iOS 101

Hands-On Challenges

iOS 101
Hands-On Challenges

Copyright © 2014 Razeware LLC.

All rights reserved. No part of this book or corresponding materials (such as text, images, or source code) may be reproduced or distributed by any means without prior written per- mission of the copyright owner.

This book and all corresponding materials (such as source code) are provided on an "as is" basis, without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose, and noninfringement. In no event shall the authors or copyright holders be liable for any claim, damages or other liability, whether in action of contract, tort or otherwise, arising from, out of or in connec- tion with the software or the use or other dealings in the software.

All trademarks and registered trademarks appearing in this book are the property of their respective owners.

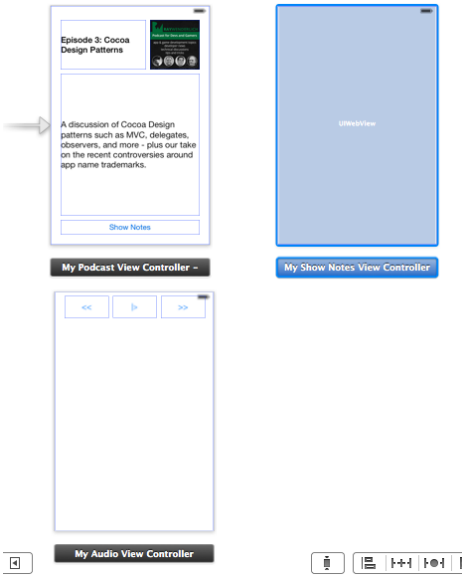
Challenge #5: A Podcast App

A few months ago we started an official raywenderlich.com podcast, where several members of the team chat about iOS development news and tech talk.

Wouldn't it be great if there was an app that let you easily browse through the episodes available so far? In this challenge, you will build exactly that – and you'll get hands-on experience with Navigation Controllers, Tab Bar Controllers, and Container View Controllers in the process.

Basic Level Walkthrough

In the resources for this challenge, you will find a starter project called MyPodcast. Open it in Xcode and open **Main.storyboard** to see what you have so far:

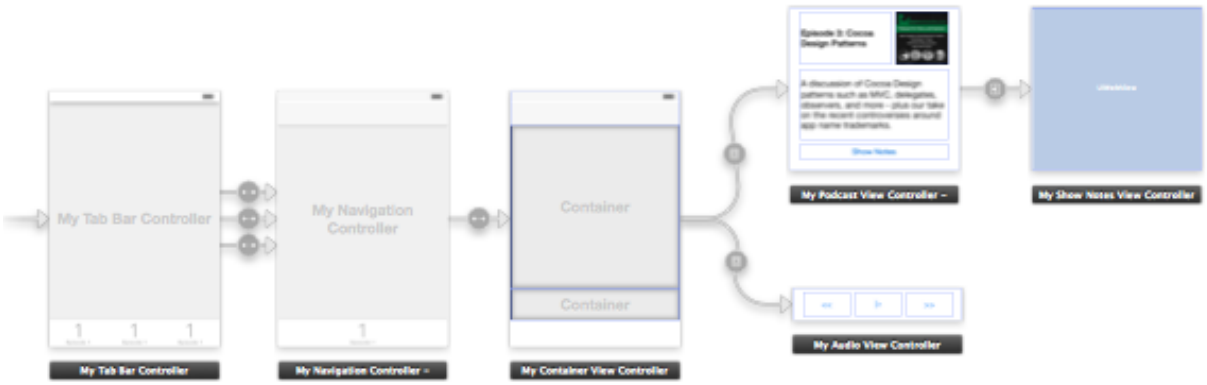


As you can see, there are three view controllers:

- **MyPodcastViewController**: Displays some metadata about a podcast, such as its title and description. Also has a button to display the show notes.
- **MyShowNotesViewController**: Contains a web view that loads the show notes for the podcast.
- **MyAudioViewController**: A reusable view controller that has controls to play an audio file. Pretend this came from another project, and you'd like to re-use this view controller in this app.

You will also see that there is a model class called **Podcast** that contains the metadata for each podcast. It also contains a static method to create three instances of the class for our first three episodes.

