Extracted from:

iOS 10 SDK Development

Creating iPhone and iPad Apps with Swift

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

iOS 10 SDK Development

Creating iPhone and iPad Apps with Swift

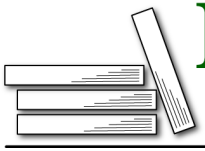
Chris Adamson
with Janie Clayton
edited by Rebecca Gulick

iOS 10 SDK Development

Creating iPhone and iPad Apps with Swift

Chris Adamson
with Janie Clayton

The Pragmatic Bookshelf
Raleigh, North Carolina



Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

The team that produced this book includes:

Publisher: Andy Hunt
VP of Operations: Janet Furlow
Executive Editor: Susannah Davidson Pfalzer
Development Editor: Rebecca Gulick
Indexing: Potomac Indexing, LLC
Copy Editor: Nicole Abramowitz
Layout: Gilson Graphics

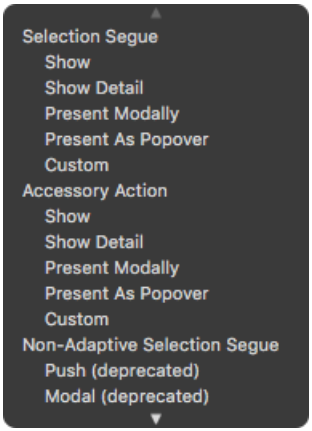
For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

Printed in the United States of America.
ISBN-13: 978-1-68050-210-7
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—March 2017

Segueing Between Scenes

At first glance, this looks like a lot of work to add what's little more than a title bar to our app, but we're now set up to start navigating between scenes. It gets pretty easy from here.



How easy? Like, *drawing one line* easy. In Main.storyboard, go to the episode list scene, select the table cell, and Control-drag from it to the player scene. At the end of the drag, a pop-up appears (shown here), listing a set of choices for Selection Segue, Accessory Action, and some deprecated choices. From the top section, choose Show. Once you release, this creates a new arrow between the episode list and player scenes.

More importantly, this gives us navigation to the player scene! Run the app again and tap on any of the table rows. The player scene slides in from the right, and the navigation bar updates to show a back button called Podcasts (the title of the previous view controller on the navigation stack) at upper-left, and clears out the title, since this scene doesn't have a navigation item. Click the back button to return to the episode list, and click another row to revisit the player. The player still plays the MP3 we hard-coded it to a few chapters back, but we can work on that next.

Updating the Player Scene for Navigation

In the storyboard, notice that the player scene now has a simulated navigation bar at its top. It may be overlapping some of the contents; if that's the case, as before, select the view controller, visit the Attributes Inspector, and turn off Adjust Scroll View Insets. Also, we can give this scene a proper title by going to the Object Library at bottom right and dragging in a Navigation Item (the icon with the back button, but without the yellow ball). Drop it anywhere in the scene, and in the Attributes Inspector, give it the title Player.

While we're here, let's attend to one last piece of business: the player view controller class has just been called ViewController since we started the project. With other view controllers in play, it's now ambiguous. Bring up the file explorer in the left pane, slowly double-click ViewController.swift, and rename it to PlayerViewController.swift. Then edit its class declaration to its new name:

navigation/PragmaticPodcasts-10-1/PragmaticPodcasts/PlayerViewController.swift
```
class PlayerViewController: UIViewController {
```

Of course, now the storyboard has a scene that refers to a ViewController that no longer exists. Select the player scene's view controller icon (the yellow ball), go to its Identity Inspector (⌥⌘3), and change the class to PlayerViewController.

Customizing Segues

Now let's figure out how to make the player use the podcast episode that we clicked on. The secrets are in that line we drew between the two scenes. When we did that, the storyboard added an arrow with an icon between the episode list and player scenes.

This arrow represents a *segue*, an object for a transition between two storyboard scenes. The little icon in the circle shows the type of segue—this one is a "push" segue, where the incoming scene pushes the old one off the side. Along with knowing which scenes are connected and how, the segue allows us to interact with the transition in code.

Segues can have attributes of their own, and it is a good habit to assign each one an identifier, so that our code can figure out which segue is being performed. Click on the segue arrow, bring up the Attributes Inspector, and enter showPlayer for the identifier (any string will do; you just have to be consistent between storyboard and code).

