutorial 5](#), we added a list and navigation links to the flow of our app, including sample text. This Tutorial 6 will add images: some symbols for icons and some photos for products and faces.

We’ll pick up here where the last tutorial left off. Ideally, you have completed the [previous tutorials in this series](#). Or, you can [download](#) the prepared project, ready to start this tutorial.

2. Label

In our Xcode project, the News tab item currently contains only a `Text` view with the word `News`, but no icon.



We want to add an icon in this tab item. The text and the icon will both label the tab item. Then SwiftUI can choose to use the text or icon or both, depending on the circumstance.

Swift (the programming language that we’re using) is pretty flexible about spacing in the code. We can put a new line or spaces almost anywhere we want. As the code gets more complex, it is beneficial to adopt a strategy for consistent spacing that neatly distinguishes between different sections of code. For example, I tend to start each modifier on a new line.

👉 Click in the code just before the period in `.tag(1)`. Hit return on the keyboard to move it down to a new line.



[Open in app](#)

We need to replace the existing `Text("News")` with a label. The easiest way to do this is to delete the old text, drag in a label, then customize the text in that new label.

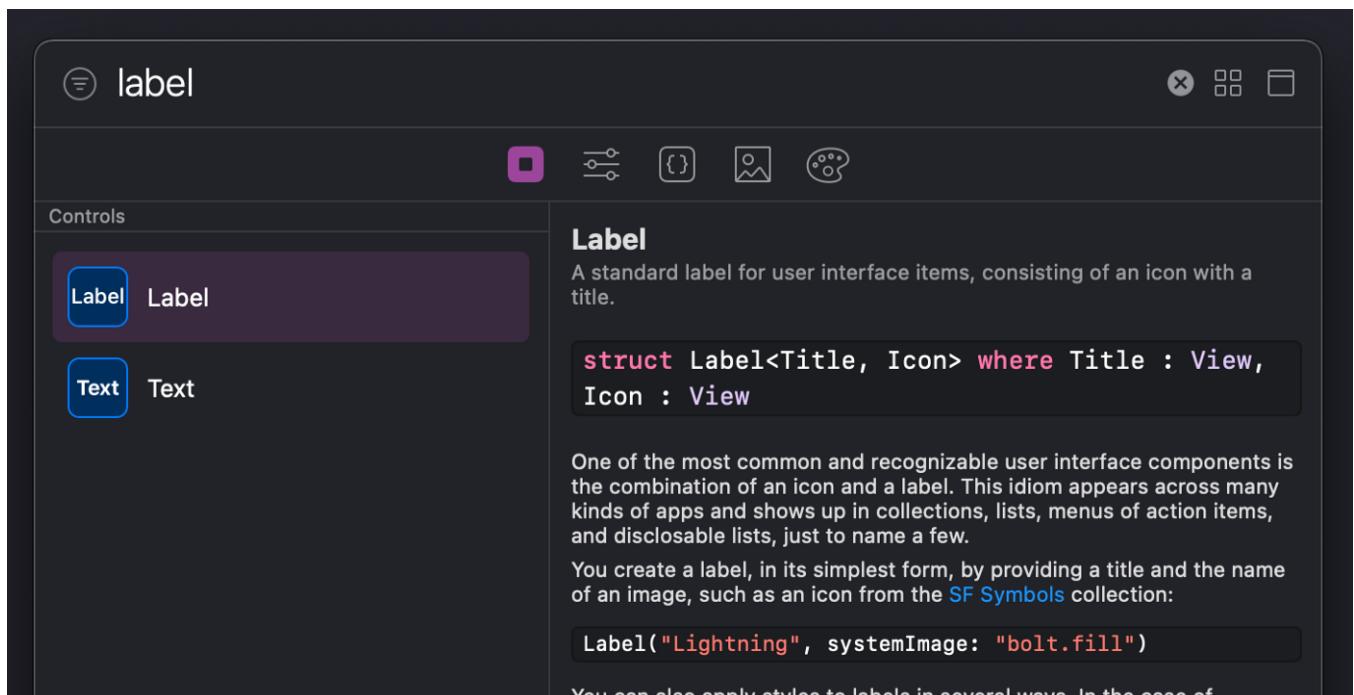
👉 In the `tabItem`, select and delete `Text("News")`.

```

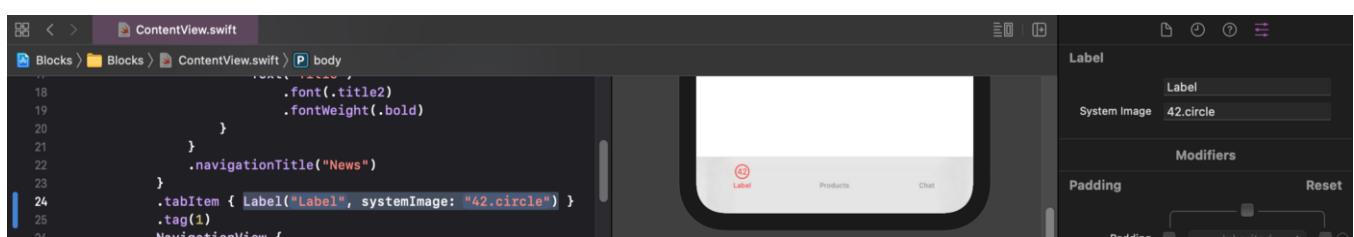
23
24     .tabItem { | }
25     .tag(1)
26     NavigationView {

```

👉 Click on the + Library button, show the Views and filter using the word `Label`.

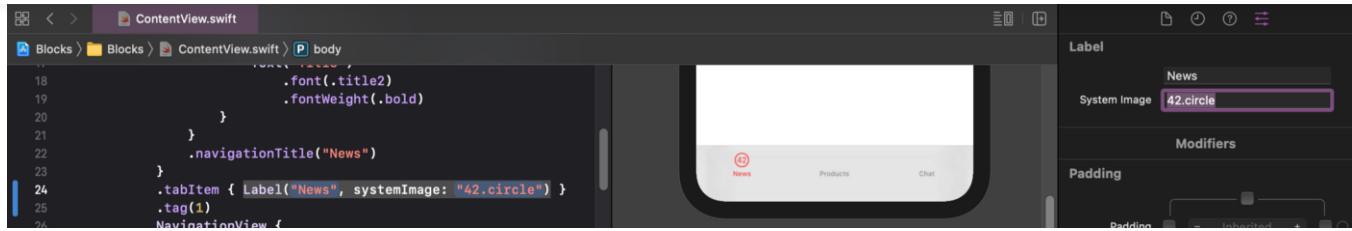


👉 Drag the `Label` from the library into the empty `tabItem` in the code.



👉 In the preview, you can see the first tab item now has the icon of 42 in a circle, and the word `Label` underneath it. In the Attributes inspector and the code, you can see that the `Label` has the word "`"Label"` and a `systemImage` of `42.circle`.



[Open in app](#)

We need to change the icon from `42.circle` to something suitable for news. If we type “news” in that field, the icon goes blank since no such system image is available. So, how do we know what system images we can use?

3. Install the SF Symbols App

SwiftUI has thousands of built-in icons or “system images” called “SF Symbols”, which we can use in our apps for tab items and other image labels. We can peruse the available SF Symbols using the SF Symbols app, available for free from Apple’s web page.

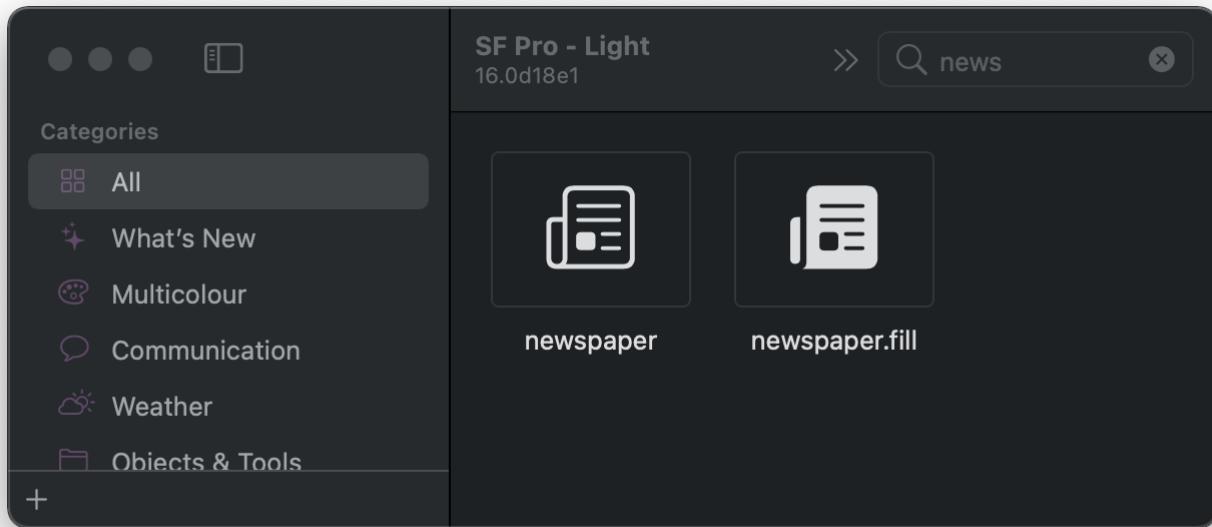
- 👉 Visit Apple’s [SF Symbols web page](#).
- 👉 Under the `Download the app` heading, click the `Download` link next to the latest version of SF Symbols (version 2.1 at the time of writing this article).
- 👉 After it finishes downloading, open the installer and install the new app on your Mac.
- 👉 In your Applications folder, open the new `SF Symbols` app.

4. Add icons to the Tab Items

Let’s find an SF Symbol to use as the icon for our News tab item.

- 👉 With the SF Symbols app open, type `news` into the search bar.

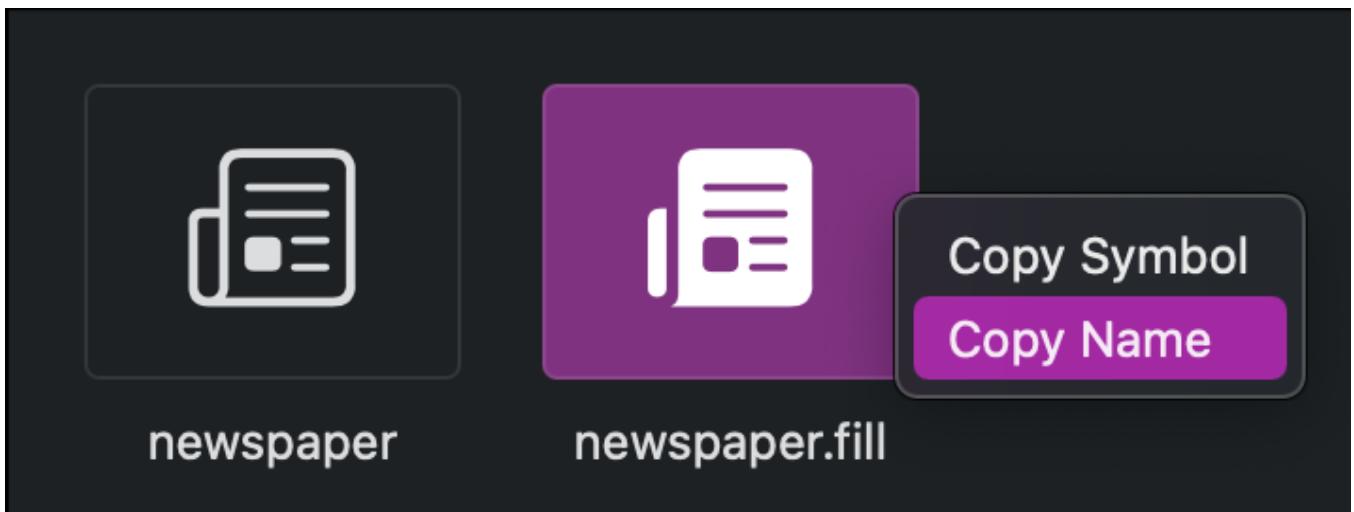


[Open in app](#)

👁 SF Symbols shows the two icons that contain the word `news` in their name.

For tab item icons, filled versions are best because they are more clearly identified when selected.

👉 Right-click (or Control click) on the `newspaper.fill` icon and choose `Copy Name` from the popup menu.

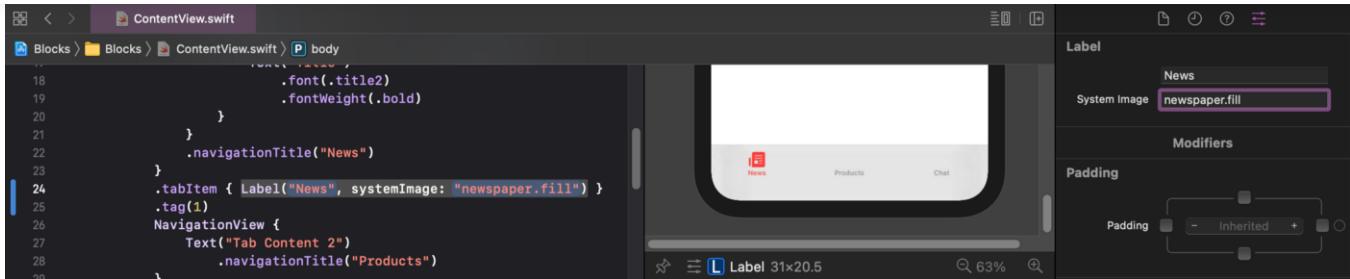


This copies the literal text `newspaper.fill` to the clipboard. But the benefit of using the SF Symbols app is choosing from a known valid symbol name.



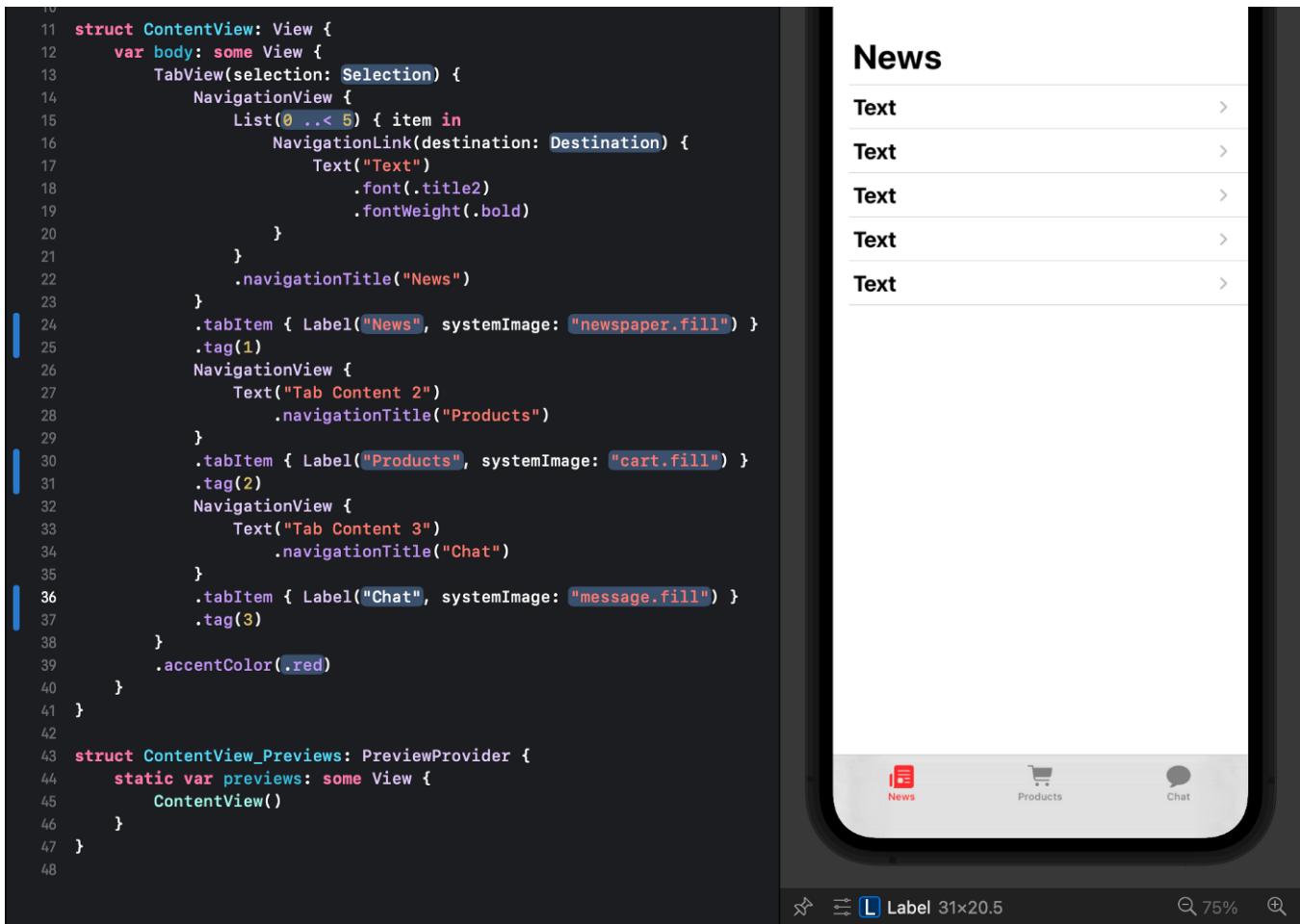


Open in app



💡 The News tab item now shows the `newspaper.fill` icon.

👉 Repeat the process for the other tab items to give them `systemImage`s of `cart.fill` (for Products) and `message.fill` (for Chat). Your code and preview should then appear as shown below.



5. Download Image Files

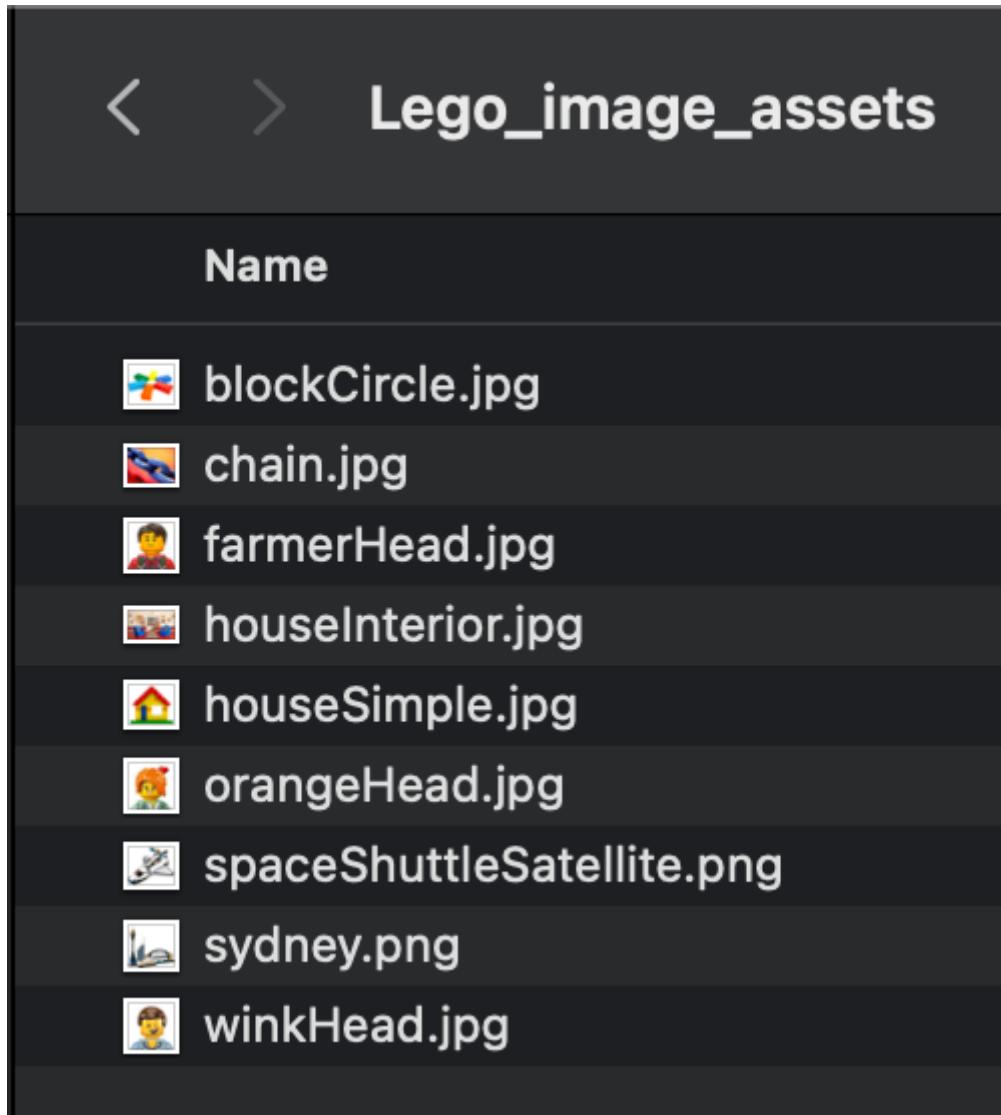
SF Symbols are great for icons. But we sometimes need larger pictures, such as photos of products or people, which aren't system images. For that, we need to add image files



[Open in app](#)

decompress automatically to create a `Lego_image_assets` folder. If not, double click the zip file to decompress it.

👉 Open the `Lego_image_assets` folder. You should see a list of photos for our app. Some are JPEG format; some are PNG format.

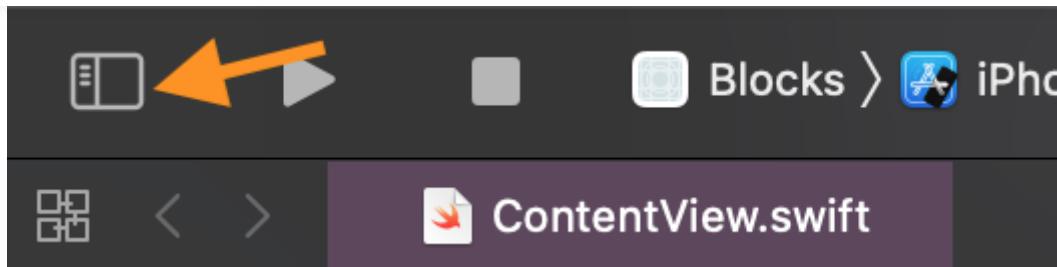


6. Add to the Assets Catalog

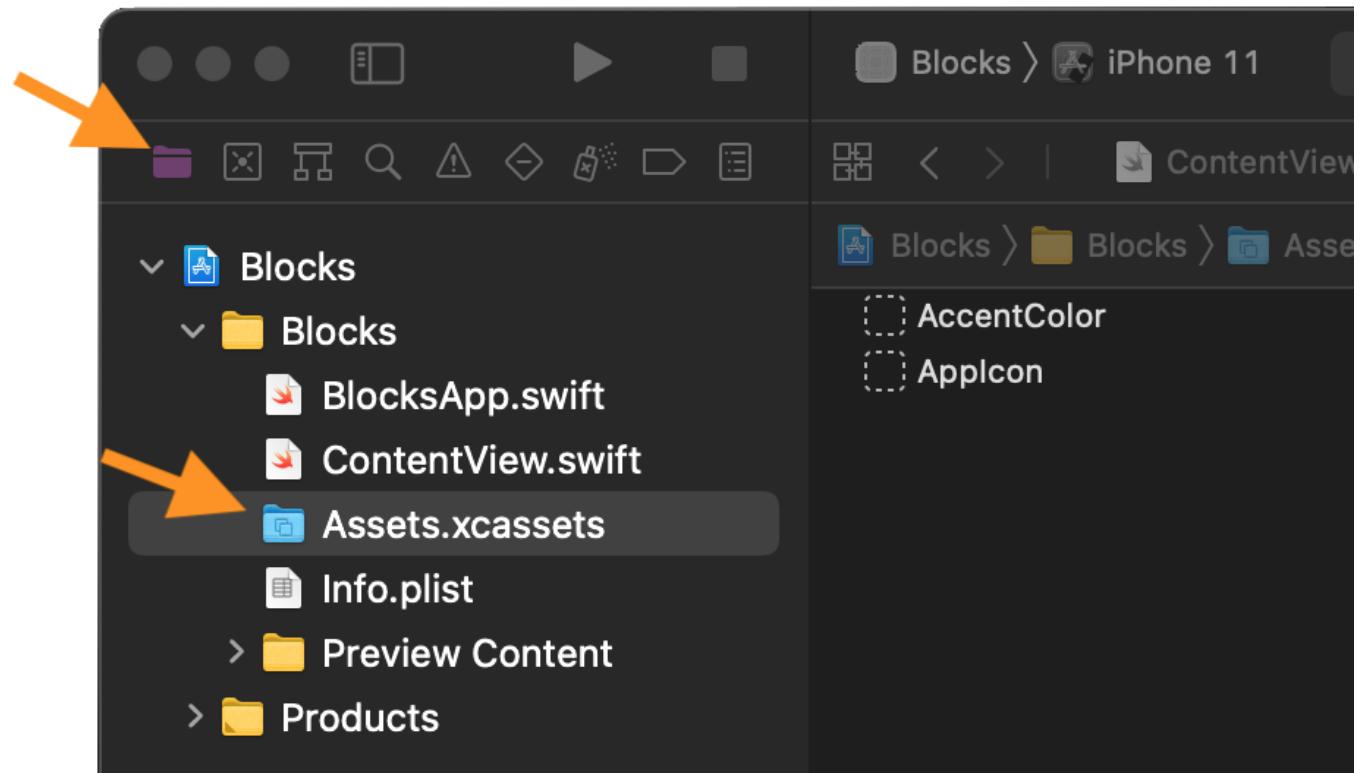
Xcode can handle images as individual files, but it is simpler to manage them inside an “assets catalog”. You can have several asset catalogs in a project, but for now, we’ll use the default “Assets” catalog that Xcode included with the project template.

👉 If Xcode isn’t currently showing the Project Navigator, click on the `Hide` or `Show` the ~~Project Navigator~~ button near the far left of the toolbar.



[Open in app](#)

👉 In your Xcode project, select the `Assets.xcassets` item in the Project navigator.



👀 You can see a placeholder for the `AccentColor` and `AppIcon`. We're not using either of those at this stage.

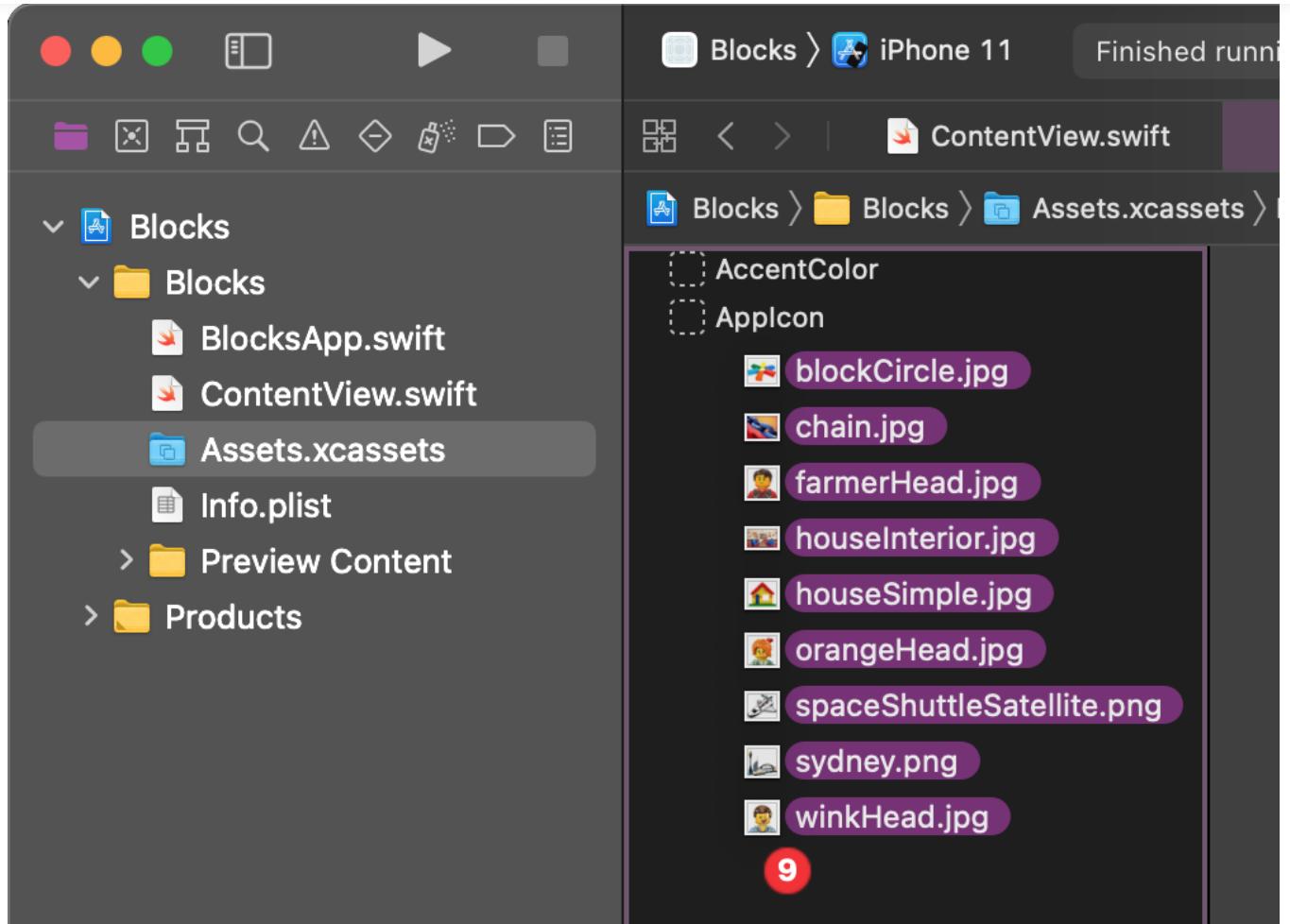
👉 Back in the Finder, select all of the files inside the `Lego_image_assets` folder. One way to do this is to choose one file and choose `Select All` from the `Edit` menu in the Finder.

👉 Drag all of the files (by dragging one, with all selected) into Xcode into the middle panel below the `AppIcon` placeholder.

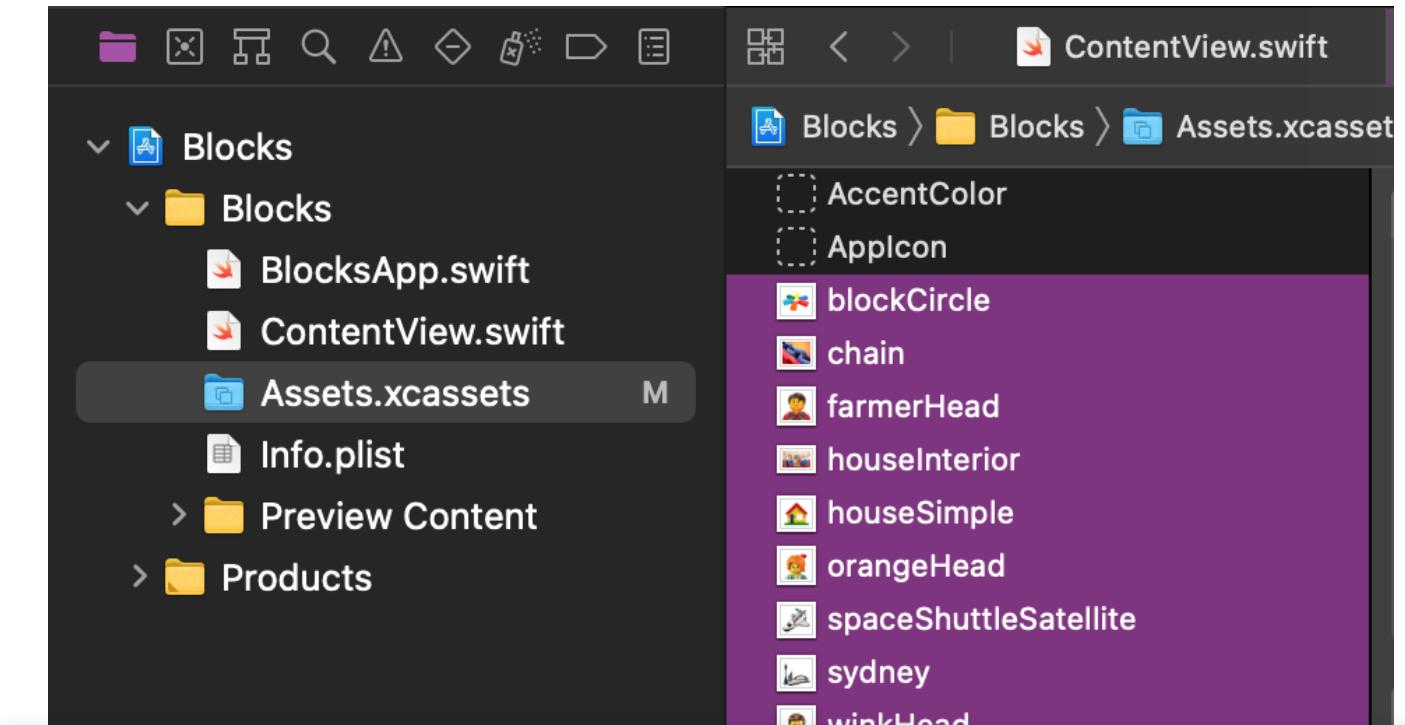




Open in app



👁 You should see the images listed in the assets catalog.




[Open in app](#)

👉 As you can see, the assets catalog defaults to providing three slots for each image.

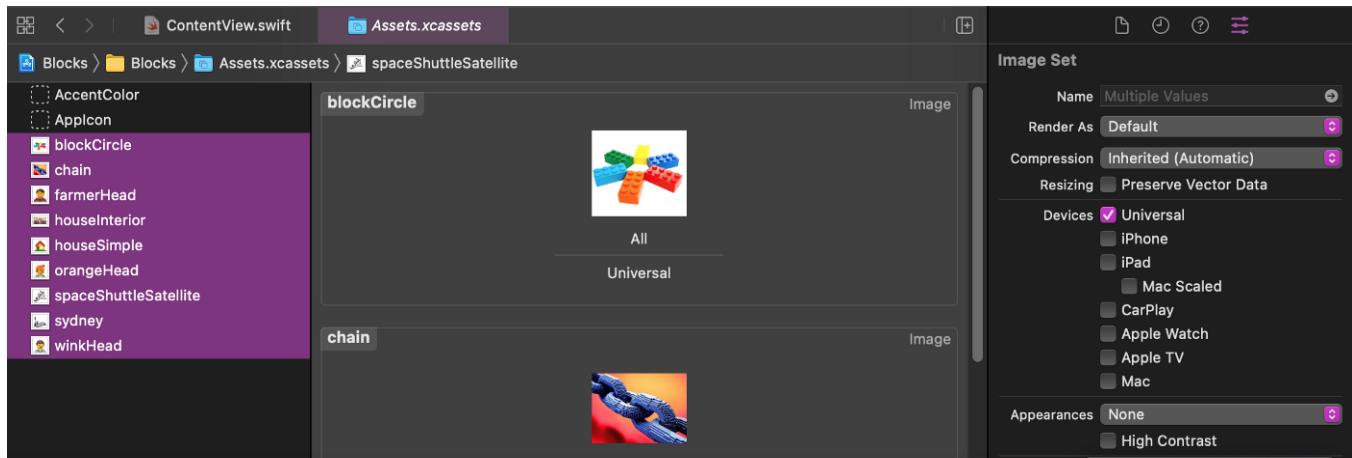


The iPhone, iPad (and Mac) have a screen with one of several resolutions. The original standard screens have one pixel per point. Later devices with “Retina” screens have $2 \times 2 = 4$ pixels per point. Some, such as the iPhone X, have $3 \times 3 = 9$ pixels per point. The asset catalog has placeholders for each of these image resolutions, labeled as `1x`, `2x` and `3x`.

In this case, we only provided one image file for each image asset, which Xcode has placed in the `1x` slot. Since there is no specific version of this image for the other resolutions, Xcode will scale the main image file as needed.

Let’s explicitly tell Xcode to scale each of these images for any needed size.

👉 With all the photos still selected in the assets catalog: in the Attributes inspector, change the `Scales` from `Individual Scales` to `Single Scale`.



[Open in app](#)

As a general guide, for an image that was created as a drawing (such as using Sketch, Figma, or Illustrator apps), it's best to include each resolution (i.e. 1x, 2x, 3x) as its PNG file. For a photo JPEG (such as `farmerHead`), it's usually sufficient to include one version with enough pixel resolution to look good when scaled for different devices.

8. Commit Changes

As you've done before:

1. Choose Commit from the Source Control menu.
 2. Enter a description such as: Added images
 3. Click on the Commit button.
- Notice that this time the commit includes many new files: all of the images files and the “JSON” files that Xcode created behind the scenes for the assets catalog.



[Open in app](#)

The screenshot shows the Xcode Assets.xcassets editor with a hierarchical tree structure:

- Blocks (selected)
- Blocks
 - Assets.xcassets
 - blockCircle.imageset
 - blockCircle.jpg
 - Contents.json
 - chain.imageset
 - chain.jpg
 - Contents.json
 - farmerHead.imageset
 - Contents.json

9. Next...

Our app now has some icons and images. We can see the icons in the tab items, but we haven't yet added the photo images to any of the scenes.

Next, in [Tutorial 7](#), we will start to create our layout for text and images in the cells.

See upcoming tutorials in the [table of contents](#).

!? If you have any questions or comments, please add a response below.



[Open in app](#)

Next Level Swift

Next Level Swift

Next Level Swift aims at sharing knowledge and insights into better programming for iOS and is dedicated to help...

[medium.com](https://medium.com/@nextlevelsdk)

We are always looking for talented and passionate Swift developers! Check out our writer's section and find out how you can share your knowledge with the Next Level Swift Community!

