# Creating and Distributing iOS Frameworks

*Michael Katz on June 7, 2016*

Have you ever wanted to share a chunk of code between two or more of your apps, or wanted to share a part of your program with other developers?

Maybe you wanted to modularize your code similarly to how the iOS SDK separates its API by functionality, or perhaps you want to distribute your code in the same way as popular 3rd parties do.

In this iOS frameworks tutorial you'll learn how to do all of the above!

In iOS 8 and Xcode 6, Apple provided a new template, **Cocoa Touch Framework**. As you'll see, it makes creating custom frameworks much easier than before.

*C'mon on in and learn how to create and distribute iOS frameworks!*

Frameworks have three major purposes:

- Code encapsulation
- Code modularity
- Code reuse

You can share your framework with your other apps, team members, or the iOS community. When combined with Swift's access control, frameworks help define strong, testable interfaces between code modules.

In Swift parlance, a **module** is a compiled group of code that is distributed together. A framework is one type of module, and an app is another example.

In this iOS frameworks tutorial, you'll extract a piece of an existing app and set it free, and by doing so, you'll learn the ins and outs of frameworks by:

- Creating a new framework for the rings widget
- Migrating the existing code and tests
- Importing the whole thing back into the app.
- Packing it up as an uber-portable CocoaPods
- Bonus: Setting up a repository for your framework

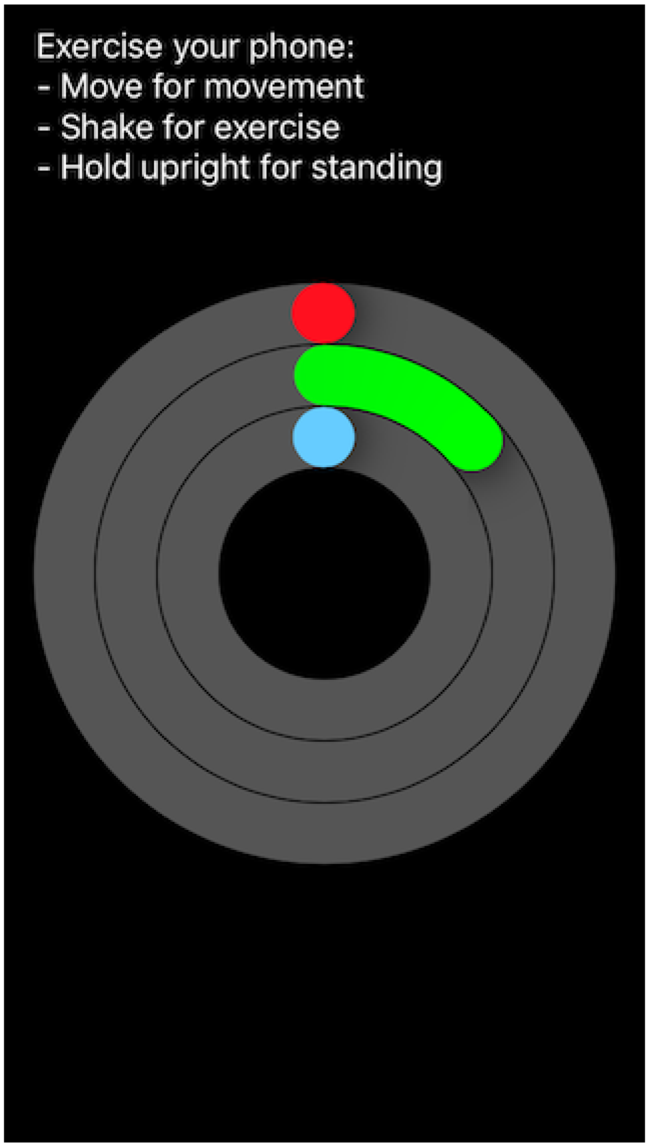
By the time you're done, the app will behave exactly as it did before, but it will use the portable framework you developed! :]

## Getting Started

Download the starter project.

**Phonercise** is a simple application that replicates the Apple Watch Activity app, except it measures your phone's physical activity. The three rings on the main view represent movement, standing and exercise.

To get the most out of the project, you'll need to build and run on an actual iOS device, and turn on the volume. Go ahead now, build and run!


Exercise your phone:
- Move for movement
- Shake for exercise
- Hold upright for standing

Move the phone around to get credit for movement, and "exercise" the phone by shaking it vigorously — that'll get its heartrate up. To get credit for standing up, just hold the phone upright.

The logic for the app is pretty simple:

- **ActionViewController.swift** contains the view lifecycle and motion logic.
- All the view logic is in the files in the **Three Ring View** folder, where you'll find **ThreeRingView.swift**, which handles the view, and **Fanfare.swift**, which handles the audio. The other files handle the custom drawing of the shading, gradient and shapes.

The ring controls are pretty sweet. They've got an addictive quality and they're easy to understand. Wouldn't it be nice to use them in a number of applications beyond this fun, but completely silly app? Frameworks to the rescue!

## Creating a Framework

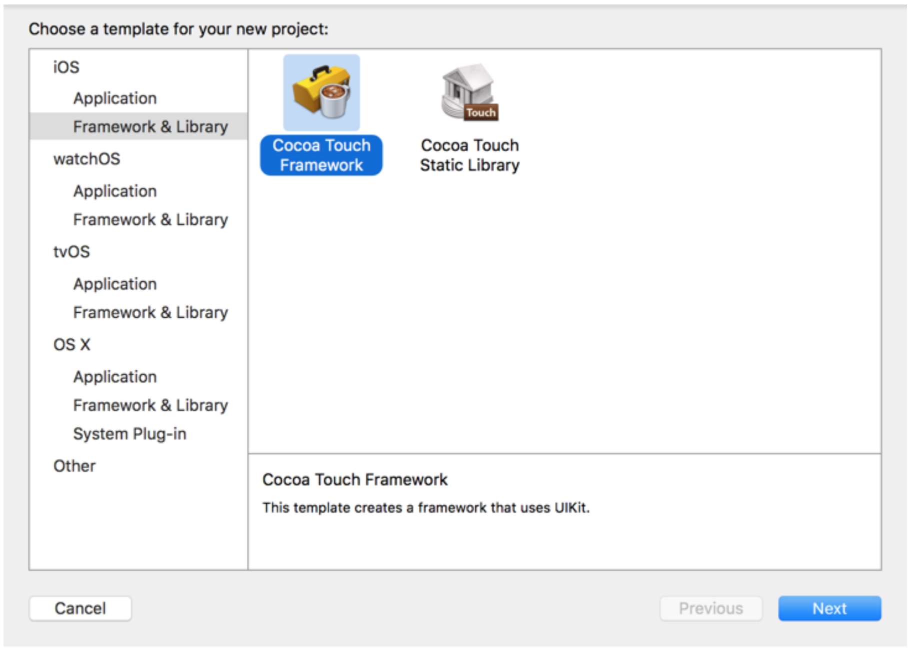
Frameworks are self-contained, reusable chunks of code and resources that you can import into any number of apps and even share across iOS, tvOS, watchOS, and macOS apps.

If you've programmed in other languages, you may have heard of node modules, packages, gems, jars, etc. Frameworks are the Xcode version of these. Some examples of common frameworks in the iOS SDK are: `Foundation`, `UIKit`, `AVFoundation`, `CloudKit`, etc.

## Framework Set Up

In Xcode 6, Apple introduced the **Cocoa Touch Framework** template along with **access control**, so creating frameworks has never been easier. The first thing to do is to create the project for the framework.