In code, when a segue is about to be performed, the view controller currently on the screen gets two method callbacks prior to the transition actually taking place:

- shouldPerformSegue(withIdentifier:sender:) gives the current view controller an opportunity to permit or veto the named segue. If it returns false, the segue is not performed.
- prepare(for:sender:) is called right before the transition. The first parameter is a UIStoryboardSegue, an object that provides the identifier and both the source and destination view controllers. This gives the current view controller an opportunity to prepare the destination scene before it even appears.

The latter method is what we need to prepare the player scene. When prepare(for:sender:) is called, we can get the PodcastEpisode that was tapped on, and

set up the player scene with its audio URL and title. Then, when the scene flies in, it'll be all ready for the user to tap Play.

That said, we don't want the episode list modifying the player scene's fields directly—they could change, after all. So let's go over to PlayerViewController.swift and give it a way of accepting an episode to display and play:

navigation/PragmaticPodcasts-10-1/PragmaticPodcasts/PlayerViewController.swift

```
Line 1  var episode : PodcastEpisode? {
        didSet {
          loadViewIfNeeded()
          titleLabel.text = episode?.title
    5     if let url = episode?.enclosureURL {
            set (url: url)
          }
          if let imageURL = episode?.iTunesImageURL {
            let session = URLSession(configuration: .default)
   10       let dataTask = session.dataTask(with: imageURL) { dataMb, _, _ in
              if let data = dataMb {
                DispatchQueue.main.async {
                  self.logoView.image = UIImage(data: data)
                }
   15         }
            }
            dataTask.resume()
          }
        }
   20   }
```

This gives us an episode property, with a *lot* going on in its didSet. Let's take it step by step:

- We start with line 3, handling an important problem with segues, one that isn't immediately obvious. A view in a storyboard isn't actually loaded until it needs to be onscreen…but that means the view *won't* be loaded when prepare(for:sender:) is about to segue to this scene, which in turn means that *none* of the IBOutlets (like titleLabel) will be available yet. Manually forcing the view to load if needed, with loadViewIfNeeded(), fixes this problem.
- On line 4, we set the contents of the titleLabel from the PodcastEpisode's title.
- Lines 5-7 unwrap the episode's URL and use it to call setURL(). This method was one of the first things we wrote, and sets up the AVPlayer to play the podcast audio.
- Finally, there's a big section from lines 8-18 that populates the big image view. If we can unwrap an imageURL, we create a URLSession (line 9) and give it a URLSessionDataTask (line 10). This works like the other image-loading

data tasks that we've written before: its closure (lines 10-16) looks to see if we got any data (line 11), and if so, dispatches to the main queue (lines 12-14) to create a new UIImage and set it as the image of logoView. Finally, after declaring this whole dataTask, we start it on line 17.

Also, while we're in PlayerViewController.swift, we can now take out the code we had set up to play a hard-coded URL. Delete the entire viewDidLoad(), as well as the line titleLabel.text = url.lastPathComponent in set(url:).

Now the player is ready to receive a PodcastEpisode when it's on the receiving end of a segue. Switch over to EpisodeListViewController.swift and write a new prepare(for:sender:) method.

navigation/PragmaticPodcasts-10-1/PragmaticPodcasts/EpisodeListViewController.swift

```
Line 1  override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
2     if segue.identifier == "showPlayer",
3       let playerVC = segue.destination as? PlayerViewController,
4       let indexPath = table.indexPathForSelectedRow {
5       let episode = feeds[indexPath.section].episodes[indexPath.row]
6       playerVC.episode = episode
7     }
8   }
```

Lines 2 and 3 test the segue to see if the segue is the one we expect and that the destination can be cast to PlayerViewController. Line 4 uses the table's indexPathForSelectedRow to figure out which row was tapped and get its corresponding PodcastEpisode on line 5. If all of that works, we simply set the episode property of the PlayerViewController, triggering the didSet that we just wrote in that class.

And that's all we need. Run the app and give it a try: scroll down to any row and tap it. The player will slide in with its label set to the episode we tapped on, and when you tap Play, that episode will load and start playing.

Look at that—or more accurately, *listen* to that. We now have an honest-to-gosh working podcast player! It downloads and parses a feed, shows its episode, lets us pick one, shows some of its metadata, and plays its audio. Maybe it's no Overcast or Pocket Casts yet, but we're only getting started with our iOS development career, right?