Sign up for Next Level Swift Newsletter

By Next Level Swift

Get all the latest articles, posts and news straight to your mailbox! [Take a look.](#)

[Get this newsletter](#)

Emails will be sent to research2learn@yahoo.co.uk.
[Not you?](#)



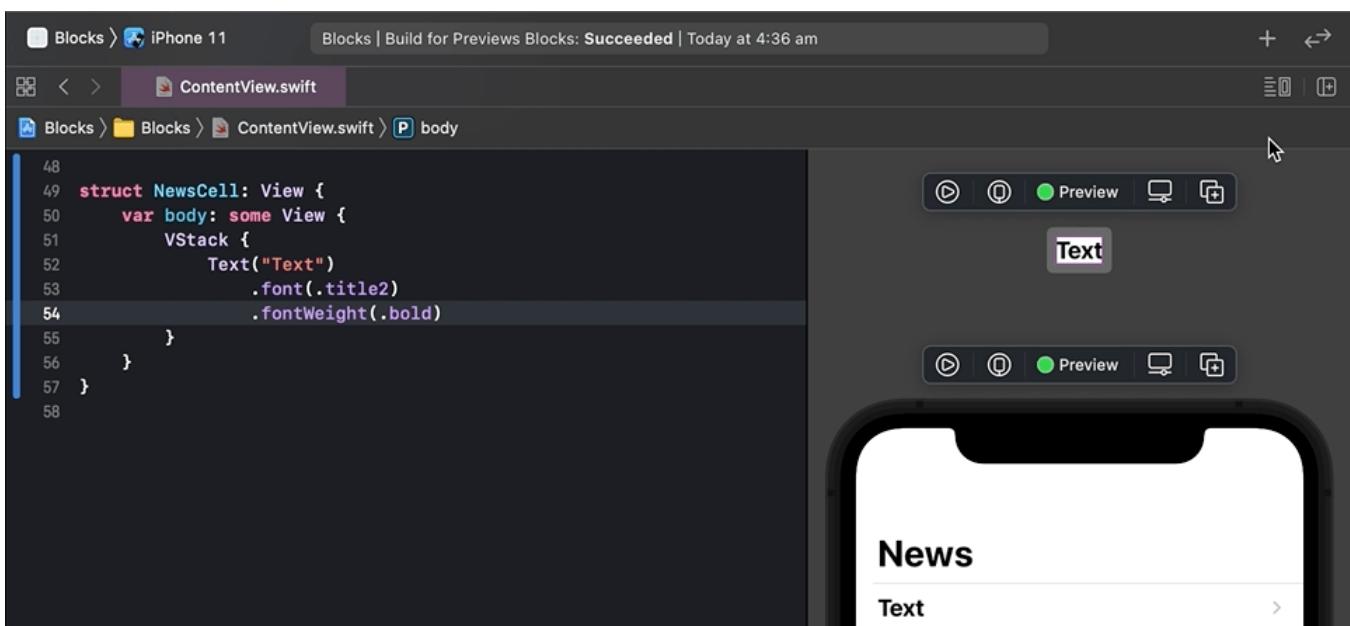
[Open in app](#)Published in Next Level Swift · [Following](#) ▾Tom Brodhurst-Hill · [Following](#)

May 14, 2021 · 8 min read ★

...

Extract aSubview and Preview

Build an App Like Lego, with SwiftUI — Tutorial 7



1. Introduction

Good designers break up an app into components, where each scene is made up of several subviews, which stack together like Lego blocks. Those subviews might be further broken down into smaller subviews. Each component can be shared across multiple parent/super views, scenes, and even different apps. This is commonly referred to as a “design system”.

Read more about building a design system, to be shared between designers and developers, in the article “[Build an App Like Lego — An Intro for Designers](#)”.

Good developers also break up their app code into reusable components. Instead of building “massive views”, they extract code for each reusable subview into its own file,



[Open in app](#)

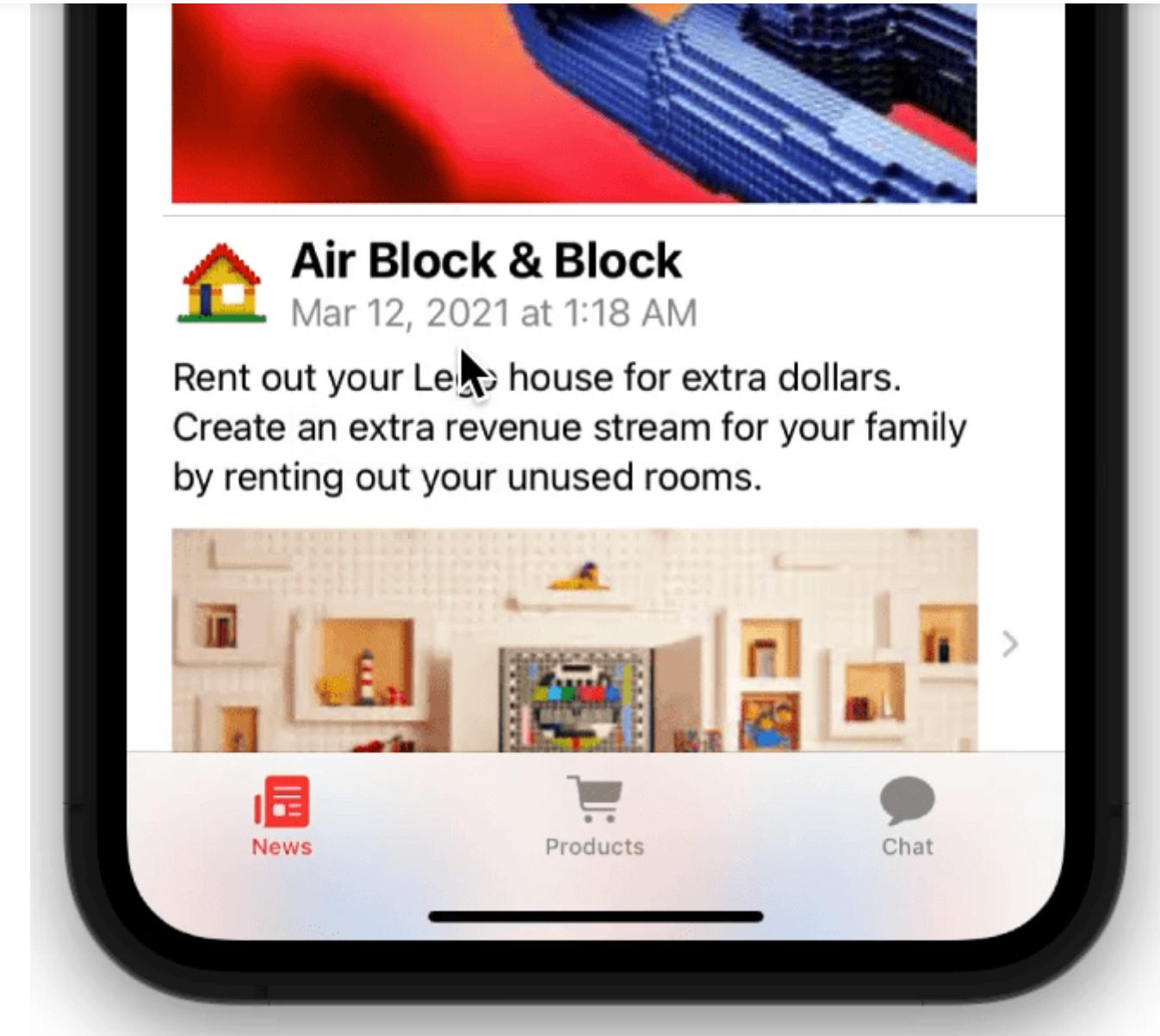
At the end of [Tutorial 6](#) we saw our sample images in the assets catalog. Now we will prepare a cell into which we can add those images (in the next Tutorial 8).

We'll pick up here where the last tutorial left off. Ideally, you have completed the [previous tutorials in this series](#). Or, you can [download](#) the prepared project, ready to start this tutorial.

2. News List Cell

We want to ultimately get the cells in the News list to look like those that we previewed as the end goal in [Tutorial 1](#).

The screenshot shows an iPhone displaying a news article. The top status bar indicates the time is 1:22 and shows signal, Wi-Fi, and battery icons. The main content area has a dark header with the word "News" in large white letters. Below the header is a news card for an article titled "Lego Block Chain". The article features a small icon of colorful lego blocks, the title "Lego Block Chain", and the date "Mar 12, 2021 at 1:18 AM". The text of the article reads: "Add your own Lego block to the chain of transactions to guarantee security by trusting a whole crowd of people you've never met. What can possibly go wrong?". Below the text is a large thumbnail image showing a close-up of blue and red lego blocks. At the bottom of the screen are four navigation icons: a house (Home), a magnifying glass (Search), a bookmark (Bookmark), and a profile (User).

[Open in app](#)

As you can see, the desired layout includes a small image, the title, the date (in gray), some detailed text, and a large detail image. At least one text view is allowed to wrap over multiple lines. The fonts show varied styles. The cell height grows to fit its contents.

To achieve this, we need to create our own custom layout, including several subviews.

Imagined as Lego blocks, it looks something like this:



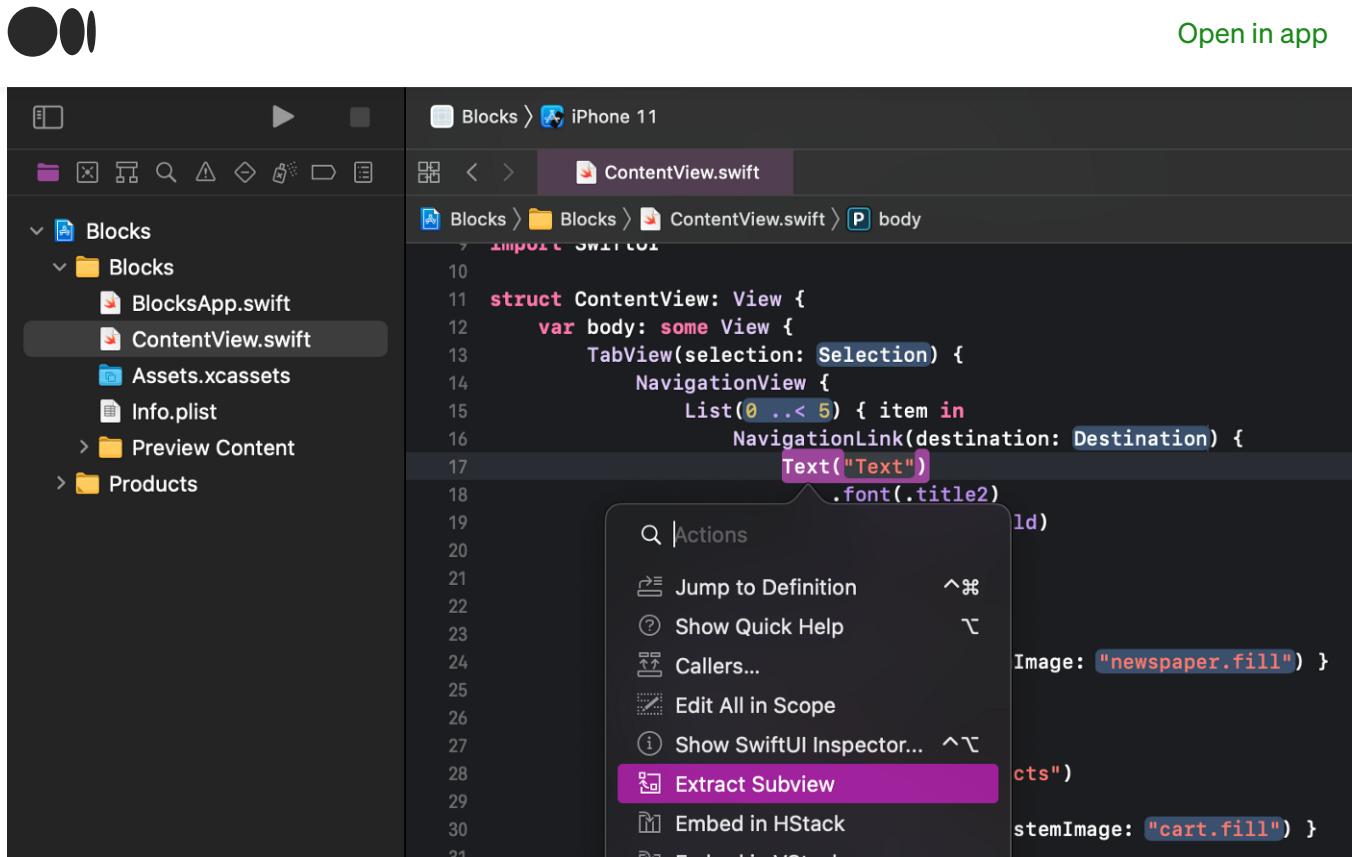
[Open in app](#)

3. Extract a Subview

So far, in this custom cell, we have only created the text title. We need to add several other subviews (text and image views). If we do this in the main `ContentView` code, it will become cluttered. Let's avoid the clutter by first extracting out `Text("Title")` into a new view of its own, where we can add the other subviews.

- 👉 In the Xcode project's File navigator, select the `ContentView.swift` file, if not already selected.
- 👉 In the code, Command click near the start of `Text("Text")` and choose `Extract Subview` from the menu of actions.





💡 Xcode pulls the `Text` and its modifiers out of the main `struct ContentView` code, into a new `struct ExtractedView` code block at the end of the file.

```
11 struct ContentView: View {  
12     var body: some View {  
13         TabView(selection: Selection) {  
14             NavigationView {  
15                 List(0 ..< 5) { item in  
16                     NavigationLink(destination: Destination) {  
17                         ExtractedView()  
18                     }  
19                 }  
20             }  
21         }  
22     }  
23 }
```

```
45 }
46
47 struct ExtractedView: View {
48     var body: some View {
49         Text("Text")
50             .font(.title2)
51             .fontWeight(.bold)
52     }
53 }
```

`ExtractedView` is just a placeholder name.



[Open in app](#)

👉 While the `ExtractedView` text is still highlighted, type `NewsCell` to replace that name, and hit `return` (on the keyboard). You should see it renamed in both locations. If not, undo and repeat the previous step to `Extract Subview`.

```

11 struct ContentView: View {
12     var body: some View {
13         TabView(selection: Selection) {
14             NavigationView {
15                 List(0 ..< 5) { item in
16                     NavigationLink(destination: Destination) {
17                         NewsCell()
18                     }
19                 }
20             }
21         }
22     }
23 }
```

```

43
44
45
46
47 struct NewsCell: View {
48     var body: some View {
49         Text("Text")
50             .font(.title2)
51             .fontWeight(.bold)
52     }
53 }
```

4. Add a Preview

For now, we have both the `ContentView` of our whole app and the `NewsCell` in the same code file. We will eventually create a separate code file for each view. Each SwiftUI view file usually contains its own preview code, which is responsible for generating the preview we've used throughout this tutorial.

The current preview code is fairly simple. It just contains the name of the view that we have been previewing: `ContentView`. The `()` after the name of the view basically just means “make one of these”. So, `ContentView()` means “make a `ContentView` here”.

```

40
41 struct ContentView_Previews: PreviewProvider {
42     static var previews: some View {
43         ContentView()
44     }
45 }
```



[Open in app](#)

👉 In the code, after the `var preview` line, insert a new blank line and start typing `NewsCell`. You should see Xcode autocomplete it and offer `NewsCell` from a popup menu.

The screenshot shows a portion of Xcode's code editor. Line 43 contains the text `News|`. A tooltip on the right says `Cannot find 'News' in scope`. Below the cursor, a completion dropdown menu is open, showing `NewsCell` (selected with a purple background), `LOG_NEWS`, and another `NewsCell`.

```
41 struct ContentView_Previews: PreviewProvider {  
42     static var previews: some View {  
43         News|  
44             S NewsCell  
45             V LOG_NEWS  
46     }  
47 }
```

👉 Complete entering `NewsCell` by either typing or hit `return` to choose it in the menu. Type `()` at the end.

👁️ Check that the preview pane now shows the additional `NewsCell` preview. You may need to click the `Resume` button in the top right of the preview to refresh it.





Open in app

```

11 struct ContentView: View {
12     var body: some View {
13         TabView(selection: Selection) {
14             NavigationView {
15                 List(0 ..< 5) { item in
16                     NavigationLink(destination: Destination) {
17                         NewsCell()
18                     }
19                 }
20                 .navigationTitle("News")
21             }
22             .tabItem { Label("News", systemImage: "newspaper.fill") }
23             .tag(1)
24             NavigationView {
25                 Text("Tab Content 2")
26                 .navigationTitle("Products")
27             }
28             .tabItem { Label("Products", systemImage: "cart.fill") }
29             .tag(2)
30             NavigationView {
31                 Text("Tab Content 3")
32                 .navigationTitle("Chat")
33             }
34             .tabItem { Label("Chat", systemImage: "message.fill") }
35             .tag(3)
36         }
37         .accentColor(.red)
38     }
39 }
40
41 struct ContentView_Previews: PreviewProvider {
42     static var previews: some View {
43         NewsCell()
44         ContentView()
45     }
46 }
47
48 struct NewsCell: View {
49     var body: some View {
50         Text("Text")
51         .font(.title2)
52         .fontWeight(.bold)
53     }
54 }
```



5. Preview Size That Fits

As you can see, the new preview of the `NewsCell` is taking up an entire iPhone screen. Since it is intended to be a repeating cell row in a list, it would be better for the preview to instead show just the size that fits its content.

👉 In the preview code, click in the `NewsCell()` line (if that line is not already selected). In the Attributes inspector change the `Preview Layout` to `Size That Fits`.





Open in app

The screenshot shows the Xcode interface with the code editor displaying `ContentView.swift`. The code defines a `NewsCell` struct with a `body` property containing some text. A preview of the `NewsCell` is shown in the preview pane, which is now smaller due to the added `.previewLayout(.sizeThatFits)`. The preview pane also shows a list of news items.

```

40
41 struct ContentView_Previews: PreviewProvider {
42     static var previews: some View {
43         NewsCell()
44             .previewLayout(.sizeThatFits)
45         ContentView()
46     }
47 }
48
49 struct NewsCell: View {
50     var body: some View {
51         Text("Text")
52             .font(.title2)
53             .fontWeight(.bold)
54     }
55 }
56

```

- As shown above, you should see that Xcode has added a new `.previewLayout(.sizeThatFits)` to the `NewsCell()`. The preview of the `NewsCell` is now much smaller — the minimum size needed to fit its content.

6. VStack

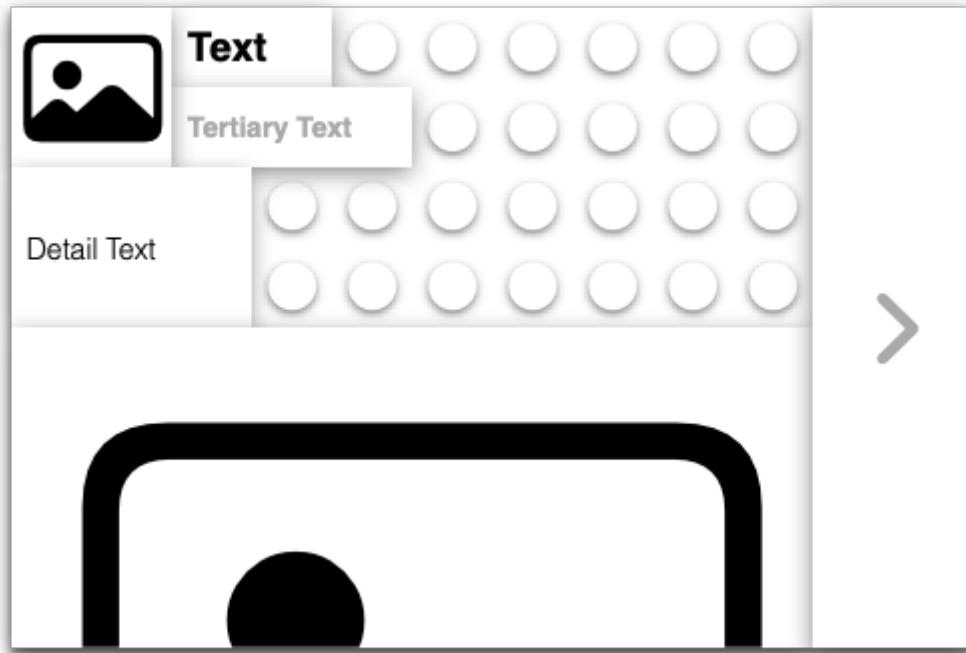
Now we start to really build with blocks, like Lego.

Recall that the aim of this tutorial is to build a custom view (which we've named "NewsCell") that implements a Lego block structure that can display a news article such as:



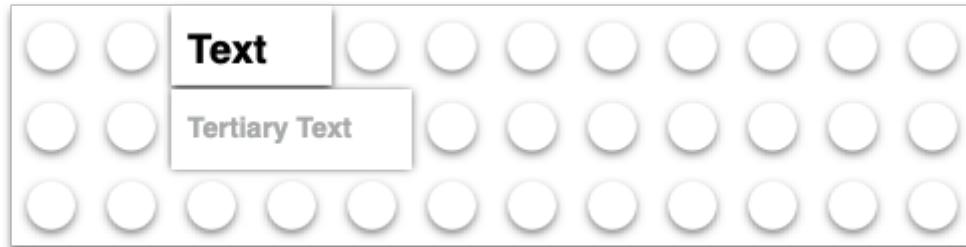
Our `NewsCell` needs to eventually display any text and images that we give it. We



[Open in app](#)

So far, our NewsCell only contains one Text view.

We need to add another text view for the gray wording, under the first Text view, which looks like this in Lego blocks:



To add views together, like this, in a vertical stack, we need to embed them in SwiftUI's "VStack".

👉 In the preview, Command click on the Text . In the popup menu, choose Embed in VStack .

```

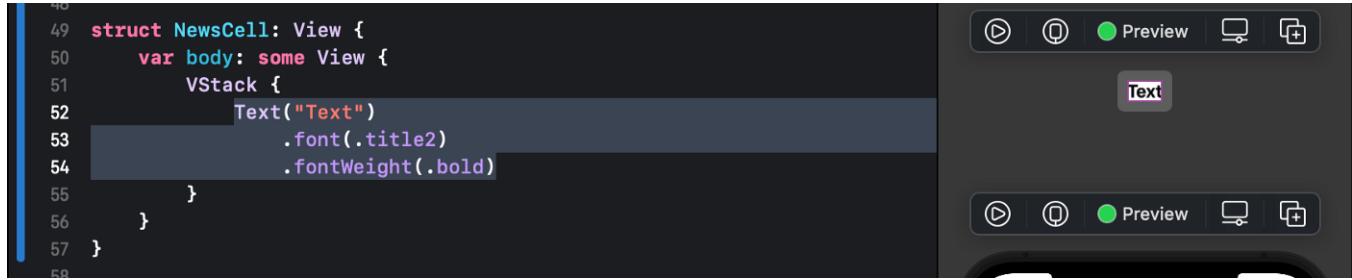
46     }
47 }
48
49 struct NewsCell: View {
50     var body: some View {
51         Text("Text")
52             .font(.title2)

```



[Open in app](#)

- 💡 You can see that the code now has a new `VStack` with the title `Text` inside it.



```

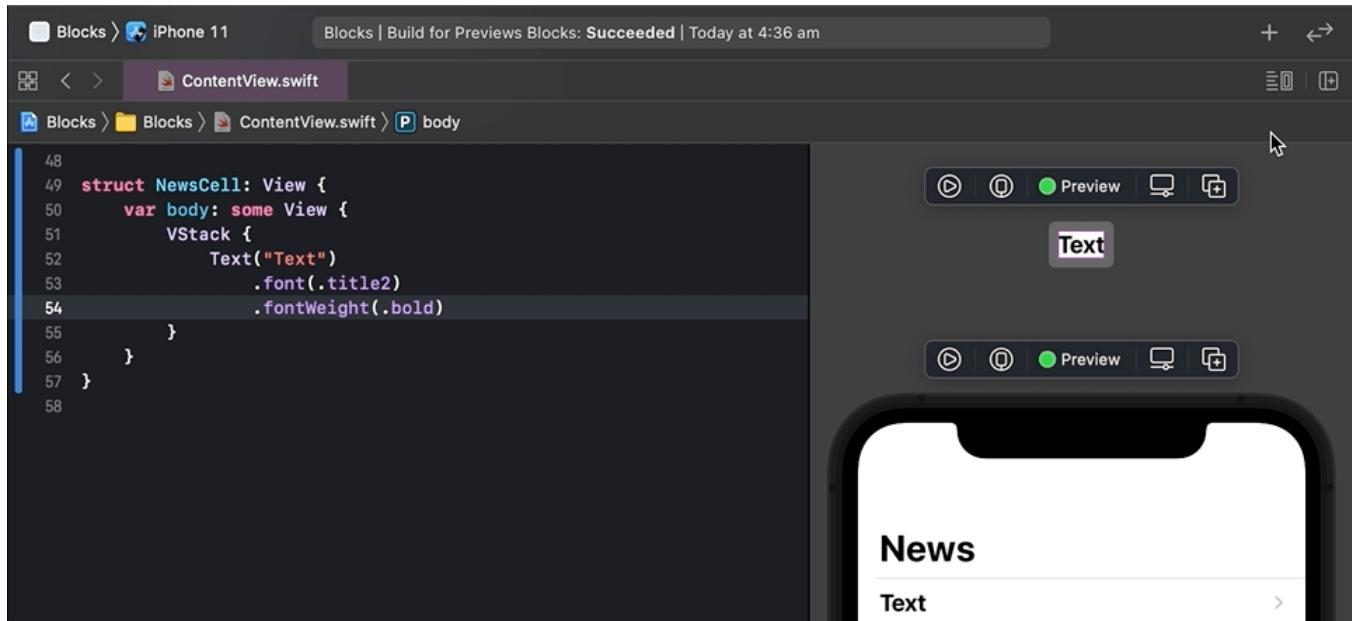
49 struct NewsCell: View {
50     var body: some View {
51         VStack {
52             Text("Text")
53                 .font(.title2)
54                 .fontWeight(.bold)
55         }
56     }
57 }

```

Nothing changed in the preview, since the `VStack` only contains the same original `Text` view.

Now let's add a second `Text` view (for the gray subtitle) to the `VStack`.

👉 As you've done in [earlier tutorials](#), drag the `Text` view from the Library into the preview, directly below the existing title text view. When it's in the right place to release, you will see a colored line below the title text. If it's hard to see, remember that you can zoom in using the magnifying glass buttons in the bottom right of the preview pane, or use a reverse pinch on a trackpad.



```

48
49 struct NewsCell: View {
50     var body: some View {
51         VStack {
52             Text("Text")
53                 .font(.title2)
54                 .fontWeight(.bold)
55         }
56     }
57 }

```

- 💡 In the code, the `VStack` now contains two `Text` subviews.

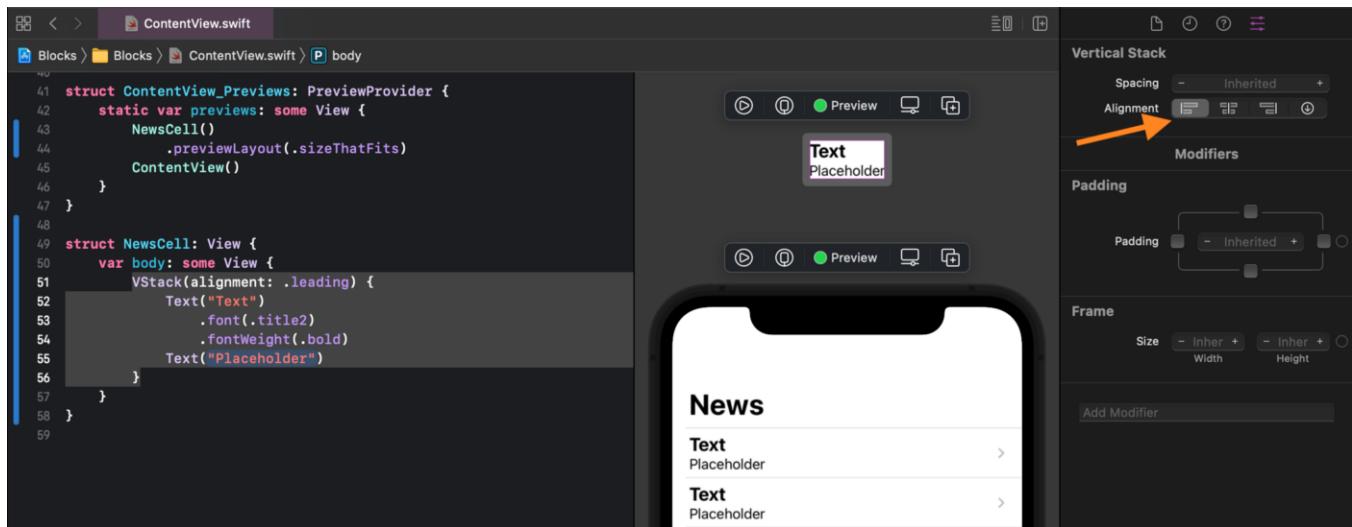
7 VStack Alignment



[Open in app](#)

many other languages, the leading edge is on the right. Fortunately, SwiftUI uses the alignment terms `leading` and `trailing`, so it can adapt to any language orientation.

👉 Check that the `vStack` is still selected in the code. (If not, you can simply click on it.) In the Attributes inspector, under the `Vertical Stack` heading, click on the leading alignment icon.



👁️ The two `Text` views are now aligned on the leading edge of the `vStack`. The first preview shows just one news cell. The second preview shows a list of news cells, each with the updated changes.

8. Text Color

Let's replace the "Placeholder" text, so we can see how it compares with our desired layout.

👉 Click on the `Placeholder` text, in the preview or the code. You might have to click twice. In the Attributes inspector, change the wording from `Placeholder` to `Tertiary Text`. Hit `return` on the keyboard.

👉 Change the `Font Color` to `Gray`.





Open in app

The screenshot shows the Xcode interface. On the left is the code editor with `ContentView.swift` open, displaying the following code:

```

41 struct ContentView_Previews: PreviewProvider {
42     static var previews: some View {
43         NewsCell()
44             .previewLayout(.sizeThatFits)
45         ContentView()
46     }
47 }
48
49 struct NewsCell: View {
50     var body: some View {
51         VStack(alignment: .leading) {
52             Text("Text")
53                 .font(.title2)
54                 .fontWeight(.bold)
55             Text("Tertiary Text")
56                 .foregroundColor(Color.gray)
57         }
58     }
59 }
60

```

On the right, there are two preview windows. The top one shows a small preview of a text element with the text "Text" and "Tertiary Text". The bottom one shows a larger iPhone X preview of the `NewsCell` component, which displays "News" at the top, followed by two text elements: "Text" (in bold, black font) and "Tertiary Text" (in gray font). To the right of the preview is a "Text" inspector panel with various styling options like font, color, and padding.

9. Commit Changes

As you've done before:

1. Choose Commit from the Source Control menu.
2. Enter a description such as: Created NewsCell with custom layout
3. Click on the Commit button.

10. Recap

We created a custom layout for our news cell. We added a vertical stack view, text, and modifiers.

We will complete this layout in [Tutorial 8](#) by adding images and more text.

See upcoming tutorials in the [table of contents](#).

!? If you have any questions or comments, please add a response below.

This series is released via [Next Level Swift](#). Subscribe to keep updated and never miss a new Tutorial of this series!

[Next Level Swift](#)



[Open in app](#)

medium.com

We are always looking for talented and passionate Swift developers! Check out our writer's section and find out how you can share your knowledge with the Next Level Swift Community!

Sign up for Next Level Swift Newsletter

By Next Level Swift

Get all the latest articles, posts and news straight to your mailbox! [Take a look.](#)

[Get this newsletter](#)

Emails will be sent to research2learn@yahoo.co.uk.
[Not you?](#)



[Open in app](#)Published in Next Level Swift · [Following](#)Tom Brodhurst-Hill · [Following](#)

May 19, 2021 · 8 min read

...

Image Views, Resizing, Aspect Ratio

Build an App Like Lego, with SwiftUI — Tutorial 8



1. Introduction

Images, such as photos and drawings, come in various sizes and resolutions. When showing an image in an app, the available rectangular area is usually quite different from that of the image. So, we need to stipulate how the image should be resized.

Should the image stretch and shrink? If the aspect ratio of the image differs from the display area, should it fill the area and crop the image, or fit what it can and leave edge gaps?



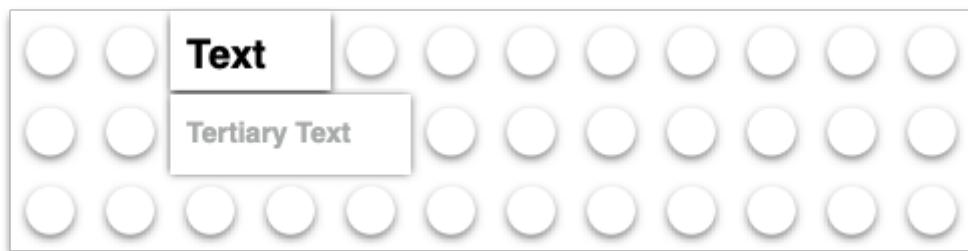
[Open in app](#)

project. In this Tutorial 8 we will add image views to the news cell, to show those image files. We will also add more text views.

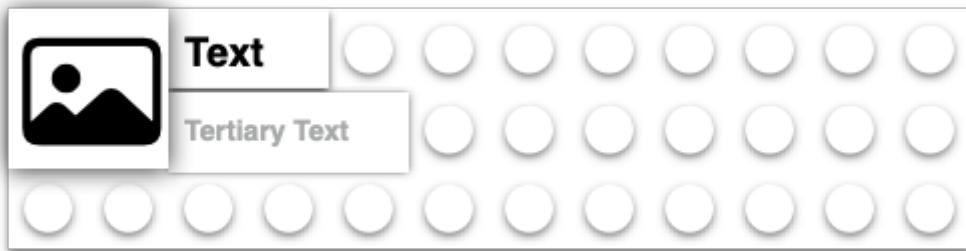
We'll pick up here where the last tutorial left off. Ideally, you have completed the [previous tutorials in this series](#). Or, you can [download](#) the prepared project, ready to start this tutorial.

2. Add an Image in an HStack

Our NewsCell currently contains two text views.



We need to add the small image horizontally next to the existing content.



As you might guess, following our use of the VStack in the previous tutorial, SwiftUI provides an “HStack” view to layout subviews horizontally.

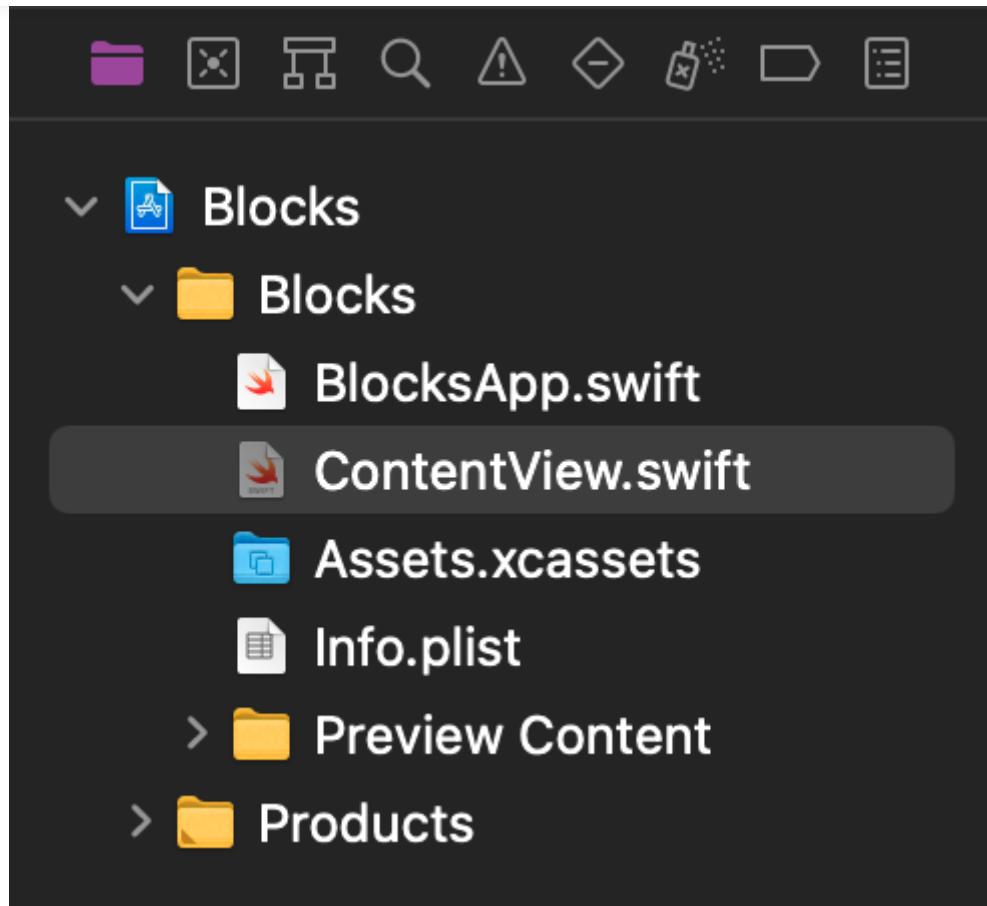
We need to embed our `VStack` (which contains the two `Text` views) in a new `HStack`. Then, we'll add the image to that `HStack`.

👉 If not already open, open the “Blocks” Xcode project and select the `ContentView.swift` file in the File navigator.