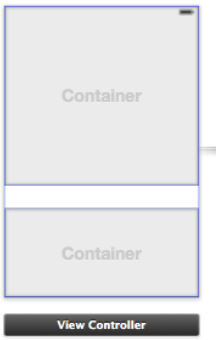
Your job in this challenge is to put these view controllers into the following configuration:



Here you have:

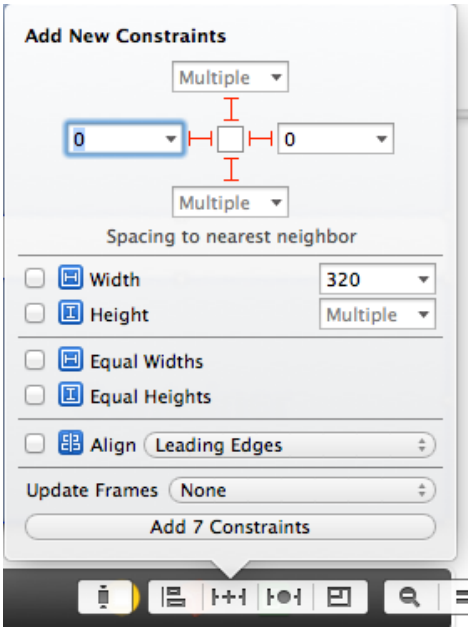
- A **tab bar controlle**r, with one tab for each episode
- Each episode will be contained in a **navigation controller**, so you can navigate between the container controller view and the detail view controller
- The container view controller will consist of two parts: the podcast view controller and the audio view controller, contained within **container views**

Let's build this from the inside out, starting with the container view controller. To do this, drag a new view controller into your canvas to the left of the podcast view controller. Drag two container views into the new view controller, and delete the new view controllers that it embeds by default. At this point you should have something that looks like this:

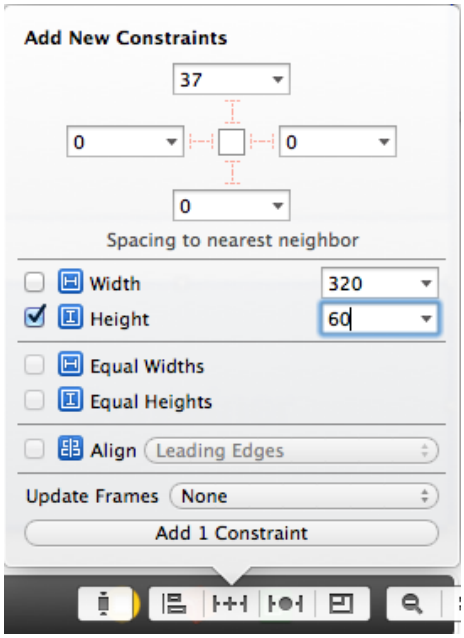


Now, set up Auto Layout for this view controller by doing the following:

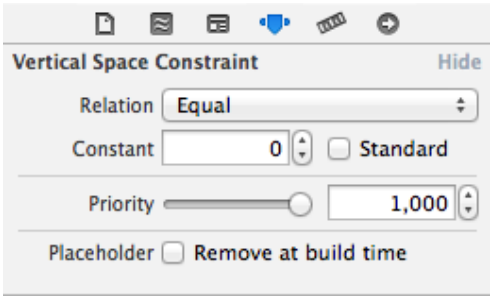
• Select both view controllers and pin them to all edges:



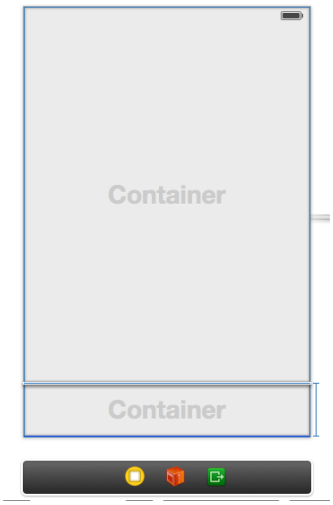
• Select the bottom view controller and constrain its height to 60:



• Select the vertical space constraint between the two container views, and set it to 0:

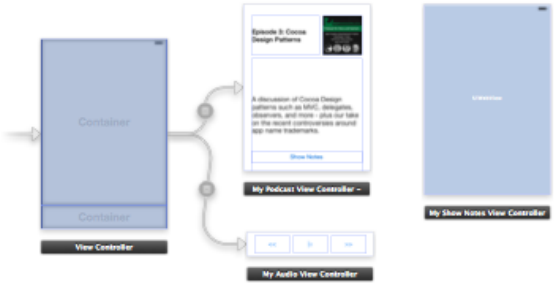


- Finally, click **Resolve Auto Layout Issues\Update All Frames in View Controller** to make the frames match your new constraints. At this point, you should see the following:



Next, control-drag from the top container view to the podcast view controller, and select **embed** from the popup. Similarly, control-drag from the bottom container view to the audio view controller, and select **embed** from the popup.

