

Creating views

Video: Frames and layouts
4 min

Reading: Frame and layouts examples
20 min

Video: Stacks and views
4 min

Video: Creating views using stacks
3 min

Reading: Stacks cheat sheet
10 min

Reading: Exercise: Creating a simple view using stacks
30 min

Reading: Solution: Creating a simple view using stacks
10 min

Directions [View](#) [Self review](#)

Frame and layouts examples

Overview

In the previous video, you learned about the important tools for modifying the layout of your user interface. In this reading, you'll work through examples of using those tools. By the end, you will understand when to consider modifying a view layout. You will be also able to identify the frame of a view element and explain the different tools.

To try out these examples, create a SwiftUI app project in Xcode. Set the target to your favorite iPhone and open **ContentView.swift**. The code that defines the content view will be something like this:

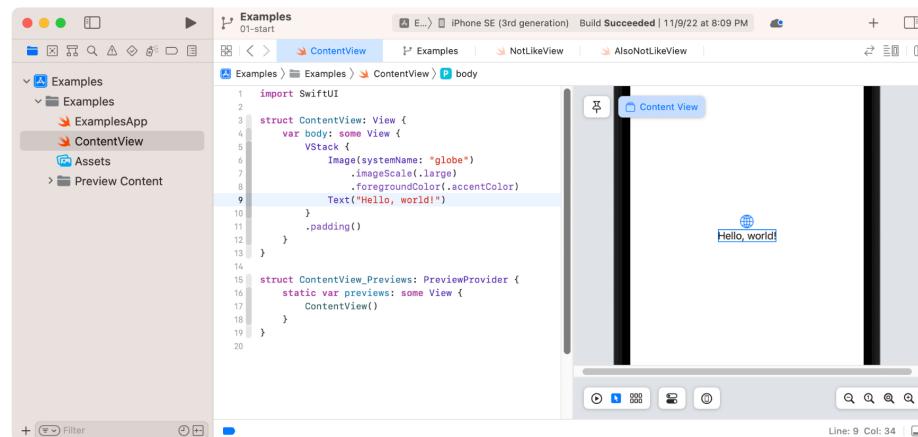
```
1 struct ContentView: View {
2     var body: some View {
3         VStack {
4             Image(systemName: "globe")
5                 .imageScale(.large)
6                 .foregroundColor(.accentColor)
7             Text("Hello, world!")
8         }
9         .padding()
10    }
11 }
```

The examples that follow use or modify the code inside the **VStack**.

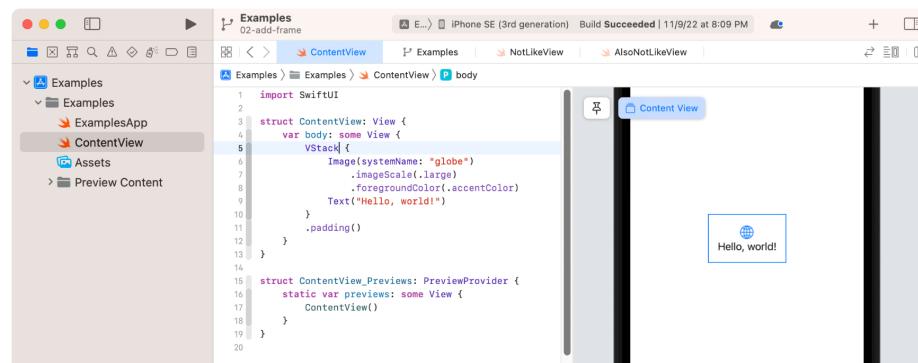
Frame, the basic unit of layouts

Every view that is displayed on the screen is a rectangle. Views that don't appear to be a rectangle are drawn in a rectangle. This rectangle is called the frame of the view.

In the Xcode project you created, set the mode of the Canvas to Selectable (the middle of the three buttons on the lower left). Now select the line with the **Text** in the code editor. The Canvas displays a box around the text. This box is the frame of the **Text**.



Select the **VStack** and now the rectangle encloses the image and **Text** view.