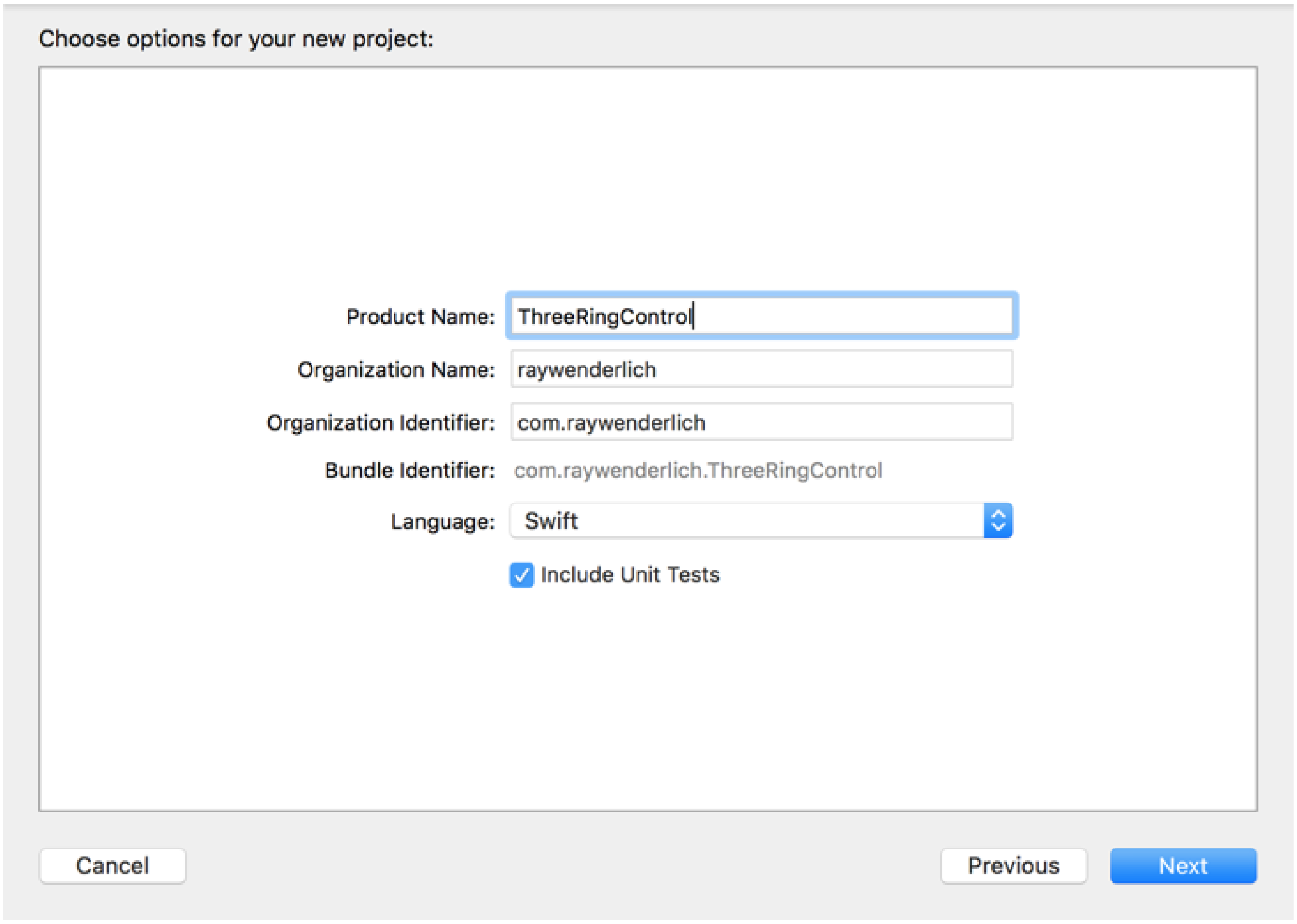
1. Create a new project. In Xcode, go to **File/New/Project**.
2. Choose **iOS/Framework & Library/Cocoa Touch Framework** to create a new framework.



Cocoa Touch Framework

This template creates a framework that uses UIKit.

3. Click **Next**.
4. Set the **Product Name** to `ThreeRingControl`. Use your own `Organization Name` and `Organization Identifier`. Check **Include Unit Tests**. That's right! You're going to have automated tests ensure your framework is bug free.


Choose options for your new project:

Product Name: ThreeRingControl

Organization Name: raywenderlich

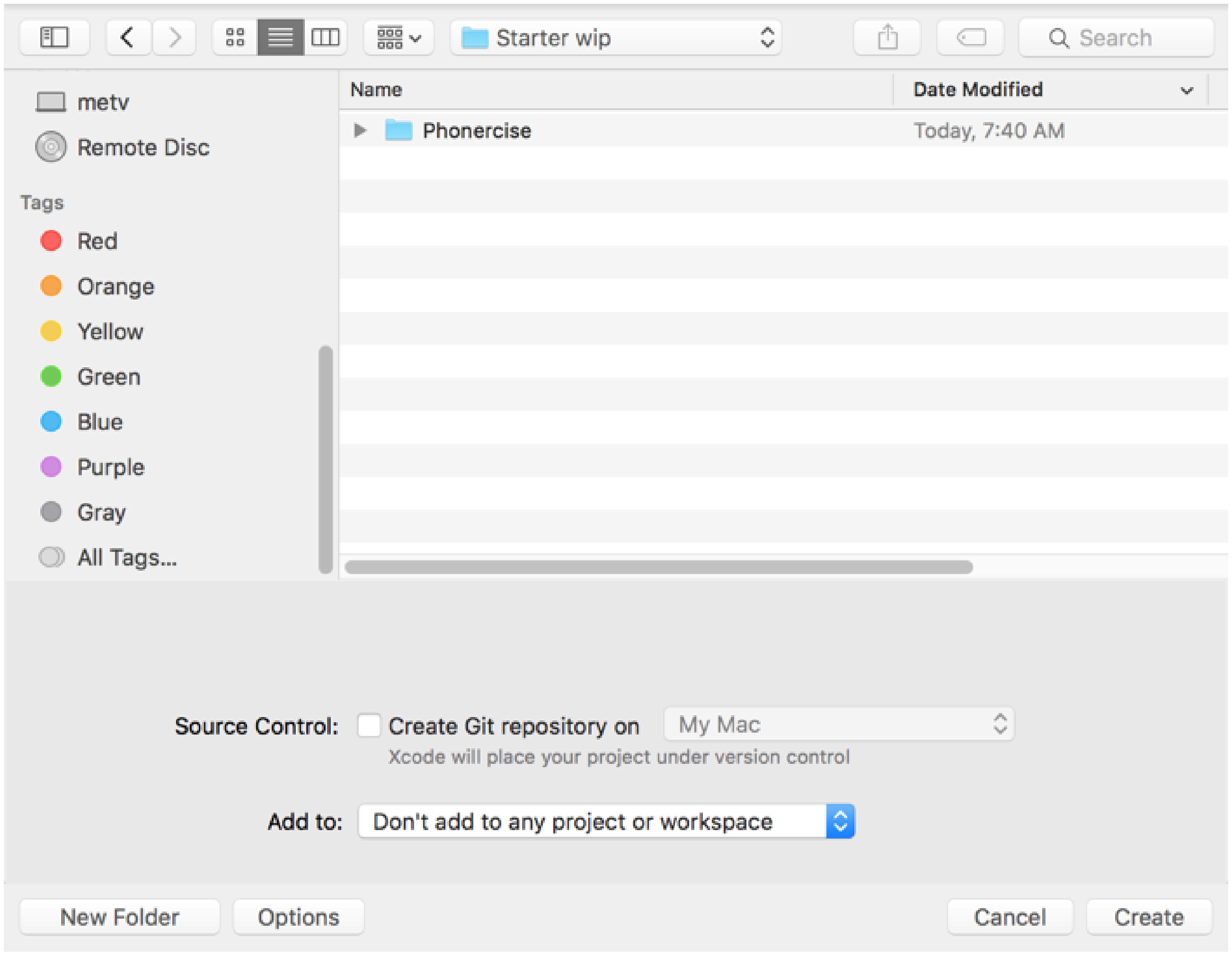
Organization Identifier: com.raywenderlich