Finally, drag the "initial view controller" arrow from the podcast view controller to your new container view controller so it shows up when you run the app. Your storyboard should now look like this:



Build and run, and you should see both view controllers contained within your single new view controller:

Nice! Next, you should create a view controller for this new view controller that takes a **Podcast** object as a parameter, and passes the appropriate data to each contained view controller.

Create a new file with the **iOS\Cocoa Touch\Objective-C class** template, name it **MyContainerViewController**, and make it a subclass of **UIViewController**. Then open **MyContainerViewController.h** and replace it with the following:

```
@class Podcast;

@interface MyContainerViewController : UIViewController

@property (strong, nonatomic) Podcast *podcast;

@end
```

This simply adds a property for the podcast to display inside this view controller. Next switch to **MyContainerViewController.m** and import each of the contained view controllers and the model:

```
#import "MyPodcastViewController.h"
#import "MyAudioViewController.h"
#import "Podcast.h"
```

Also add this new method:

```
- (void)prepareForSegue:(UIStoryboardSegue *)segue
```

```
    sender:(id)sender {
    if ([segue.identifier isEqualToString:@"EmbedPodcast"]) {
      MyPodcastViewController *vc = (MyPodcastViewController *)
        segue.destinationViewController;
      vc.podcast = self.podcast;
    }
    if ([segue.identifier isEqualToString:@"EmbedAudio"]) {
      MyAudioViewController *vc = (MyAudioViewController *)
        segue.destinationViewController;
      vc.audioFile = self.podcast.audioFile;
    }
}
```

The way you set up view controllers inside container views is to override **prepareForSegue:sender:** like you see here, and simply set up any properties when the embed segue is about to occur. Here you simply set the appropriate property for each view controller.

Note that it distinguishes between the segues by identifier – so for this to work, you have to set the identifier for each segue. So open **Main.storyboard** and set the Identifier of the top segue to **EmbedPodcast**, and the identifier of the bottom segue to **EmbedAudio**.

While you are still in **Main.storyboard**, you need to associate your new view controller to your new class. So select your new view controller, and in the identity inspector set the class to **MyContainerViewController**.

One last thing. To test that this all works, open **MyContainerViewController.m** and add this to the top of **prepareForSegue:sender:**:

```
  self.podcast = [Podcast podcasts][1];
```

This is a temporary line that sets the podcast property to the second podcast – you'll remove this in a moment.

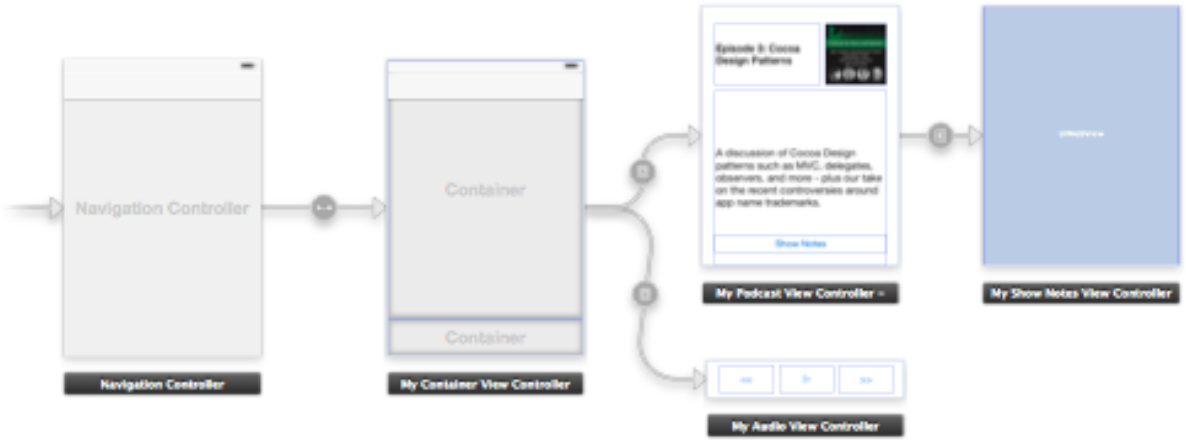
Build and run, and now you should be able to see the info about the second podcast – and play the episode using the controls at the bottom:

Once you've verified it's working so far, delete that test line that sets **self.podcast**.