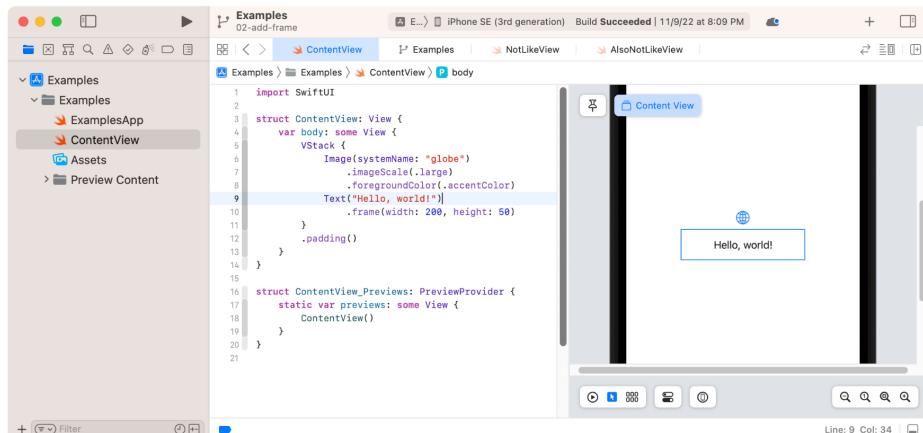


Setting the frame

Now you'll set the frame for the `Text`. Change the code as follows:

```
1 Text("Hello, world!")
2     .frame(width: 200, height: 50)
```

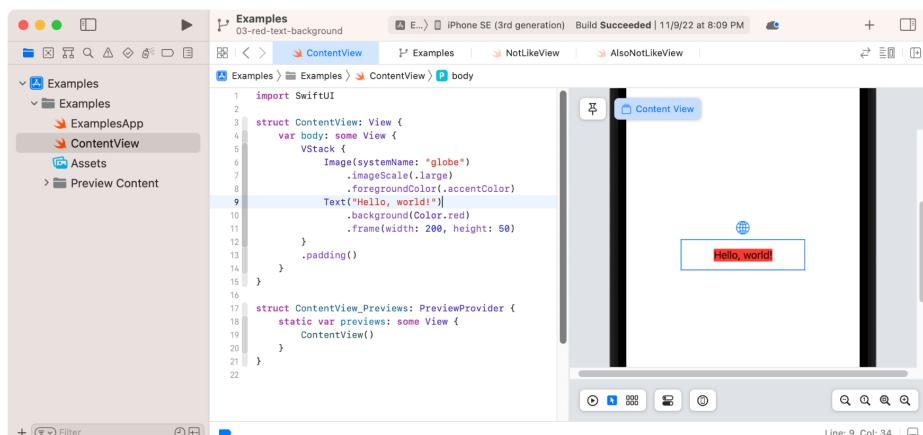
The frame for the `Text` appears to change on the canvas, but that's not what is really happening.



Set the background color of the `Text` to red:

```
1 Text("Hello, world!")
2     .background(Color.red)
3     .frame(width: 200, height: 50)
```

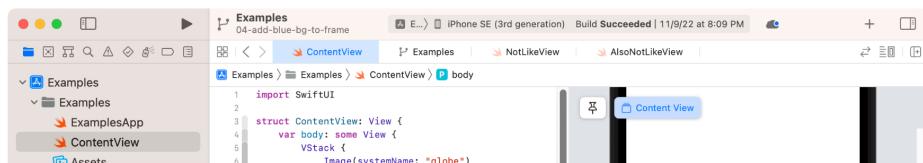
The red background is displayed only in the original `Text` frame.



Now add a blue background to the `.frame()` modifier:

```
1 Text("Hello, world!")
2     .background(Color.red)
3     .frame(width: 200, height: 50)
4     .background(Color.blue)
```

The `.frame()` modifier creates a new invisible view with the specified size. The `Text` is inside that new view.



```

1   .imageScale(.large)
2   .foregroundColor(.accentColor)
3   Text("Hello, world!")
4   .background(Color.red)
5   .frame(width: 200, height: 50)
6   .background(Color.blue)
7   }
8 }
9 struct ContentView_Previews: PreviewProvider {
10 static var previews: some View {
11     ContentView()
12 }
13 }
14 }
15 }
16 }
17 }
18 }
19 }
20 }
21 }
22 }
23 }

```

Line: 9 Col: 34 | ⌂

When to modify your layout

You may be wondering if all these invisible views can slow your app down. The basic answer is no. SwiftUI is very efficient when working with views, including the invisible ones for view modifiers.

However, SwiftUI is very good at laying out an interface on many different devices and platforms. The best idea is to use a modification only when it's important. For example, a logo may need to be a certain size or you want a portion of the app to display a certain way.

Start with no modifications. Add them when you need them. When you do add them, preview them on any device that your app runs on.

Organizing groups of views

You can think of your interface as different groups of related views. For example, the stopwatch in the Apple Clock app has two vertical groups. In the digital version, the first is a horizontal group that displays the time. Next is a horizontal group with a lap button, a control to switch between the digital and analog versions and another button to start and stop timing.