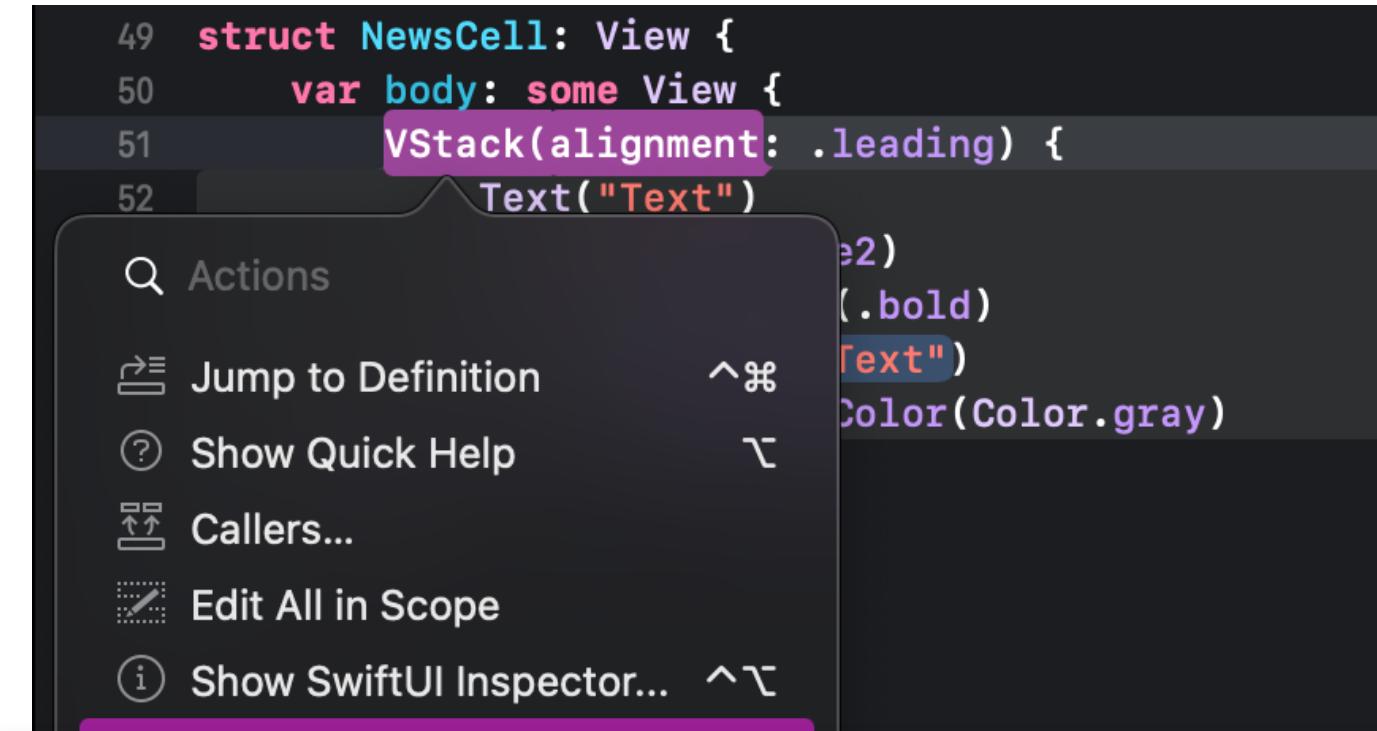




Open in app



👉 Command click on the `VStack` in the code. In the Actions popup menu, choose Embed in HStack .



[Open in app](#)

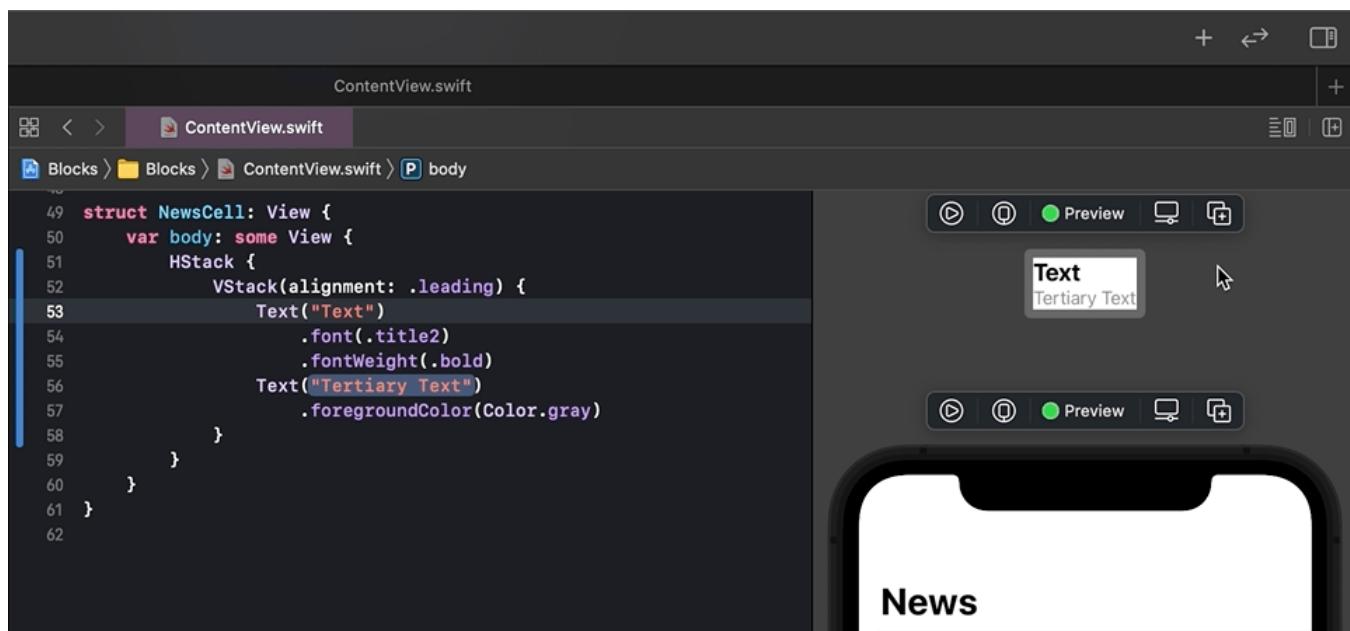
```

49 struct NewsCell: View {
50     var body: some View {
51         HStack {
52             VStack(alignment: .leading) {
53                 Text("Text")
54                     .font(.title2)
55                     .fontWeight(.bold)
56                 Text("Tertiary Text")
57                     .foregroundColor(Color.gray)
58             }
59         }
60     }
61 }
62

```

👉 Drag the `Image` view from the Library into the code, to insert a new line after the `HStack`.

🐞 Previously, we dragged a `Text` view into a `VStack` in the preview. Unfortunately, if we try to drag the image here into the preview, Xcode doesn't let us position it correctly. So, we're dragging it into the code instead.



The screenshot shows the Xcode interface. On the left, the code editor displays the following Swift code:

```

50 struct NewsCell: View {
51     var body: some View {
52         HStack {
53             Image("Image Name")
54                 VStack(alignment: .leading) {
55                     Text("Text")
56                         .font(.title2)
57                         .fontWeight(.bold)
58                     Text("Tertiary Text")
59                         .foregroundColor(Color.gray)
60                 }
61         }
62 }

```

On the right, there are two preview windows. The top one shows a dark gray background with a white rectangular box containing the text "Text" and "Tertiary Text". The bottom one shows a similar view on a simulated iPhone screen.

For now, we will add the system “photo” image as the placeholder. Later we will make this a property that we can change. We can use an SF Symbol icon for this. You might recall in [Tutorial 6](#) that we displayed an SF Symbol as an image in a `Label` in a tab item.

- 👉 In the code, click on the “Image Name” placeholder and type to replace it with `systemName: "photo"`.

The screenshot shows the Xcode interface after the update. The code editor now includes the `systemName: "photo"` line:

```

48 struct NewsCell: View {
49     var body: some View {
50         HStack {
51             Image(systemName: "photo")
52                 VStack(alignment: .leading) {
53                     Text("Text")
54                         .font(.title2)
55                         .fontWeight(.bold)
56                     Text("Tertiary Text")
57                         .foregroundColor(Color.gray)
58                 }
59         }
60     }
61 }

```

The preview window shows the word "Text" with a small photo icon next to it, indicating the placeholder image has been applied.

- 👁️ Check that the photo image appears in the preview. If not, check that you have typed the name exactly right, matching lowercase/uppercase.

The image is too small, compared to our intended layout. We will deal with that shortly.

3. Image Modifiers

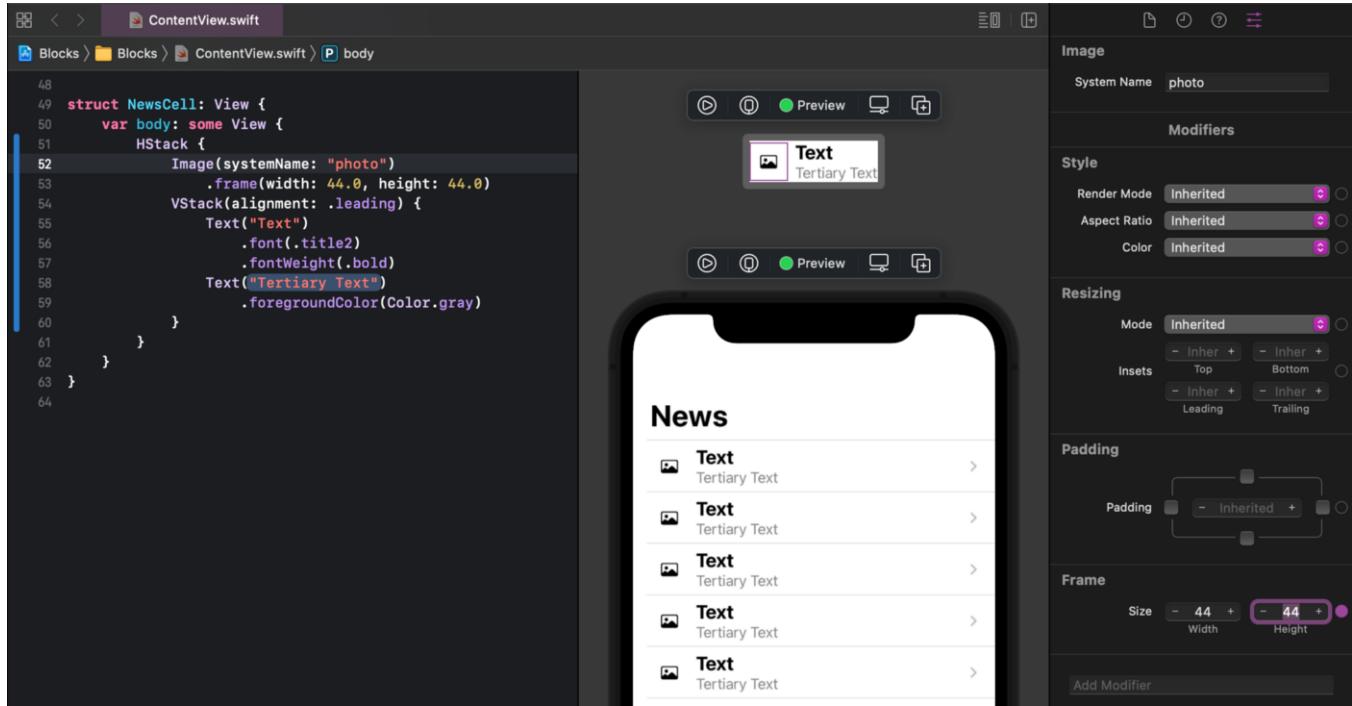
The image is showing at its original size. But we want to stretch (or shrink) it to fit a defined frame, with a consistent width and height.





Open in app

👉 With the `Image` still selected in the code, in the Attributes inspector, enter the `Frame Width` and `Height` as `44`. As usual, hit `return` to tell Xcode to update the preview with the new attributes.



👀 What happened? As you can see by the thin square outline in the left of the `NewsCell` preview, the image's frame has been correctly set to `44 x 44` points. However, the image content has not resized to fit its frame.

We need to:

1. Modify the image to be resizable, so it will stretch or shrink to fit its frame.
2. Modify the image to respect its original aspect ratio, so it doesn't distort when it resizes.

SwiftUI provides modifiers for images called `resizable` and `aspectRatio`. They only apply to images and not all view types.

When we add modifiers to a view, the order of those modifiers matters. For example, after adding the `frame` modifier to our image, that modified view is no longer a pure image. We can't add image modifiers after applying a frame. So, we need to add the

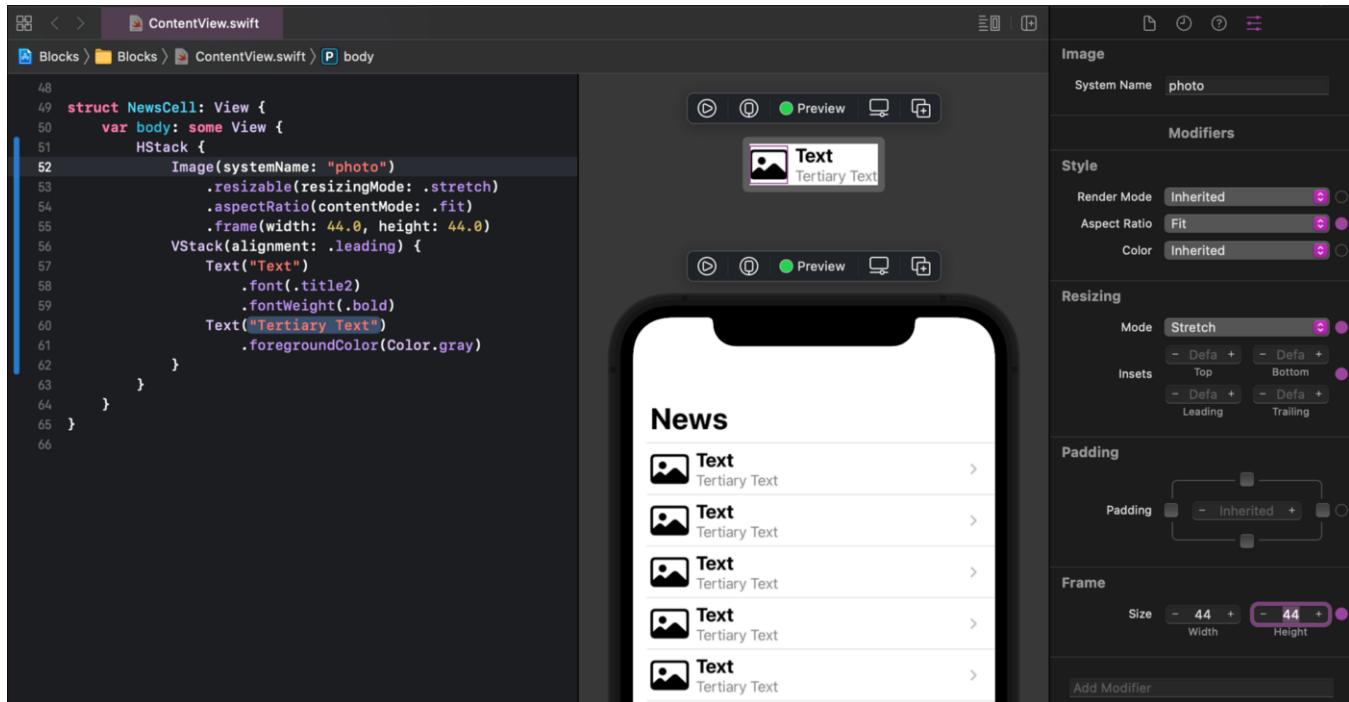




Open in app

update to the latest version. Otherwise, you will need to manually enter the code, without using the Attributes inspector.

👉 In the Attributes inspector, set the Aspect Ratio to Fit . Set the Resizing Mode to Stretch .



👁️ The NewsCell layout is looking pretty good so far. With the `frame` modifier after the image modifiers (`resizable` and `aspectRatio`), the image is sizing correctly for our layout.

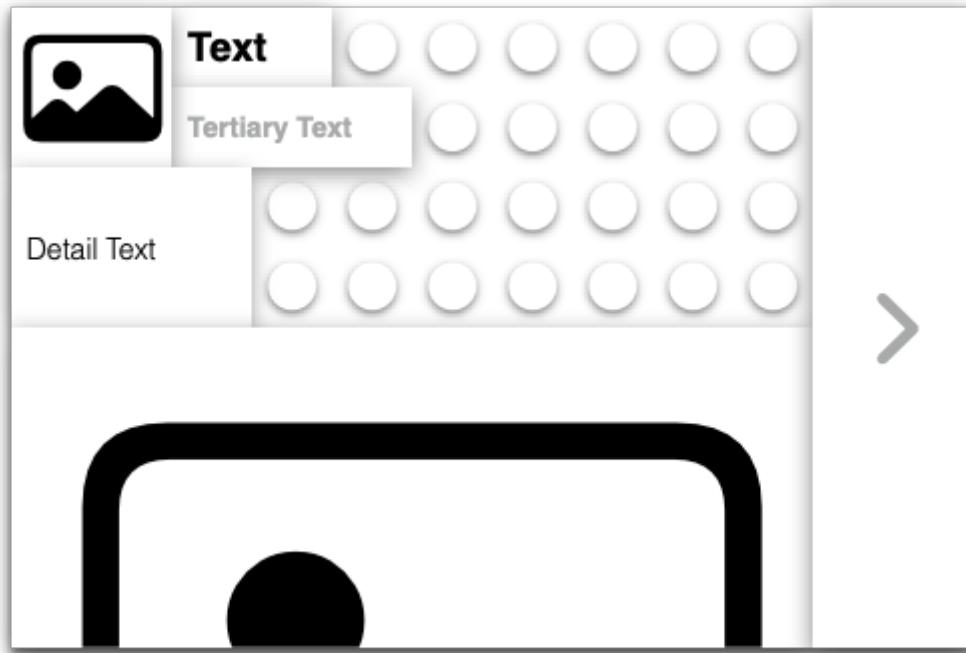
4. Add Another VStack

We're nearly there. We just need to add a detailed text view and a large image, vertically below the blocks (views) that we've positioned so far.





Open in app



Follow the steps below. They are fairly brief since you have already learned the detail in the previous steps. Refer back if you need to.

- 👉 Command click on the `HStack` and choose `Embed in VStack`.
- 👉 From the Library, drag a `Text` view and an `Image` view to insert new lines in the code, just before the closing `}` of the new `VStack`.

The screenshot shows the Xcode interface with the following components:

- Code Editor:** Displays the `NewsCell` struct definition in Swift. A blue vertical bar highlights the `HStack` element. Lines 65 and 66 show the insertion of new code: `Text("Placeholder")` and `Image("Image Name")`.
- Library:** Shows a `Text` view with a placeholder image and the text "Tertiary Text".
- Preview:** Shows a preview of the news cell with the added `Text` and `Image` views.
- Simulator:** Shows a mobile device screen displaying the news cell with the added `Text` and `Image` views.

```

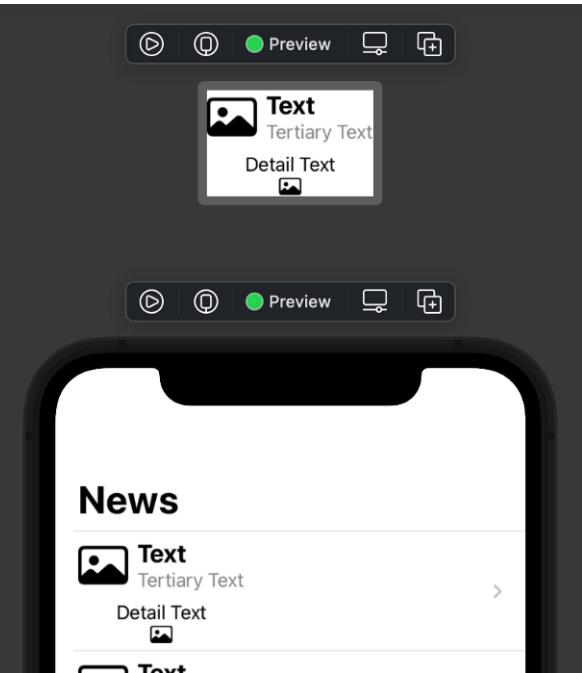
48
49 struct NewsCell: View {
50     var body: some View {
51         VStack {
52             HStack {
53                 Image(systemName: "photo")
54                     .resizable(resizingMode: .stretch)
55                     .aspectRatio(contentMode: .fit)
56                     .frame(width: 44.0, height: 44.0)
57                 VStack(alignment: .leading) {
58                     Text("Text")
59                         .font(.title2)
60                         .fontWeight(.bold)
61                     Text("Tertiary Text")
62                         .foregroundColor(Color.gray)
63                 }
64             }
65             Text("Placeholder")
66             Image("Image Name")
67         }
68     }
69 }
```



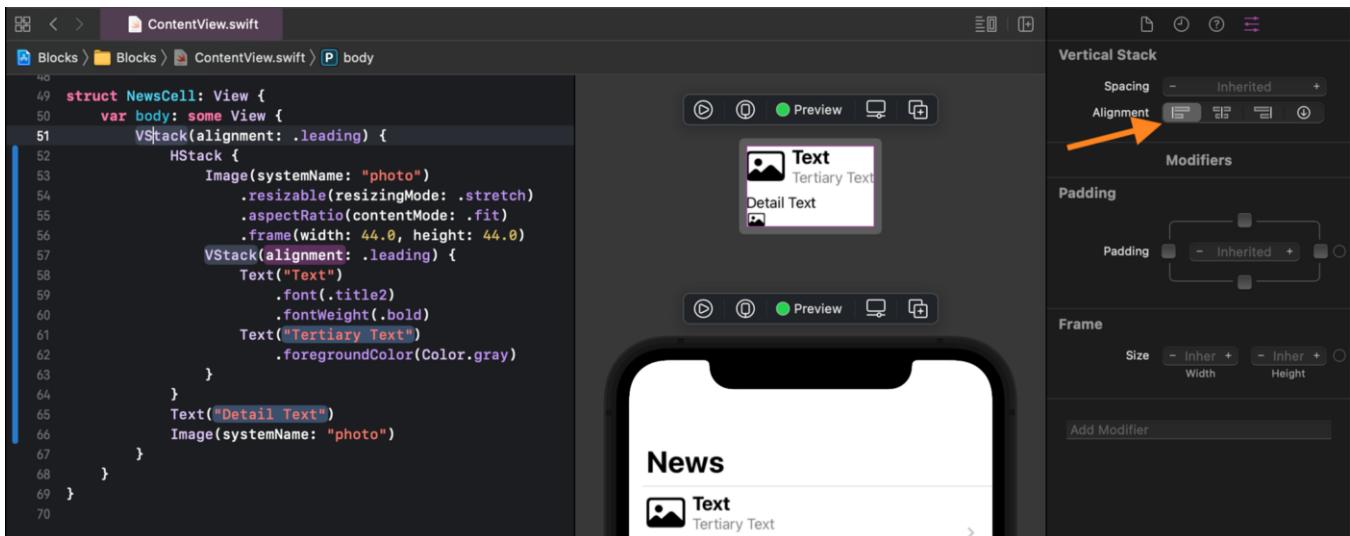

[Open in app](#)

```

46
49 struct NewsCell: View {
50     var body: some View {
51         VStack {
52             HStack {
53                 Image(systemName: "photo")
54                     .resizable(resizingMode: .stretch)
55                     .aspectRatio(contentMode: .fit)
56                     .frame(width: 44.0, height: 44.0)
57             VStack(alignment: .leading) {
58                 Text("Text")
59                     .font(.title2)
60                     .fontWeight(.bold)
61                 Text("Tertiary Text")
62                     .foregroundColor(Color.gray)
63             }
64         }
65     }
66     Text("Detail Text")
67     Image(systemName: "photo")
68 }
69 }
```



👉 In the code, click on the top `VStack`. In the Attributes inspector, set the `Alignment` to `leading`.



👉 Click on the new bottom image in the preview (in the top preview of the news cell, not the bottom preview of the list).

👉 In the Attributes inspector, set the `Aspect Ratio` to `Fill` and the `Resizing Mode` to `Stretch`.




[Open in app](#)

The image stretches to fill all of the available areas.

The image stretches to fill all of the available areas.

Let's limit the image's height, so it won't dwarf the text and the rest of the screen.

Set the image's Frame Height to 240 (and hit return). Leave the Width blank (default).



[Open in app](#)

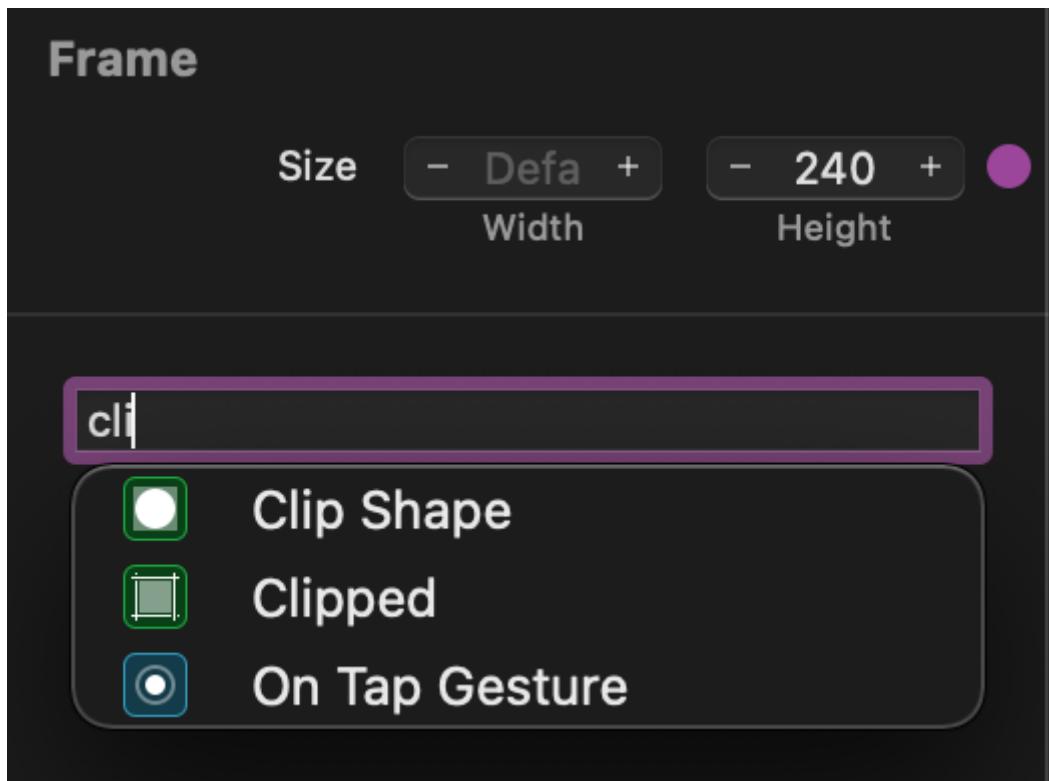
overlapping other views.

We need to clip the image to cut off any part that exceeds its frame. In the Attributes inspector, unfortunately, there is no obvious option to clip the image to its frame.

Fear not — we can add other modifiers just by clicking in the `Add Modifier` field at the bottom of the Attributes inspector.

👉 In the `Add Modifier` field, type `cli`.

🐞 If clicking in the `Add Modifier` the field doesn't show the popup menu, try hitting the down arrow on the keyboard.



👉 Select the `Clipped` modifier from the popup menu.

👁️ The code now has a new `.clipped()` modifier on the second `Image`. In the preview, the image is now clipped to its frame. The Attributes inspector has added a `Clipped` section.




[Open in app](#)

Code Snippet (ContentView.swift):

```

49 struct NewsCell: View {
50     var body: some View {
51         VStack(alignment: .leading) {
52             HStack {
53                 Image(systemName: "photo")
54                     .resizable(resizingMode: .stretch)
55                     .aspectRatio(contentMode: .fit)
56                     .frame(width: 44.0, height: 44.0)
57             }
58             VStack(alignment: .leading) {
59                 Text("Text")
60                     .font(.title2)
61                     .fontWeight(.bold)
62                 Text("Tertiary Text")
63                     .foregroundColor(Color.gray)
64             }
65             Text("Detail Text")
66             Image(systemName: "photo")
67                 .resizable(resizingMode: .stretch)
68                 .aspectRatio(contentMode: .fill)
69                 .frame(height: 240.0)
70                 .clipped()
71         }
72     }
73 }

```

Preview and Inspector:

- Image**: System Name: photo
- Modifiers**: None
- Style**: Render Mode: Inherited, Aspect Ratio: Fill, Color: Inherited
- Resizing**: Mode: Stretch, Insets: Top, Bottom, Leading, Trailing
- Padding**: Padding: - Inherited +
- Frame**: Size: Width 240, Height 240
- Clipped**: No Inspectable Parameters

Ideally, we'd like the image to only use the full 240 points as the maximum height, but use less if the image is smaller. Fortunately, `frame` accepts `maxHeight`, for this purpose.

👉 In the code, in the `frame(...)`, change `height` to `maxHeight`.

Code Snippet (ContentView.swift):

```

49 struct NewsCell: View {
50     var body: some View {
51         VStack(alignment: .leading) {
52             HStack {
53                 Image(systemName: "photo")
54                     .resizable(resizingMode: .stretch)
55                     .aspectRatio(contentMode: .fit)
56                     .frame(width: 44.0, height: 44.0)
57             }
58             VStack(alignment: .leading) {
59                 Text("Text")
60                     .font(.title2)
61                     .fontWeight(.bold)
62                 Text("Tertiary Text")
63                     .foregroundColor(Color.gray)
64             }
65             Text("Detail Text")
66             Image(systemName: "photo")
67                 .resizable(resizingMode: .stretch)
68                 .aspectRatio(contentMode: .fill)
69                 .frame(maxHeight: 240.0)
70                 .clipped()
71         }
72     }
73 }

```

Preview and Inspector (updated):

- Image**: System Name: photo
- Modifiers**: None
- Style**: Render Mode: Inherited, Aspect Ratio: Fill, Color: Inherited
- Resizing**: Mode: Stretch, Insets: Top, Bottom, Leading, Trailing
- Padding**: Padding: - Inherited +
- Frame**: Max Height: 240
- Clipped**: No Inspectable Parameters



[Open in app](#)

💡 The final Lego block in our design, the disclosure indicator (right pointing chevron), is automatically added to each of our cells by the List, as you can see in the full scene preview.

5. Commit Changes

As you've done before:

1. ➡ Choose Commit from the Source Control menu.
2. ➡ Enter a description such as: NewsCell: HStack and images
3. ➡ Click on the Commit button.

6. Recap

We created a custom layout for our news cell. We added stack views, images, text and modifiers so that it should layout properly, given any text and image content.

Next, in [Tutorial 9](#), we will move the NewsCell code to its own separate file.

See upcoming tutorials in the [table of contents](#). Follow the author to be notified of more articles.

❗ If you have any questions or comments, please add a response below.

This series is released via [Next Level Swift](#). Subscribe to keep updated and never miss a new Tutorial of this series!

[Next Level Swift](#)

Next Level Swift

Next Level Swift aims at sharing knowledge and insights into better programming for iOS and is dedicated to help...

[medium.com](https://medium.com/nextlevelsdk)



[Open in app](#)

Community!

Sign up for Next Level Swift Newsletter

By Next Level Swift

Get all the latest articles, posts and news straight to your mailbox! [Take a look.](#)

[Get this newsletter](#)

Emails will be sent to research2learn@yahoo.co.uk.

[Not you?](#)



[Open in app](#)Published in Next Level Swift · [Following](#) ▾Tom Brodhurst-Hill · [Following](#)

May 26, 2021 · 4 min read ★

...

Move a View to its Own File

Build an App Like Lego, with SwiftUI — Tutorial 9

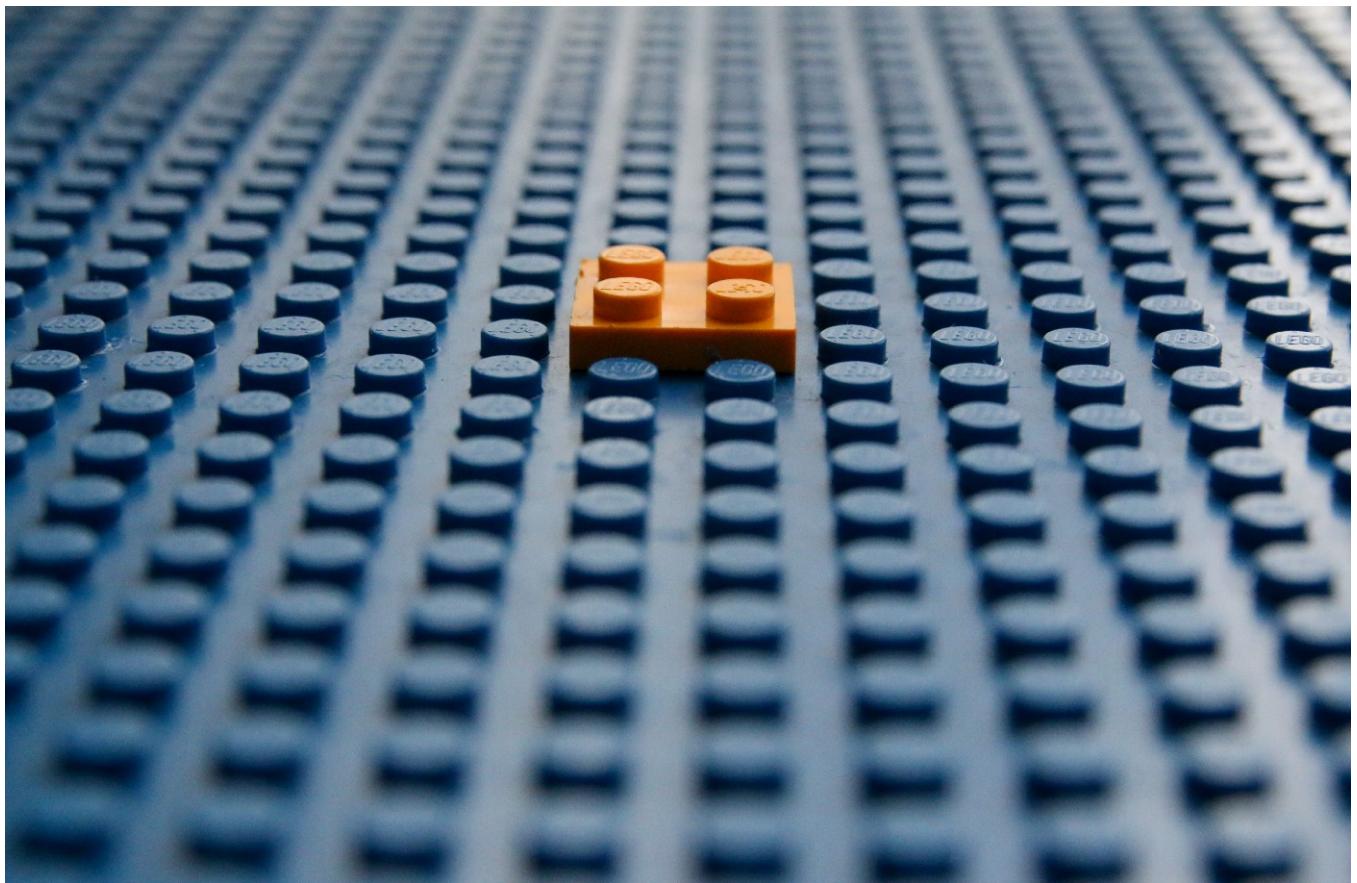


Photo by [Glen Carrie](#) on [Unsplash](#)

1. Introduction

Xcode provides a menu command to `Extract Subview` from a larger view or scene (which we did in [Tutorial 7](#)). But that extracted view still resides in the same file as the original super view. It is best practice to separate each view into its own file.

We need to “refactor” our existing code by “separating the concerns” into separate files.



[Open in app](#)

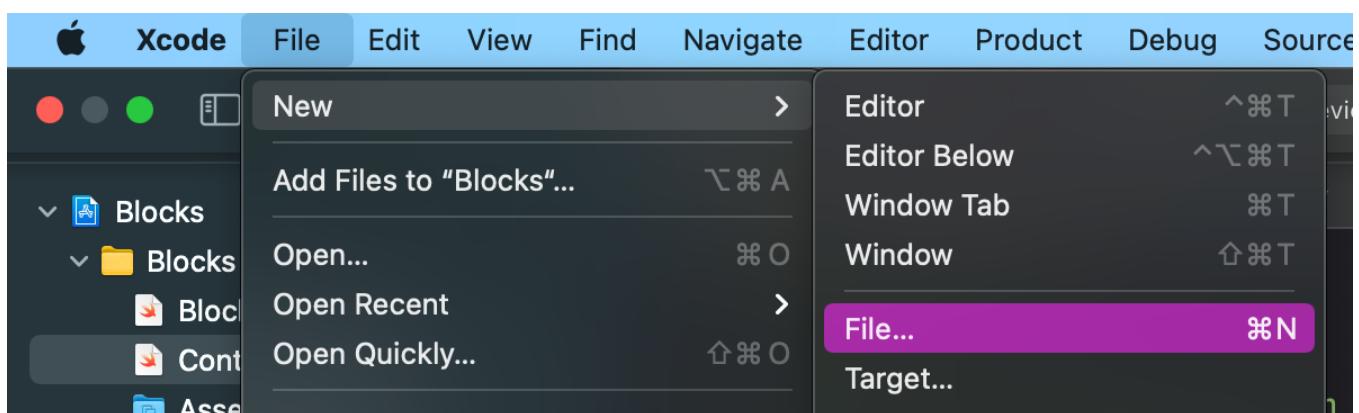
In this short Tutorial 9, we will move the `NewsCell` view into its own `NewsCell.swift` file, out of the parent `ContentView.swift` file.

We'll pick up here where the last tutorial left off. Ideally, you have completed the [previous tutorials in this series](#). Or, you can [download](#) the prepared project, ready to start this tutorial.

2. Move NewsCell to its own file

At the end of the previous [Tutorial 8](#), we had the `ContentView` and `NewsCell` code structures (or “structs”) both in the same `ContentView.swift` file. Let’s move `NewsCell` into its file so that we can maintain them separately.

👉 In Xcode, in the File menu, select. `New > File....`



👉 Make sure that `iOS` is selected in the top bar. Choose the `SwiftUI View` option from the template. Click `Next`.