Next, you will put this view controller inside a navigation controller. This will allow you to have a navigation stack, and when the user taps the Show Notes button it will push the show notes view controller on to the navigation stack, giving the user an easy way to go back to where they started.

To do this, open **Main.storyboard**, select the container view controller and select **Editor\Embed In\Navigation Controller**. This will embed the container view controller into the navigation controller, and you will see a navigation bar appear on the top of the container view controller.

Next, control-drag from the **Show Notes** button to the Show Notes View Controller, and select **push** from the popup that appears. At this point, your storyboard should look like this:



Next, you need to create a view controller to back this navigation controller so you can pass the podcast model object through, just like last time.

To do this, create a new file with the **iOS\Cocoa Touch\Objective-C class** template, name it **MyNavigationController**, and make it a subclass of **UIViewController**. Then open **MyNavigationController.h** and replace it with the following:

```
@class Podcast;

@interface MyNavigationController : UINavigationController

@property (strong, nonatomic) Podcast *podcast;

@end
```

Just as before, this simply has a property for the podcast to display.

Switch to **MyNavigationController.m** and add these imports to the top of the file:

```
#import "Podcast.h"
#import "MyContainerViewController.h"
```

Unlike the container view controller, in navigation controllers you don't put the initial setup code in **prepareForSegue:sender:**; instead, you put it in **viewDidLoad**. So replace **viewDidLoad** with the following:

```
- (void)viewDidLoad
{
  [super viewDidLoad];

  // Testing only, you'll delete this line in a moment
  self.podcast = [Podcast podcasts][2];

  // Pass podcast to container view controller
  MyContainerViewController *vc =
    [self.viewControllers objectAtIndex:0];
  vc.podcast = self.podcast;
}
```

The first line is a testing line – it sets the podcast to display to the third podcast, to check that everything is working OK. You will delete this in a moment.

Next, a navigation controller contains a property called **viewControllers** that lists all of the view controllers currently on the stack. Here you pick out the first item (the root view controller), which you know is a container view controller, and pass the podcast model object through.

Next, open **Main.storyboard**, select the navigation view controller, and in the Identity Inspector set the class to your new **MyNavigationController** to link the two.

The final step is to set up the podcast view controller to pass the podcast info to the show notes view controller. Still in **Main.storyboard**, select the push segue between the podcast and show notes view controllers and set the Identifier to **Show Notes**. Then open **MyPodcastViewController.m** and add this to the bottom of the file:

```
- (void)prepareForSegue:(UIStoryboardSegue *)segue
sender:(id)sender {
  if ([segue.identifier isEqualToString:@"ShowNotes"]) {
    MyShowNotesViewController *vc = (MyShowNotesViewController *)
      segue.destinationViewController;
    vc.podcast = self.podcast;
  }
}
```

As usual, before this segue occurs you have an opportunity to pass data to the new view controller, so you pass the podcast through here.

Build and run, and the info for the third podcast will appear (Cocoa Design Patterns). Even better, if you tap the Show Notes you should see the show notes page appear:

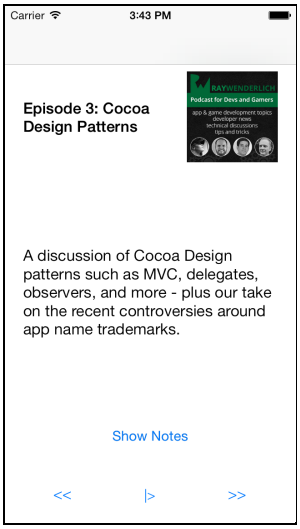
You may have noticed, however, that the container view controller appears behind the navigation bar, which probably isn't what you want:

Here's how you can fix this:

- Open Main.storyboard and resize the top container view controller so the top is below the navigation bar
- Delete the constraint between the top of the top container and the superview
- Control-drag between the top of the top container and the white space right above, choose **Top Layout Guide**
- Select that new constraint and set the constant to 0