The code you've been working with is a vertical group, a **VStack**. For a horizontal group, you use **HStack**. The code that follows is a horizontal group of three buttons.

```

1 HStack {
2     Button("One") {}
3     Button("Two") {}
4     Button("Three") {}
5 }

```

Remove the background colors and frame from the **Text**. Now add the **HStack** with the buttons below the **Text**.

```

1 Examples
2 Examples
3 ExamplesApp
4 ContentView
5 Assets
6 Preview Content
7 Examples Examples ContentView body
8 Examples Examples ContentView body
9 import SwiftUI
10 struct ContentView: View {
11     var body: some View {
12         VStack {
13             Image(systemName: "globe")
14             .imageScale(.large)
15             .foregroundColor(.accentColor)
16             Text("Hello, world!")
17             HStack {
18                 Button("One") {}
19                 Button("Two") {}
20                 Button("Three") {}
21             }
22         }
23     }
24 }
25
26 struct ContentView_Previews: PreviewProvider {
27     static var previews: some View {
28         ContentView()
29     }
30 }
31
32 }
33
34 }
35
36 }
37
38 }
39
40 }
41
42 }
43
44 }
45
46 }
47
48 }
49
50 }
51
52 }
53
54 }
55
56 }
57
58 }
59
60 }
61
62 }
63
64 }
65
66 }
67
68 }
69
70 }
71
72 }
73
74 }
75
76 }
77
78 }
79
80 }
81
82 }
83
84 }
85
86 }
87
88 }
89
90 }
91
92 }
93
94 }
95
96 }
97
98 }
99
100 }
101
102 }
103
104 }
105
106 }
107
108 }
109
110 }
111
112 }
113
114 }
115
116 }
117
118 }
119
120 }
121
122 }
123
124 }
125
126 }
127
128 }
129
130 }
131
132 }
133
134 }
135
136 }
137
138 }
139
140 }
141
142 }
143
144 }
145
146 }
147
148 }
149
150 }
151
152 }
153
154 }
155
156 }
157
158 }
159
160 }
161
162 }
163
164 }
165
166 }
167
168 }
169
170 }
171
172 }
173
174 }
175
176 }
177
178 }
179
180 }
181
182 }
183
184 }
185
186 }
187
188 }
189
190 }
191
192 }
193
194 }
195
196 }
197
198 }
199
200 }
201
202 }
203
204 }
205
206 }
207
208 }
209
210 }
211
212 }
213
214 }
215
216 }
217
218 }
219
220 }
221
222 }
223
224 }
225
226 }
227
228 }
229
230 }
231
232 }
233
234 }
235
236 }
237
238 }
239
240 }
241
242 }
243
244 }
245
246 }
247
248 }
249
250 }
251
252 }
253
254 }
255
256 }
257
258 }
259
260 }
261
262 }
263
264 }
265
266 }
267
268 }
269
270 }
271
272 }
273
274 }
275
276 }
277
278 }
279
280 }
281
282 }
283
284 }
285
286 }
287
288 }
289
290 }
291
292 }
293
294 }
295
296 }
297
298 }
299
300 }
301
302 }
303
304 }
305
306 }
307
308 }
309
310 }
311
312 }
313
314 }
315
316 }
317
318 }
319
320 }
321
322 }
323
324 }
325
326 }
327
328 }
329
330 }
331
332 }
333
334 }
335
336 }
337
338 }
339
340 }
341
342 }
343
344 }
345
346 }
347
348 }
349
350 }
351
352 }
353
354 }
355
356 }
357
358 }
359
360 }
361
362 }
363
364 }
365
366 }
367
368 }
369
370 }
371
372 }
373
374 }
375
376 }
377
378 }
379
380 }
381
382 }
383
384 }
385
386 }
387
388 }
389
390 }
391
392 }
393
394 }
395
396 }
397
398 }
399
400 }
401
402 }
403
404 }
405
406 }
407
408 }
409
410 }
411
412 }
413
414 }
415
416 }
417
418 }
419
420 }
421
422 }
423
424 }
425
426 }
427
428 }
429
430 }
431
432 }
433
434 }
435
436 }
437
438 }
439
440 }
441
442 }
443
444 }
445
446 }
447
448 }
449
450 }
451
452 }
453
454 }
455
456 }
457
458 }
459
460 }
461
462 }
463
464 }
465
466 }
467
468 }
469
470 }
471
472 }
473
474 }
475
476 }
477
478 }
479
480 }
481
482 }
483
484 }
485
486 }
487
488 }
489
490 }
491
492 }
493
494 }
495
496 }
497
498 }
499
500 }
501
502 }
503
504 }
505
506 }
507
508 }
509
510 }
511
512 }
513
514 }
515
516 }
517
518 }
519
520 }
521
522 }
523
524 }
525
526 }
527
528 }
529
530 }
531
532 }
533
534 }
535
536 }
537
538 }
539
540 }
541
542 }
543
544 }
545
546 }
547
548 }
549
550 }
551
552 }
553
554 }
555
556 }
557
558 }
559
560 }
561
562 }
563
564 }
565
566 }
567
568 }
569
570 }
571
572 }
573
574 }
575
576 }
577
578 }
579
580 }
581
582 }
583
584 }
585
586 }
587
588 }
589
590 }
591
592 }
593
594 }
595
596 }
597
598 }
599
600 }
601
602 }
603
604 }
605
606 }
607
608 }
609
610 }
611
612 }
613
614 }
615
616 }
617
618 }
619
620 }
621
622 }
623
624 }
625
626 }
627
628 }
629
630 }
631
632 }
633
634 }
635
636 }
637
638 }
639
640 }
641
642 }
643
644 }
645
646 }
647
648 }
649
650 }
651
652 }
653
654 }
655
656 }
657
658 }
659
660 }
661
662 }
663
664 }
665
666 }
667
668 }
669
670 }
671
672 }
673
674 }
675
676 }
677
678 }
679
680 }
681
682 }
683
684 }
685
686 }
687
688 }
689
690 }
691
692 }
693
694 }
695
696 }
697
698 }
699
700 }
701
702 }
703
704 }
705
706 }
707
708 }
709
710 }
711
712 }
713
714 }
715
716 }
717
718 }
719
720 }
721
722 }
723
724 }
725
726 }
727
728 }
729
730 }
731
732 }
733
734 }
735
736 }
737
738 }
739
740 }
741
742 }
743
744 }
745
746 }
747
748 }
749
750 }
751
752 }
753
754 }
755
756 }
757
758 }
759
760 }