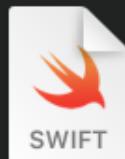


[Open in app](#)

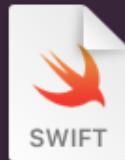
Choose a template for your new file:

[iOS](#)[macOS](#)[watchOS](#)[tvOS](#)

Source

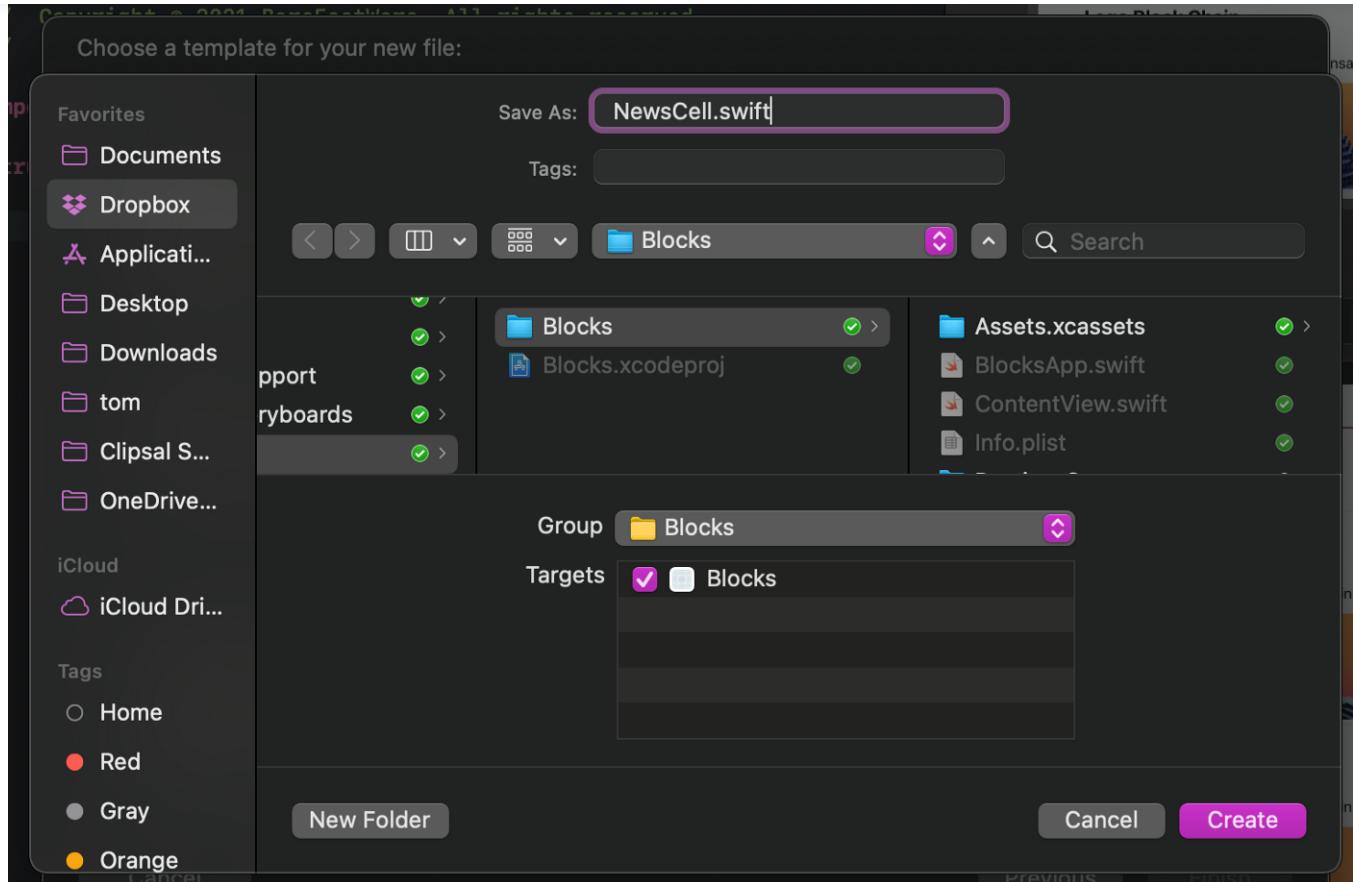
[Swift File](#)[Cocoa Touch Class](#)[Header File](#)[IIG File](#)

User Interface

[SwiftUI View](#)[Storyboard](#)[UITest Case](#)[CR File](#)[View Controller](#)

👉 In the Save As dialog, change the name of the new file from `SwiftUIView.swift` to `NewsCell.swift`. Ensure that the file will be saved into the same folder as the



[Open in app](#)

⌚ Xcode creates new `struct NewsCell` code, with a placeholder `Text` view. Note that Xcode complains that now there is an `Invalid redeclaration of 'NewsCell'` since we have previously created `struct NewsCell` in the `ContentView.swift` file.

```

1 // 
2 //  NewsCell.swift
3 //  Blocks
4 // 
5 //  Created by Tom Brodhurst-Hill on 1/3/21.
6 //  Copyright © 2021 BareFetWare. All rights reserved.
7 // 
8 
9 import SwiftUI
10
11 struct NewsCell: View { ✖ Invalid redeclaration of 'NewsCell'
12     var body: some View {
13         Text("Hello, World!")
14     }
15 }
16
17 struct NewsCell_Previews: PreviewProvider {
18     static var previews: some View {
19         NewsCell()
20     }
21 }

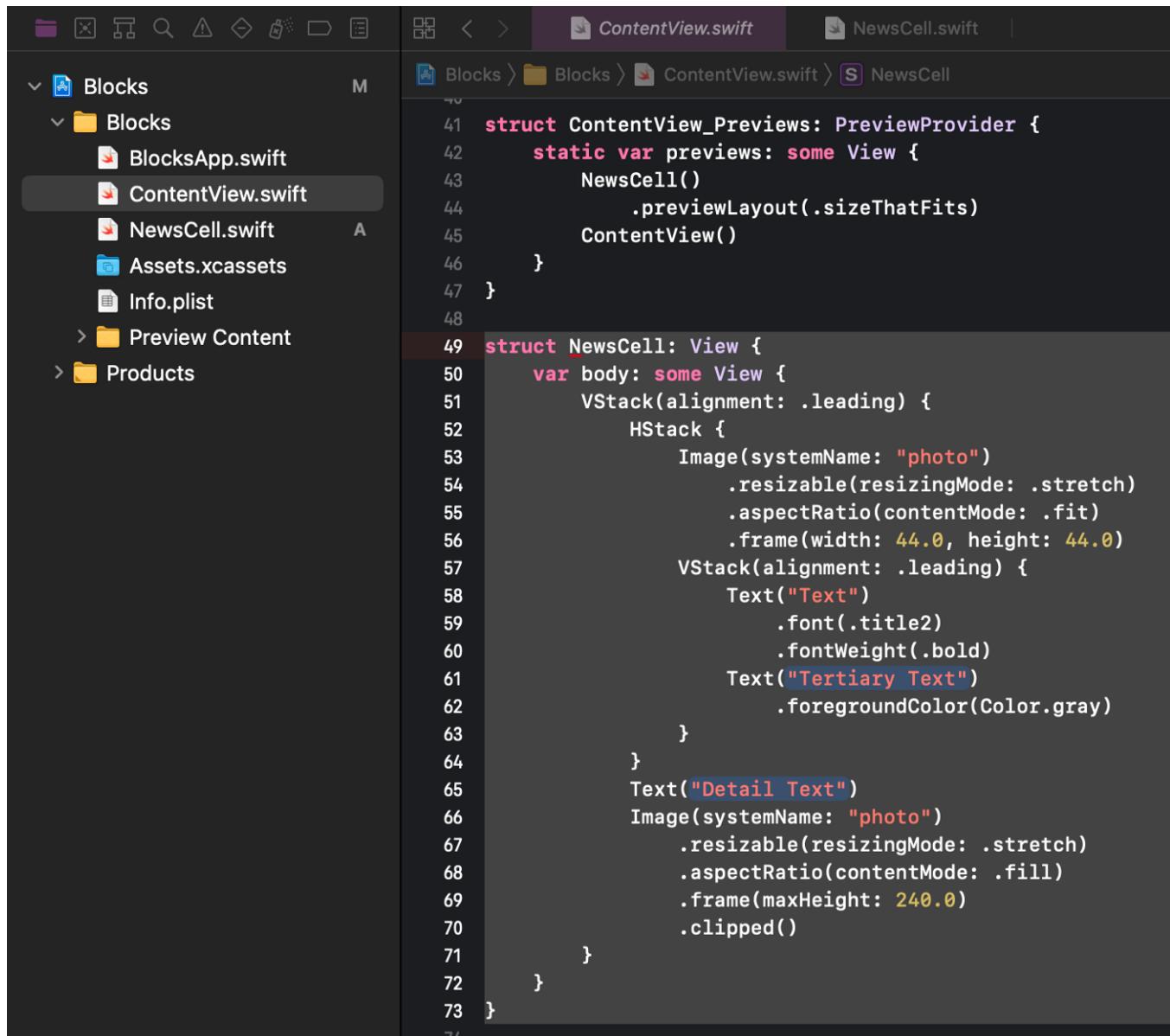
```



[Open in app](#)

Let's move the `struct NewsCell` code from the `ContentView.swift` file, to this new dedicated `NewsCell.swift` file.

- 👉 In the Project navigator (on the left), click on the `ContentView.swift` file. Select all of the code in the `struct NewsCell` block of code.



The screenshot shows the Xcode interface. The Project navigator on the left has a tree view with 'Blocks' expanded, showing 'BlocksApp.swift', 'ContentView.swift' (which is selected and highlighted in blue), 'NewsCell.swift', 'Assets.xcassets', and 'Info.plist'. Below 'Blocks' is a 'Preview Content' section and a 'Products' section. The main editor area shows the `ContentView.swift` file. The code is as follows:

```

41 struct ContentView_Previews: PreviewProvider {
42     static var previews: some View {
43         NewsCell()
44             .previewLayout(.sizeThatFits)
45         ContentView()
46     }
47 }
48
49 struct NewsCell: View {
50     var body: some View {
51         VStack(alignment: .leading) {
52             HStack {
53                 Image(systemName: "photo")
54                     .resizable(resizingMode: .stretch)
55                     .aspectRatio(contentMode: .fit)
56                     .frame(width: 44.0, height: 44.0)
57             VStack(alignment: .leading) {
58                 Text("Text")
59                     .font(.title2)
60                     .fontWeight(.bold)
61                 Text("Tertiary Text")
62                     .foregroundColor(Color.gray)
63             }
64         }
65         Text("Detail Text")
66         Image(systemName: "photo")
67             .resizable(resizingMode: .stretch)
68             .aspectRatio(contentMode: .fill)
69             .frame(maxHeight: 240.0)
70             .clipped()
71     }
72 }
73 }
74

```

- 👉 Cut the `struct NewsCell` code (such as by using the `Edit` menu's `Cut` command).
- 👉 Select the `NewsCell.swift` file (in the Project navigator). Select the placeholder `struct NewsCell` code.





Open in app

The screenshot shows the Xcode interface with the file structure on the left and the code editor on the right.

File Structure:

- Blocks

 - BlocksApp.swift
 - ContentView.swift
 - NewsCell.swift (selected)
 - Assets.xcassets
 - Info.plist

- Preview Content
- Products

Code Editor (NewsCell.swift):

```
1 //  
2 //  NewsCell.swift  
3 //  Blocks  
4 //  
5 //  Created by Tom Brodhurst-Hill on 9/3/21.  
6 //  Copyright © 2021 BareFetWare. All rights reserved.  
7 //  
8  
9 import SwiftUI  
10  
11 struct NewsCell: View {  
12     var body: some View {  
13         Text("Hello, World!")  
14     }  
15 }  
16  
17 struct NewsCell_Previews: PreviewProvider {  
18     static var previews: some View {  
19         NewsCell()  
20     }  
21 }  
22
```

👉 Paste the `struct NewsCell` code over the template code.





Open in app

The screenshot shows the Xcode interface with the project 'Blocks' open. The left sidebar shows files like BlocksApp.swift, ContentView.swift, and NewsCell.swift. The main editor window displays the code for NewsCell.swift:

```

1 // 
2 //  NewsCell.swift
3 //  Blocks
4 //
5 //  Created by Tom Brodhurst-Hill on 9/3/21.
6 //  Copyright © 2021 BareFeetWare. All rights reserved.
7 //
8
9 import SwiftUI
10
11 struct NewsCell: View {
12     var body: some View {
13         VStack(alignment: .leading) {
14             HStack {
15                 Image(systemName: "photo")
16                     .resizable(resizingMode: .stretch)
17                     .aspectRatio(contentMode: .fit)
18                     .frame(width: 44.0, height: 44.0)
19             }
20             VStack(alignment: .leading) {
21                 Text("Text")
22                     .font(.title2)
23                     .fontWeight(.bold)
24                 Text("Tertiary Text")
25                     .foregroundColor(Color.gray)
26             }
27             Text("Detail Text")
28             Image(systemName: "photo")
29                 .resizable(resizingMode: .stretch)
30                 .aspectRatio(contentMode: .fill)
31                 .frame(maxHeight: 240.0)
32                 .clipped()
33         }
34     }
35 }
36
37 struct NewsCell_Previews: PreviewProvider {
38     static var previews: some View {
39         NewsCell()
40     }
41 }
42

```

3. Move the preview code

Our original `ContentView.swift` file and the new `NewsCell.swift` file both contain a preview of the `NewsCell`. We should move the preview to only be in the same file as our moved `NewsCell` code.

👉 Back in `ContentView.swift`, select the `NewsCell()` in the preview, including the `.previewLayout(.sizeThatFits)`.





Open in app

```

Blocks < > | NewsCell.swift | ContentView.swift
Blocks > Blocks > ContentView.swift > P previews
37     .accentColor(.red)
38 }
39 }
40
41 struct ContentView_Previews: PreviewProvider {
42     static var previews: some View {
43         NewsCell()
44             .previewLayout(.sizeThatFits)
45         ContentView()
46     }
47 }
48

```

👉 Cut it (such as by choosing Cut from the Edit menu).

👉 Switch to NewsCell.swift . Select the NewsCell() in the preview. Paste to replace it.

```

Blocks < > | NewsCell.swift | ContentView.swift
Blocks > Blocks > NewsCell.swift > P previews
32     .clipped()
33 }
34 }
35 }
36
37 struct NewsCell_Previews: PreviewProvider {
38     static var previews: some View {
39         NewsCell()
40             .previewLayout(.sizeThatFits)
41     }
42 }
43

```

👁️ Xcode shows the preview of the NewsCell in the NewsCell.swift file and no longer in the ContentView.swift file. You might need to refresh using the Resume button in the preview.

👉 Run the app to confirm that there are no errors.

4. Commit changes

As you've done before:



[Open in app](#)

3. Click on the `Commit` button.

5. Next...

Our `NewsCell` code is now in its own `NewsCell.swift` file. We will create a separate file when we start to build each new view instead of exacting it.

Next, in [Tutorial 10](#), we will extract the text and image properties to create different instances of the `NewsCell`.

See upcoming tutorials in the [table of contents](#). Follow the author to be notified of more articles.

If you have any questions or comments, please add a response below.

This series is released via [Next Level Swift](#). Subscribe to keep updated and never miss a new Tutorial of this series!

Next Level Swift

Next Level Swift

Next Level Swift aims at sharing knowledge and insights into better programming for iOS and is dedicated to help...

[medium.com](https://medium.com/next-level-swift)

We are always looking for talented and passionate Swift developers! Check out our writer's section and find out how you can share your knowledge with the Next Level Swift Community!



[Open in app](#)

Sign up for Next Level Swift Newsletter

By Next Level Swift

Get all the latest articles, posts and news straight to your mailbox! [Take a look.](#)

[Get this newsletter](#)

Emails will be sent to research2learn@yahoo.co.uk.
[Not you?](#)



[Open in app](#)Published in Next Level Swift · [Following](#) ▾Tom Brodhurst-Hill · [Following](#)

Jun 2, 2021 · 8 min read ★

...

Define Properties for a View

Build an App Like Lego, with SwiftUI — Tutorial 10



Photo by [Adyant Pankaj](#) on [Unsplash](#)

1. Introduction

A component is most flexible when we declare “properties” to customize differently for each instance.

For a view component, the properties are best declared as subviews, such as an image or text, or as a view attribute such as color. We usually set a property when the instance



[Open in app](#)

In the previous [Tutorial 9](#), we moved the `NewsCell` code into its own file. It had static text and images, but they were the same for every cell.

In this Tutorial 10, we will add some properties to our code, so that each instance of the news cell can contain different images and text.

We'll pick up here where the last tutorial left off. Ideally, you have completed the [previous tutorials in this series](#). Or, you can [download](#) the prepared project, ready to start this tutorial.

2. Add properties

In our `struct NewsCell` code we have three `Text` subviews and two `Image` subviews. Currently, they are “hardcoded” with placeholder values like `"Detail Text"` and `systemName: "photo"`. Let’s refactor the code to bring out these five subviews as properties of the `NewsCell`.

- 👉 Select the `NewsCell.swift` file, in the File navigator.
- 👉 Insert the following `let` lines of code between the `struct NewsCell` line and the `var body`.



[Open in app](#)

👁️ It should look like this in Xcode:

```
11 struct NewsCell: View {  
12  
13     let image: Image  
14     let text: Text  
15     let detailText: Text  
16     let tertiaryText: Text  
17     let largeImage: Image  
18  
19     var body: some View {  
20         VStack(alignment: .leading) {  
21             HStack {
```

You don't need to worry about the mechanics of these lines of code. But here's a crash course on the principles involved here:

1. `let` declares that each property will be assigned once and then not change.
Technically, `let` declares this property as a “constant”. Alternatively, we would use `var` if we needed to vary (change) a property after it was first created.
2. The word after `let` is the name of the property. We can use any name we like, that



[Open in app](#)

text” is named `detailText`. This is referred to as “camel case” style since it has a hump (or humps) of upper case letters in the middle of the name.

3. The colon (:) means “this property (on the left) is that type (on the right)”. For example, `detailText: Text` means that the `detailText` property is a `Text` view.
4. In Swift, it is standard practice to use a suffix for the variable name that matches the type. For example, `largeImage` is an `Image`.
5. Since this `NewsCell` is a view, it should only be concerned about the subviews it contains and not the data “model” that populates it. For example, the second line of text, in gray, is currently showing the date. But it could contain any text in the future. That’s why we name the text views according to visual prominence (such as `tertiaryText`) rather than the data model (such as `dateText`).

3. Parameters and arguments

⌚ Xcode shows an error on the preview, complaining that the `NewsCell()` instance is missing arguments (values) for parameters `image`, `text` etc.

```
43
44 struct NewsCell_Previews: PreviewProvider {
45     static var previews: some View {
46         NewsCell() // ⚠️ Missing arguments for parameters 'image', 'text', 'detailText', 'tertiaryText', 'largeImage' in call
47             .previewLayout(.sizeThatFits)
48     }
49 }
```

Let’s pause to explain some buzzword technical terms.

We have created a `NewsCell` “type”. Specifically, this one is a `struct` (structure), with properties like `text` and `image`. A type is just the blueprint plan. To create an “instance” (a single occurrence) of that type, we need to provide any required properties in the parentheses of an instance. Each property in the parentheses appears with a name and value. The name is called the “parameter” and the value is called the “argument”, but these terms are often interchanged.

For example, consider this `NewsCell` instance:



[Open in app](#)

This `NewsCell` instance has five parameters. The first parameter is `image` with the argument `Image(systemName: "photo")`.

Now, back to our code.

We need to provide values for the properties in the `NewsCell()` instance in the preview.

👉 Delete the `()` after `NewsCell`. Re-type the opening `(`. Xcode should offer to autocomplete with all the new parameters and placeholders for the arguments (values).

```
43
44 struct NewsCell_Previews: PreviewProvider {
45     static var previews: some View {
```



[Open in app](#)

- 👉 Choose that autocomplete option, by hitting `Return` or double-clicking in the popup menu.

```
43
44 struct NewsCell_Previews: PreviewProvider {
45     static var previews: some View {
46         NewsCell(image: Image, text: Text, detailText: Text, tertiaryText: Text, largeImage: Image)
47             .previewLayout(.sizeThatFits)
48     }
49 }
```

- 👉 Replace the first `Image` argument placeholder with `Image(systemName: "photo")`.

```
43
44 struct NewsCell_Previews: PreviewProvider {
45     static var previews: some View {
46         NewsCell(image: Image(systemName: "photo"), text: Text, detailText: Text, tertiaryText: Text, largeImage: Image)
47             .previewLayout(.sizeThatFits)
48     }
49 }
```

- ⌚ As you can see, the `NewsCell(...)` line of code containing the arguments is getting long and will get longer as we fill in each argument.

Let's wrap each parameter and argument on its own line to make it more readable. Remember that Swift allows us to format code spacing how we like, without affecting the execution of the code.

- 👉 Click just before the first parameter `image` (just after the opening `(`) and hit `Return`. Click just before the `text` parameter and hit `Return`. Repeat for all the parameters.

```
43
44 struct NewsCell_Previews: PreviewProvider {
45     static var previews: some View {
46         NewsCell(
47             image: Image(systemName: "photo"),
48             text: Text,
49             detailText: Text,
50             tertiaryText: Text,
51             largeImage: Image)
52             .previewLayout(.sizeThatFits)
53     }
54 }
```



[Open in app](#)

```

44 struct NewsCell_Previews: PreviewProvider {
45     static var previews: some View {
46         NewsCell(
47             image: Image(systemName: "photo"),
48             text: Text("Text"),
49             detailText: Text("Detail Text"),
50             tertiaryText: Text("Tertiary Text"),
51             largeImage: Image(systemName: "photo"))
52         .previewLayout(.sizeThatFits)
53     }
54 }
```

👉 After the last `Image(...)`, click between the two closing parentheses: `)`). Hit `return` to move the second `)` to a new line.

```

45
44 struct NewsCell_Previews: PreviewProvider {
45     static var previews: some View {
46         NewsCell(
47             image: Image(systemName: "photo"),
48             text: Text("Text"),
49             detailText: Text("Detail Text"),
50             tertiaryText: Text("Tertiary Text"),
51             largeImage: Image(systemName: "photo")
52         )
53         .previewLayout(.sizeThatFits)
54     }
55 }
```

👉 Let's ensure that the code is neatly indented. Select all the code in this file (such as by choosing `Select All` in the `Edit` menu). Hit `Control — i` to re-indent the code (or, in the `Editor` menu, select `Structure > Re-Indent`).

👁️ Xcode moves the `.previewLayout` modifier to line up with the closing `)` in the preview code.

```

46
44 struct NewsCell_Previews: PreviewProvider {
45     static var previews: some View {
46         NewsCell(
47             image: Image(systemName: "photo"),
48             text: Text("Text"),
49             detailText: Text("Detail Text"),
50             tertiaryText: Text("Tertiary Text"),
51             largeImage: Image(systemName: "photo")
52         )
53     }
54 }
```





Open in app

4. Implement the properties

We have defined the properties of our `NewsCell`, but the code inside `var body` doesn't yet actually use them. We need to replace the occurrences of the `Text` and `Image` views in the code with those property names.

👉 In the `var body` code, select the first `Image(systemName: "photo")` code. Type `image` to replace it.

```
11 struct NewsCell: View {  
12  
13     let image: Image  
14     let text: Text  
15     let detailText: Text  
16     let tertiaryText: Text  
17     let largeImage: Image  
18  
19     var body: some View {  
20         VStack(alignment: .leading) {  
21             HStack {  
22                 im|  
23             }  
24         }  
25     }  
26  
27     image: Image  
28  
29     imageScale(_ scale:)  
30  
31     Image  
32  
33     IMP  
34  
35     import - Import Statement  
36  
37     imaxabs(_ j:)  
38  
39     imaxdiv(_ __numer:, _ __denom:)  
40  
41     IMAXBEL  
42  
43     image: Image  
44  
45     Text("Detail Text")  
46 }
```

- As you type, Xcode offers to autocomplete with various options. You can choose the desired property (marked with a **v** for “variable” in the popup menu) — in this case:
`image .`

👉 Similarly, select `Text("Text")`. Replace it with `text`.

👉 Repeat for all of the properties, by replacing the other two `Text` views and one `Image` view with the corresponding property names, as shown below:



[Open in app](#)

```
11 struct NewsCell: View {  
12  
13     let image: Image  
14     let text: Text  
15     let detailText: Text  
16     let tertiaryText: Text  
17     let largeImage: Image  
18  
19     var body: some View {  
20         VStack(alignment: .leading) {  
21             HStack {  
22                 image  
23                     .resizable(resizingMode: .stretch)  
24                     .aspectRatio(contentMode: .fit)  
25                     .frame(width: 44.0, height: 44.0)  
26             VStack(alignment: .leading) {  
27                 text  
28                     .font(.title2)  
29                     .fontWeight(.bold)  
30                 tertiaryText  
31                     .foregroundColor(Color.gray)  
32             }  
33         }  
34         detailText  
35         largeImage  
36             .resizable(resizingMode: .stretch)  
37             .aspectRatio(contentMode: .fill)  
38             .frame(maxHeight: 240.0)  
39             .clipped()  
40         }  
41     }  
42 }
```

NewsCell now requires that all five properties are given a value when we create an instance, using NewsCell().

5. Add properties to the instance in ContentView

👉 In the Product menu, choose Build.





Open in app

```

Blocks | Build for Previews Failed ✘ 1
Content View.swift NewsCell.swift
Blocks > Blocks > NewsCell.swift > body
1 // 
2 //  NewsCell.swift
3 //  Blocks
4 //
5 //  Created by Tom Brodhurst-Hill on 1/3/21.
6 //  Copyright © 2021 BareFetWare. All rights reserved.
7 //
8
9 import SwiftUI
10
11 struct NewsCell: View {

```

👉 Click on the error (stop sign) in the path bar. Xcode shows the file containing the error. Click on `ContentView.swift` in the popup menu.

👁️ As you can see, the `NewsCell()` instance in the `ContentView` now has the error `Missing arguments for parameters`, the same error that we fixed earlier in the `NewsCell` preview.

```

11 struct ContentView: View {
12     var body: some View {
13         TabView(selection: .constant(1)) {
14             NavigationView {
15                 List(0 ..< 5) { item in
16                     NavigationLink(destination: Destination) {
17                         NewsCell() ⚡ Missing arguments for parameters 'image', 'text', 'detailText', 'tertiaryText', 'largeImage' in call
18                     }
19                 }
20             }
21         }
22     }
23 }

```

For now, we'll just fill this `NewsCell()` with the same placeholder data as we had in the preview. In the next tutorial, we will connect it to a data model.

👉 Copy the populated `NewsCell(...)` instance from the preview in `NewsCell.swift` to replace the empty `NewsCell()` instance in `ContentView`.

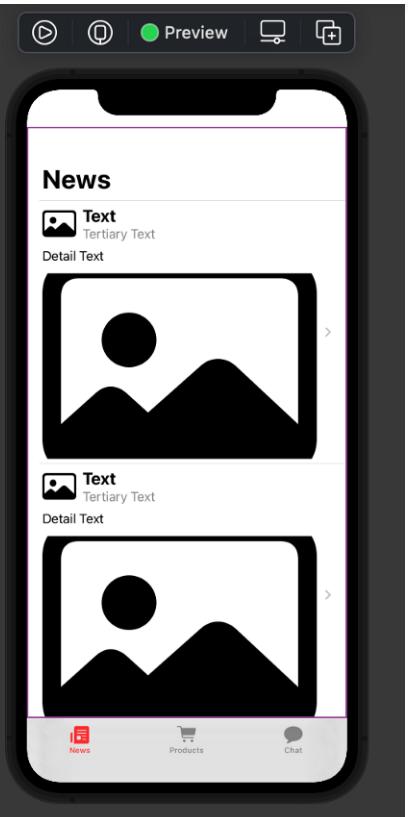




Open in app

```

8
9 import SwiftUI
10
11 struct ContentView: View {
12     var body: some View {
13         TabView(selection: $selection) {
14             NavigationView {
15                 List(0 ..< 5) { item in
16                     NavigationLink(destination: Destination) {
17                         NewsCell(
18                             image: Image(systemName: "photo"),
19                             text: Text("Text"),
20                             detailText: Text("Detail Text"),
21                             tertiaryText: Text("Tertiary Text"),
22                             largeImage: Image(systemName: "photo")
23                         )
24                     }
25                 }
26                 .navigationTitle("News")
27             }
28             .tabItem { Label("News", systemImage: "newspaper.fill") }
29             .tag(1)
30             NavigationView {
31                 Text("Tab Content 2")
32                 .navigationTitle("Products")
33             }
34             .tabItem { Label("Products", systemImage: "cart.fill") }
35             .tag(2)
36             NavigationView {
37                 Text("Tab Content 3")
38                 .navigationTitle("Chat")
39             }
40         }
41     }
42 }
```



- ⌚ Check that the preview shows the new placeholder text and images. You will probably need to refresh the preview using the Try Again or Resume button.

6. Commit changes

As you've done before:

- 👉 Choose Commit from the Source Control menu.
- 👉 Enter a description such as: NewsCell: refactored with properties
- 👉 Click on the Commit button.

7. Next...

Our NewsCell now has properties, so we can give each instance different property values. Currently, however, each instance in our News scene is exactly the same.

Next, in [Tutorial 11](#), we will add a data model with multiple news articles, which we will use in different instances of a NewsCell .



[Open in app](#)

This series is released via [Next Level Swift](#). Subscribe to keep updated and never miss a new Tutorial of this series!

Next Level Swift

Next Level Swift

Next Level Swift aims at sharing knowledge and insights into better programming for iOS and is dedicated to help...

[medium.com](https://medium.com/next-level-swift)

We are always looking for talented and passionate Swift developers! Check out our writer's section and find out how you can share your knowledge with the Next Level Swift Community!

Sign up for Next Level Swift Newsletter

By Next Level Swift

Get all the latest articles, posts and news straight to your mailbox! [Take a look.](#)

[Get this newsletter](#)

Emails will be sent to research2learn@yahoo.co.uk.
[Not you?](#)



[Open in app](#)Published in Next Level Swift · [Following](#) ▾Tom Brodhurst-Hill · [Following](#)

Jun 9, 2021 · 6 min read ★

...

Create a Model and Mock Instances

Build an App Like Lego, with SwiftUI — Tutorial 11



Photo by [Apolo Photographer](#) on [Unsplash](#)

1. Introduction

Broadly speaking, an app has two facets: model and view. A view is something you see or touch or otherwise interact with directly as a human. A model is something behind the scenes that stores data or calculates results.

A view's job is to show visual layout, color, text, images, buttons, tab bar, and other



[Open in app](#)

In this tutorial, we will build a model and mock data.

So far in our app, we have created only views, like `NewsCell`. At the end of the previous [Tutorial 10](#) we refactored the `NewsCell` to expose properties that we can customize to create different `NewsCell` instances.

In this [Tutorial 11](#), we will add a model called `Article` to contain the raw news article information. In the next tutorial, we will display those articles in news cells.

We'll pick up here where the last tutorial left off. Ideally, you have completed the [previous tutorials in this series](#). Or, you can [download](#) the prepared project, ready to start this tutorial.

2. Model Types

Swift stores words and names in a type called a `String`. A `String` is a model, which stores a “string” of characters, such as letters, numbers, spaces, and punctuation, to make a word, sentence, paragraph or longer. In contrast to a `Text` view, a `String` only stores the “string” of characters, not the visual characteristics like font, color, and size.

Swift stores dates and times in the `Date` type.

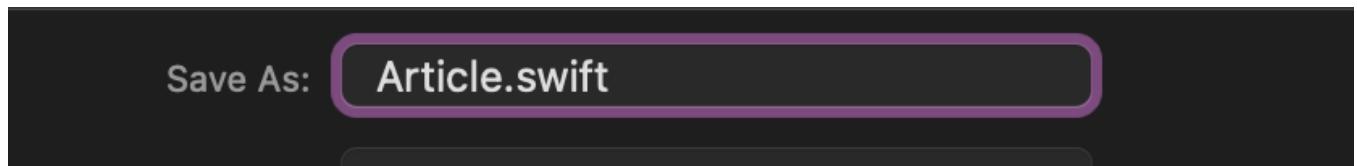
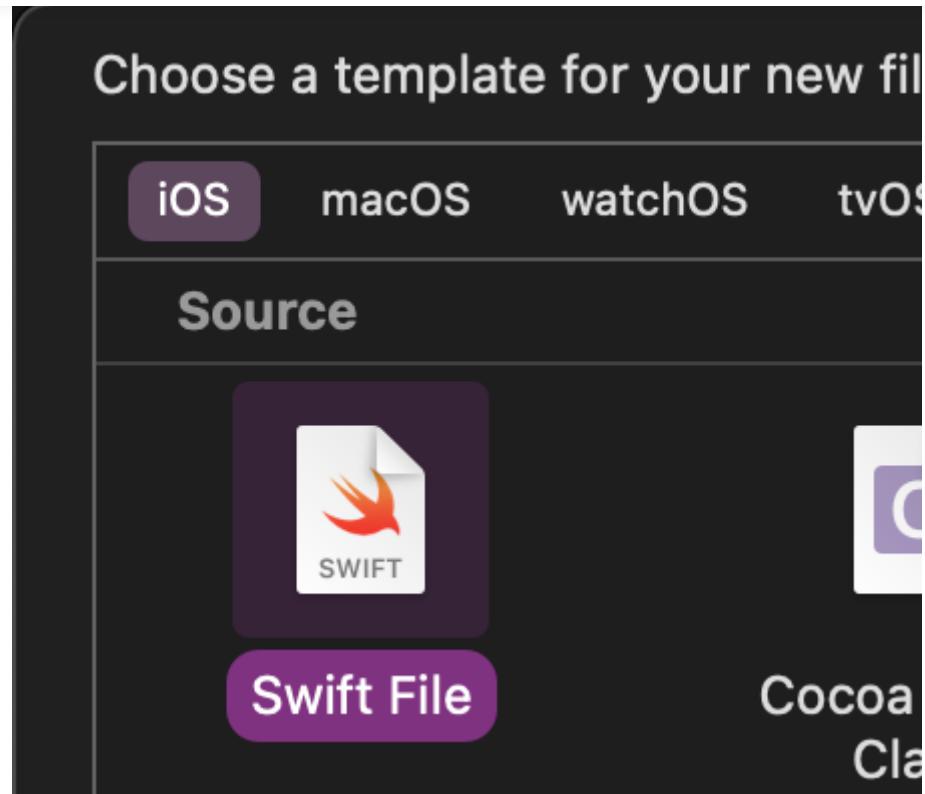
Numbers are usually stored as an `Int` if they are integers (whole numbers like `0`, `1`, `234`, `-78`) or as `Double` if they can have a fractional component (like `0.78`, `1.23` or `-45.6`).

3. Create the Model Code

Each news article will consist of a title, date, detail, and the name of two images. Let's create an `Article` type to store this information for each article.

👉 In Xcode, use the `File` menu's `New File...` command to create a new file called



[Open in app](#)

👉 Type or paste the code below. It's good practice to type in the code to familiarise yourself with Xcode's editing and autocompletion features.



[Open in app](#)

⌚ Your Article code should look like this:

The screenshot shows the Xcode interface with the project navigation bar at the top. The left sidebar shows the project structure under the 'Blocks' folder, including 'BlocksApp.swift', 'ContentView.swift', 'Article.swift' (which is selected), 'NewsCell.swift', 'Assets.xcassets', 'Info.plist', 'Preview Content', and 'Products'. The main editor area displays the 'Article.swift' code:

```
1 //  
2 //  Article.swift  
3 //  Blocks  
4 //  
5 //  Created by Tom Brodhurst-Hill on 5/3/21.  
6 //  Copyright © 2021 BareFeetWare. All rights reserved.  
7 //  
8  
9 import Foundation  
10  
11 struct Article {  
12     let title: String  
13     let date: Date  
14     let detail: String  
15     let smallImageName: String  
16     let largeImageName: String  
17 }
```

That's it! We have created our first model.

ℹ️ For the curious, here's a breakdown explanation of the code above:

1. `import Foundation` imports (into the app project) all of the foundation (fundamental) Swift types, such as dates. If we didn't do this, Xcode's compiler would complain that it *Cannot find type 'Date' in scope*.
2. `struct Article` declares that we are creating a new type (specifically a “structure”)



[Open in app](#)

4. The word following `let` sets the name of each property, such as `title`, `date`, `detail`. We can use any wording we like, but we should follow the guidelines for naming Swift variables.
5. The colon (`:`) tells Swift that the property name on the left is of the type on the right. For example, the `title` is a `String`, which means it will be a string of letters, numbers, spaces, whereas `date` will be the special `Date` type that stores the time and day in a year.

4. Create Mock Articles

In a completed app, we would source the news articles from the internet, by requesting them from a news server. Since we don't have any network fetching functionality yet, we need to "mock" some model data that we can use to test our models. Mock data gives examples of what we expect in our finished app.

- 👉 Create another file, using the `Swift File` template. Call it `Article+Mock.swift`.
- 👉 Type in the code below. You might want to copy and paste the long quoted strings.



[Open in app](#)

💡 When done, the code should look like this:

The screenshot shows the Xcode interface with the project navigation bar at the top. Below it is the project structure sidebar showing a folder named 'Blocks' containing files like 'BlocksApp.swift', 'ContentView.swift', 'Article.swift', and 'Article+Mock.swift'. The main editor area displays the 'Article+Mock.swift' file with the following code:

```

1 // 
2 //  Article+Mock.swift
3 //  Blocks
4 //
5 //  Created by Tom Brodhurst-Hill on 3/5/21.
6 //  Copyright © 2021 BareFeetWare. All rights reserved.
7 //
8
9 import Foundation
10
11 extension Article {
12
13     static let blockChain: Self = .init(
14         title: "Lego Block Chain",
15         date: Date(),
16         detail: "Add your own Lego block to the chain of transactions to guarantee
17             security by trusting a whole crowd of people you've never met. What can
18             possibly go wrong?",
19         smallImageName: "blockCircle",
20         largeImageName: "chain"
21     )
22
23     static let airBlock: Self = .init(
24         title: "Air Block & Block",
25         date: Date(),
26         detail: "Rent out your Lego house for extra dollars. Create an extra revenue
27             stream for your family by renting out your unused rooms.",
28         smallImageName: "houseSimple",
29         largeImageName: "houseInterior"
30     )
31 }

```

ℹ️ Don't worry about understanding the code, but here is an explanation:

1. The file is named `Article+Mock.swift` because it extends (or adds to) the functionality of the `struct Article` type that we already have. The `+Mock` in the name indicates the **purpose** of the extension in this file. We can have multiple files that extend the same type, each with a different `+Purpose`.
2. `extension Article` adds extra functionality to the `struct Article` that we already created. In other words, it “extends” `Article` with more features. We could have ~~just added this code in the `struct Article` code, but keeping the mock values in a~~



[Open in app](#)

3. Using `static` and `: Self` declares the variable as an “instance of this type”. That just means that anywhere in the code that expects an `Article`, we can just provide `.blockChain` or `.airBlock`. This makes it really easy to use these mock articles wherever we need them.
4. This code is all inside the `extension Article`, which gives it the same context as if it was all inside `struct Article`. This means that, in this context, Swift understands that `Self` means `Article` and that `.init(...)` means `Article(...)`. For example, we could have written `static let blockChain: Article = Article()` instead of `static let blockChain: Self = .init()`, since they mean the same thing in this context. But the latter is more generic and reusable.
5. Each pair of parameters and argument provides a value for that property. For example, `title: "Lego Block Chain"` sets the `title` property to the string `"Lego Block Chain"`.
6. Recall from our earlier [Tutorial 6](#) that we added a bunch of image files, including those named `houseSimple` and `houseInterior`. If you like, you can have another look by selecting the `Assets.catalog` in the Project navigator. We are quoting those asset names here for the image names.
7. `Date()` creates a new instance of a date. Since we have provided no properties in this instance of `Date`, it will use its default value, which happens to be “right now” — that is, today, right this second, when the code runs.