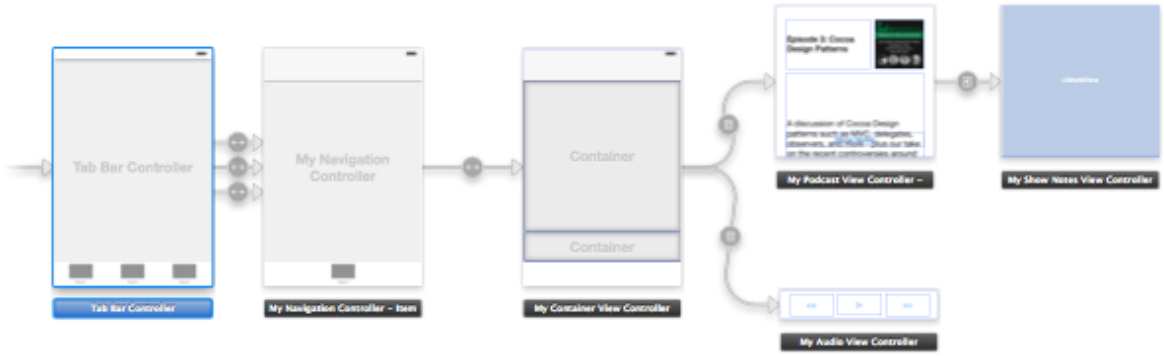
Basically what that did was switch the constraint between the top of the top container view from the **Superview** to the **Top Layout Guide**. The top layout guide is a special value that means "the topmost area, below any other views that might be up there like toolbars, navigation bars, etc."

Build and run, and it should look much better:



Also delete that test line in **MyNavigationController.m**'s **viewDidLoad**.

OK you're on to the final step – making a tab for each episode. To do this, open **Main.storyboard**, select your navigation controller, and select **Editor\Embed In\Tab Bar Controller**. It will create one tab item by default, but to create two more control-drag from the tab bar controller to the navigation controller and select **Relationship Segue\view controllers** two times. At this point your storyboard should look like this:



As usual, you need to create a class for the tab bar controller to pass the model data through. Create a new file with the **iOS\Cocoa Touch\Objective-C class** template, name it **MyTabBarController**, and make it a subclass of **UITabBarController**.

Back in **Main.storyboard**, select your tab bar controller and set its class to **UITabBarController** in the Identity Inspector.

Then open **MyTabBarController.m** and add these lines to the top of the file:

```
#import "Podcast.h"
#import "MyNavigationController.h"
```

Also add this new method:

```
- (void)setupViewController:(int)index
  withPodcast:(Podcast *)podcast {
  MyNavigationController *vc = self.viewControllers[index];
  vc.podcast = podcast;
  vc.tabBarItem.title = [NSString stringWithFormat:@"Episode %d",
    podcast.episodeNumber];
  vc.tabBarItem.image = [UIImage imageNamed:[NSString
    stringWithFormat:@"%d.png", podcast.episodeNumber]];
}
```

Usually you can set the tab bar item title and image inside the Storyboard editor, but in this case you can't because you're using the same view controller multiple times (so it needs a different tab bar item for each case). So here you'll set the tab bar item for each view controller programmatically before you display it.

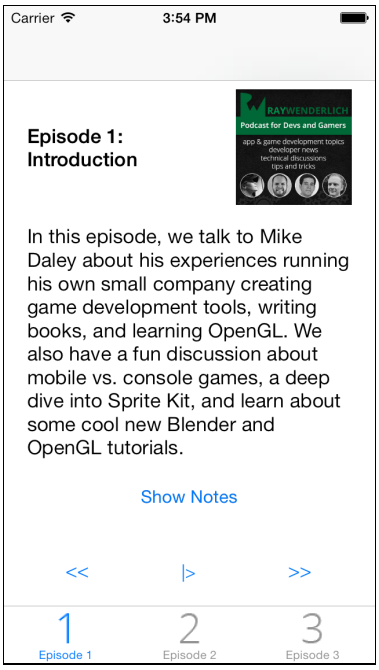
This is a helper method that looks in the tab bar controller's **viewControllers** array (one for each tab) and sets the podcast to the appropriate value. It also sets up the tab bar item with an image and title based on the episode.

Finally, replace viewDidLoad with the following:

```
- (void)viewDidLoad
{
  [super viewDidLoad];
  NSArray *podcasts = [Podcast podcasts];
  for (int i = 0; i < 3; i++) {
    [self setupViewController:i withPodcast:podcasts[i]];
  }
}
```

This calls the helper method with the appropriate podcast from the podcasts array.

Build and run, and enjoy some podcasts! :]



Uber Haxx0r Challenge

Your uber haxx0r challenge is to instrument this app using Reveal (http://www.revealapp.com) and take a look at the views to get a good idea of the view hierarchy.

Next, draw two diagrams:

- A simplified version of the view hierarchy when you're viewing and listening a podcast episode
- For the same screen, a diagram of the view controller hierarchy

If you have a good understanding of the difference here, congratulations – you have reached the uber haxx0r level!