```

Line: 10 Col: 19 | ⌂

The current code describes a view that is grouped in the vertical direction. It contains three elements: an image, some text and a horizontal group of buttons.

Designing your interface as groups of horizontal and vertical views makes designing for SwiftUI easier.

Adding space

The **Text** and the buttons of the **HStack** are too close together. A **Spacer** adds space. Where that space is added depends on the direction of the views in the group. For example, if you add a **Spacer** between the first and second buttons, it adds space in the horizontal direction.

To add space in the vertical direction, add the **Spacer** between the **Text** and the **HStack** with the buttons:

```

1 Text("Hello, world!")
2 Spacer()
3 HStack {
4     Button("One") {}
5 }

```

```

5     Button("Two") {}
6     Button("Three") {}
7 }

```

Now the buttons are at the bottom of the screen and the **Text** is at the top.

Adding padding

Sometimes a **Spacer** adds too much space. The **Spacer** between the **Text** and button group pushed the buttons to the bottom of the screen. It also moved the **Text** and image to the top. Sometimes you want less space between views. The most common reason is matching the Apple Human Interface Guidelines for spacing.

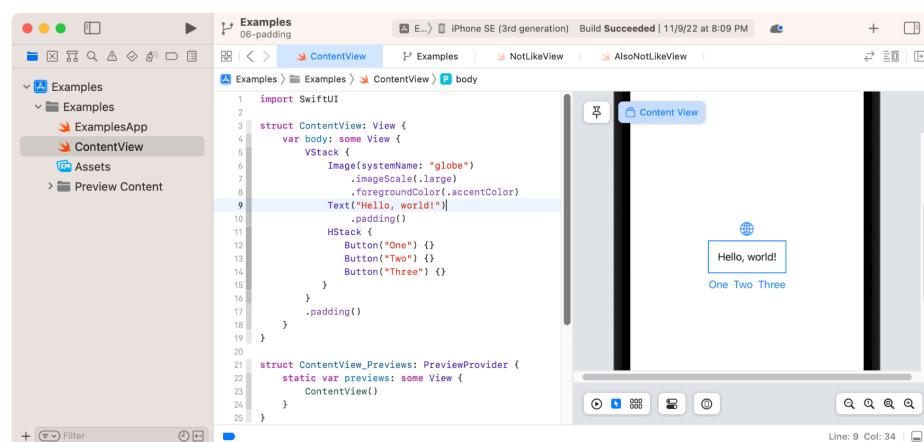
For this, you can use the **.padding()** modifier. By default, this adds the system-defined amount of space on all four sides of a view. It's also possible to choose which sides and how much space.