👉 After entering all the code, test that it compiles by selecting `Build` from the `Product` menu. If there are any errors, fix them by checking that your code matches the above code.

5. Commit Changes

As you've done before:

1. 👉 Choose `Commit` from the `Source Control` menu.
2. 👉 Enter a description such as: `Created Article model and mock instances`



[Open in app](#)

We have created a new `Article` model and a couple of mock instances.

Next, in [Tutorial 12](#), we will extend the `NewsCell` view so it can display the data from the `Article` model.

See upcoming tutorials in the [table of contents](#). Follow the author to be notified of more articles.

💬 If you have any questions or comments, please add a response below.

This series is released via [Next Level Swift](#). Subscribe to keep updated and never miss a new Tutorial of this series!

Next Level Swift

Next Level Swift

Next Level Swift aims at sharing knowledge and insights into better programming for iOS and is dedicated to help...

[medium.com](https://medium.com/@tom_brodhurst_hill)

We are always looking for talented and passionate Swift developers! Check out our writer's section and find out how you can share your knowledge with the Next Level Swift Community!



[Open in app](#)[Get this newsletter](#)

Emails will be sent to research2learn@yahoo.co.uk.
[Not you?](#)



[Open in app](#)Published in Next Level Swift · [Following](#) ▾Tom Brodhurst-Hill · [Following](#)

Jun 15, 2021 · 7 min read ★

...

Connect Model and View

Build an App Like Lego, with SwiftUI — Tutorial 12

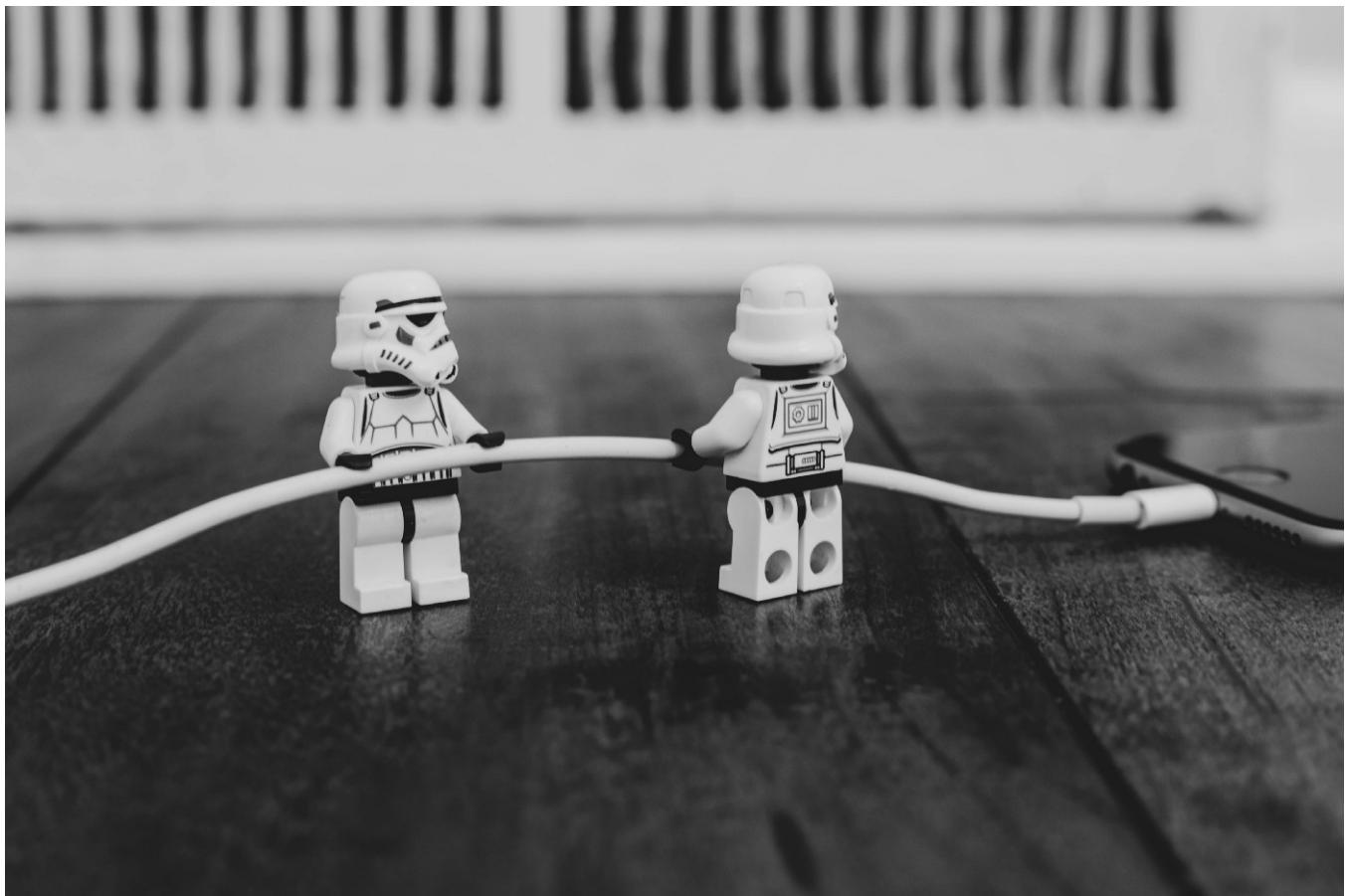


Photo by [Will Porada](#) on [Unsplash](#)

1. Introduction

A properly architected app separates the view components (what you see and touch) from the model components (what you can't see, such as data and calculations). At some point, we need to display some model data in a view. We need some code that dictates how to “present” it or “control” it, which is why it is sometimes described as a “presenter” or a “controller”



[Open in app](#)

present it in the view.

In this tutorial, we will create an extension of a view to display data from a model. We will also use the Attributes inspector to select which mock model to use for each view.

In the previous [Tutorial 11](#), we created an `Article` model and a couple of mock instances. In this Tutorial 12, we will display those articles in news cells.

We will pick up here where the last tutorial left off. Ideally, you have completed the [previous tutorials](#) in this series. Or, you can [download](#) the prepared project, ready to start this tutorial.

2. Extend NewsCell to Accept an Article

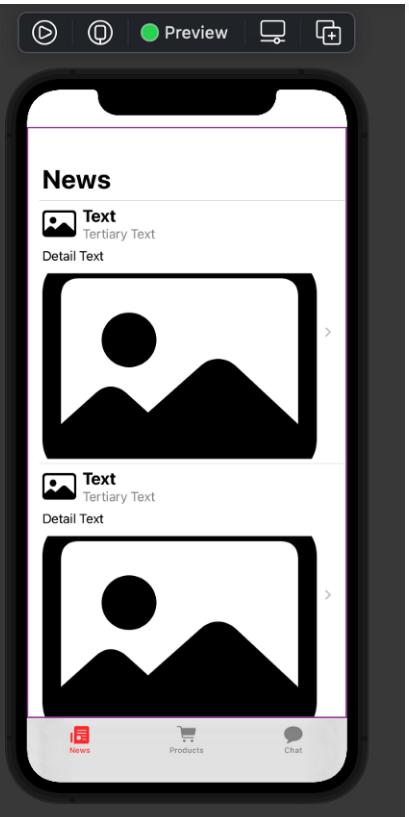
Now we have an `Article` model and a `NewsCell` view. We need a way to show the data from an `Article` in a `NewsCell`. There are several ways to do this, but the neatest and most extendable way I've found is to create an extension of the view (`NewsCell`) that expects a model (`Article`) as the only parameter.

💡 Look at `ContentView.swift`. Recall that in [Tutorial 10](#) we temporarily created a `NewsCell` instance by providing all of the expected view properties: `image`, `text`, etc.



[Open in app](#)

```
8 import SwiftUI
9
10
11 struct ContentView: View {
12     var body: some View {
13         TabView(selection: $selection) {
14             NavigationView {
15                 List(0 ..< 5) { item in
16                     NavigationLink(destination: Destination) {
17                         NewsCell(
18                             image: Image(systemName: "photo"),
19                             text: Text("Text"),
20                             detailText: Text("Detail Text"),
21                             tertiaryText: Text("Tertiary Text"),
22                             largeImage: Image(systemName: "photo")
23                         )
24                     }
25                 }
26                 .navigationTitle("News")
27             }
28             .tabItem { Label("News", systemImage: "newspaper.fill") }
29             .tag(1)
30             NavigationView {
31                 Text("Tab Content 2")
32                 .navigationTitle("Products")
33             }
34             .tabItem { Label("Products", systemImage: "cart.fill") }
35             .tag(2)
36             NavigationView {
37                 Text("Tab Content 3")
38                 .navigationTitle("Chat")
39             }
40         }
41     }
42 }
```



This NewsCell instance is currently just a placeholder.



[Open in app](#)

Now that we have an `Article` model, we would like to replace this `NewsCell` placeholder with the data from `Article` (model) instances.

For example:



[Open in app](#)

To facilitate this, we need to define an alternative `init` (initialization) for `NewsCell` that will take an `Article`.

In other words, we already have an `init` (a way to create) for `NewsCell` where we specify each `Text` and `Image` subview. It would be convenient for us to have another `init` which would instead just need an `Article`. The code will extract each part of the `Article` to make the required `Text` and `Image` subviews.

Let's do that now.

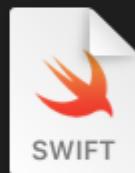


[Open in app](#)

Choose a template for your new file

iOS**macOS****watchOS****tvOS**

Source

**Swift File****Cocoa Class****Header File****IIG Reference**

User Interface

**SwiftUI View****Storyboard**

[Open in app](#)

👉 In the new file, select the default `struct NewsCell_Article` code.

The screenshot shows the Xcode interface with the project navigation on the left and the code editor on the right. The project structure is as follows:

- Blocks** folder:
 - Blocks** folder
 - `BlocksApp.swift`
 - `ContentView.swift`
 - `Article.swift`
 - `Article+Mock.swift`
 - `NewsCell.swift`
 - `NewsCell+Article.swift` (highlighted with a grey background)
 - `Assets.xcassets`
 - `Info.plist`
 - `Preview Content`
 - `Products`

In the code editor, the file `NewsCell+Article.swift` contains the following Swift code:

```
1 //  
2 //  NewsCell+Article.swift  
3 //  Blocks  
4 //  
5 //  Created by Tom Brodhurst-Hill on 9/5/21.  
6 //  Copyright © 2021 BareFeetWare. All rights reserved.  
7 //  
8  
9 import SwiftUI  
10  
11 struct NewsCell_Article: View {  
12     var body: some View {  
13         Text("Hello, World!")  
14     }  
15 }  
16  
17 struct NewsCell_Article_Previews: PreviewProvider {  
18     static var previews: some View {  
19         NewsCell_Article()  
20     }  
21 }
```

👉 Delete it and replace it with the following code:



[Open in app](#)

In the preview, replace the template `NewsCell_Article()` instance with:

⌚ Your code should look like this:





Open in app

The screenshot shows the Xcode interface with the 'Blocks' project open. The 'Blocks' folder contains several files: Article.swift, Article+Mock.swift, ContentView.swift, NewsCell.swift, and NewsCell+Article.swift. The 'NewsCell+Article.swift' file is currently selected and shown in the editor. The code defines an extension for NewsCell that initializes it with an Article object, setting up text, detailText, tertiaryText, and images. It also defines a PreviewProvider for generating previews of the cell. To the right, a preview window displays a mobile application interface titled 'Air Block & Block' with a timestamp of '2021-05-25 03:41:01 +0000'. The preview shows a room with two blue armchairs, a small table, and a red rug.

```

1 // 
2 //  NewsCell+Article.swift
3 //  Blocks
4 //
5 //  Created by Tom Brodhurst-Hill on 9/5/21.
6 //  Copyright © 2021 BareFetWare. All rights reserved.
7 //
8
9 import SwiftUI
10
11 extension NewsCell {
12     init(article: Article) {
13         self.init(
14             image: Image(article.smallImageName),
15             text: Text(article.title),
16             detailText: Text(article.detail),
17             tertiaryText: Text(String(describing: article.date)),
18             largeImage: Image(article.largeImageName)
19         )
20     }
21 }
22
23 struct NewsCell_Article_Previews: PreviewProvider {
24     static var previews: some View {
25         NewsCell(article: .airBlock)
26             .previewLayout(.sizeThatFits)
27     }
28 }

```

👉 Check that your code compiles, by selecting `Build` from the `Product` menu. Fix any errors in the code.

ℹ️ Let's walk through what's happening here, in the code. Again, you don't need to worry about these inner workings — this is just for those who want to understand the code:

1. `extension NewsCell` just means: Add this new code to the existing functionality of `NewsCell`.
2. `init(article: Article)` means: Define a new way of initializing/instantiating a `NewsCell`, that just requires a parameter named `article` with an argument (value) that is an `Article`.
3. `self.init(image...)` means: Create a `NewsCell` using the existing properties/parameters of `image`, `text` etc, with the arguments/values listed below for each.
4. `article.smallImageName` means: the article's `smallImageName`. Getting the property of an instance like this is referred to as "dot notation".
5. `image: Image(article.smallImageName)` means: For the expected `image` parameter/property, create an `Image` using the `smallImageName` of the provided



[Open in app](#)

detailText .

7. For the `tertiaryText`, we want to show the `article`'s date. `Text()` expects a `String` argument but the `article`'s date is a `Date`, not a `String`. To create a `String` from a `Date`, we can use the `String(describing:)` initialiser function. This will give a basic `String` description of the `Date`. It won't be pretty, but it will do for now, and we'll fix it later.

- 👉 If the preview hasn't updated, click on the `Resume` button in the top right of the preview pane.
- 👀 Confirm that the preview shows the “Air Block & Block” article data, as shown above. Of course, the date will show as your current local date.

3. Select a Mock Instance from the Attributes Inspector

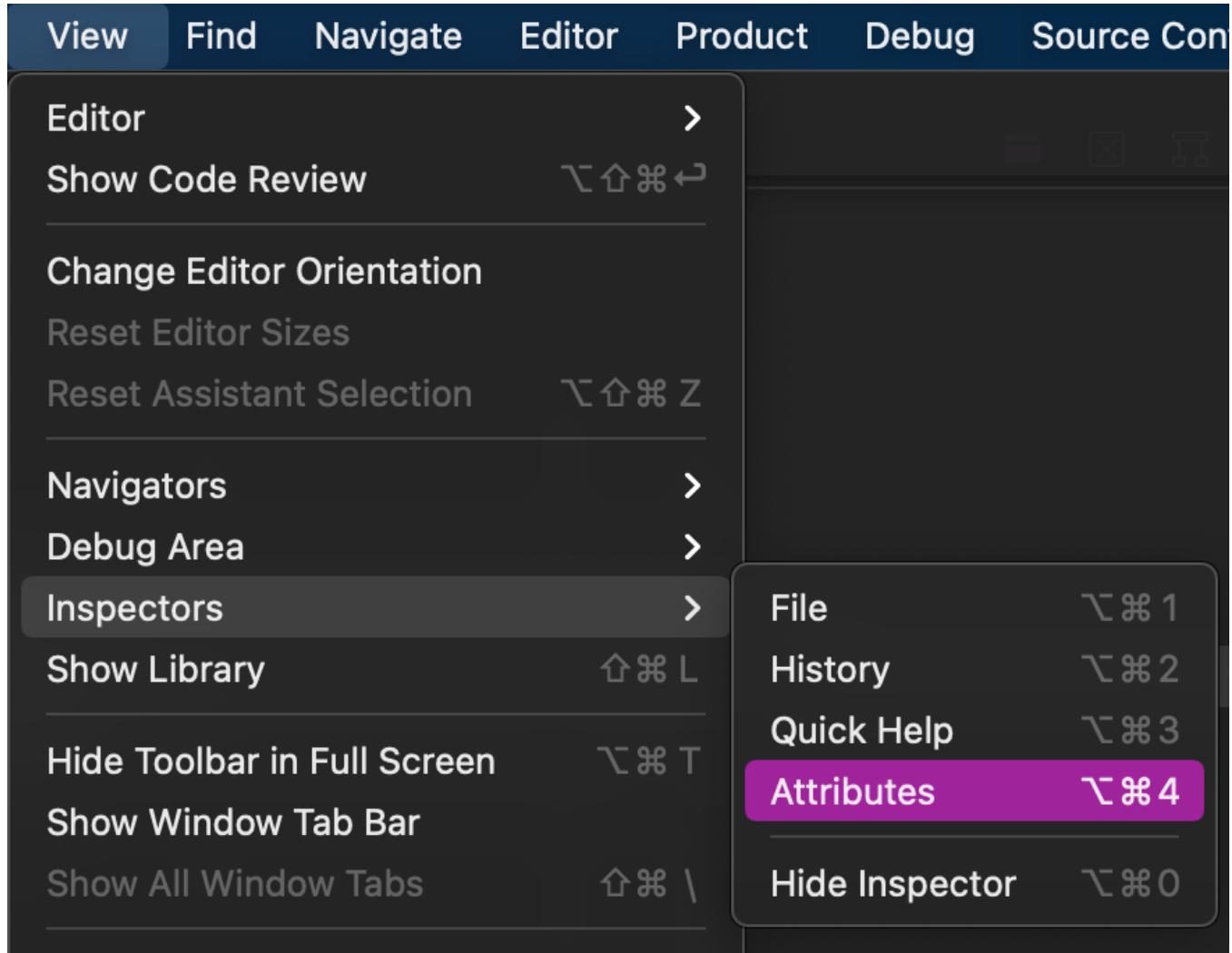
- 👉 Click on the `NewsCell(...)` instance in the preview code.

Make sure that the Attributes inspector (right panel) is showing. If not, you can display it through the `View` menu.

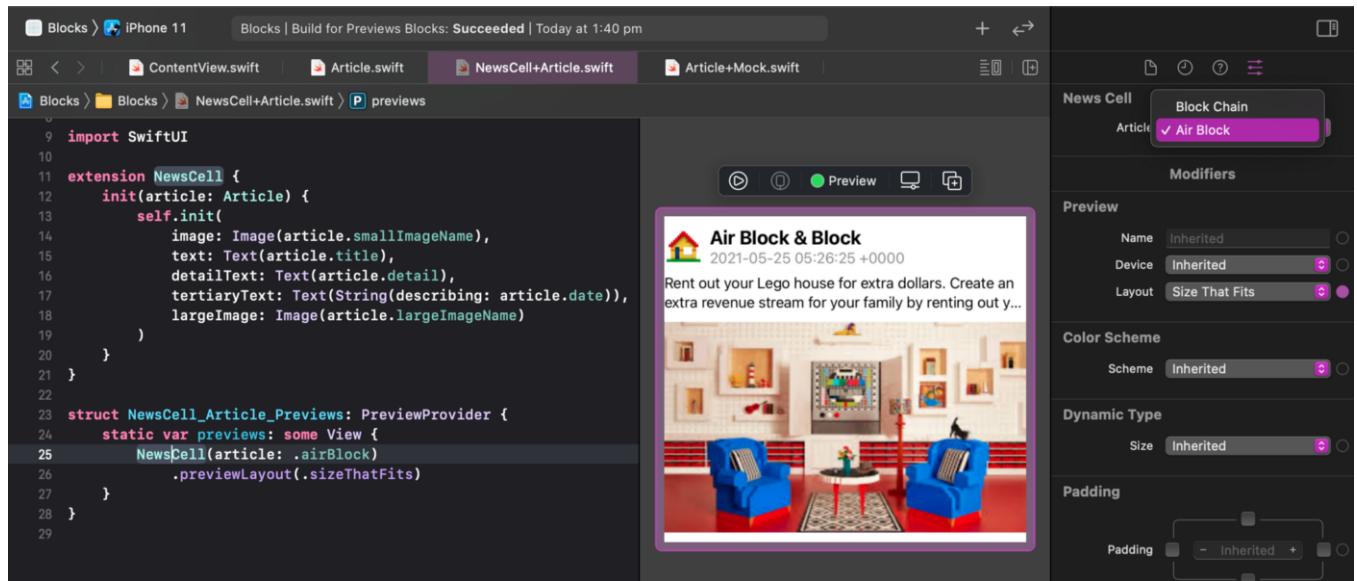




Open in app



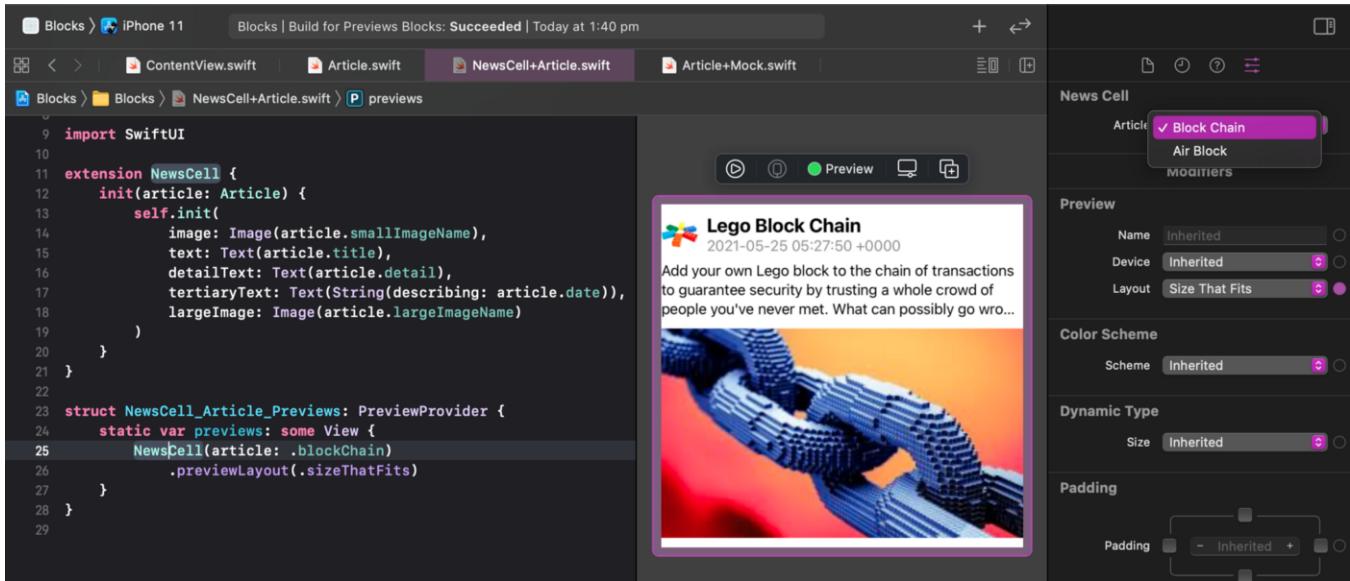
👉 At the top of the Attributes inspector, click on the Article popup menu.



[Open in app](#)

the camel case of our variable names (like `airBlock`) to more human-readable separate words in title case (like `Air Block`).

👉 In the Attributes inspector, change the `Article` popup menu from `Air Block` to `Block Chain`. Confirm that the preview updates to show the different `Article`.



Then change it back to `Air Block`.

4. Show Articles in the List

Now that we have created the `NewsCell`'s `init(article: Article)` initializer, let's use an article in the list display in `ContentView`.

👉 In `ContentView.swift`, locate the `NewsCell(` line of code. Double click on the opening parenthesis `(`, which will select the entire multiline contents between the opening and closing parentheses.



[Open in app](#)

```

11 struct ContentView: View {
12     var body: some View {
13         TabView(selection: Selection) {
14             NavigationView {
15                 List(0 ..< 5) { item in
16                     NavigationLink(destination: Destination) {
17                         NewsCell(
18                             image: Image(systemName: "photo"),
19                             text: Text("Text"),
20                             detailText: Text("Detail Text"),
21                             tertiaryText: Text("Tertiary Text"),
22                             largeImage: Image(systemName: "photo")
23                         )
24                     }
25                 }
26             }
27         }
28     }
29 }
```

👉 Hit delete on the keyboard.

👉 Type the opening (again. This time, in the auto-completion popup menu, you should see the additional option of the NewsCell(article:) initializer. Select it.



The screenshot shows the Xcode code editor with the following code:

```

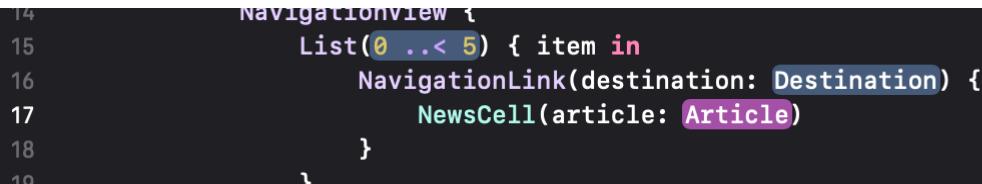
11 struct ContentView: View {
12     var body: some View {
13         TabView(selection: Selection) {
14             NavigationView {
15                 List(0 ..< 5) { item in
16                     NavigationLink(destination: Destination) {
17                         NewsCell(|)
18                     }
19                 }
20             }
21         }
22     }
23 }
```

A completion dropdown menu is open at the cursor position (|). It contains two suggestions:

- M (article:)** (highlighted)
- M (image:text:detailText:tertiaryText:largeImage:)**

A red circle with a question mark icon is shown next to the first suggestion, indicating a missing argument.

👁️ Xcode places an Article placeholder where it expects an instance of an Article .



The screenshot shows the Xcode code editor with the following code:

```

14             NavigationView {
15                 List(0 ..< 5) { item in
16                     NavigationLink(destination: Destination) {
17                         NewsCell(article: Article)
18                     }
19                 }
20             }
21         }
22     }
23 }
```

The word "Article" is highlighted in purple, indicating it is a placeholder for a type.

👉 With the Article placeholder selected, type a period (.). Xcode should offer a popup menu of the available instances of Article (which we created in an earlier tutorial).





Open in app

```

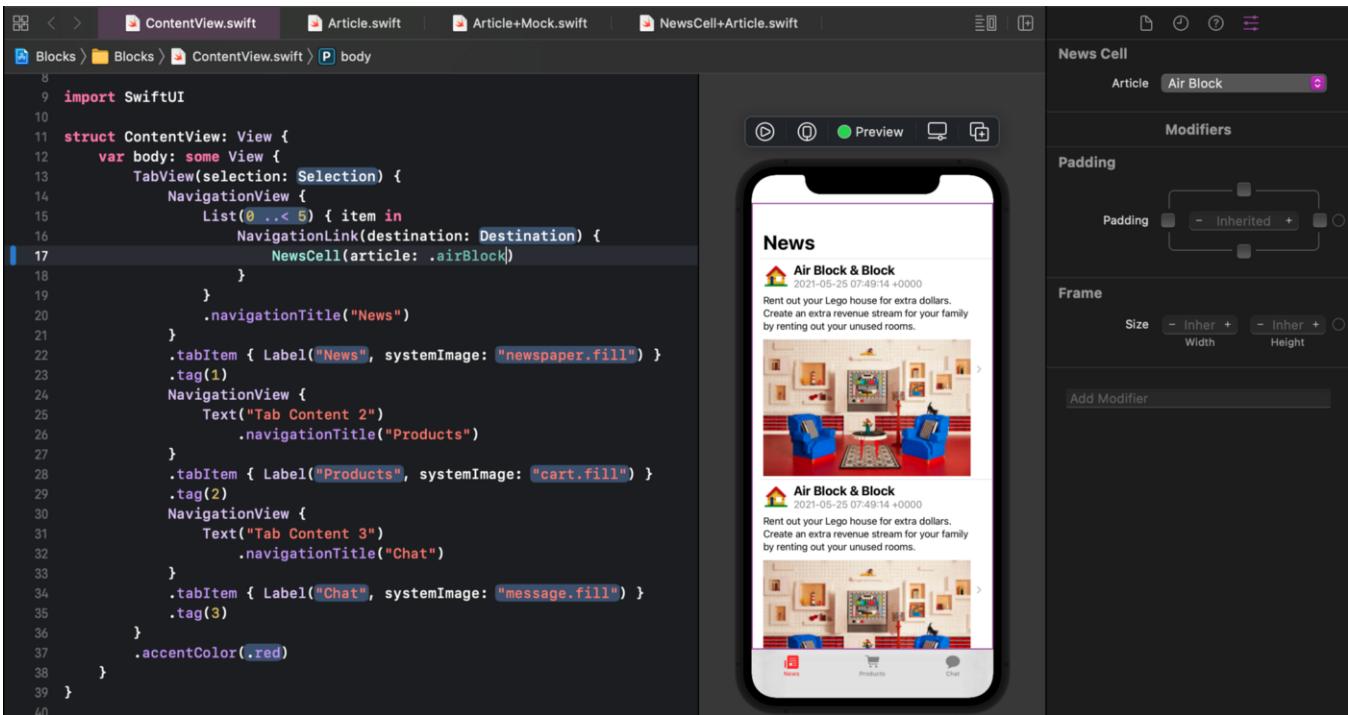
14
15     List(0 ..< 5) { item in
16         NavigationLink(destination: Destination) {
17             NewsCell(article: .)
18         }
19     }
20     .navigationTitle("News")
21 }
22 .tabItem { Label("News", systemImage: "newspaper.fill") }
23 .tag(1)

```

A code completion dropdown is shown, listing several options: `airBlock`, `blockChain`, `mocks`, and `init(title:date:detail:smallImageName:largeImageName:)`. The option `airBlock` is highlighted with a green checkmark.

👉 Select `airBlock` from the popup menu.

👉 Check that the list of news cells in the preview all show the `airBlock` article. If necessary, build the app (select `Build` from the `Product` menu) and refresh the preview.



5. Commit Changes

As you've done before:

1. 👉 Choose `Commit` from the `Source Control` menu.
2. 👉 Enter a description such as: `NewsCell` can display an `Article`
3. 👉 Click on the `Commit` button.



[Open in app](#)

Next, in [Tutorial 13](#), we will loop through the Article instances to show them all in the list.

See upcoming tutorials in the [table of contents](#). Follow the author to be notified of more articles.

! If you have any questions or comments, please add a response below.

This series is released via [Next Level Swift](#). Subscribe to keep updated and never miss a new Tutorial of this series!

Next Level Swift

Next Level Swift

Next Level Swift aims at sharing knowledge and insights into better programming for iOS and is dedicated to help...

[medium.com](https://medium.com/@nextlevelsdk)

We are always looking for talented and passionate Swift developers! Check out our writer's section and find out how you can share your knowledge with the Next Level Swift Community!

Sign up for Next Level Swift Newsletter

By [Next Level Swift](#)



[Open in app](#)

[Open in app](#)Published in Next Level Swift · [Following](#) ▾Tom Brodhurst-Hill · [Following](#)

Jun 23, 2021 · 7 min read ★

...

Show an Array of Identifiable Items in a List

Build an App Like Lego, with SwiftUI — Tutorial 13



Photo by [Eric & Niklas](#) on [Unsplash](#)

1. Introduction

In Swift and most programming languages, an ordered sequence of items is called an `Array`. In SwiftUI, we often want to display an array of items in a `List`. An `Array` is a model structure. A `List` is a view structure.

A `List` needs to uniquely identify each item. In other words, it needs the items in an `Array` to be `Identifiable`. For example, we could identify each item by a unique name or a unique number. Then the `List` can control the items' positions and perform



[Open in app](#)

In the previous [Tutorial 12](#), we displayed an article in each news cell in our list. But each cell showed the same article.

In this Tutorial 13, we will show one `NewsCell` for each unique `Article` in an `Array`. We will make the articles `Identifiable` so we can display them in a `List`.

We will pick up here where the last tutorial left off. Ideally, you have completed the [previous tutorials in this series](#). Or, you can [download](#) the prepared project, ready to start this tutorial.

2. Create an Array

First, let's create an array of articles, using the mock articles that we created earlier.

👉 Open the `Article+Mock.swift` file that we created in an earlier tutorial. At the end of that file, add a blank line and then this code:





Open in app

Your code should then look like this:

```
11 extension Article {
12
13     static let blockChain: Self = .init(
14         title: "Lego Block Chain",
15         date: Date(),
16         detail: "Add your own Lego block to the chain of transactions
17             to guarantee security by trusting a whole crowd of people
18             you've never met. What can possibly go wrong?",  

19         smallImageName: "blockCircle",
20         largeImageName: "chain"
21     )
22
23     static let airBlock: Self = .init(
24         title: "Air Block & Block",
25         date: Date(),
26         detail: "Rent out your Lego house for extra dollars. Create
27             an extra revenue stream for your family by renting out
28             your unused rooms.",  

29         smallImageName: "houseSimple",
30         largeImageName: "houseInterior"
31     )
32
33 }
34
35 extension Article {
36     static let mocks: [Self] = [.blockChain, .airBlock]
37 }
```

 Here's a quick overview of this new code:

1. The second extension is separated just for visual clarity. We could have combined all of the code in this file into one extension.

2. Self-inside-extension Article is the same as explicitly writing Article



[Open in app](#)

means an array of that type. An array is just a sequence of items, in order. For example, an array of fruits could be `let fruits: [Fruit] = [.apple, .orange, .banana]`.

4. We've chosen the variable name `mocks` for the array of mock articles. Arrays are usually named as a plural.
5. We wrap the items in the array in square brackets, with the items separated by commas.
6. We've assigned `mocks` to be the array of `[.blockChain, .airBlock]`, which contains the two instances already defined above.

3. Show an Array in a List

Now we will show the information from our `mocks` array of articles in a `List`.

👉 In `ContentView.swift`, in the `List(...)`, click on the template `0...<5` to select it.

```
11 struct ContentView: View {
12     var body: some View {
13         TabView(selection: Selection) {
14             NavigationView {
15                 List(0 ..< 5) { item in
16                     NavigationLink(destination: Destination) {
17                         NewsCell(article: .airBlock)
18                     }
19                 }
20             }
21         }
22     }
23 }
```

👉 Replace it with `Article.mocks`. As you type, Xcode will prompt to autocomplete, so you don't have to type the whole thing.

The screenshot shows a portion of a Swift file in Xcode. At line 15, there is a call to `List(A)`. A completion dropdown menu is open, showing suggestions: `S Article`, `P App`, and `E Axis`. The suggestion `S Article` is highlighted with a purple background.

```
13     TabView(selection: Selection) {
14         NavigationView {
15             List(A) { item in
16                 NavigationLink(destination: Destination) {
17                     NewsCell(article: .airBlock)
18                 }
19             }
20         }
21     }
22 }
```





Open in app

```

14         NavigationView {
15             List(Article.) { item in
16                 Navig M init(title:date:detail:smallIm
17                     @ Type
18                 }
19             }
20             .navigation
21         }

```

A screenshot of Xcode showing a code editor with a popover open over the word 'Article.'. The popover contains several items: 'M init(title:date:detail:smallIm' (with a red circle icon), '@ Type', 'V airBlock', 'V blockChain', 'V mocks', and '@ self'. The 'mocks' option is highlighted.

```

14     NavigationView {
15         List(Article.mocks) { item in
16             NavigationLink( ⓘ Initializer 'init(_:rowContent:)' requires that 'Article' conform to
17                 NewsCell(a
18             )
19         }

```

A screenshot of Xcode showing a code editor with an error popover. The error message says: 'Initializer 'init(_:rowContent:)' requires that 'Article' conform to 'Identifiable''. A red stop sign icon is visible next to the error message.

4. Identifiable

👉 As you can see, Xcode shows an error popover.

👉 Click on the red stop sign icon in the error popover to reveal all of the words in the error description.

Xcode is saying that the `List` won't accept an array of articles, because `Article` does not "conform to `Identifiable`". But what does that mean?

It means that `List` needs a way to identify each article as distinct from the other articles in the `mocks` array. The simplest way to do that is to just add a unique identifier to each article. It could be a unique name string, but we will just number them.

👉 Open the `Article.swift` file (by selecting it in the File navigator in the left pane).

👉 After `struct Article`, add `: Identifiable`. This tells Xcode that we promise to make `Article` "conform" to the `Identifiable` protocol.

```

11 struct Article: Identifiable { ⓘ Type 'Article' does not conform to protocol 'Identifiable'
12     let title: String
13     let date: Date
14     let detail: String
15     let smallImageName: String

```



[Open in app](#)

⌚ Xcode keeps us to our promise, by complaining that Type 'Article' does not conform to protocol 'Identifiable'. Note that this stop sign contains a dot, not a cross like previous errors.

👉 Click on the stop sign containing the dot.

⌚ The error popover expands and offers to Fix the error by adding protocol stubs .

```

11 struct Article: Identifiable {
12     let title: String
13     let date: Date
14     let detail: String
15     let smallImageName: String
16     let largeImageName: String
17 }
18

```

👉 Click on the Fix button in the popover.

⌚ Xcode adds placeholder code to fix the error.

```

10
11 struct Article: Identifiable {
12     var id: ObjectIdentifier
13
14     let title: String
15     let date: Date
16     let detail: String
17     let smallImageName: String
18     let largeImageName: String
19 }
20

```

👉 Since we will use simple integers (whole numbers) for the id, replace ObjectIdentifier with Int .

👉 Since we will not change (vary) the id of each article, once assigned, change var to let .



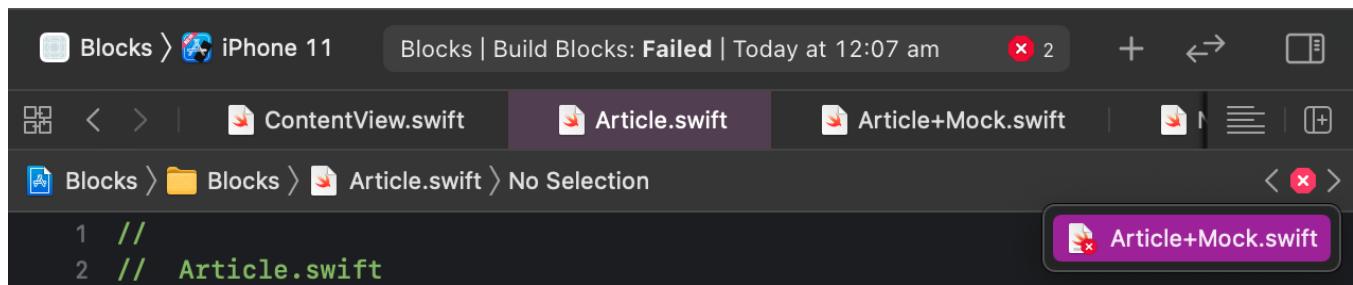
[Open in app](#)

```
11 struct Article: Identifiable {  
12     let id: Int  
13     let title: String  
14     let date: Date  
15     let detail: String  
16     let smallImageName: String  
17     let largeImageName: String  
18 }  
19
```

👉 Build the app (in the Products menu).