Bundle Identifier: com.raywenderlich.ThreeRingControl

Language: Swift

[✓] Include Unit Tests

5. Click **Next**.
6. In the file chooser, choose to create the project at the same level as the `Phonercise` project.



Phonercise — Today, 7:40 AM

Source Control: ☐ Create Git repository on My Mac
Xcode will place your project under version control

Add to: Don't add to any project or workspace

7. Click **Create**.

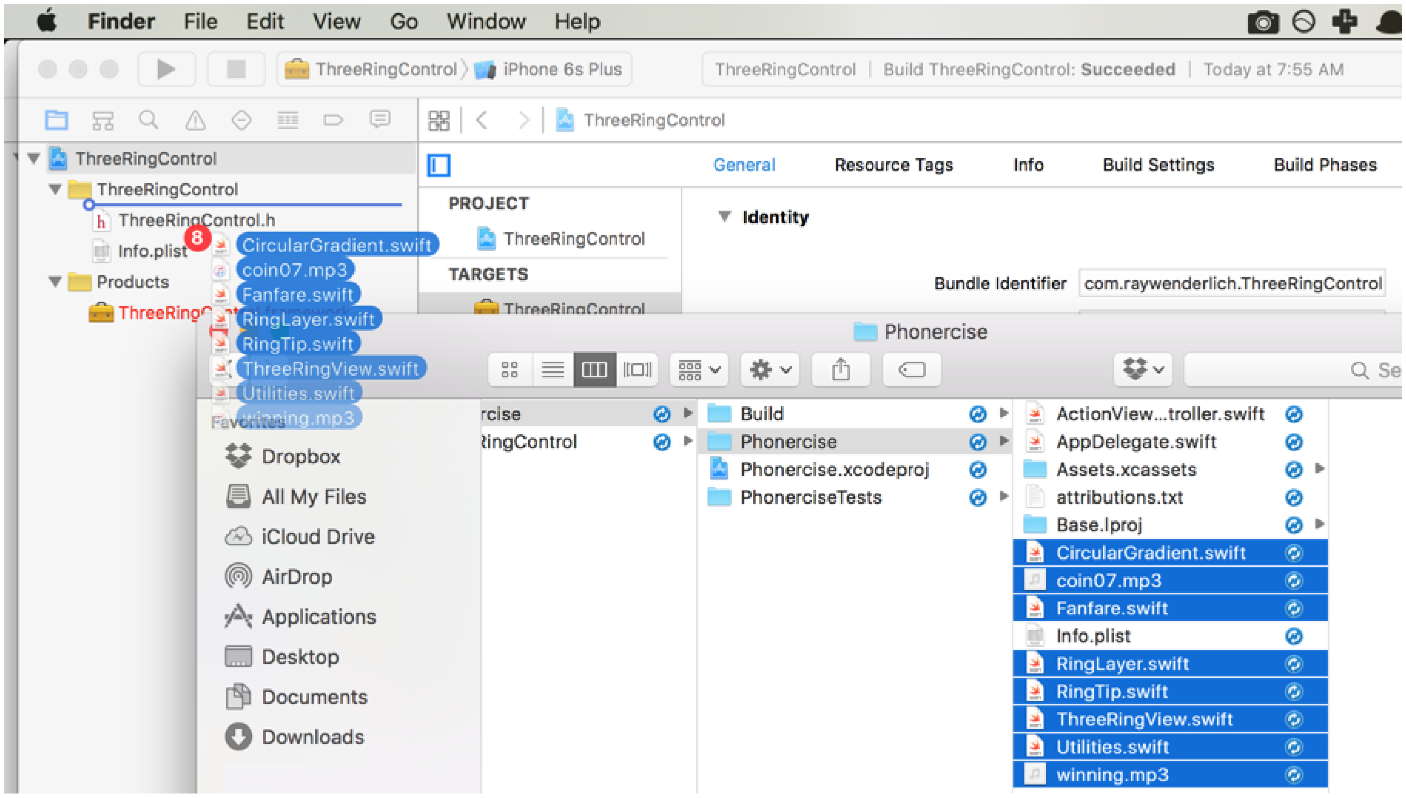
Now you have a project (albeit a boring one) that creates a framework!

## Add Code and Resources

Your current state is a framework without code, and that is about as appealing as straight chocolate without sugar. In this section, you'll put the pod in CocoaPods by adding the existing files to the framework.

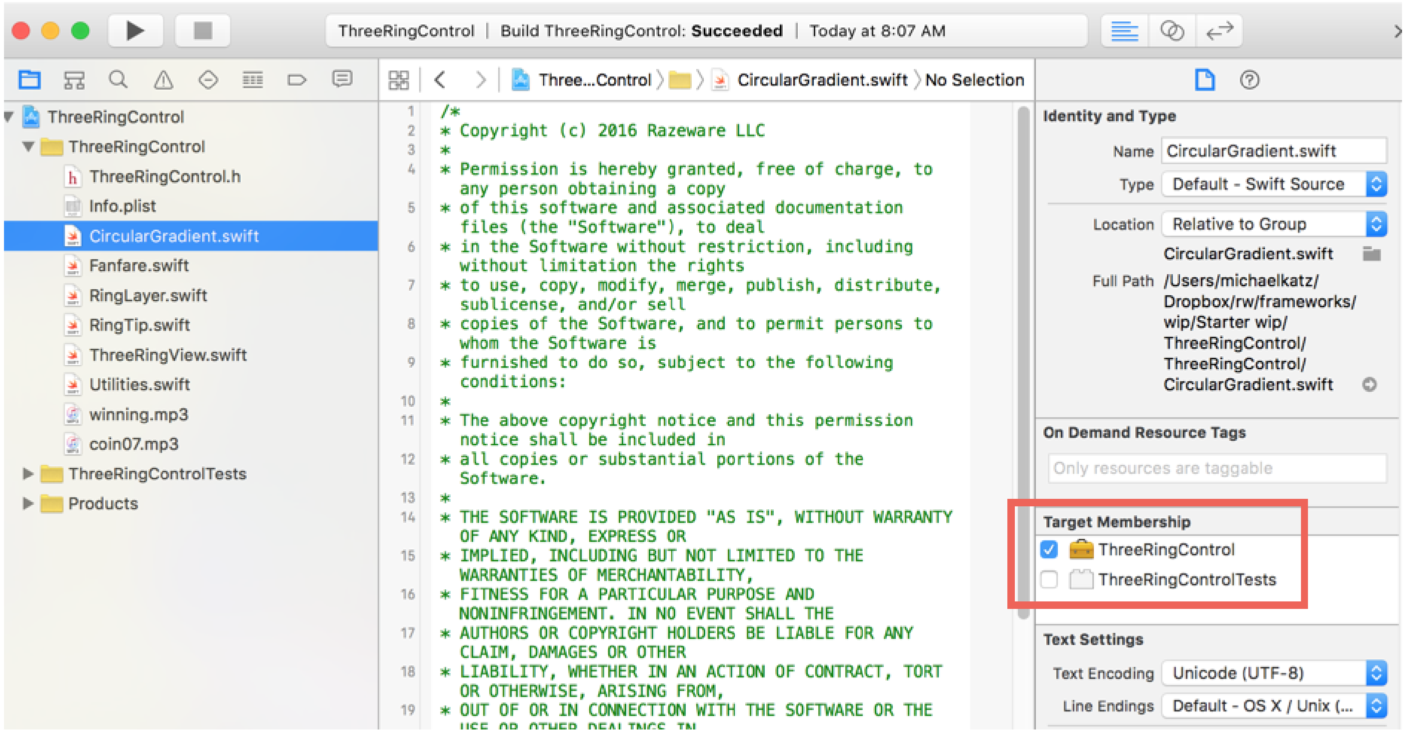
From the **Phonercise** source directory, drag the following eight files into the **ThreeRingControl** project in Xcode:

- CircularGradient.swift
- coin07.mp3
- Fanfare.swift
- RingLayer.swift
- RingTip.swift
- ThreeRingView.swift
- Utilities.swift
- winning.mp3



Make sure to check **Copy items if needed**, so that the files actually copy into the new project instead of just adding a reference. Frameworks need their own code, not references, to be independent.

Double-check that each of the files has **Target Membership** in **ThreeRingControl** to make sure they appear in the final framework. You can see this in the **File Inspector** for each file.



```
/*
* Copyright (c) 2016 Razeware LLC
*
* Permission is hereby granted, free of charge, to
  any person obtaining a copy
* of this software and associated documentation
  files (the "Software"), to deal
* in the Software without restriction, including
  without limitation the rights
* to use, copy, modify, merge, publish, distribute,
  sublicense, and/or sell
* copies of the Software, and to permit persons to
  whom the Software is
* furnished to do so, subject to the following
  conditions:
*
* The above copyright notice and this permission
  notice shall be included in
* all copies or substantial portions of the
  Software.
*
* THE SOFTWARE IS PROVIDED "AS IS", 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 AN ACTION OF CONTRACT, TORT
  OR OTHERWISE, ARISING FROM,
* OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE
```

Identity and Type

Name: CircularGradient.swift
Type: Default - Swift Source
Location: Relative to Group
CircularGradient.swift
Full Path: /Users/michaelkatz/Dropbox/rw/frameworks/wip/Starter wip/ThreeRingControl/ThreeRingControl/CircularGradient.swift

On Demand Resource Tags

Only resources are taggable

Target Membership

✓ ThreeRingControl
☐ ThreeRingControlTests

Text Settings

Text Encoding: Unicode (UTF-8)
Line Endings: Default - OS X / Unix (…

Build the framework project to make sure that you get `Build Succeeded` with no build warnings or errors.

## Add the Framework to the Project

Close the **ThreeRingControl** project, and go back to the **Phonercise** project. Delete the six files under the **Three Ring View** group as well as the two MP3 files in **Helper Files**. Select **Move to Trash** in the confirmation dialog.

Build the project, and you'll see several predictable errors where Xcode complains about not knowing what the heck a `ThreeRingView` is. Well, you'll actually see messages along the lines of "`Use of undeclared type 'ThreeRingView'`", among others.

Adding the Three Ring Control framework project to the workspace is the solution to these problems.

## Add the Framework to the Project

Right-click on the root **Phonercise** node in the project navigator. Click **Add Files to "Phonercise"**. In the file chooser, navigate to and select **ThreeRingControl.xcodeproj**. This will add **ThreeRingControl.xcodeproj** as a sub-project.

**Note:** It isn't strictly necessary to add the framework project to the app project; you could just add the `ThreeRingControl.framework` output.

However, combining the projects makes it easier to develop both the framework and app simultaneously. Any changes you make to the framework project are automatically propagated up to the app. It also makes it easier for Xcode to resolve the paths and know when to rebuild the project.

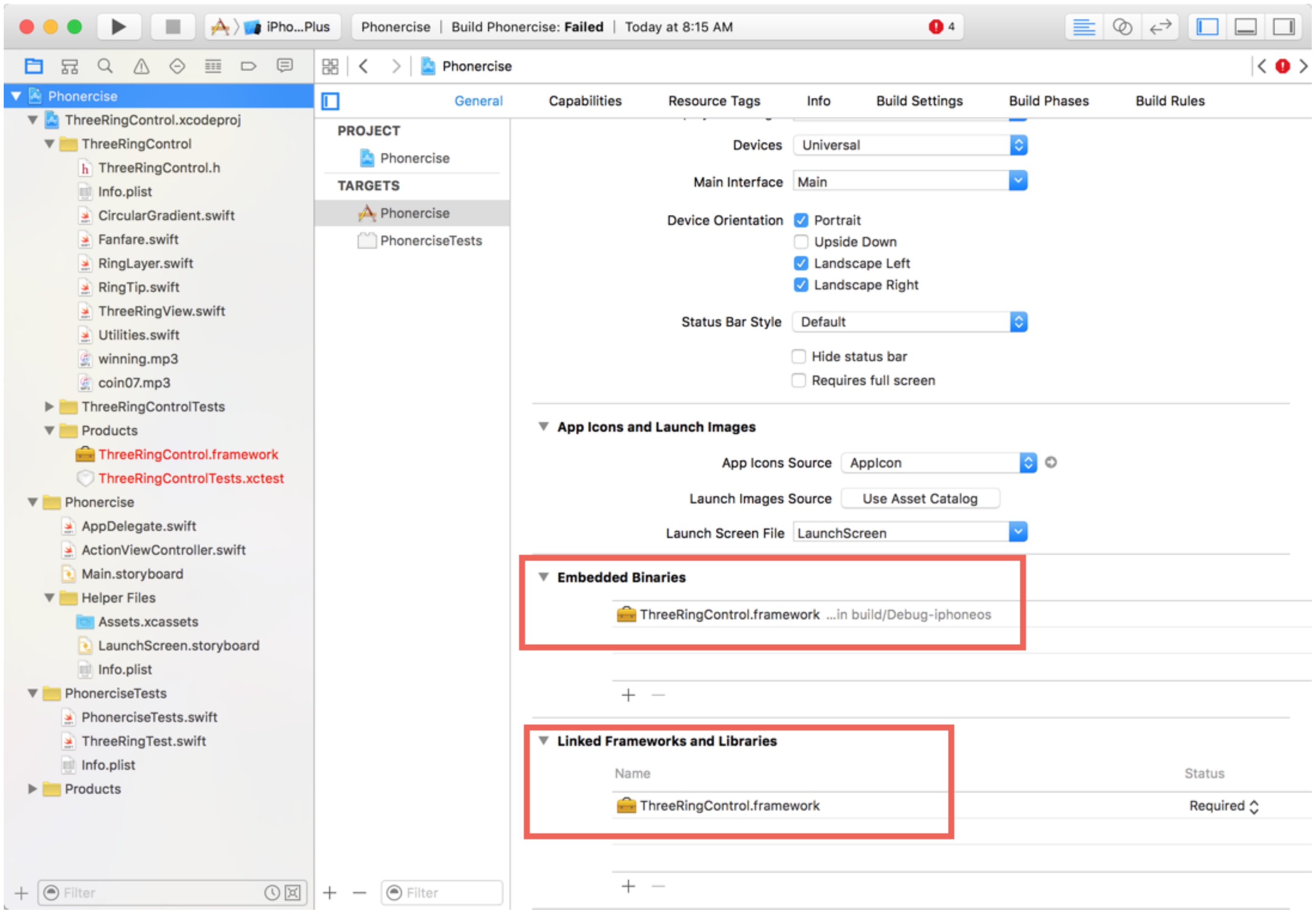
Even though the two projects are now together in the workspace, **Phonercise** still doesn't get **ThreeRingControl**. It's like they're sitting in the same room, but Phonercise can't see the new framework.