Remove the **Spacer** between the **Text** and button group. Now, add padding to the **Text**:

```

1 Text("Hello, world!")
2     .padding()
3 HStack {
4     Button("One") {}
5     Button("Two") {}
6     Button("Three") {}
7 }

```



Now there is the recommended space between the bottom of the **Text** view and the top of the button group. There is also space between the top of the **Text** view and the image.

Alignment

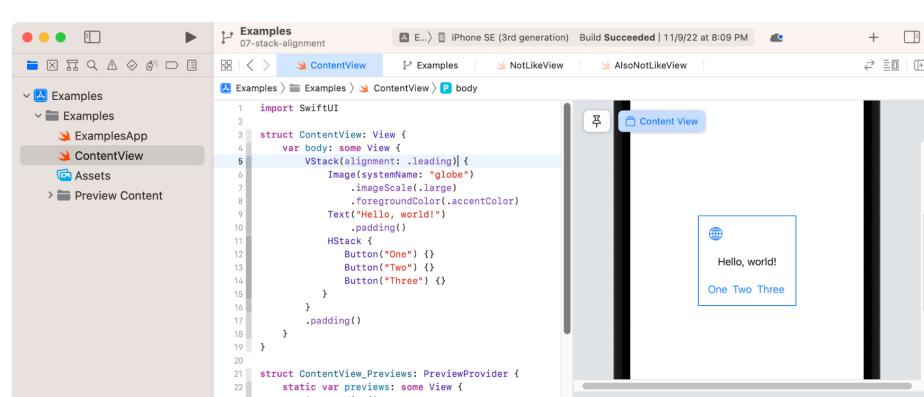
The current interface is centered on the iPhone screen. Sometimes you want your views to align in different ways.

There are two main ways to change the alignment.

The first uses the Stack views alignment argument. A **VStack** arranges its child views vertically but aligns them in the horizontal direction. Similarly, an **HStack** arranges its child views horizontally, but aligns them in a vertical direction.

Change the **VStack** so it aligns items to its leading edge:

```
1 VStack(alignment: .leading) {
```





Although it looks like the text and buttons are still centered, that's not really the case. Examine the frames for the **Text** and for the **HStack** in the Canvas. They are both as wide as the **VStack**.

Add a **.frame** modifier to the **VStack** with a width of 300.

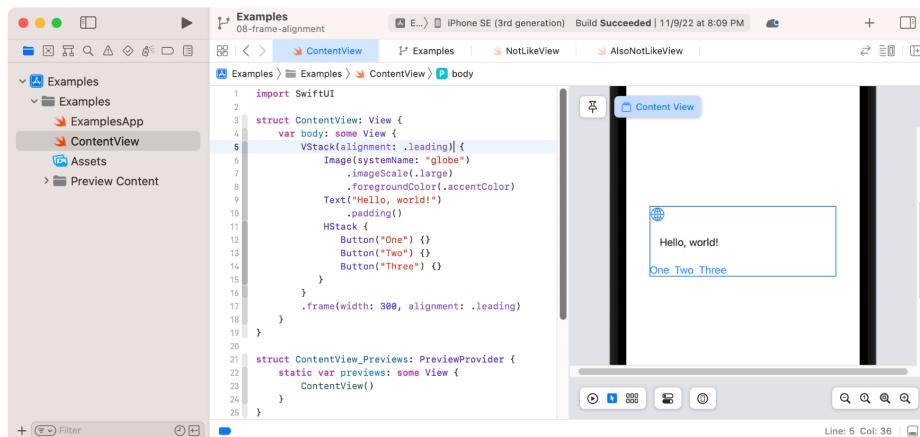
```
1 struct ContentView: View {  
2     var body: some View {  
3         VStack(alignment: .leading) {  
4             ...  
5         }  
6         .frame(width: 300)  
7     }  
8 }
```

The views inside the **VStack** don't change position. The **.frame** modifier added a new invisible view.

Now you can try the second way to change alignment. The frame modifier also takes an argument for alignment.

Add a leading alignment to the **VStack** frame:

```
1     .frame(width: 300, alignment: .leading)
```



There are many different types of alignment including leading, trailing, top, center and so on. Use the code you've created to experiment with the different types. Try using them in the stacks and in frames. Try adding frames with different alignments.

Conclusion

In this reading, you explored tools you can use to modify the layout of your interface. You learned that a frame is just another invisible view and how to identify one on the Xcode Canvas. You also learned when to modify your layout. Plus, you experimented with stacks of views, tried adding space with the **Spacer** and then with padding and, finally, you explored two ways to align views.

[Mark as completed](#)

[Like](#) [Dislike](#) [Report an issue](#)