5. Add missing arguments

⌚ Now Xcode complains about errors in the Article+Mock.swift file.



👉 Switch to the Article+Mock.swift file by selecting it in the error popup menu.

⌚ Xcode shows two occurrences of the same error: Each mock article is missing an argument (value) for the new `id` parameter (property) that we added.





Open in app

```

11 extension Article {
12
13     static let blockChain: Self = .init(
14         title: "Lego Block Chain", ⚡ Missing argument for parameter 'id' in call
15         date: Date(),
16         detail: "Add your own Lego block to the chain of transactions to
17             guarantee security by trusting a whole crowd of people you've
18             never met. What can possibly go wrong?",
19         smallImageName: "blockCircle",
20         largeImageName: "chain"
21     )
22
23     static let airBlock: Self = .init(
24         title: "Air Block & Block", ⚡ Missing argument for parameter 'id' in call
25         date: Date(),
26         detail: "Rent out your Lego house for extra dollars. Create an extra
27             revenue stream for your family by renting out your unused rooms.",
28         smallImageName: "houseSimple",
29         largeImageName: "houseInterior"
30     )
31 }
32
33 extension Article {
34     static let mocks: [Self] = [.blockChain, .airBlock]
35 }
```

👉 Tap on the stop sign containing the dot. In the popover error detail, click on the Fix button.

```

11 extension Article {
12
13     static let blockChain: Self = .init(
14         title: "Lego Block Chain", ⚡ Missing argument for parameter 'id' in call
15         date: Date(),
16         detail: "Add your own L
17             guarantee security
18             never met. What can possibly go wrong?",
19         smallImageName: "blockCircle",
20         largeImageName: "chain"
21     )
22
23 }
```

👁️ Xcode adds the missing id parameter with a placeholder for the expected Int value.





Open in app

- 👉 Replace the `Int` placeholder with the value `1`. Click after the comma and hit Return to separate the parameters onto their own lines.

```

13     static let blockChain: Self = .init(
14         id: 1,
15         title: "Lego Block Chain",
16         date: Date(),
17         details: "Add your own Lego block to the chain of transactions to"

```

- 👉 Similarly, in the second instance, add `id: 2`.

```

21
22     static let airBlock: Self = .init(
23         id: 2,
24         title: "Air Block & Block",
25         date: Date(),
26         details: "Rent out your house for extra dollars. Create an extra revenue stream for your family by renting out your unused rooms."

```

- 👉 Build the app again and check that there are now no errors.

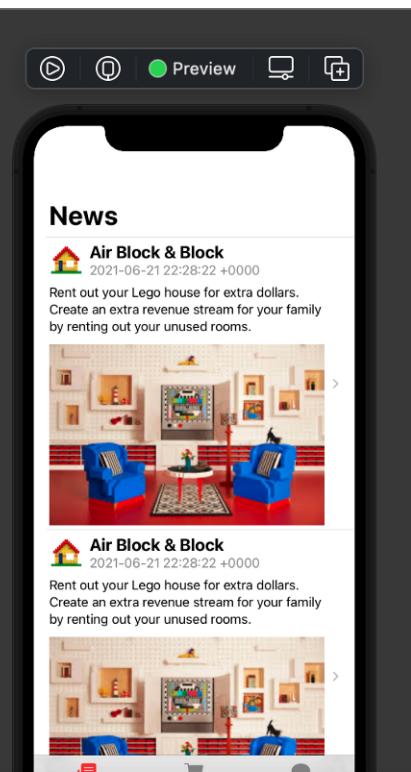
6. Array in a List

- 👉 Switch to the `ContentView.swift` file, which contains our `List`. If necessary, Resume the preview.

```

1 //
2 // ContentView.swift
3 // Blocks
4 //
5 // Created by Tom Brodhurst-Hill on 4/3/21.
6 // Copyright © 2021 BareFeetWare. All rights reserved.
7 //
8
9 import SwiftUI
10
11 struct ContentView: View {
12     var body: some View {
13         TabView(selection: $selection) {
14             NavigationView {
15                 List(Article.mocks) { item in
16                     NavigationLink(destination: Destination) {
17                         NewsCell(article: item)
18                     }
19                 }
20                 .navigationTitle("News")
21             }
22             .tabItem { Label("News", systemImage: "newspaper.fill") }
23             .tag(1)
24             NavigationView {
25                 Text("Tab Content 2")
26                 .navigationTitle("Products")
27             }
28             .tabItem { Label("Products", systemImage: "cart.fill") }
29             .tag(2)

```



[Open in app](#)

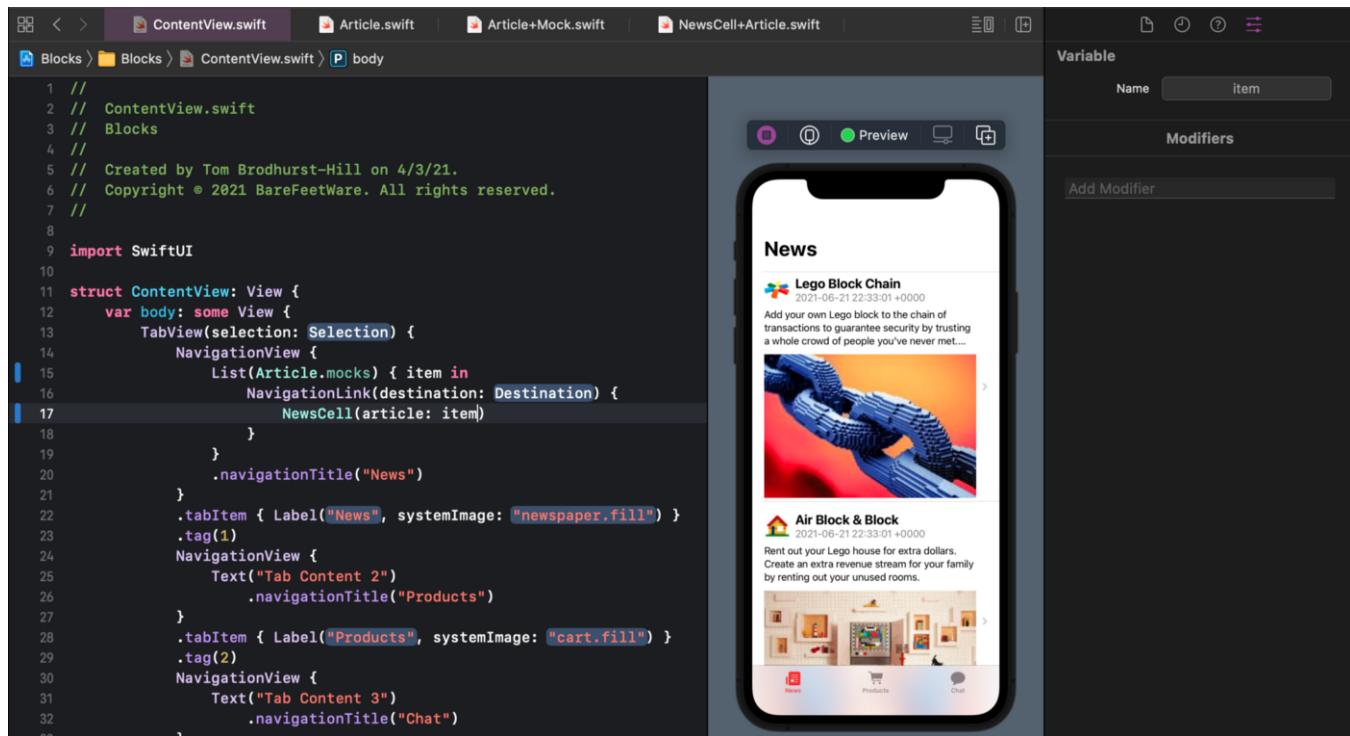
👉 When the preview is running, scroll the News scene by dragging it up and down.

👁 You should see that there are now just two news cells displayed in the list. Recall that the previous template code showed six cells.

The `List(Article.mocks)` tells SwiftUI to make a list containing one cell/row for each item in the `Article's mocks` array, which is just two.

But the preview is still displaying all news cells with only the `.airBlock` article. That makes sense, since the `NewsCell` code inside the `List` has the argument `.airBlock`). We need to replace that static (unchanging) instance with each `item` that the `List` provides.

👉 In the `NewsCell(...)`, replace the `.airBlock` with `item`.



👁 The preview now shows two news cells populated with different articles. You can scroll up and down to check.

7. Rename a Variable

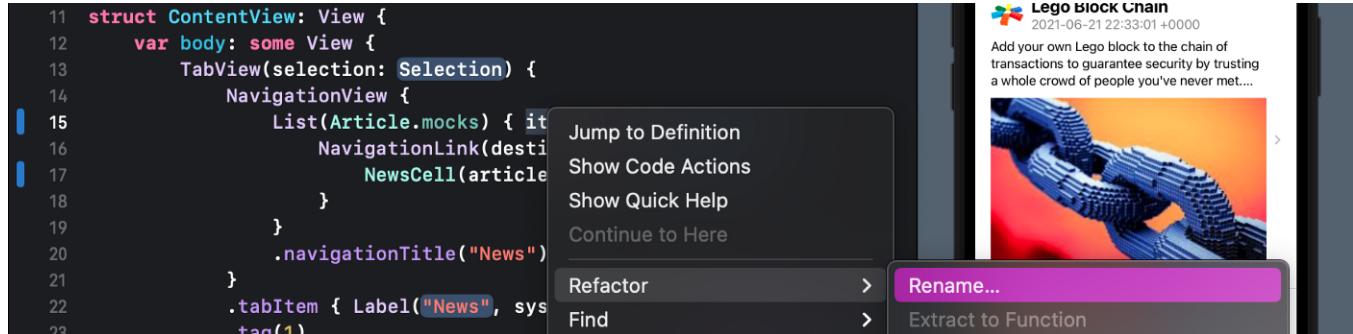
The variable name `item` is pretty generic. As our code gets more complex, naming



[Open in app](#)

👉 Control click on either of the `item` occurrences in the `List` code.

👉 In the popup menu, select Refactor > Rename....



👁️ Xcode focuses on the code that will be affected and selects both occurrences of `item`.



👉 With the `item` text still selected, type to replace it with `article`. You should see both instances change as you type.

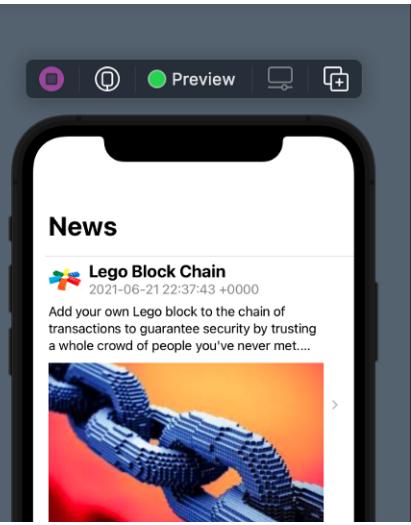


[Open in app](#)

```

1 // ContentView.swift
2 // Blocks
3 //
4 //
5 // Created by Tom Brodhurst-Hill on 4/3/21.
6 // Copyright © 2021 BareFeetWare. All rights reserved.
7 //
8
9 import SwiftUI
10
11 struct ContentView: View {
12     var body: some View {
13         TabView(selection: $selection) {
14             NavigationView {
15                 List(Article.mocks) { article in
16                     NavigationLink(destination: Destination(article: article)) {
17                         NewsCell(article: article)
18                     }
19                 }
20             }
21             .navigationTitle("News")
22         }
23     }
24 }

```



8. Commit Changes

As you've done before:

1. ➡ Choose Commit from the Source Control menu.
2. ➡ Enter a description such as: Article: Identifiable and mocks
3. ➡ Click on the Commit button.

9. Next...

Congratulations! The list of news articles is now visually complete. It is showing all the currently available articles.

Next, in [Tutorial 14](#), we will build a `ProductCell`, using all the tools you've learned so far.

See upcoming tutorials in the [table of contents](#). Follow the author to be notified of more articles. If you liked this article, clap for it (up to 50 times).

💬 If you have any questions or comments, please add a response below.

This series is released via [Next Level Swift](#). Subscribe to keep updated and never miss



[Open in app](#)

We are always looking for talented and passionate Swift developers! Check out our writer's section and find out how you can share your knowledge with the Next Level Swift Community!

Sign up for Next Level Swift Newsletter

By Next Level Swift

Get all the latest articles, posts and news straight to your mailbox! [Take a look.](#)

[Get this newsletter](#)

Emails will be sent to research2learn@yahoo.co.uk.
[Not you?](#)



[Open in app](#)Published in Next Level Swift · [Following](#) ▾Tom Brodhurst-Hill · [Following](#)

Jun 29, 2021 · 9 min read ★

...

Create a Cell by Stacking Subviews

Build an App Like Lego, with SwiftUI — Tutorial 14

Sydney Opera House & Harbour Bridge

Capture the architectural essence and splendor of Sydney with this magnificent set that brings together the iconic Sydney Opera House™, Sydney Harbour Bridge, Sydney Tower and Deutsche Bank Place, in an inspirational skyline setting. Each individual LEGO® structure provides a unique and rewarding building experience with true-to-life color and relative scale depiction. Sydney's sparkling harbor is represented in the

\$75 [Buy](#)

In SwiftUI, we can build a view from subview components by dragging them from a library, so that they click into place vertically and horizontally, like Lego® blocks.

In this Tutorial 14, we will add text subviews, a button, and a resizing image.

In the previous [Tutorial 13](#), we finished connecting the news Article model instances with the NewsCells in the News scene. Now we will build a ProductCell for our Products scene.

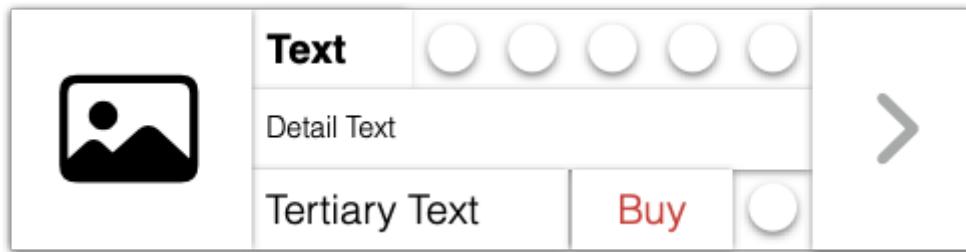


[Open in app](#)

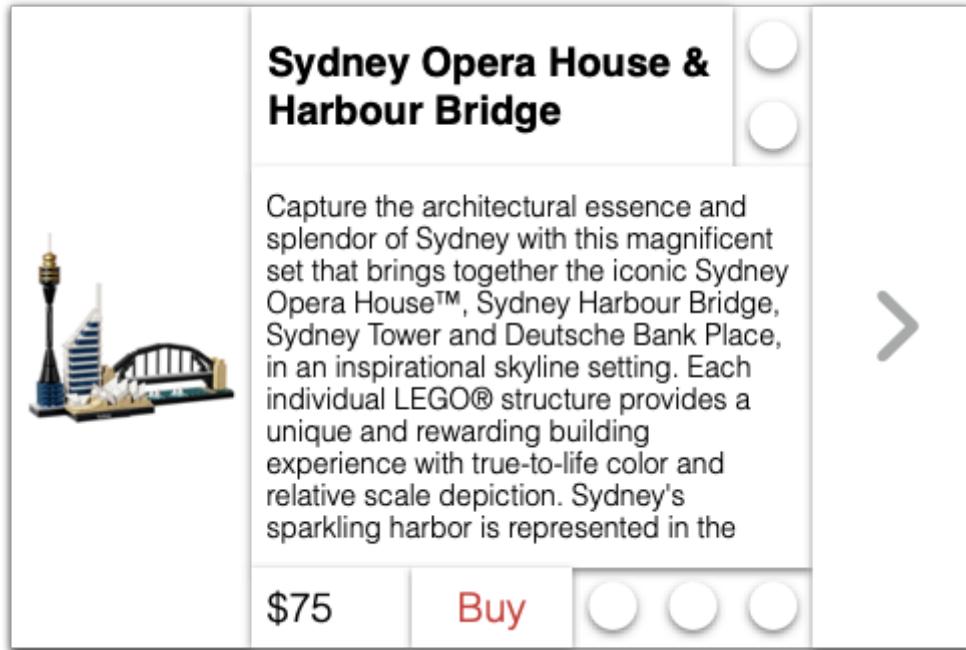
We will pick up here where the last tutorial left off. Ideally, you have completed the [previous tutorials in this series](#). Or, you can [download](#) the prepared project, ready to start this tutorial.

2. Product Cell

We're going to create a product cell to show the image, name description, and price of a product, along with a button to buy it. We will assemble the Lego® blocks like this:



The layout above shows placeholders for the image and text blocks, which we will be able to replace for each different instance. For example, to sell the Sydney Harbour Bridge, we could substitute the image and text blocks as shown below.

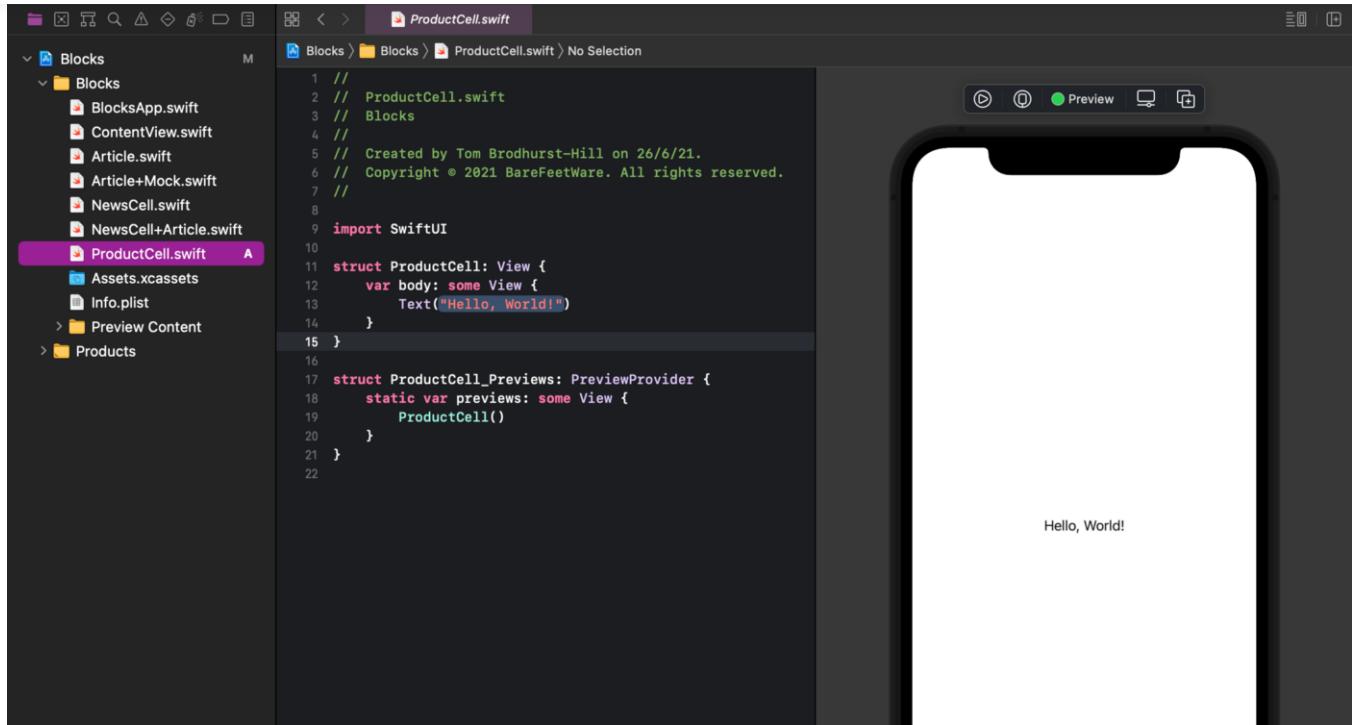


👉 In Xcode's File menu, select New > File . Choose the SwiftUI View template, and name it `ProductCell.swift` .





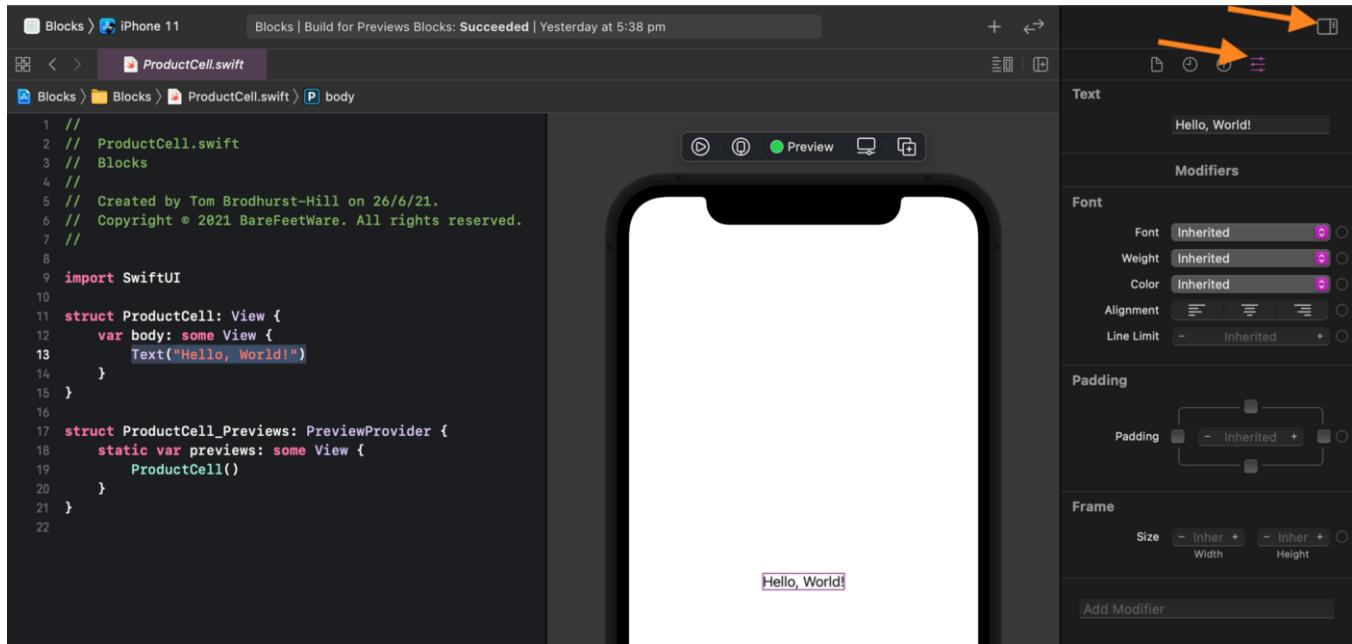
Open in app



3. Text Attributes

Now we will style the text subview using the Attributes inspector.

👉 Ensure that the Attributes inspector panel is visible on the right. If not, click on the Show Inspectors and the Show Attributes inspector buttons in the top right (as shown below).

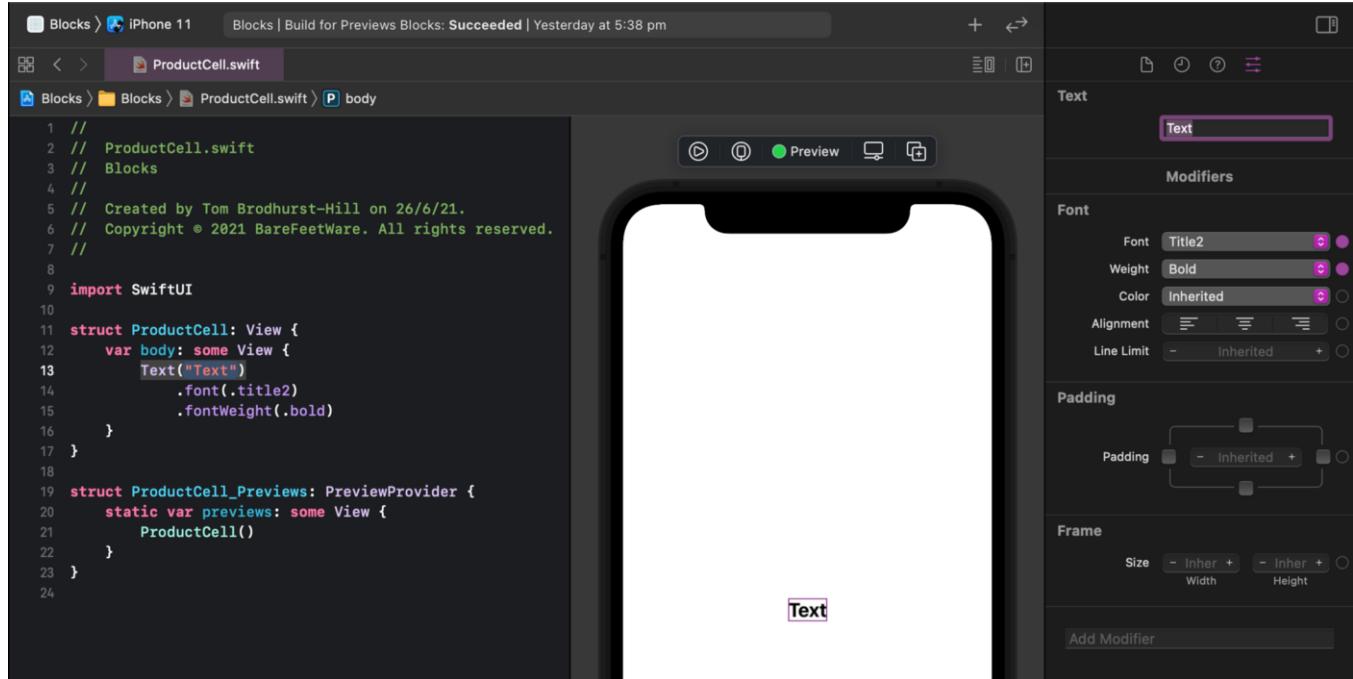


[Open in app](#)

👉 In the Attributes inspector, change the text to be just the word `Text`. Hit `return`. Set the `Font` to `Title 2`. Set the `Font Weight` to `Bold`.

🐞 You might need to first select `Title`, to prompt Xcode to also list `Title 2`.

👁️ Check that your code, preview, and Attributes inspector appear as below.

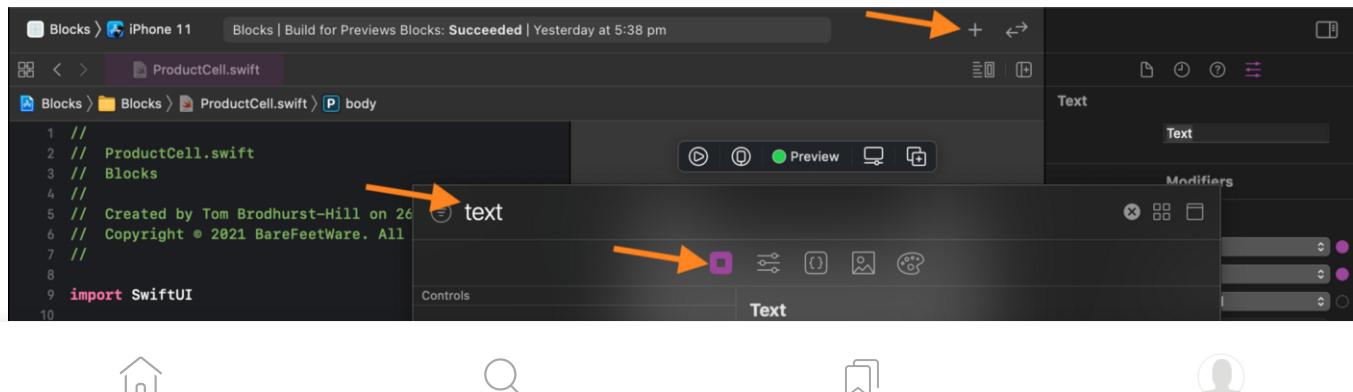


4. AddSubviews in a Vertical Stack

Let's add the other Lego blocks (subviews) to our view.

👉 Click on the Library button (the `+` symbol) above the top right of the preview pane.

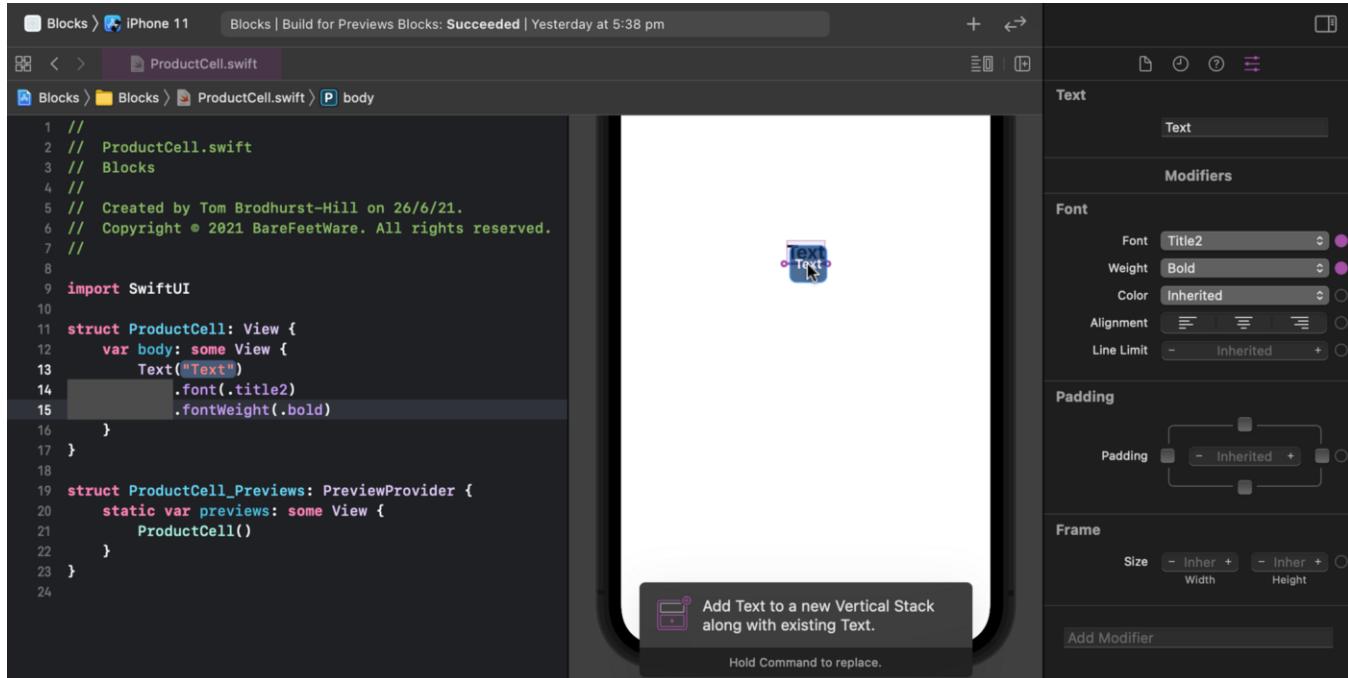
👉 The Library appears. Ensure that the Views tab is selected. Type the word `text` into the search bar. Select the `Text` item in the list, as shown below.



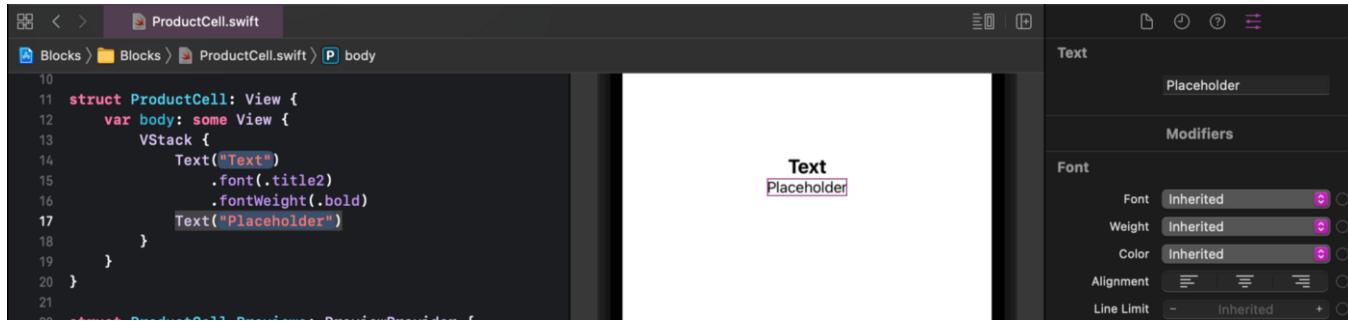
[Open in app](#)

👉 Drag the `Text` item from the Library into the preview, just below the existing `Text` subview (Lego block).

👁️ As shown below, Xcode shows where the new subview will be positioned, like one Lego block below the other.



👁️ In the code, Xcode adds a `VStack` (vertical stack) containing the original `Text` and the new `Placeholder` text.



👉 In the preview, click on the new `Placeholder` text subview, so that you can see its attributes.

👉 In the Attributes inspector, change the new text subview to contain the words `Detail Text`. Hit `return` so Xcode applies the change. Change the `Color` to `Gray`.





Open in app

```

10
11 struct ProductCell: View {
12     var body: some View {
13         VStack {
14             Text("Text")
15                 .font(.title2)
16                 .fontWeight(.bold)
17             Text("Detail Text")
18                 .foregroundColor(Color.gray)
19         }
20     }
21 }

```

👉 Again from the Views Library, drag in a third text subview, just below the Detail Text. Before you release the dragged object, ensure that the caption at the bottom says Insert Text in Vertical Stack. If it instead says Add Text to a new Vertical Stack... then you need to move the pointer up a few pixels until the caption changes.

```

10
11 struct ProductCell: View {
12     var body: some View {
13         VStack {
14             Text("Text")
15                 .font(.title2)
16                 .fontWeight(.bold)
17             Text("Detail Text")
18                 .foregroundColor(Color.gray)
19         }
20     }
21 }
22
23 struct ProductCell_Previews: PreviewProvider {
24     static var previews: some View {
25         ProductCell()
26     }
27 }

```

⌚ Check that your code appears as shown below, with all three Text subviews in just one VStack. Otherwise, undo and try again.

```

10
11 struct ProductCell: View {
12     var body: some View {
13         VStack {
14             Text("Text")
15                 .font(.title2)
16                 .fontWeight(.bold)
17             Text("Detail Text")
18                 .foregroundColor(Color.gray)
19             Text("Placeholder")
20         }
21 }

```





Open in app

```

10
11 struct ProductCell: View {
12     var body: some View {
13         VStack {
14             Text("Text")
15                 .font(.title2)
16                 .fontWeight(.bold)
17             Text("Detail Text")
18                 .foregroundColor(Color.gray)
19             Text("Tertiary Text")
20         }
21     }
22 }

```

Text
Detail Text
Tertiary Text

Font
Inherited
Weight Inherited
Color Inherited
Alignment Inherited
Line Limit - Inherited +

5. Vertical Stack Alignment

Our vertical stack contains the three text subviews, stacked like Lego® blocks. A `VStack` defaults to center aligning its subviews, but we want this `VStack` to align them against the leading edge (the left edge in English systems).

- 👉 In the preview, select the `VStack`, by clicking just to the left or right of the `Text` subview. Alternatively, click in the white space away from the text subviews, then once anywhere on the text subviews.
- 👉 Confirm that Xcode shows the `VStack` selected in the code, preview and Attributes inspector, as shown below.

```

10
11 struct ProductCell: View {
12     var body: some View {
13         VStack {
14             Text("Text")
15                 .font(.title2)
16                 .fontWeight(.bold)
17             Text("Detail Text")
18                 .foregroundColor(Color.gray)
19             Text("Tertiary Text")
20         }
21     }
22 }

```

Text
Detail Text
Tertiary Text

Vertical Stack
Spacing - Inherited +
Alignment **Leading** **Center** **Trailing** **Justified**

Modifiers

- 👉 In the Attributes inspector, next to `Alignment`, click on the leading (first) icon.

```

10
11 struct ProductCell: View {
12     var body: some View {
13         VStack(alignment: .leading) {
14             Text("Text")
15                 .font(.title2)
16                 .fontWeight(.bold)
17             Text("Detail Text")
18                 .foregroundColor(Color.gray)
19             Text("Tertiary Text")
20         }
21     }
22 }

```

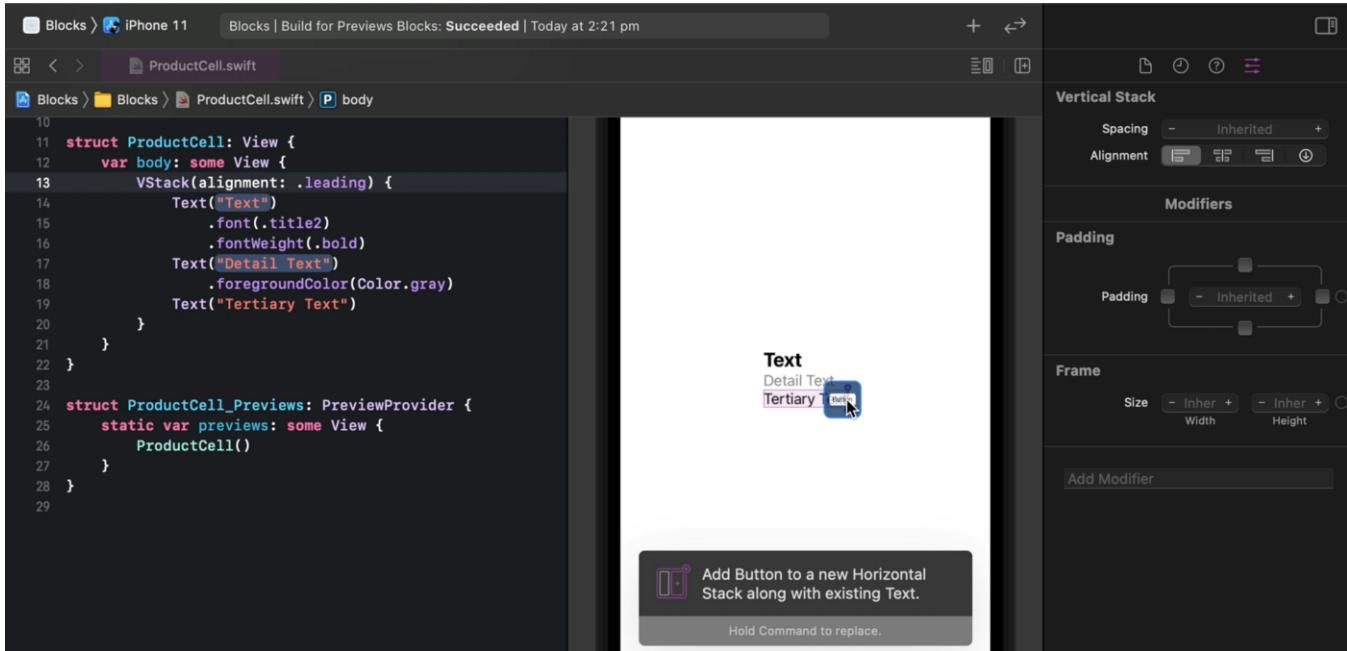
Text
Detail Text
Tertiary Text

Vertical Stack
Spacing - Inherited +
Alignment **Leading** **Center** **Trailing** **Justified**

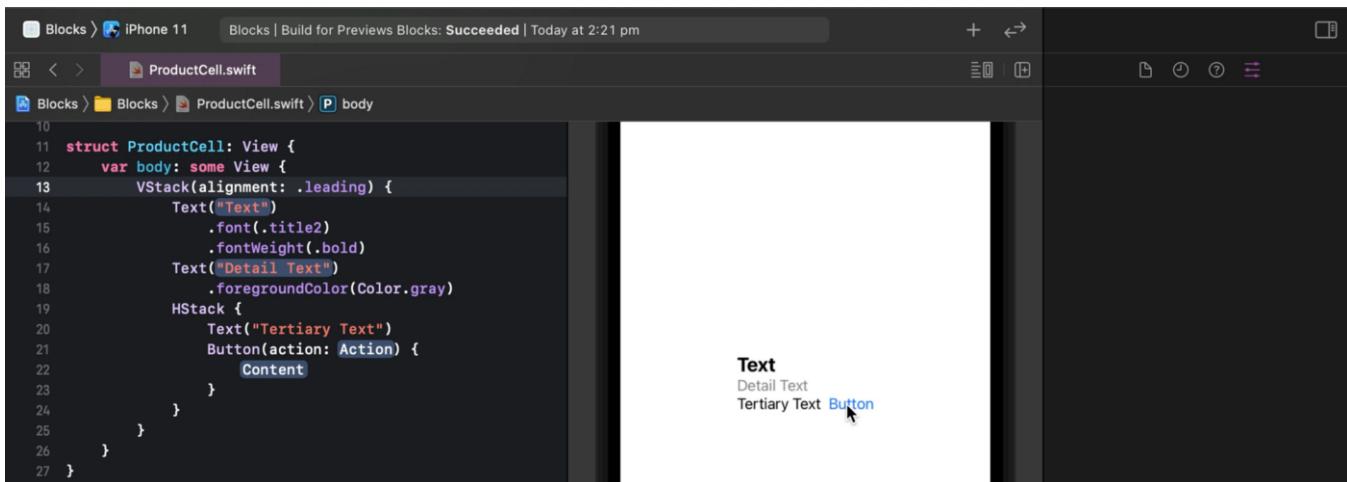
Modifiers

[Open in app](#)

👉 Show the Views Library again, but this time locate the `Button` item (at the top of the list). Drag `Button` into the preview to the right of the `Tertiary Text`. Ensure that the insertion bar shows only to the right of the `Tertiary Text` and not all of the text subviews.