Try linking the framework to the app's target to fix this problem. First, expand the **ThreeRingControl** project to see the **Products** folder, and then look for for `ThreeRingControl.framework` beneath it. This file is the output of the framework project that packages up the binary code, headers, resources and metadata.

Select the top level **Phonercise** node to open the project editor. Click the **Phonercise** target, and then go to the **General** tab.

Scroll down to the **Embedded Binaries** section. Drag `ThreeRingControl.framework` from the **Products** folder of **ThreeRingControl.xcodeproj** onto this section.

You just added an entry for the framework in both **Embedded Binaries** and **Linked Frameworks and Binaries**.



Phonercise | Build Phonercise: **Failed** | Today at 8:15 AM

```
▼ Phonercise
  ▼ ThreeRingControl.xcodeproj
    ▼ ThreeRingControl
        ThreeRingControl.h
        Info.plist
        CircularGradient.swift
        Fanfare.swift
        RingLayer.swift
        RingTip.swift
        ThreeRingView.swift
        Utilities.swift
        winning.mp3
        coin07.mp3
    ▶ ThreeRingControlTests
    ▼ Products
        ThreeRingControl.framework
        ThreeRingControlTests.xctest
  ▼ Phonercise
      AppDelegate.swift
      ActionViewController.swift
      Main.storyboard
    ▼ Helper Files
        Assets.xcassets
        LaunchScreen.storyboard
        Info.plist
  ▼ PhonerciseTests
      PhonerciseTests.swift
      ThreeRingTest.swift
      Info.plist
  ▶ Products
```

PROJECT: Phonercise
TARGETS: Phonercise, PhonerciseTests

General | Capabilities | Resource Tags | Info | Build Settings | Build Phases | Build Rules

| | |
|---|---|
| Devices | Universal |
| Main Interface | Main |
| Device Orientation | [✓] Portrait<br>[ ] Upside Down<br>[✓] Landscape Left<br>[✓] Landscape Right |
| Status Bar Style | Default |
| | [ ] Hide status bar<br>[ ] Requires full screen |

▼ App Icons and Launch Images

| | |
|---|---|
| App Icons Source | AppIcon |
| Launch Images Source | Use Asset Catalog |
| Launch Screen File | LaunchScreen |

▼ Embedded Binaries

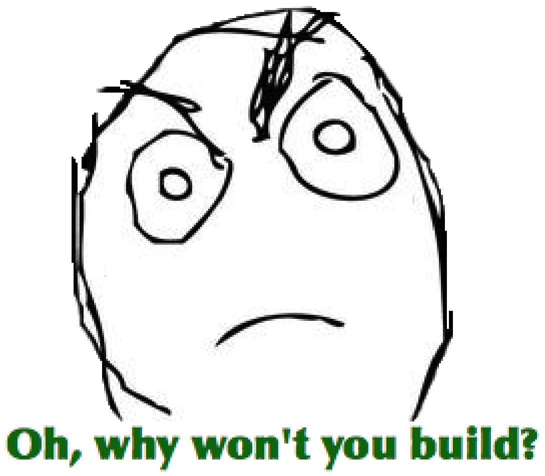
ThreeRingControl.framework ...in build/Debug-iphoneos

▼ Linked Frameworks and Libraries

| Name | Status |
|---|---|
| ThreeRingControl.framework | Required |

Now the app knows about the framework and where to find it, so that should be enough, right?

Build the **Phonercise** project. More of the same errors.



## Access Control

Your problem is that although the framework is part of the project, the project's code doesn't know about it — out of sight, out of mind.

Go to **ActionViewController.swift**, and add the following line to the list of imports at the top of the file.

```
import ThreeRingControl
```

It's critical, but this inclusion won't fix the build errors. This is because Swift uses **access control** to let you determine whether constructs are visible to other files or modules.

By default, Swift makes everything **`internal`** or visible only within its own module.

To restore functionality to the app, you have to update the access control on two **Phonercise** classes.

Although it's a bit tedious, the process of updating access control improves modularity by hiding code not meant to appear outside the framework. You do this by leaving certain functions with no access modifier, or by explicitly declaring them **`internal`**.

Swift has three levels of access control. Use the following rules of thumb when creating your own frameworks:

- **Public**: for code called by the app or other frameworks, e.g., a custom view.
- **Internal**: for code used between functions and classes within the framework, e.g., custom layers in that view.
- **Private**: for code used within a single file, e.g., a helper function that computes layout heights.

Ultimately, making frameworks is so much easier than in the past, thanks to the new Cocoa Touch Framework template. Apple provided both of these in Xcode 6 and iOS 8.

## Update the Code

When **ThreeRingView.swift** was part of the Phonercise app, internal access wasn't a problem. Now that it's in a separate module, it must be made **`public`** for the app to use it. The same is true of the code in **Fanfare.swift**.

Open **ThreeRingView.swift** inside of **`ThreeRingControlProject`**.

Make the class public by adding the **`public`** keyword to the class definition, like so:

```
public class ThreeRingView : UIView {
```

**`ThreeRingView`** will now be visible to any app file that imports the **`ThreeRingControl`** framework.

Add the **`public`** keyword to:

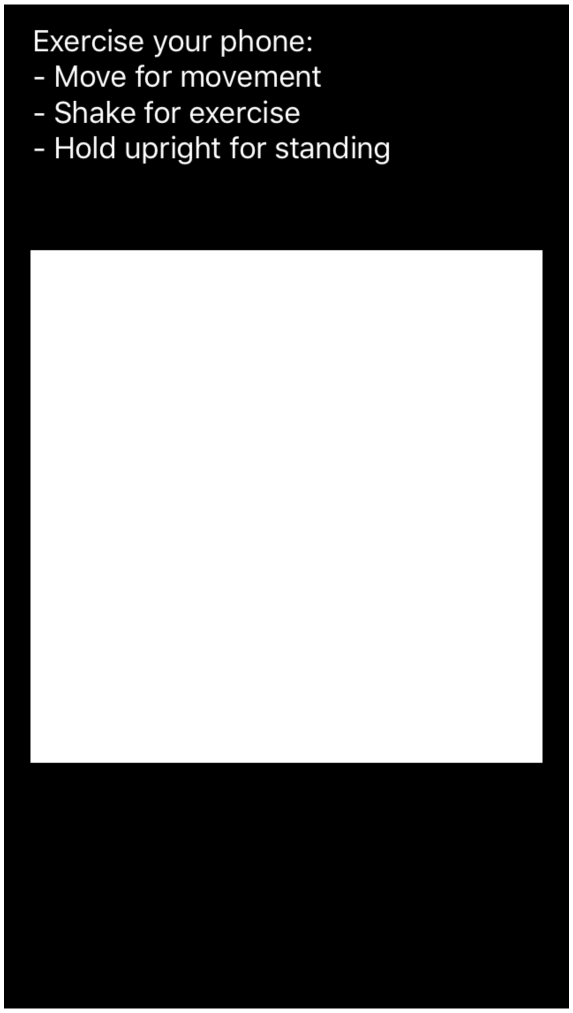
- Both **`init`** functions
- The variables **`RingCompletedNotification`**, **`AllRingsCompletedNotification`**, **`layoutSubviews()`**
- Everything marked **`@IBInspectable`** — there will be nine of these

**Note**: You might wonder why you have to declare **`inits`** as public. Apple explains this and other finer points of access control in their Access Control Documentation.

