Swift Scroll View School

Hands-On Challenges

Swift Scroll View School Hands-On Challenges

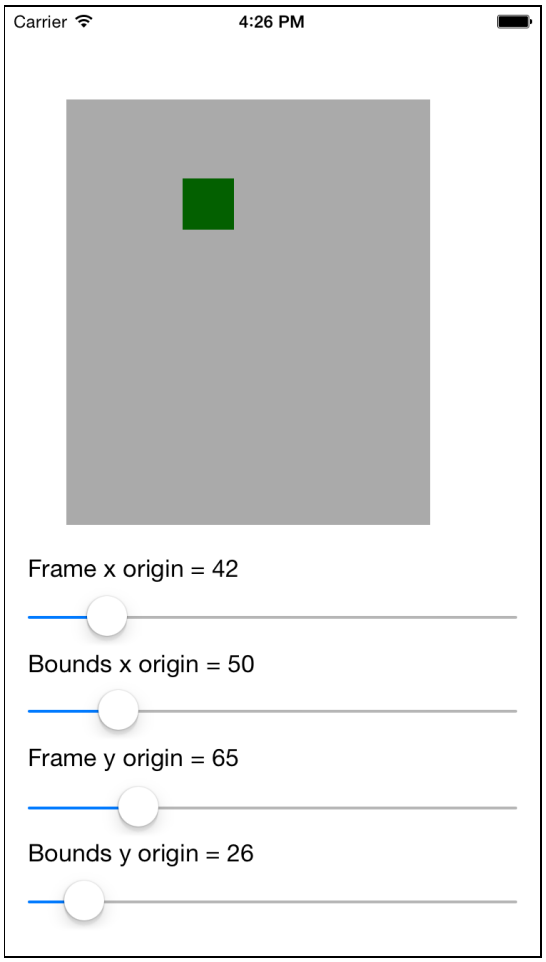
Challenge A: Frame and Bounds

A scroll view is all about presenting more content than is visible in its frame. You've seen how a plain old view can do something similar by changing its bounds origin.

In the demo, you saw a sample app with a couple of sliders to change the x coordinate of the frame and bounds origin points. In this mini-challenge, you'll add another set of sliders for the y coordinate.

This will help you visualize the difference between frame and bounds. If you're already familiar with how they work, feel free to go right ahead to the next video and get started with scroll views!

Here's what the app will look like at the end of the challenge:

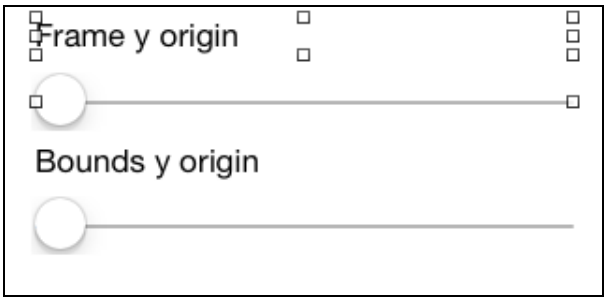


Let's get started!

More user interface

First, you'll need more user interface elements. Open **Main.storyboard** and start by dragging out two labels and two sliders.

Arrange them underneath the existing sliders. Make them take up the entire width of the screen with the standard margins on the sides.



Set the top label to "Frame y origin" and the bottom label to "Bounds y origin" as shown above. You'll update this text from code, but it's nice to fill it in so you know which is which at design time too.

Next: outlets! You'll need outlets for each of these views. Open the assistant editor and make sure **ViewController.swift** is displayed. Control-drag from each view to the code. Name the label outlets `frameYOriginLabel` and `boundsYOriginLabel`, and the slider outlets `frameYSlider` and `boundsYSlider`.

Finally: actions! With the assistant editor still open, control-drag from each slider over to the code. Make sure to select Action rather than Outlet as the connection type. Name the frame slider action `frameYSliderChanged` and the bounds slider action `boundsYSliderChanged`.

That's it for the interface! Switch over to **ViewController.swift** for the implementation.

Frame and bounds

First, you'll want to make sure the correct values are on screen when the app launched. Add the following code to `viewDidLoad`, just before the call to `updateValues()`:

```
frameYSlider.value = Float(containerView.frame.origin.y)
boundsYSlider.value = Float(containerView.bounds.origin.y)
```

This will update the sliders to match the initial position of the container.

Next, find `updateValues` and add the following code to the end of that method:

```
frameYOriginLabel.text =
  "Frame y origin = \(Int(containerView.frame.origin.y))"
boundsYOriginLabel.text =
  "Bounds y origin = \(Int(containerView.bounds.origin.y))"
```

This will update the text of the labels to match the position of the container. As you move the sliders, this method is also called to keep everything up to date.

Finally, all you need to do is add the implementation to the two new action methods. Find `frameYSliderChanged` and add the following implementation:

```
containerView.frame.origin.y = CGFloat(frameYSlider.value)
updateValues()
```

As expected, this code updates the frame origin's y-coordinate with the slider value and then calls `updateValues()` to update the display.

Find boundsYSliderChanged and add the following implementation:

```
containerView.bounds.origin.y = CGFloat(boundsYSlider.value)
updateValues()
```

Similarly, this will update the bounds origin's y-coordinate with the slider value and then update the display.

That's it for this mini-challenge! Build and run the app, and move the sliders around. Try to predict exactly what will change on screen before you change one of the values. Soon, you'll be a master of frame and bounds and ready to move ahead with scrolling!