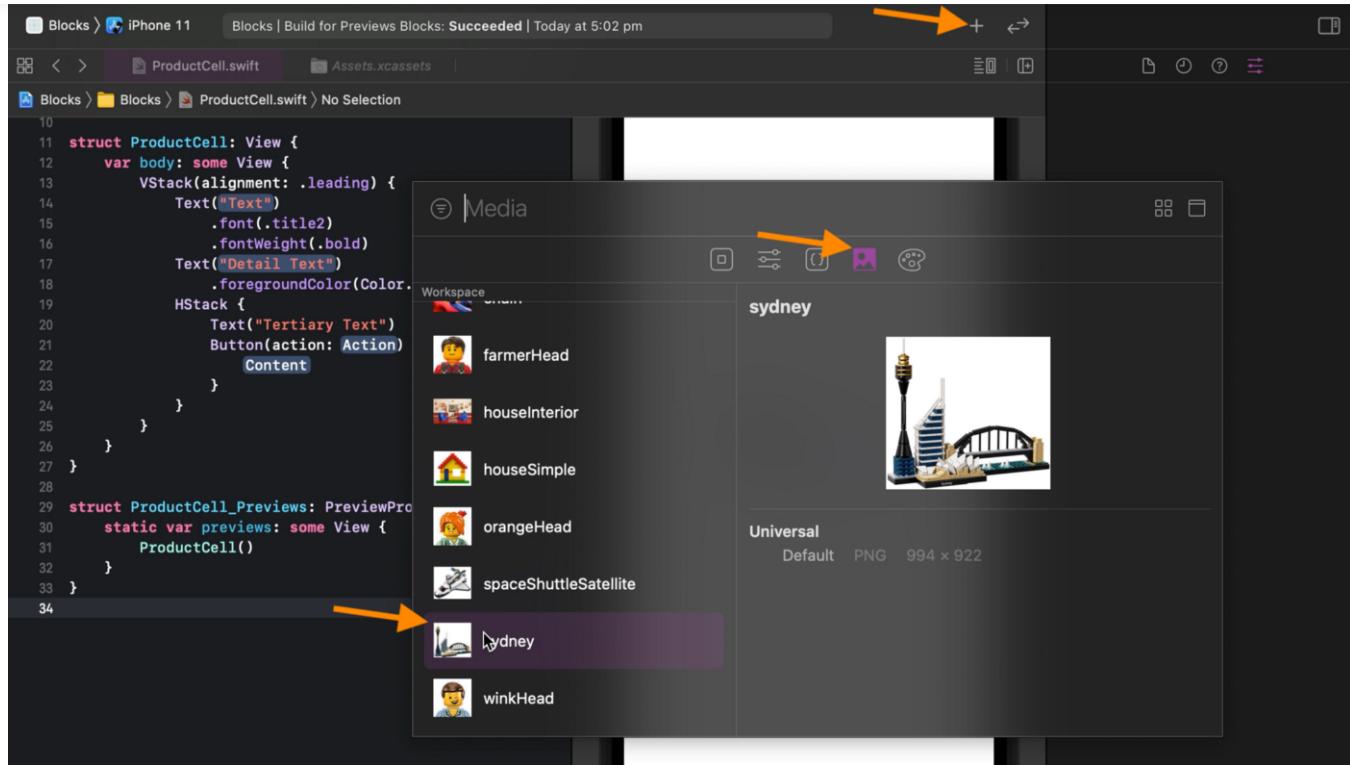
👁️ Xcode creates an `HStack` (horizontal stack), containing the `Tertiary Text` and the `Button` side by side.



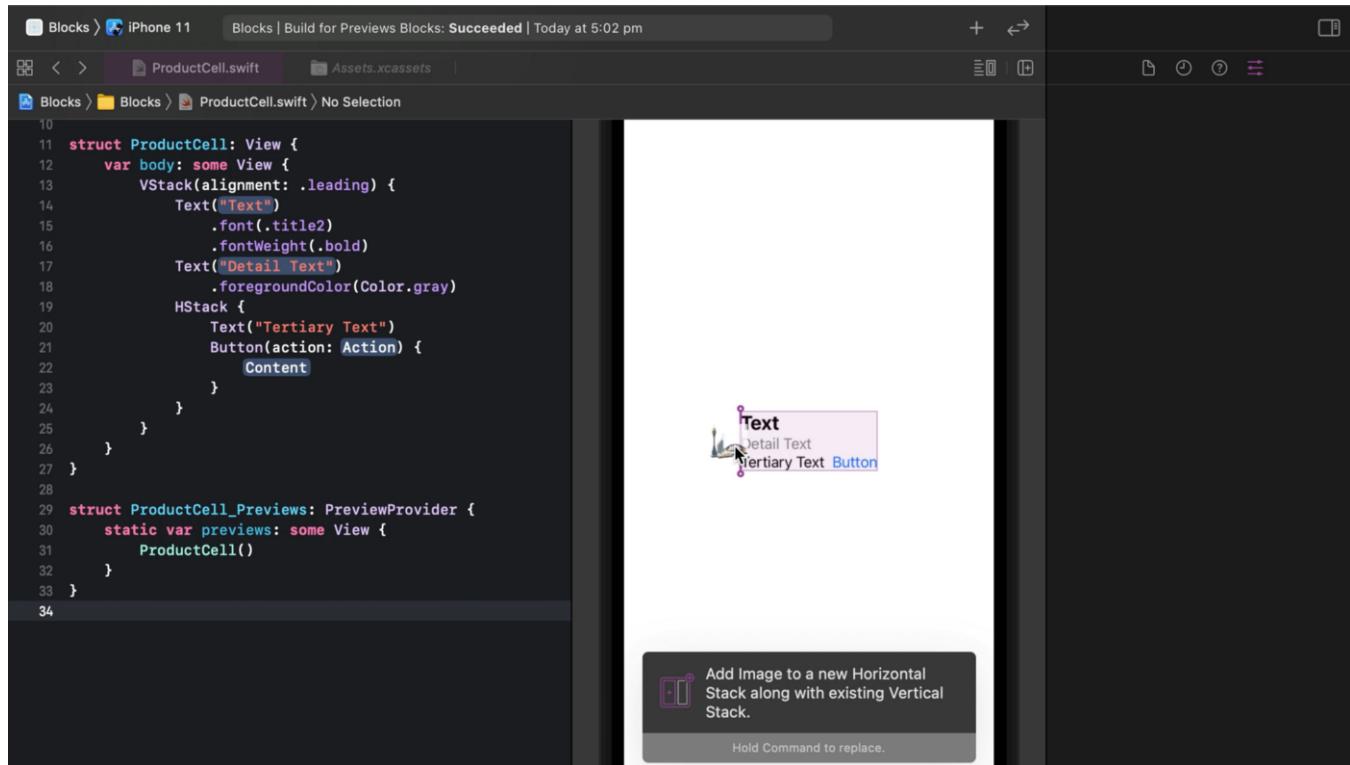
7. Add a Stretching Image in a Horizontal Stack

👉 Click the `+` button again to show the Library. Click on the `Media Library` tab to




[Open in app](#)


👉 Drag the `Sydney` image to the left of the text subviews. This time, ensure that the insertion bar appears next to **all** the text subviews, as shown below.



🌟 We could have instead dragged `Image` from the Views Library, but it defaults to



[Open in app](#)

- ⌚ The image loads but is so huge that we can barely see it on the iPhone screen. It also pushes all of the other content off-screen.

```
Blocks > iPhone 11 Blocks | Build for Previews Blocks: Succeeded | Today at 5:02 pm
Blocks > ProductCell.swift Assets.xcassets
Blocks > Blocks > ProductCell.swift No Selection

10
11 struct ProductCell: View {
12     var body: some View {
13         HStack {
14             Image("sydney")
15             VStack(alignment: .leading) {
16                 Text("Text")
17                     .font(.title2)
18                     .fontWeight(.bold)
19                 Text("Detail Text")
20                     .foregroundColor(Color.gray)
21             HStack {
22                 Text("Tertiary Text")
23                 Button(action: Action) {
24                     Content
25                 }
26             }
27         }
28     }
29 }
30
31
32 struct ProductCell_Previews: PreviewProvider {
33     static var previews: some View {
34         ProductCell()
35     }
36 }
```

We need to restrict the image to be 60 points wide, make it stretch/shrink to fit, and keep its aspect ratio.

- 👉 In the code, click on the `Image("Sydney")` line of code. We can't easily click to select it in the preview itself since it is out of bounds.
- 👉 In the Attributes inspector, set the `Frame Width` to `60`, and hit `return`.




[Open in app](#)

```

Blocks > iPhone 11   Blocks | Build for Previews Blocks: Succeeded | Today at 5:02 pm
Blocks > ProductCell.swift  Assets.xcassets
Blocks > ProductCell.swift > body
10
11 struct ProductCell: View {
12     var body: some View {
13         HStack {
14             Image("sydney")
15                 .frame(width: 60.0)
16             VStack(alignment: .leading) {
17                 Text("Text")
18                     .font(.title2)
19                     .fontWeight(.bold)
20                 Text("Detail Text")
21                     .foregroundColor(Color.gray)
22             HStack {
23                 Text("Tertiary Text")
24                 Button(action: Action) {
25                     Content
26                 }
27             }
28         }
29     }
30 }
31
32 struct ProductCell_Previews: PreviewProvider {
33     static var previews: some View {
34         ProductCell()
35     }
36 }
37
38

```

⌚ As you can see by the tall rectangular outline, the image is now restricted to a width of 60 points, allowing room for the rest of the content to appear again. However, the image content is drawing beyond its frame.

We need to restrict the image to shrink/stretch to only fill its frame.

👉 In the Attributes inspector, set the Resizing Mode to Stretch



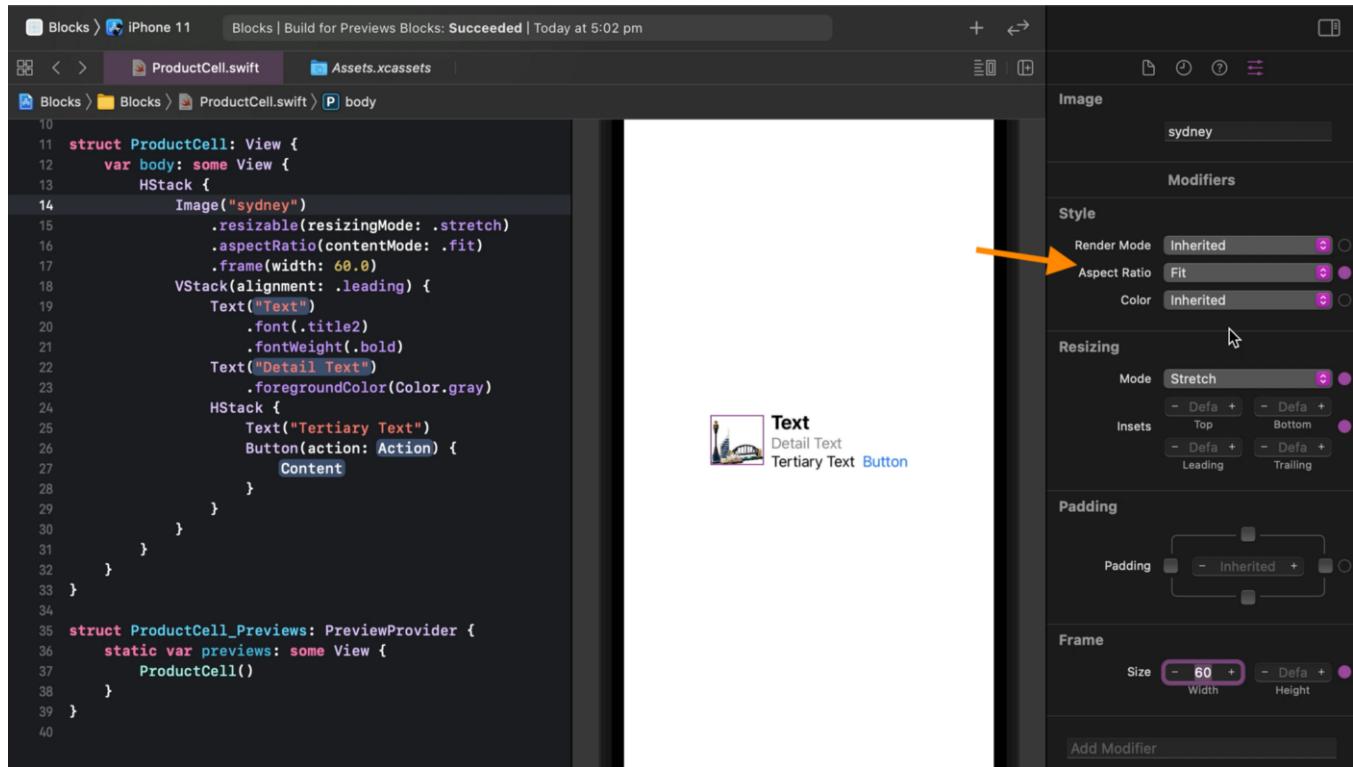


Open in app

- ⌚ The image is now stretching to fit its frame, but it is distorted — not stretching vertically by the same proportion as horizontally.

We can fix the vertical/horizontal distortion by setting the aspect ratio to fit.

- 👉 In the Attributes inspector, set the Aspect Ratio to Fit .



8. Image Symbol

Since the Library only facilitates adding an `Image` that uses an asset file, we used the `Sydney` image as a placeholder. However, it would be better to show something more general, like the “photo” system image. As we discussed earlier in [Tutorial 6](#), SwiftUI provides thousands of “SF Symbol” icons, including one called “photo”.

- 👉 In the code, after `Image` , select `("Sydney")` . Delete it and type an opening parenthesis again. Xcode shows a popup list of autocomplete options. Scroll down to the `Image(systemName:)` option and select it.



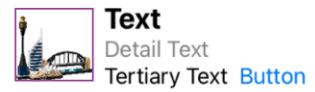


Open in app

```

11 struct ProductCell: View {
12     var body: some View {
13         HStack {
14             Image()
15                 .scale(.label)
16                 .orientation(.label)
17                 .decorative()
18                 .decorative(.bundle)
19                 .decorative(.scale)
20                 .decorative(.scale, orientation: .label)
21                 .systemName()
22                 .uiImage()
23             (systemName: String) -> Image
24

```



⌚ Xcode enters `Image(systemName: String)`, with the `String` placeholder selected.

```

11 struct ProductCell: View {
12     var body: some View {
13         HStack {
14             Image(systemName: String)
15                 .resizable(resizingMode: .stretch)
16                 .aspectRatio(contentMode: .fit)
17                 .frame(width: 60.0)
18             VStack(alignment: .leading) {
19                 Text("Text")
20                     .font(.title2)
21                     .fontWeight(.bold)
22                 Text("Detail Text")
23                     .foregroundColor(Color.gray)

```



👉 Replace `String` with "photo".

```

11 struct ProductCell: View {
12     var body: some View {
13         HStack {
14             Image(systemName: "photo")
15                 .resizable(resizingMode: .stretch)
16                 .aspectRatio(contentMode: .fit)
17                 .frame(width: 60.0)
18             VStack(alignment: .leading) {
19                 Text("Text")
20                     .font(.title2)
21                     .fontWeight(.bold)
22                 Text("Detail Text")
23                     .foregroundColor(Color.gray)

```

⌚ Xcode shows the photo icon, as the placeholder image in our cell.

9. Preview Size That Fits

The `ProductCell` layout is looking pretty good now.

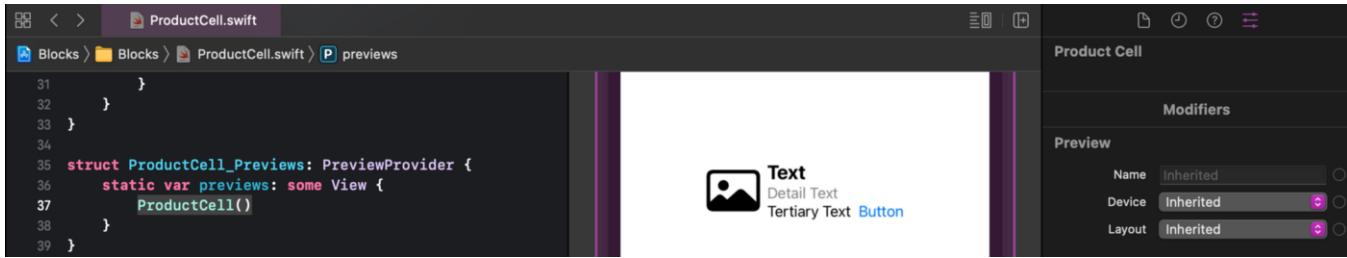
The preview is showing on a full-screen iPhone. This view is intended to be just part of



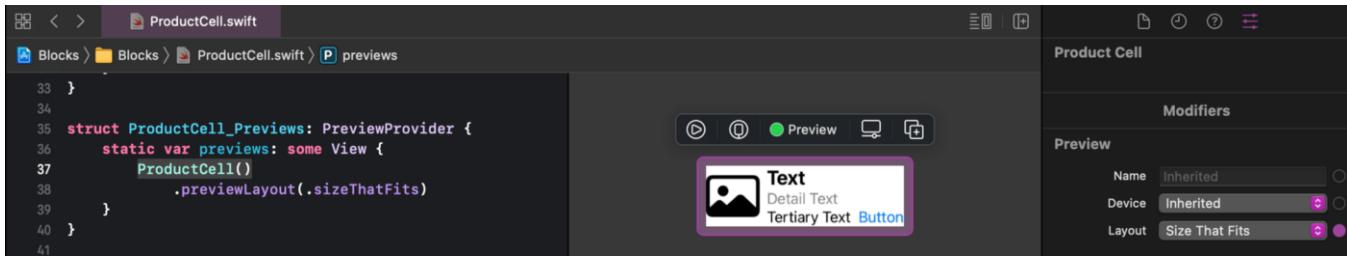


Open in app

- 👉 Click anywhere in the white space of the preview, outside of the subviews that we've added.



- 👉 In the Attributes inspector, set the Preview Layout to Size That Fits .

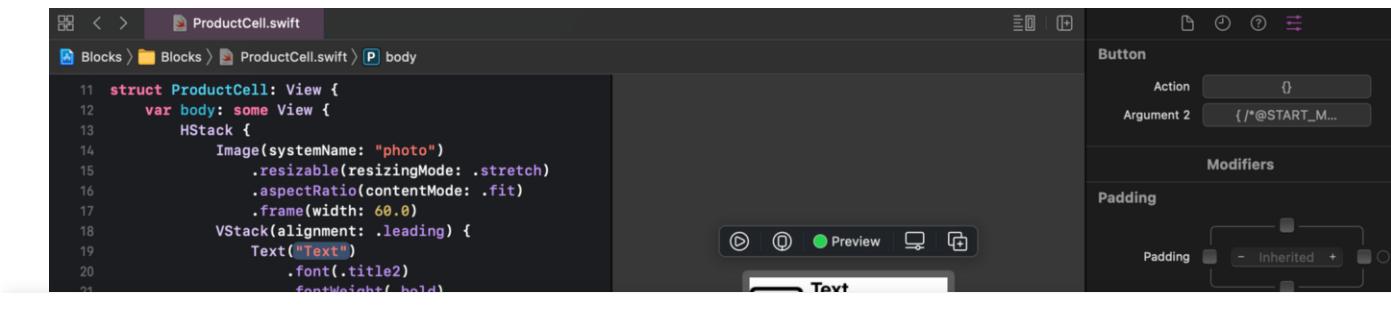


🐞 We could have set the preview layout to Size That Fits at the very beginning, before adding the subviews. This would have made it a bit easier to manage. Unfortunately, Xcode (up to at least version 12.5) doesn't allow us to drag subviews in to add an outer VStack or HStack , without the extra space on the outside. So, we deferred resizing the preview until after that was done.

10. Change the Button Title

The button is currently displaying the default title `Button` . Let's change it to `Buy` to match our design.

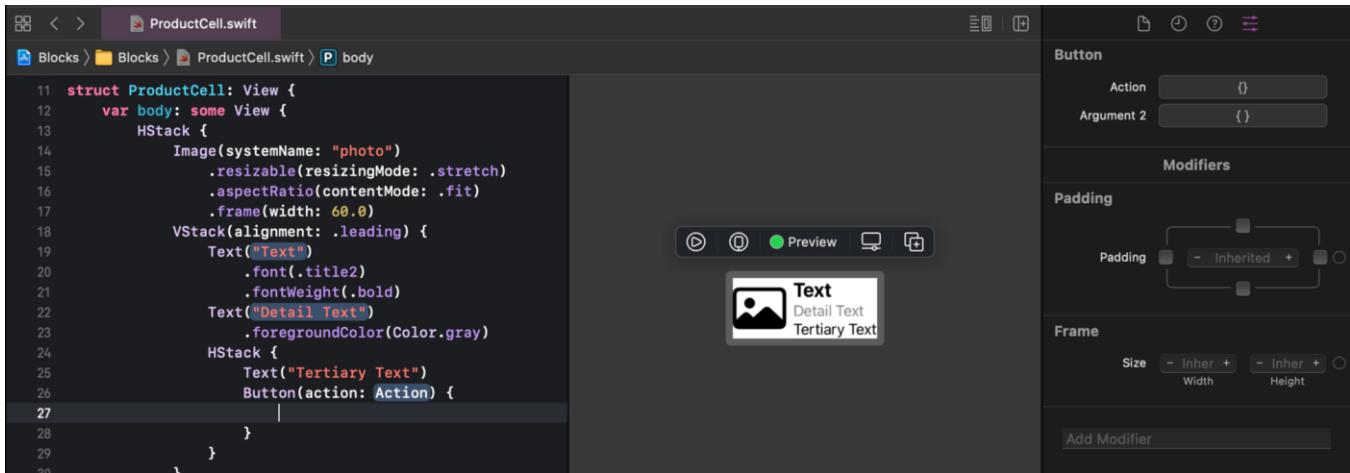
- 👉 In the preview, click once in the `Button` .



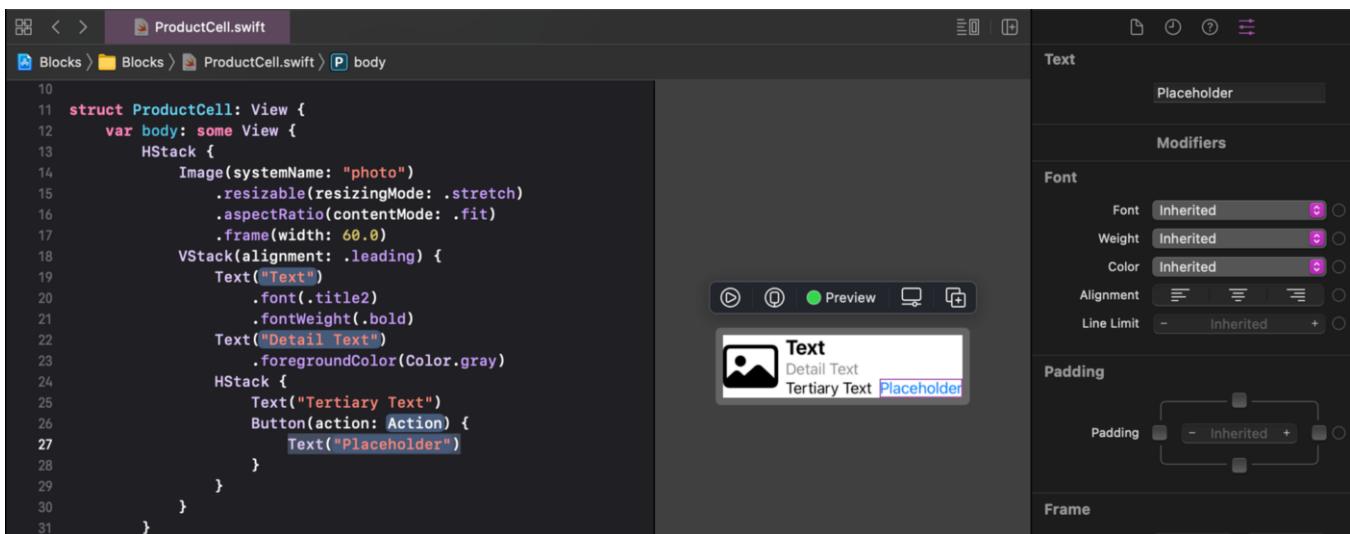
[Open in app](#)

🐞 As you can see, the Attributes inspector doesn't provide any way to change the title of the button. It just shows `Argument 2` with a disabled button.

👉 In the code, click on the `Content` below the `Button` line of code. Hit `delete` on the keyboard to remove it.



👉 Click the `+` button to show the Library. From the Views Library, drag a `Text` subview into the code, where the `Content` used to be in the `Button`.



👉 In the Attributes inspector, change the `Text` word from `Placeholder` to `Buy` and hit `return`.





Open in app

The screenshot shows the Xcode interface with the file `ProductCell.swift` open. The code defines a `ProductCell` struct with a `body` property. Inside, there's an `HStack` containing an `Image` and a `VStack`. The `VStack` has alignment set to `.leading` and contains three `Text` elements: "Text" (bold), "Detail Text" (gray), and "Tertiary Text". Below this is another `HStack` with "Tertiary Text" and a `Button` labeled "Buy". To the right, the Xcode preview pane shows a dark-themed interface with a photo placeholder, the three text labels, and the "Buy" button.

```

10
11 struct ProductCell: View {
12     var body: some View {
13         HStack {
14             Image(systemName: "photo")
15                 .resizable(resizingMode: .stretch)
16                 .aspectRatio(contentMode: .fit)
17                 .frame(width: 60.0)
18         VStack(alignment: .leading) {
19             Text("Text")
20                 .font(.title2)
21                 .fontWeight(.bold)
22             Text("Detail Text")
23                 .foregroundColor(Color.gray)
24             HStack {
25                 Text("Tertiary Text")
26                 Button(action: Action) {
27                     Text("Buy")
28                 }
29             }
30         }
31     }
32 }

```

- ⌚ Check that your code and preview appear as shown above.

11. Commit Changes

As you've done before:

- 👉 Choose Commit from the Source Control menu.
- 👉 Enter a description such as: `ProductCell`
- 👉 Click on the `Commit` button.

12. Next...

We have created a `ProductCell`, including subviews for text, an image and a button.

In [Tutorial 15](#) we will create a `Product` model, some mock products and show them in a list in the app.

See upcoming tutorials in the [table of contents](#). Follow the author to be notified of more articles. If you liked this article, clap for it (up to 50 times).

💬 If you have any questions or comments, please add a response below.

This series is released via [Next Level Swift](#). Subscribe to keep updated and never miss



[Open in app](#)

Next Level Swift

Next Level Swift aims at sharing knowledge and insights into better programming for iOS and is dedicated to help...

medium.nextlevelsweet.com

We are always looking for talented and passionate Swift developers! Feel free to check out our writer's section and find out how you can share your knowledge with the Next Level Swift Community!

Sign up for Next Level Swift Newsletter

By Next Level Swift

Get all the latest articles, posts and news straight to your mailbox! [Take a look.](#)

[Get this newsletter](#)

Emails will be sent to research2learn@yahoo.co.uk.
[Not you?](#)



[Open in app](#)Published in Next Level Swift · [Following](#) ▾Tom Brodhurst-Hill · [Following](#)

Jul 7, 2021 · 6 min read ★

...

Create a Model, Mock, and List of Cells

Build an App Like Lego, with SwiftUI — Tutorial 15

The screenshot shows the Xcode interface with several files open in the left sidebar: ProductCell.swift, Product+Mock.swift, ProductCell+Product.swift, Article.swift, and Article+M. The main editor area contains Swift code for a ProductCell extension and a ProductCell+Product.swift struct. The right side of the screen displays the Xcode Preview pane, which shows a detailed view of a "Space Shuttle" product cell. The preview includes a small image of a space shuttle, the product name "Space Shuttle", a description of the shuttle's features, and a price of "75.0". The Xcode interface also shows various settings for the product cell, such as modifiers, color schemes, dynamic types, padding, and frame sizes.

```

1 // 
2 //  ProductCell+Product.swift
3 //  Blocks
4 //
5 //  Created by Tom Brodhurst-Hill on 10/3/21.
6 //  Copyright © 2021 BareFetWare. All rights reserved.
7 //

8 import SwiftUI

9
10 extension ProductCell {
11     init(product: Product) {
12         self.init(
13             image: Image(product.imageName),
14             text: Text(product.name),
15             detailText: Text(product.description),
16             tertiaryText: Text(String(describing: product.price)))
17     }
18 }
19
20 }

21 struct ProductCell_Product_Previews: PreviewProvider {
22     static var previews: some View {
23         ProductCell(product: .spaceShuttle)
24             .previewLayout(.sizeThatFits)
25     }
26 }
27 }
28 
```

1. Introduction

A properly architected app is built from the blocks of a view, a model, and some kind of controller or presenter to inject the model into the view. Keeping the responsibilities separated makes the blocks reusable and more easily understood.

SwiftUI marks the end of the need for a “view controller”, which was a hallmark of the previous UIKit way of building apps. That is thankfully true. However, we still logically need some small piece of code to inject the model properties into a view, so I will refer to it here as the “controller” or “presenter”. In reality, this controller is just a simple small extension of the view that maps each model property into a view property.

In a later tutorial, we will also introduce the “view model”.



[Open in app](#)

In the previous [Tutorial 14](#), we created a `ProductCell` view. In this Tutorial 15, we will create a `Product` model and mock values to be shown in that view.

This is similar to when we created the `Article` model for the `NewsCell`. Refer back to the [earlier tutorials](#) if anything here seems unfamiliar.

We will pick up here where the last tutorial left off. Ideally, you have completed the [previous tutorials in this series](#). Or, you can [download](#) the prepared project, ready to start this tutorial.

2. Extract Properties of a View

In Xcode, select the `ProductCell.swift` file that we created in the previous [Tutorial 14](#).

👉 Insert the following properties before the `var` body:





Open in app

⌚ Your code should look like this:

```

11 struct ProductCell: View {
12
13     let image: Image
14     let text: Text
15     let detailText: Text
16     let tertiaryText: Text
17
18     var body: some View {
19         HStack {
20             Image(systemName: "photo")
21                 .resizable(resizingMode: .stretch)

```

⌚ Xcode now complains in the PreviewProvider code (at the bottom of the file) that the ProductCell() instance is missing arguments.

```

40
41 struct ProductCell_Previews: PreviewProvider {
42     static var previews: some View {
43         ProductCell() ⚡ Missing arguments for parameters 'image', 'text', 'detailText', 'tertiaryText' in call
44             .previewLayout(.sizeThatFits)
45     }
46 }
47

```

👉 Click on the stop icon in the error pop over to expand the detail. Click the Fix button.

```

40
41 struct ProductCell_Previews: PreviewProvider {
42     static var previews: some View {
43         ProductCell()
44             .previewLayout( ⚡ Missing arguments for parameters 'image', 'text', 'detailText',
45         'tertiaryText' in call
46     }
47

```

Missing arguments for parameters 'image', 'text', 'detailText', 'tertiaryText' in call
Insert 'image: <#Image#>, text: <#Text#>, detailText: <#Text#>, tertiaryText: <#Text#>' Fix



[Open in app](#)

Structure > Re-Indent to tidy up the indenting, if needed.

```

40
41 struct ProductCell_Previews: PreviewProvider {
42     static var previews: some View {
43         ProductCell(
44             image: Image,    ✘ Cannot convert value of type 'Text' to expected argument type 'Image'
45             text: Text,
46             detailText: Text,
47             tertiaryText: Text
48         )
49         .previewLayout(.sizeThatFits)
50     }
51 }
```

👉 Replace each of the placeholders (`Image` and `Text`) in the `PreviewProvider` code with the copied placeholders from the `var` body. For example, replace `Image` with `Image(systemName: "photo")`.



```

40
41 struct ProductCell_Previews: PreviewProvider {
42     static var previews: some View {
43         ProductCell(
44             image: Image(systemName: "photo"),
45             text: Text("Text"),
46             detailText: Text("Detail Text"),
47             tertiaryText: Text("Tertiary Text")
48         )
49         .previewLayout(.sizeThatFits)
50     }
51 }
```

👁 Check that the code compiles without error and that the preview displays correctly.

👉 Replace each of the placeholder `Image` and `Text` views in the `var` body with the corresponding new property names. For example, replace `Image(systemName: "photo")` with `image`.



[Open in app](#)

```

1 // ProductCell.swift
2 // Blocks
3 //
4 // Created by Tom Brodhurst-Hill on 10/3/21.
5 // Copyright © 2021 BareFeetWare. All rights reserved.
6 //
7 //
8
9 import SwiftUI
10
11 struct ProductCell: View {
12
13     let image: Image
14     let text: Text
15     let detailText: Text
16     let tertiaryText: Text
17
18     var body: some View {
19         HStack {
20             image
21                 .resizable(resizingMode: .stretch)
22                 .aspectRatio(contentMode: .fit)
23                 .frame(width: 60.0)
24             VStack(alignment: .leading) {
25                 text
26                     .font(.title2)
27                     .fontWeight(.bold)
28                 detailText
29                     .foregroundColor(Color.gray)
30             HStack {
31                 tertiaryText
32                     Button(action: Action) {
33                         Text("Buy")
34                     }
35                 }
36             }
37         }
38     }
39 }
40
41 struct ProductCell_Previews: PreviewProvider {
42     static var previews: some View {
43         ProductCell(
44             image: Image(systemName: "photo"),
45             text: Text("Text"),
46             detailText: Text("Detail Text"),
47             tertiaryText: Text("Tertiary Text")
48         )
49         .previewLayout(.sizeThatFits)
50     }
51 }
52

```



3. Product Model

👉 Create a new file, using the `Swift File` template. Name it `Product.swift`.



[Open in app](#)

👉 Add the Product structure code below:

👁️ Confirm that your code looks like this, with no errors:



[Open in app](#)

Blocks > Blocks > Product.swift > No Selection

```
1 //  
2 // Product.swift  
3 // Blocks  
4 //  
5 // Created by Tom Brodhurst-Hill on 10/3/21.  
6 // Copyright © 2021 BareFeetWare. All rights reserved.  
7 //  
8  
9 import Foundation  
10  
11 struct Product: Identifiable {  
12     let id: Int  
13     let name: String  
14     let description: String  
15     let price: Double  
16     let imageName: String  
17 }  
18
```

4. Product Mocks

👉 Create a new Swift File named `Product+Mock.swift`.