The next step is to do essentially the same thing as you did for **ThreeRingView.swift**, and add the **`public`** keyword to the appropriate parts of **Fanfare.swift**. For your convenience, this is already done.

Note that the following variables are public: **`ringSound`**, **`allRingSound`**, and **`sharedInstance`**. The function **`playSoundsWhenReady()`** is public as well.

Now build and run. The good news is that the errors are gone, and the bad news is that you've got a big, white square. Not a ring in sight. Oh no! What's going on?


Exercise your phone:
- Move for movement
- Shake for exercise
- Hold upright for standing

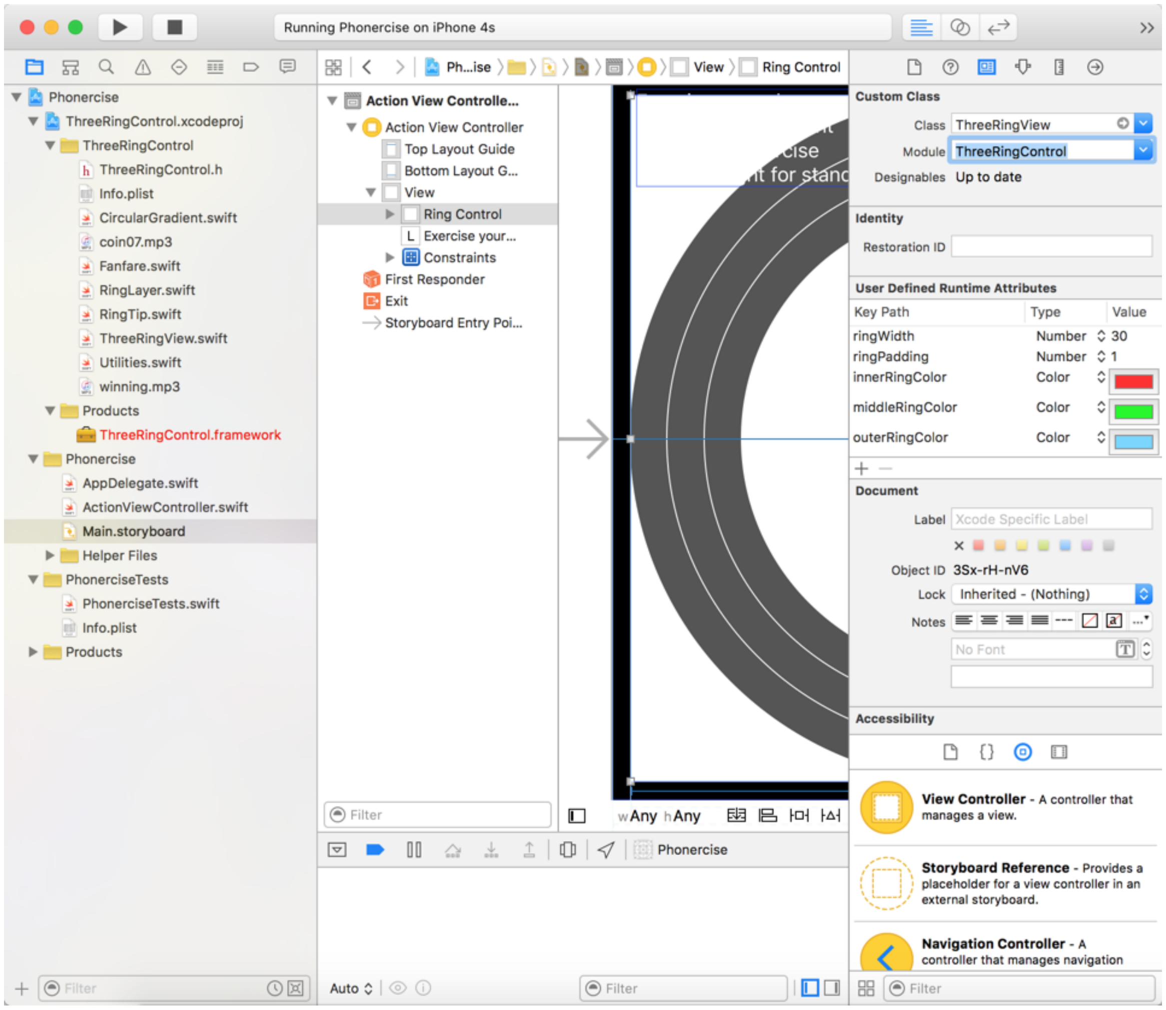
## Update the Storyboard

When using storyboards, references to custom classes need to have both the class name and module set in the **Identity Inspector**. At the time of this storyboard's creation, `ThreeRingView` was in the app's module, but now it's in the framework.

Update the storyboard, as explained below, by telling it where to find the custom view; this will get rid of the white square.

1. Open **Main.Storyboard** in the **Phonercise** project.
2. Select the **Ring Control** in the view hierarchy.
3. In the **Identity Inspector**, under **Custom Class**, change the **Module** to `ThreeRingControl`.

Once you set the module, Interface Builder should update and show the control in the editor area.



**Custom Class**

Class: ThreeRingView
Module: ThreeRingControl
Designables: Up to date

**Identity**

Restoration ID:

**User Defined Runtime Attributes**

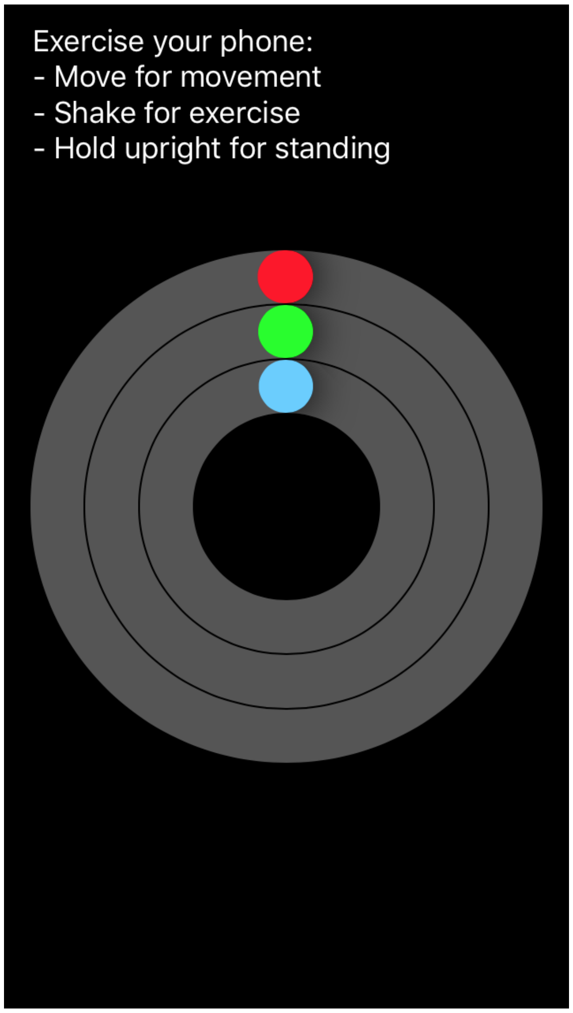
| Key Path | Type | Value |
|---|---|---|
| ringWidth | Number | 30 |
| ringPadding | Number | 1 |
| innerRingColor | Color | 🟥 |
| middleRingColor | Color | 🟩 |
| outerRingColor | Color | 🟦 |

**Document**

Label: Xcode Specific Label
Object ID: 3Sx-rH-nV6
Lock: Inherited - (Nothing)
Notes: No Font

**Accessibility**

Build and run. Now you'll get some rings.



## Update the Tests

There's one final place to update before you can actually use the framework. It's the oft-forgotten test suite. :]

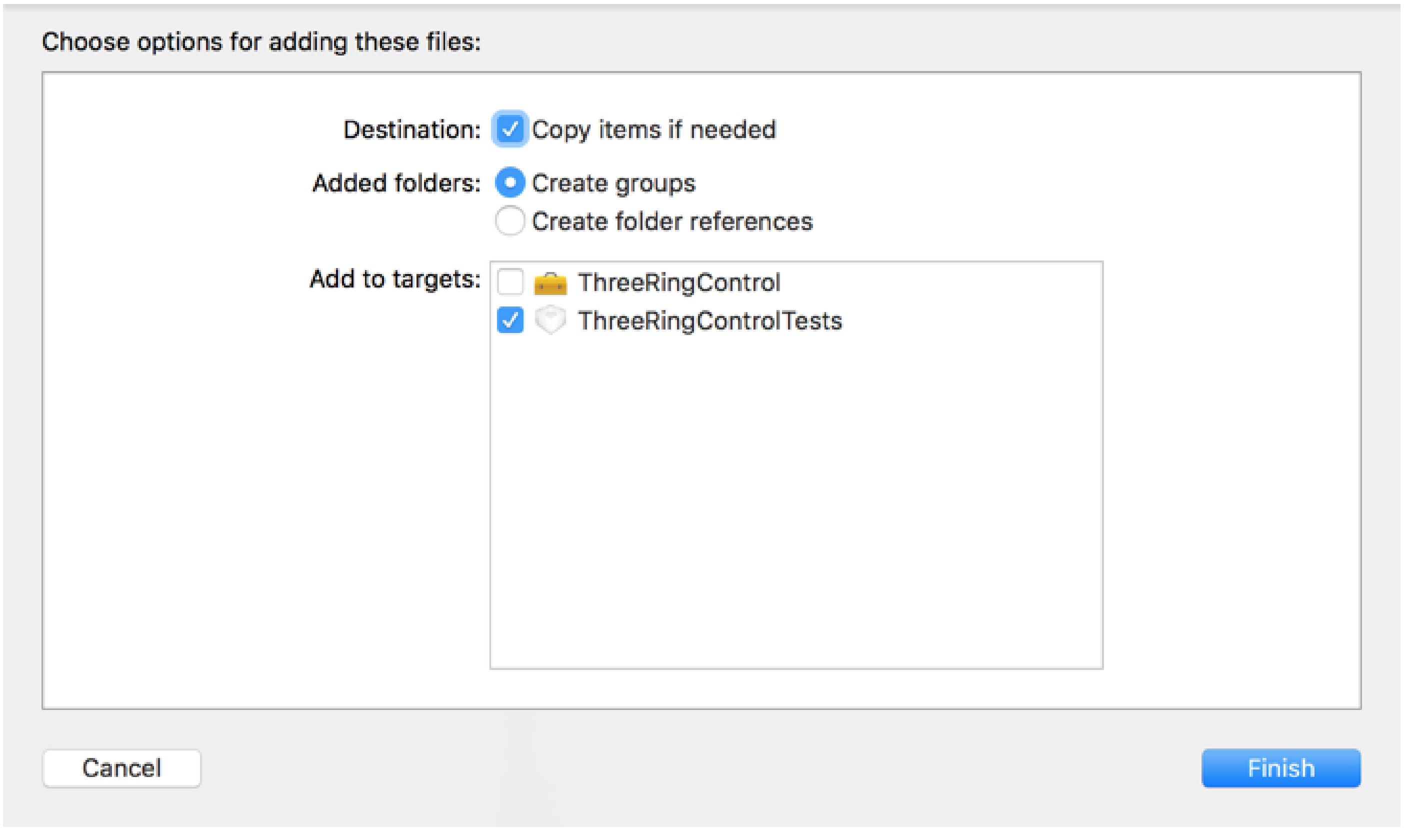
This app includes unit tests. However, the tests have a dependency on `ThreeRingControl`, so you need to fix that.

> **Note:** Before Swift 2.0, unit testing was difficult, but Swift 2.0 came along with `@testable`, making testing far easier. It means you no longer have to make all your project's items public just to be able to test them.
>
> Simply import the module you want to test using the `@testable` keyword, and your test target will have access to all internal routines in the imported module.

The first test is **ThreeRingTest.swift**, which tests functionality within **ThreeRingView.swift** and should be part of the **ThreeRingControl** framework.

From the **PhonerciseTests** group in the app project, drag **ThreeRingTest.swift** to the **ThreeRingControlTests** group in the framework project.

Select **Copy items if needed**, and make sure you select the **ThreeRingControlTests** target and not the framework target.


Choose options for adding these files:

Destination: [✓] Copy items if needed

Added folders: [✓ Create groups] Create folder references

Add to targets:
- [ ] ThreeRingControl
- [✓] ThreeRingControlTests

Then delete the **ThreeRingTest.swift** from the app project. Move it to the trash.