👉 Add the extension code from the following:



[Open in app](#)

- 👁 Check that the code compiles without any errors, and looks like this:



[Open in app](#)

Blocks > Blocks > Product+Mock.swift > No Selection

```

1  //
2  //  Product+Mock.swift
3  //  Blocks
4  //
5  //  Created by Tom Brodhurst-Hill on 10/3/21.
6  //  Copyright © 2021 BareFeetWare. All rights reserved.
7  //
8
9  import Foundation
10
11 extension Product {
12
13     static let sydney: Self = .init(
14         id: 1,
15         name: "Sydney Opera House & Harbour Bridge",
16         description: "Capture the architectural essence and splendor of Sydney with
17             this magnificent set that brings together the iconic Sydney Opera House™,
18             Sydney Harbour Bridge, Sydney Tower and Deutsche Bank Place, in an
19             inspirational skyline setting. Each individual LEGO® structure provides a
20             unique and rewarding building experience with true-to-life color and
21             relative scale depiction. Sydney's sparkling harbor is represented in the
22             tiled baseplate, adding an extra dimension and feel of authenticity to
23             this detailed recreation of one of the world's most glamorous cities.",
24         price: 98.0,
25         imageName: "sydney"
26     )
27
28     static let spaceShuttle: Self = .init(
29         id: 2,
30         name: "Space Shuttle",
31         description: "Carry out daring space missions with the Space Shuttle Explorer,
32             featuring an authentic white, black and gray color scheme, large engines,
33             opening payload bay with robotic arm, satellite with foldout wings and a
34             minifigure cockpit with tinted canopy. This 3-in-1 LEGO® Creator model
             rebuilds into a Moon Station or a Space Rover for further outer-space
             adventures. Also includes a minifigure.",
35         price: 75.0,
36         imageName: "spaceShuttleSatellite"
37     )
38
39 }
40
41 extension Product {
42     static let mocks: [Self] = [.sydney, .spaceShuttle]
43 }
44

```

5. Product Presenter

Now we have a `Product` model and a `ProductCell` view. We need to define how a `Product`'s properties are presented in a `ProductCell`.



[Open in app](#)

⌚ As you're typing, Xcode should offer autocompletions to make it easier, such as this one after typing `init(`:

```
9 import SwiftUI
10
11 extension ProductCell {
12     init(product: Product) {
13         self.init| ✖ Type of expression is ambiguous without mor...
14     }
15 }
```

M `init(product:)`

M `init(image:text:detailText:tertiaryText:)`

M `init(image: Image, text: Text, detailText: Text, tertiaryText: Text) -`



[Open in app](#)

👉 In the `var previews`, replace the `ProductCell_Product()` template code with:

👁️ Confirm that the code compiles and the preview shows as below:





Open in app

The screenshot shows the Xcode interface with the ProductCell.swift file open. The code defines a ProductCell view with a preview provider for the Space Shuttle product. A callout bubble highlights the preview area, showing a Space Shuttle model with its details: Name (Space Shuttle), Device (Inherited), Layout (Size That Fits), Color Scheme (Inherited), Dynamic Type (Inherited), Padding (Inherited), and Frame (Width and Height both set to Inherited). The preview itself shows a white Space Shuttle model with black and grey accents, a cockpit window, and a payload bay.

```

1 // 
2 //  ProductCell+Product.swift
3 //  Blocks
4 //
5 //  Created by Tom Brodhurst-Hill on 10/3/21.
6 //  Copyright © 2021 BareFeetWare. All rights reserved.
7 //
8
9 import SwiftUI
10
11 extension ProductCell {
12     init(product: Product) {
13         self.init(
14             image: Image(product.imageName),
15             text: Text(product.name),
16             detailText: Text(product.description),
17             tertiaryText: Text(String(describing: product.price)))
18     }
19 }
20
21 struct ProductCell_Product_Previews: PreviewProvider {
22     static var previews: some View {
23         ProductCell(product: .spaceShuttle)
24             .previewLayout(.sizeThatFits)
25     }
26 }
27
28

```

6. List of ProductCells

We're nearly there. We have a `Product` model, with a `mocks` array of sample data. We have a `ProductCell` view and a way to present a `Product` model in the cell.

Now we just need a scene to display the products in a list.

In the `ContentView.swift` file, we already have a placeholder scene for the products, complete with a navigation view. We just need to replace the placeholder `Text("Tab Content 2")` with a list of products.

👉 Switch to the `ContentView.swift` file.

👉 From the Views Library, drag a `Navigation Link` to insert a line above the `Text("Tab Content 2")` line of code.





Open in app

```

11 struct ContentView: View {
12     var body: some View {
13         TabView(selection: Selection) {
14             NavigationView {
15                 List(Article.mocks) { article in
16                     NavigationLink(destination: Destination) {
17                         NewsCell(article: article)
18                     }
19                 }
20                 .navigationTitle("News")
21             }
22             .tabItem { Label("News", systemImage: "newspaper.fill") }
23             .tag(1)
24             NavigationView {
25                 NavigationLink(destination: Destination) {
26                     Label Content
27                 }
28                 Text("Tab Content 2")
29                 .navigationTitle("Products")
30             }
31             .tabItem { Label("Products", systemImage: "cart.fill") }
32             .tag(2)

```

👉 Delete the `Text("Tab Content 2")` line of code.

```

24         NavigationView {
25             NavigationLink(destination: Destination) {
26                 Label Content
27             }
28             .navigationTitle("Products")
29         }
30         .tabItem { Label("Products", systemImage: "cart.fill") }
31         .tag(2)

```

👉 Command click on the `NavigationLink` code. In the Actions popup menu, select `Embed in List`.





Open in app

```

11 struct ContentView: View {
12     var body: some View {
13         Embed in ZStack
14             .background(LinearGradient(gradient: Gradient(colors: [Color.pink, Color.purple]), startPoint: .top, endPoint: .bottom))
15             .frame(maxWidth: .infinity, maxHeight: .infinity)
16             .ignoresSafeArea()
17
18             Embed in List
19                 .listStyle(.plain)
20
21             Group {
22                 Make Conditional
23                 Repeat
24                 Embed...
25                 Extract to Variable
26                 Extract to Method
27                 Extract All Occurrences
28
29             }
30
31         NavigationView {
32             List(0 ..< 5) { item in
33                 NavigationLink(destination: Destination) {
34                     Label Content
35                 }
36                 Text("Tab Content 2")
37                     .navigationTitle("Products")
38
39             }
40
41         }
42
43         Image(systemName: "newspaper.fill")
44             .frame(width: 100, height: 100)
45             .clipShape(Circle())
46             .font(.largeTitle)
47             .padding()
48             .background(LinearGradient(gradient: Gradient(colors: [Color.pink, Color.purple]), startPoint: .top, endPoint: .bottom))
49             .ignoresSafeArea()
50
51         Text("Article in List")
52             .font(.title)
53             .padding()
54
55         Text("Navigation View")
56             .font(.title)
57             .padding()
58
59         Text("List Item")
60             .font(.title)
61             .padding()
62
63         Text("Image")
64             .font(.title)
65             .padding()
66
67         Text("Text")
68             .font(.title)
69             .padding()
70
71         Text("Background")
72             .font(.title)
73             .padding()
74
75         Text("Conditional")
76             .font(.title)
77             .padding()
78
79         Text("Repeat")
80             .font(.title)
81             .padding()
82
83         Text("Embed")
84             .font(.title)
85             .padding()
86
87         Text("Extract to Variable")
88             .font(.title)
89             .padding()
90
91         Text("Extract to Method")
92             .font(.title)
93             .padding()
94
95         Text("Extract All Occurrences")
96             .font(.title)
97             .padding()
98
99         Text("Navigation View")
100            .font(.title)
101            .padding()
102
103         Text("List Item")
104            .font(.title)
105            .padding()
106
107         Text("Image")
108            .font(.title)
109            .padding()
110
111         Text("Text")
112            .font(.title)
113            .padding()
114
115         Text("Background")
116            .font(.title)
117            .padding()
118
119         Text("Conditional")
120            .font(.title)
121            .padding()
122
123         Text("Repeat")
124            .font(.title)
125            .padding()
126
127         Text("Embed")
128            .font(.title)
129            .padding()
130
131         Text("Extract to Variable")
132            .font(.title)
133            .padding()
134
135         Text("Extract to Method")
136            .font(.title)
137            .padding()
138
139         Text("Extract All Occurrences")
140            .font(.title)
141            .padding()
142
143         Text("Navigation View")
144            .font(.title)
145            .padding()
146
147         Text("List Item")
148            .font(.title)
149            .padding()
150
151         Text("Image")
152            .font(.title)
153            .padding()
154
155         Text("Text")
156            .font(.title)
157            .padding()
158
159         Text("Background")
160            .font(.title)
161            .padding()
162
163         Text("Conditional")
164            .font(.title)
165            .padding()
166
167         Text("Repeat")
168            .font(.title)
169            .padding()
170
171         Text("Embed")
172            .font(.title)
173            .padding()
174
175         Text("Extract to Variable")
176            .font(.title)
177            .padding()
178
179         Text("Extract to Method")
180            .font(.title)
181            .padding()
182
183         Text("Extract All Occurrences")
184            .font(.title)
185            .padding()
186
187         Text("Navigation View")
188            .font(.title)
189            .padding()
190
191         Text("List Item")
192            .font(.title)
193            .padding()
194
195         Text("Image")
196            .font(.title)
197            .padding()
198
199         Text("Text")
200            .font(.title)
201            .padding()
202
203         Text("Background")
204            .font(.title)
205            .padding()
206
207         Text("Conditional")
208            .font(.title)
209            .padding()
210
211         Text("Repeat")
212            .font(.title)
213            .padding()
214
215         Text("Embed")
216            .font(.title)
217            .padding()
218
219         Text("Extract to Variable")
220            .font(.title)
221            .padding()
222
223         Text("Extract to Method")
224            .font(.title)
225            .padding()
226
227         Text("Extract All Occurrences")
228            .font(.title)
229            .padding()
230
231         Text("Navigation View")
232            .font(.title)
233            .padding()
234
235         Text("List Item")
236            .font(.title)
237            .padding()
238
239         Text("Image")
240            .font(.title)
241            .padding()
242
243         Text("Text")
244            .font(.title)
245            .padding()
246
247         Text("Background")
248            .font(.title)
249            .padding()
250
251         Text("Conditional")
252            .font(.title)
253            .padding()
254
255         Text("Repeat")
256            .font(.title)
257            .padding()
258
259         Text("Embed")
260            .font(.title)
261            .padding()
262
263         Text("Extract to Variable")
264            .font(.title)
265            .padding()
266
267         Text("Extract to Method")
268            .font(.title)
269            .padding()
270
271         Text("Extract All Occurrences")
272            .font(.title)
273            .padding()
274
275         Text("Navigation View")
276            .font(.title)
277            .padding()
278
279         Text("List Item")
280            .font(.title)
281            .padding()
282
283         Text("Image")
284            .font(.title)
285            .padding()
286
287         Text("Text")
288            .font(.title)
289            .padding()
290
291         Text("Background")
292            .font(.title)
293            .padding()
294
295         Text("Conditional")
296            .font(.title)
297            .padding()
298
299         Text("Repeat")
300            .font(.title)
301            .padding()
302
303         Text("Embed")
304            .font(.title)
305            .padding()
306
307         Text("Extract to Variable")
308            .font(.title)
309            .padding()
310
311         Text("Extract to Method")
312            .font(.title)
313            .padding()
314
315         Text("Extract All Occurrences")
316            .font(.title)
317            .padding()
318
319         Text("Navigation View")
320            .font(.title)
321            .padding()
322
323         Text("List Item")
324            .font(.title)
325            .padding()
326
327         Text("Image")
328            .font(.title)
329            .padding()
330
331         Text("Text")
332            .font(.title)
333            .padding()
334
335         Text("Background")
336            .font(.title)
337            .padding()
338
339         Text("Conditional")
340            .font(.title)
341            .padding()
342
343         Text("Repeat")
344            .font(.title)
345            .padding()
346
347         Text("Embed")
348            .font(.title)
349            .padding()
350
351         Text("Extract to Variable")
352            .font(.title)
353            .padding()
354
355         Text("Extract to Method")
356            .font(.title)
357            .padding()
358
359         Text("Extract All Occurrences")
360            .font(.title)
361            .padding()
362
363         Text("Navigation View")
364            .font(.title)
365            .padding()
366
367         Text("List Item")
368            .font(.title)
369            .padding()
370
371         Text("Image")
372            .font(.title)
373            .padding()
374
375         Text("Text")
376            .font(.title)
377            .padding()
378
379         Text("Background")
380            .font(.title)
381            .padding()
382
383         Text("Conditional")
384            .font(.title)
385            .padding()
386
387         Text("Repeat")
388            .font(.title)
389            .padding()
390
391         Text("Embed")
392            .font(.title)
393            .padding()
394
395         Text("Extract to Variable")
396            .font(.title)
397            .padding()
398
399         Text("Extract to Method")
400            .font(.title)
401            .padding()
402
403         Text("Extract All Occurrences")
404            .font(.title)
405            .padding()
406
407         Text("Navigation View")
408            .font(.title)
409            .padding()
410
411         Text("List Item")
412            .font(.title)
413            .padding()
414
415         Text("Image")
416            .font(.title)
417            .padding()
418
419         Text("Text")
420            .font(.title)
421            .padding()
422
423         Text("Background")
424            .font(.title)
425            .padding()
426
427         Text("Conditional")
428            .font(.title)
429            .padding()
430
431         Text("Repeat")
432            .font(.title)
433            .padding()
434
435         Text("Embed")
436            .font(.title)
437            .padding()
438
439         Text("Extract to Variable")
440            .font(.title)
441            .padding()
442
443         Text("Extract to Method")
444            .font(.title)
445            .padding()
446
447         Text("Extract All Occurrences")
448            .font(.title)
449            .padding()
450
451         Text("Navigation View")
452            .font(.title)
453            .padding()
454
455         Text("List Item")
456            .font(.title)
457            .padding()
458
459         Text("Image")
460            .font(.title)
461            .padding()
462
463         Text("Text")
464            .font(.title)
465            .padding()
466
467         Text("Background")
468            .font(.title)
469            .padding()
470
471         Text("Conditional")
472            .font(.title)
473            .padding()
474
475         Text("Repeat")
476            .font(.title)
477            .padding()
478
479         Text("Embed")
480            .font(.title)
481            .padding()
482
483         Text("Extract to Variable")
484            .font(.title)
485            .padding()
486
487         Text("Extract to Method")
488            .font(.title)
489            .padding()
490
491         Text("Extract All Occurrences")
492            .font(.title)
493            .padding()
494
495         Text("Navigation View")
496            .font(.title)
497            .padding()
498
499         Text("List Item")
500            .font(.title)
501            .padding()
502
503         Text("Image")
504            .font(.title)
505            .padding()
506
507         Text("Text")
508            .font(.title)
509            .padding()
510
511         Text("Background")
512            .font(.title)
513            .padding()
514
515         Text("Conditional")
516            .font(.title)
517            .padding()
518
519         Text("Repeat")
520            .font(.title)
521            .padding()
522
523         Text("Embed")
524            .font(.title)
525            .padding()
526
527         Text("Extract to Variable")
528            .font(.title)
529            .padding()
530
531         Text("Extract to Method")
532            .font(.title)
533            .padding()
534
535         Text("Extract All Occurrences")
536            .font(.title)
537            .padding()
538
539         Text("Navigation View")
540            .font(.title)
541            .padding()
542
543         Text("List Item")
544            .font(.title)
545            .padding()
546
547         Text("Image")
548            .font(.title)
549            .padding()
550
551         Text("Text")
552            .font(.title)
553            .padding()
554
555         Text("Background")
556            .font(.title)
557            .padding()
558
559         Text("Conditional")
560            .font(.title)
561            .padding()
562
563         Text("Repeat")
564            .font(.title)
565            .padding()
566
567         Text("Embed")
568            .font(.title)
569            .padding()
570
571         Text("Extract to Variable")
572            .font(.title)
573            .padding()
574
575         Text("Extract to Method")
576            .font(.title)
577            .padding()
578
579         Text("Extract All Occurrences")
580            .font(.title)
581            .padding()
582
583         Text("Navigation View")
584            .font(.title)
585            .padding()
586
587         Text("List Item")
588            .font(.title)
589            .padding()
590
591         Text("Image")
592            .font(.title)
593            .padding()
594
595         Text("Text")
596            .font(.title)
597            .padding()
598
599         Text("Background")
600            .font(.title)
601            .padding()
602
603         Text("Conditional")
604            .font(.title)
605            .padding()
606
607         Text("Repeat")
608            .font(.title)
609            .padding()
610
611         Text("Embed")
612            .font(.title)
613            .padding()
614
615         Text("Extract to Variable")
616            .font(.title)
617            .padding()
618
619         Text("Extract to Method")
620            .font(.title)
621            .padding()
622
623         Text("Extract All Occurrences")
624            .font(.title)
625            .padding()
626
627         Text("Navigation View")
628            .font(.title)
629            .padding()
630
631         Text("List Item")
632            .font(.title)
633            .padding()
634
635         Text("Image")
636            .font(.title)
637            .padding()
638
639         Text("Text")
640            .font(.title)
641            .padding()
642
643         Text("Background")
644            .font(.title)
645            .padding()
646
647         Text("Conditional")
648            .font(.title)
649            .padding()
650
651         Text("Repeat")
652            .font(.title)
653            .padding()
654
655         Text("Embed")
656            .font(.title)
657            .padding()
658
659         Text("Extract to Variable")
660            .font(.title)
661            .padding()
662
663         Text("Extract to Method")
664            .font(.title)
665            .padding()
666
667         Text("Extract All Occurrences")
668            .font(.title)
669            .padding()
670
671         Text("Navigation View")
672            .font(.title)
673            .padding()
674
675         Text("List Item")
676            .font(.title)
677            .padding()
678
679         Text("Image")
680            .font(.title)
681            .padding()
682
683         Text("Text")
684            .font(.title)
685            .padding()
686
687         Text("Background")
688            .font(.title)
689            .padding()
690
691         Text("Conditional")
692            .font(.title)
693            .padding()
694
695         Text("Repeat")
696            .font(.title)
697            .padding()
698
699         Text("Embed")
700            .font(.title)
701            .padding()
702
703         Text("Extract to Variable")
704            .font(.title)
705            .padding()
706
707         Text("Extract to Method")
708            .font(.title)
709            .padding()
710
711         Text("Extract All Occurrences")
712            .font(.title)
713            .padding()
714
715         Text("Navigation View")
716            .font(.title)
717            .padding()
718
719         Text("List Item")
720            .font(.title)
721            .padding()
722
723         Text("Image")
724            .font(.title)
725            .padding()
726
727         Text("Text")
728            .font(.title)
729            .padding()
730
731         Text("Background")
732            .font(.title)
733            .padding()
734
735         Text("Conditional")
736            .font(.title)
737            .padding()
738
739         Text("Repeat")
740            .font(.title)
741            .padding()
742
743         Text("Embed")
744            .font(.title)
745            .padding()
746
747         Text("Extract to Variable")
748            .font(.title)
749            .padding()
750
751         Text("Extract to Method")
752            .font(.title)
753            .padding()
754
755         Text("Extract All Occurrences")
756            .font(.title)
757            .padding()
758
759         Text("Navigation View")
760            .font(.title)
761            .padding()
762
763         Text("List Item")
764            .font(.title)
765            .padding()
766
767         Text("Image")
768            .font(.title)
769            .padding()
770
771         Text("Text")
772            .font(.title)
773            .padding()
774
775         Text("Background")
776            .font(.title)
777            .padding()
778
779         Text("Conditional")
780            .font(.title)
781            .padding()
782
783         Text("Repeat")
784            .font(.title)
785            .padding()
786
787         Text("Embed")
788            .font(.title)
789            .padding()
790
791         Text("Extract to Variable")
792            .font(.title)
793            .padding()
794
795         Text("Extract to Method")
796            .font(.title)
797            .padding()
798
799         Text("Extract All Occurrences")
800            .font(.title)
801            .padding()
802
803         Text("Navigation View")
804            .font(.title)
805            .padding()
806
807         Text("List Item")
808            .font(.title)
809            .padding()
810
811         Text("Image")
812            .font(.title)
813            .padding()
814
815         Text("Text")
816            .font(.title)
817            .padding()
818
819         Text("Background")
820            .font(.title)
821            .padding()
822
823         Text("Conditional")
824            .font(.title)
825            .padding()
826
827         Text("Repeat")
828            .font(.title)
829            .padding()
830
831         Text("Embed")
832            .font(.title)
833            .padding()
834
835         Text("Extract to Variable")
836            .font(.title)
837            .padding()
838
839         Text("Extract to Method")
840            .font(.title)
841            .padding()
842
843         Text("Extract All Occurrences")
844            .font(.title)
845            .padding()
846
847         Text("Navigation View")
848            .font(.title)
849            .padding()
850
851         Text("List Item")
852            .font(.title)
853            .padding()
854
855         Text("Image")
856            .font(.title)
857            .padding()
858
859         Text("Text")
860            .font(.title)
861            .padding()
862
863         Text("Background")
864            .font(.title)
865            .padding()
866
867         Text("Conditional")
868            .font(.title)
869            .padding()
870
871         Text("Repeat")
872            .font(.title)
873            .padding()
874
875         Text("Embed")
876            .font(.title)
877            .padding()
878
879         Text("Extract to Variable")
880            .font(.title)
881            .padding()
882
883         Text("Extract to Method")
884            .font(.title)
885            .padding()
886
887         Text("Extract All Occurrences")
888            .font(.title)
889            .padding()
890
891         Text("Navigation View")
892            .font(.title)
893            .padding()
894
895         Text("List Item")
896            .font(.title)
897            .padding()
898
899         Text("Image")
900            .font(.title)
901            .padding()
902
903         Text("Text")
904            .font(.title)
905            .padding()
906
907         Text("Background")
908            .font(.title)
909            .padding()
910
911         Text("Conditional")
912            .font(.title)
913            .padding()
914
915         Text("Repeat")
916            .font(.title)
917            .padding()
918
919         Text("Embed")
920            .font(.title)
921            .padding()
922
923         Text("Extract to Variable")
924            .font(.title)
925            .padding()
926
927         Text("Extract to Method")
928            .font(.title)
929            .padding()
930
931         Text("Extract All Occurrences")
932            .font(.title)
933            .padding()
934
935         Text("Navigation View")
936            .font(.title)
937            .padding()
938
939         Text("List Item")
940            .font(.title)
941            .padding()
942
943         Text("Image")
944            .font(.title)
945            .padding()
946
947         Text("Text")
948            .font(.title)
949            .padding()
950
951         Text("Background")
952            .font(.title)
953            .padding()
954
955         Text("Conditional")
956            .font(.title)
957            .padding()
958
959         Text("Repeat")
960            .font(.title)
961            .padding()
962
963         Text("Embed")
964            .font(.title)
965            .padding()
966
967         Text("Extract to Variable")
968            .font(.title)
969            .padding()
970
971         Text("Extract to Method")
972            .font(.title)
973            .padding()
974
975         Text("Extract All Occurrences")
976            .font(.title)
977            .padding()
978
979         Text("Navigation View")
980            .font(.title)
981            .padding()
982
983         Text("List Item")
984            .font(.title)
985            .padding()
986
987         Text("Image")
988            .font(.title)
989            .padding()
990
991         Text("Text")
992            .font(.title)
993            .padding()
994
995         Text("Background")
996            .font(.title)
997            .padding()
998
999         Text("Conditional")
1000            .font(.title)
1001            .padding()
1002
1003         Text("Repeat")
1004            .font(.title)
1005            .padding()
1006
1007         Text("Embed")
1008            .font(.title)
1009            .padding()
1010
1011         Text("Extract to Variable")
1012            .font(.title)
1013            .padding()
1014
1015         Text("Extract to Method")
1016            .font(.title)
1017            .padding()
1018
1019         Text("Extract All Occurrences")
1020            .font(.title)
1021            .padding()
1022
1023         Text("Navigation View")
1024            .font(.title)
1025            .padding()
1026
1027         Text("List Item")
1028            .font(.title)
1029            .padding()
1030
1031         Text("Image")
1032            .font(.title)
1033            .padding()
1034
1035         Text("Text")
1036            .font(.title)
1037            .padding()
1038
1039         Text("Background")
1040            .font(.title)
1041            .padding()
1042
1043         Text("Conditional")
1044            .font(.title)
1045            .padding()
1046
1047         Text("Repeat")
1048            .font(.title)
1049            .padding()
1050
1051         Text("Embed")
1052            .font(.title)
1053            .padding()
1054
1055         Text("Extract to Variable")
1056            .font(.title)
1057            .padding()
1058
1059         Text("Extract to Method")
1060            .font(.title)
1061            .padding()
1062
1063         Text("Extract All Occurrences")
1064            .font(.title)
1065            .padding()
1066
1067         Text("Navigation View")
1068            .font(.title)
1069            .padding()
1070
1071         Text("List Item")
1072            .font(.title)
1073            .padding()
1074
1075         Text("Image")
1076            .font(.title)
1077            .padding()
1078
1079         Text("Text")
1080            .font(.title)
1081            .padding()
1082
1083         Text("Background")
1084            .font(.title)
1085            .padding()
1086
1087         Text("Conditional")
1088            .font(.title)
1089            .padding()
1090
1091         Text("Repeat")
1092            .font(.title)
1093            .padding()
1094
1095         Text("Embed")
1096            .font(.title)
1097            .padding()
1098
1099         Text("Extract to Variable")
1100            .font(.title)
1101            .padding()
1102
1103         Text("Extract to Method")
1104            .font(.title)
1105            .padding()
1106
1107         Text("Extract All Occurrences")
1108            .font(.title)
1109            .padding()
1110
1111         Text("Navigation View")
1112            .font(.title)
1113            .padding()
1114
1115         Text("List Item")
1116            .font(.title)
1117            .padding()
1118
1119         Text("Image")
1120            .font(.title)
1121            .padding()
1122
1123         Text("Text")
1124            .font(.title)
1125            .padding()
1126
1127         Text("Background")
1128            .font(.title)
1129            .padding()
1130
1131         Text("Conditional")
1132            .font(.title)
1133            .padding()
1134
1135         Text("Repeat")
1136            .font(.title)
1137            .padding()
1138
1139         Text("Embed")
1140            .font(.title)
1141            .padding()
1142
1143         Text("Extract to Variable")
1144            .font(.title)
1145            .padding()
1146
1147         Text("Extract to Method")
1148            .font(.title)
1149            .padding()
1150
1151         Text("Extract All Occurrences")
1152            .font(.title)
1153            .padding()
1154
1155         Text("Navigation View")
1156            .font(.title)
1157            .padding()
1158
1159         Text("List Item")
1160            .font(.title)
1161            .padding()
1162
1163         Text("Image")
1164            .font(.title)
1165            .padding()
1166
1167         Text("Text")
1168            .font(.title)
1169            .padding()
1170
1171         Text("Background")
1172            .font(.title)
1173            .padding()
1174
1175         Text("Conditional")
1176            .font(.title)
1177            .padding()
1178
1179         Text("Repeat")
1180            .font(.title)
1181            .padding()
1182
1183         Text("Embed")
1184            .font(.title)
1185            .padding()
1186
1187         Text("Extract to Variable")
1188            .font(.title)
1189            .padding()
1190
1191         Text("Extract to Method")
1192            .font(.title)
1193            .padding()
1194
1195         Text("Extract All Occurrences")
1196            .font(.title)
1197            .padding()
1198
1199         Text("Navigation View")
1200            .font(.title)
1201            .padding()
1202
1203         Text("List Item")
1204            .font(.title)
1205            .padding()
1206
1207         Text("Image")
1208            .font(.title)
1209            .padding()
1210
1211         Text("Text")
1212            .font(.title)
1213            .padding()
1214
1215         Text("Background")
1216            .font(.title)
1217            .padding()
1218
1219         Text("Conditional")
1220            .font(.title)
1221            .padding()
1222
1223         Text("Repeat")
1224            .font(.title)
1225            .padding()
1226
1227         Text("Embed")
1228            .font(.title)
1229            .padding()
1230
1231         Text("Extract to Variable")
1232            .font(.title)
1233            .padding()
1234
1235         Text("Extract to Method")
1236            .font(.title)
1237            .padding()
1238
1239         Text("Extract All Occurrences")
1240            .font(.title)
1241            .padding()
1242
1243         Text("Navigation View")
1244            .font(.title)
1245            .padding()
1246
1247         Text("List Item")
1248            .font(.title)
1249            .padding()
1250
1251         Text("Image")
1252            .font(.title)
1253            .padding()
1254
1255         Text("Text")
1256            .font(.title)
1257            .padding()
1258
1259         Text("Background")
1260            .font(.title)
1261            .padding()
1262
1263         Text("Conditional")
1264            .font(.title)
1265            .padding()
1266
1267         Text("Repeat")
1268            .font(.title)
1269            .padding()
1270
1271         Text("Embed")
1272            .font(.title)
1273            .padding()
1274
1275         Text("Extract to Variable")
1276            .font(.title)
1277            .padding()
1278
1279         Text("Extract to Method")
1280            .font(.title)
1281            .padding()
1282
1283         Text("Extract All Occurrences")
1284            .font(.title)
1285            .padding()
1286
1287         Text("Navigation View")
1288            .font(.title)
1289            .padding()
1290
1291         Text("List Item")
1292            .font(.title)
1293            .padding()
1294
1295         Text("Image")
1296            .font(.title)
1297            .padding()
1298
1299         Text("Text")
1300            .font(.title)
1301            .padding()
1302
1303         Text("Background")
1304            .font(.title)
1305            .padding()
1306
1307         Text("Conditional")
1308            .font(.title)
1309            .padding()
1310
1311         Text("Repeat")
1312            .font(.title)
1313            .padding()
1314
1315         Text("Embed")
1316            .font(.title)
1317            .padding()
1318
1319         Text("Extract to Variable")
1320            .font(.title)
1321            .padding()
1322
1323         Text("Extract to Method")
1324            .font(.title)
1325            .padding()
1326
1327         Text("Extract All Occurrences")
1328            .font(.title)
1329            .padding()
1330
1331         Text("Navigation View")
1332            .font(.title)
1333            .padding()
1334
1335         Text("List Item")
1336            .font(.title)
1337            .padding()
1338
1339         Text("Image")
1340            .font(.title)
1341            .padding()
1342
1343         Text("Text")
1344            .font(.title)
1345            .padding()
1346
1347         Text("Background")
1348            .font(.title)
1349            .padding()
1350
1351         Text("Conditional")
1352            .font(.title)
1353            .padding()
1354
1355         Text("Repeat")
1356            .font(.title)
1357            .padding()
1358
1359         Text("Embed")
1360            .font(.title)
1361            .padding()
1362
1363         Text("Extract to Variable")
1364            .font(.title)
1365            .padding()
1366
1367         Text("Extract to Method")
1368            .font(.title)
1369            .padding()
1370
1371         Text("Extract All Occurrences")
1372            .font(.title)
1373            .padding()
1374
1375         Text("Navigation View")
1376            .font(.title)
1377            .padding()
1378
1379         Text("List Item")
1380            .font(.title)
1381            .padding()
1382
1383         Text("Image")
1384            .font(.title)
1385            .padding()
1386
1387         Text("Text")
1388            .font(.title)
1389            .padding()
1390
1391         Text("Background")
1392            .font(.title)
1393            .padding()
1394
1395         Text("Conditional")
1396            .font(.title)
1397            .padding()
1398
1399         Text("Repeat")
1400            .font(.title)
1401            .padding()
1402
1403         Text("Embed")
1404            .font(.title)
1405            .padding()
1406
1407         Text("Extract to Variable")
1408            .font(.title)
1409            .padding()
1410
1411         Text("Extract to Method")
1412            .font(.title)
1413            .padding()
1414
1415         Text("Extract All Occurrences")
1416            .font(.title)
1417            .padding()
1418
1419         Text("Navigation View")
1420            .font(.title)
1421            .padding()
1422
1423         Text("List Item")
1424            .font(.title)
1425            .padding()
1426
1427         Text("Image")
1428            .font(.title)
1429            .padding()
1430
1431         Text("Text")
1432            .font(.title)
1433            .padding()
1434
1435         Text("Background")
1436            .font(.title)
1437            .padding()
1438
1439         Text("Conditional")
1440            .font(.title)
1441            .padding()
1442
1443         Text("Repeat")
1444            .font(.title)
1445            .padding()
1446
1447         Text("Embed")
1448            .font(.title)
1449            .padding()
1450
1451         Text("Extract to Variable")
1452            .font(.title)
1453            .padding()
1454
1455         Text("Extract to Method")
1456            .font(.title)
1457            .padding()
1458
1459         Text("Extract All Occurrences")
1460            .font(.title)
1461            .padding()
1462
1463         Text("Navigation View")
1464            .font(.title)
1465            .padding()
1466
1467         Text("List Item")
1468            .font(.title)
1469            .padding()
1470
1471         Text
```

[Open in app](#)

```
24     NavigationView {  
25         List(Product.mocks) { product| in  
26             NavigationLink(destination: Destination) {  
27                 Label Content  
28             }  
29         }  
30     .navigationTitle("Products")  
31 }
```

👉 Replace the placeholder `Label Content` with the code below. Use Xcode's autocomplete to reduce the need to type on the keyboard.



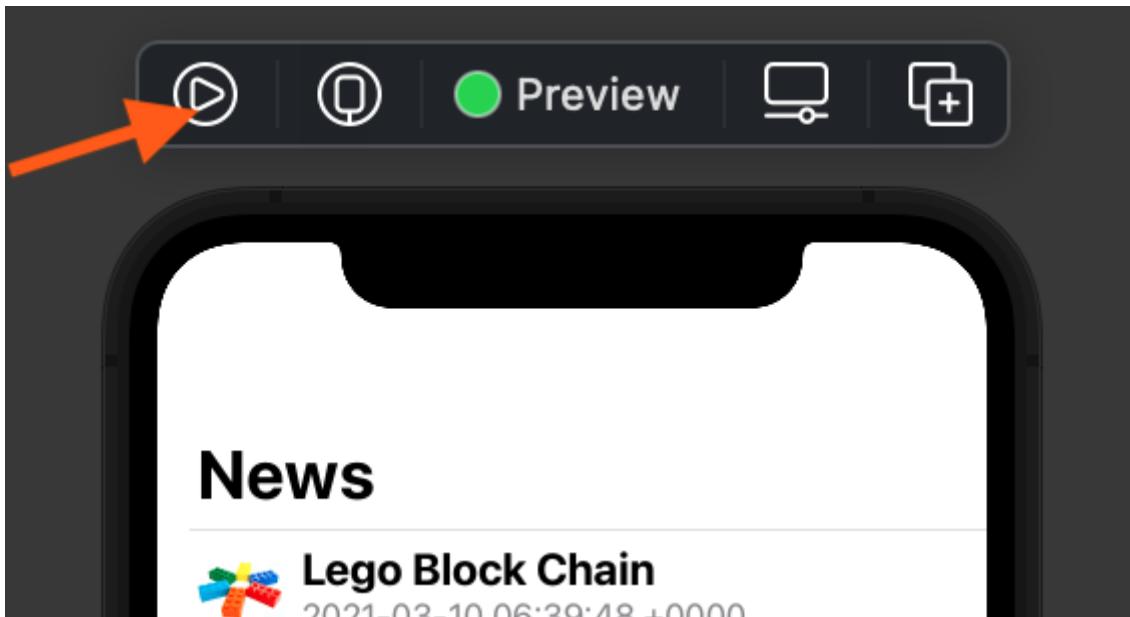
[Open in app](#)

```
24     NavigationView {  
25         List(Product.mocks) { product in  
26             NavigationLink(destination: Destination) {  
27                 ProductCell(product: product)  
28             }  
29         }  
30     .navigationTitle("Products")  
31 }
```

- 👉 Build and ensure there are no errors.

7. Live Preview

- 👉 In the preview, click the live preview button in the top left.



- 👁️ After a few seconds, the preview is live, so we can test it.

- 👉 Click on the Products tab item in the tab bar at the bottom.

- 👁️ The preview shows the two products from the mocks array, in product cells.



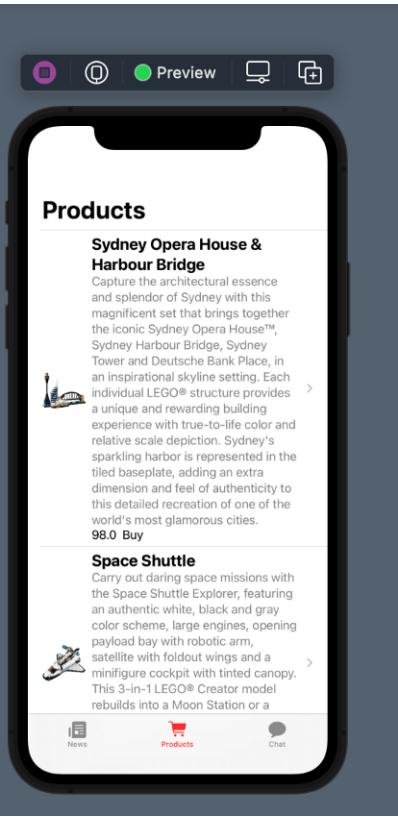


Open in app

```

1 // ContentView.swift
2 // Blocks
3 //
4 //
5 // Created by Tom Brodhurst-Hill on 4/3/21.
6 // Copyright © 2021 BareFetWare. All rights reserved.
7 //
8
9 import SwiftUI
10
11 struct ContentView: View {
12     var body: some View {
13         TabView(selection: $selection) {
14             NavigationView {
15                 List(Article.mocks) { article in
16                     NavigationLink(destination: Destination) {
17                         NewsCell(article: article)
18                     }
19                 }
20                 .navigationTitle("News")
21             }
22             .tabItem { Label("News", systemImage: "newspaper.fill") }
23             .tag(1)
24             NavigationView {
25                 List(Product.mocks) { product in
26                     NavigationLink(destination: Destination) {
27                         ProductCell(product: product)
28                     }
29                 }
30                 .navigationTitle("Products")
31             }
32             .tabItem { Label("Products", systemImage: "cart.fill") }
33             .tag(2)

```



👉 Run the app to double check that it's all working correctly in the Simulator.

8. Commit Changes

When it's all done and working, as you've done before:

1. 👉 Choose Commit from the Source Control menu.
2. 👉 Enter a description such as: Product cells show product mock data
3. 👉 Click on the Commit button.

9. Next...

Congratulations! The products list layout is complete. It is showing all the currently available products.

Next, in Tutorial 16, we will tidy up our file structure and introduce a view model to format the dollars and dates.

If you've been through the tutorial series to this point and would like to us to publish



[Open in app](#)

💬 If you have any questions or comments, please add a response below.

This series is released via [Next Level Swift](#). Subscribe to keep updated and never miss a new Tutorial of this series!

Next Level Swift

Next Level Swift

Next Level Swift aims at sharing knowledge and insights into better programming for iOS and is dedicated to help...

medium.nextlevelsweet.com

We are always looking for talented and passionate Swift developers! Feel free to check out our writer's section and find out how you can share your knowledge with the Next Level Swift Community!

Sign up for Next Level Swift Newsletter

By Next Level Swift

Get all the latest articles, posts and news straight to your mailbox! [Take a look.](#)

[Get this newsletter](#)

Emails will be sent to research2learn@yahoo.co.uk.
[Not you?](#)



[Open in app](#)