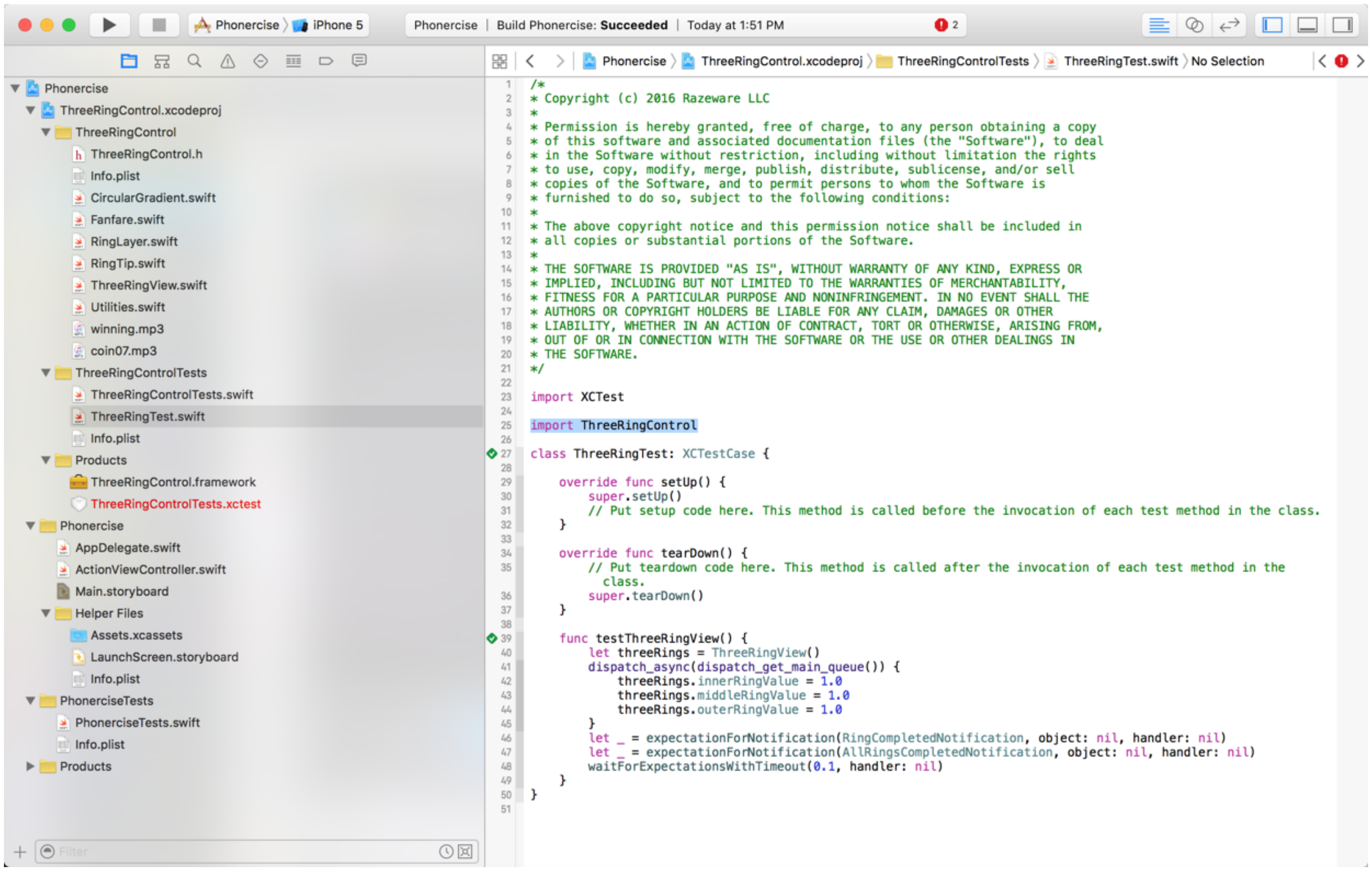
Open the copied test file, and change the import line from:

```
@testable import Phonercise
```

To:

```
import ThreeRingControl
```

Click the **run test** button in the editor’s gutter, and you’ll see a green checkmark showing that the test succeeded.



```
/*
* Copyright (c) 2016 Razeware LLC
*
* Permission is hereby granted, free of charge, to any person obtaining a copy
* of this software and associated documentation files (the "Software"), to deal
* in the Software without restriction, including without limitation the rights
* to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
* copies of the Software, and to permit persons to whom the Software is
* furnished to do so, subject to the following conditions:
*
* The above copyright notice and this permission notice shall be included in
* all copies or substantial portions of the Software.
*
* THE SOFTWARE IS PROVIDED "AS IS", 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 AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
* OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
* THE SOFTWARE.
*/

import XCTest

import ThreeRingControl

class ThreeRingTest: XCTestCase {

    override func setUp() {
        super.setUp()
        // Put setup code here. This method is called before the invocation of each test method in the class.
    }

    override func tearDown() {
        // Put teardown code here. This method is called after the invocation of each test method in the
          class.
        super.tearDown()
    }

    func testThreeRingView() {
        let threeRings = ThreeRingView()
        dispatch_async(dispatch_get_main_queue()) {
            threeRings.innerRingValue = 1.0
            threeRings.middleRingValue = 1.0
            threeRings.outerRingValue = 1.0
        }
        let _ = expectationForNotification(RingCompletedNotification, object: nil, handler: nil)
        let _ = expectationForNotification(AllRingsCompletedNotification, object: nil, handler: nil)
        waitForExpectationsWithTimeout(0.1, handler: nil)
    }
}
```

This test doesn't need **`@testable`** in its import since it is only calling **`public`** methods and variables.

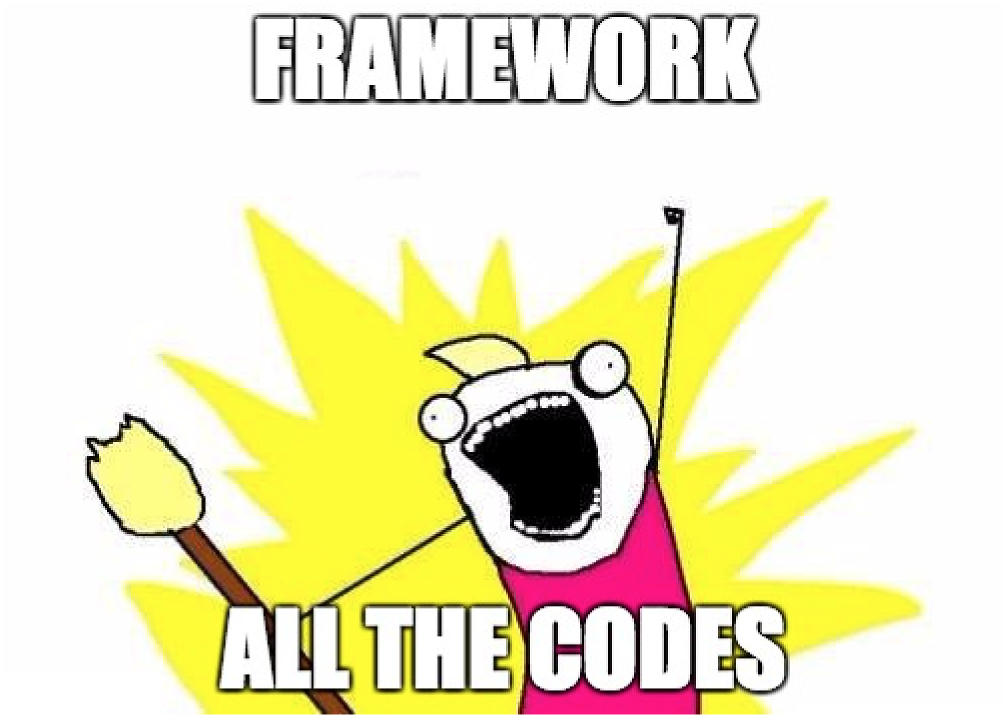
The remaining app test doesn't work this way though. Go back the to the **Phonercise** target and open **PhonerciseTests.swift**.

Add the following import to the existing imports at the top:

```
import ThreeRingControl
```

Now run the **PhonerciseTests** tests. The particular test should execute and pass.

Congratulations. You now have a working stand-alone framework and an app that uses it!



## Creating a CocoaPod

CocoaPods is a popular dependency manager for iOS projects. It's a tool for managing and versioning dependencies. Similar to a framework, a CocoaPod, or 'pod' for short, contains code, resources, etc., as well as metadata, dependencies, and set up for libraries. In fact, CocoaPods are built as frameworks that are included in the main app.

Anyone can contribute libraries and frameworks to the public repository, which is open to other iOS app developers. Almost all of the popular third-party frameworks, such as Alamofire, React native and SDWebImage, distribute their code as a pod.

Here's why you should care: by making a framework into a pod, you give yourself a mechanism for distributing the code, resolving the dependencies, including and building the framework source, and easily sharing it with your organization or the wider development community.

## Clean out the Project

Perform the following steps to remove the current link to **`ThreeRingControl`**.

1. Select **`ThreeRingControl.xcodeproj`** in the project navigator and delete it.
2. Choose **Remove Reference** in the confirmation dialog, since you'll need to keep the files on disk to create the pod.

## Install CocoaPods

If you've never used CocoaPods before, you'll need to follow a brief installation process before going any further. Go to the CocoaPods Installation Guide and come back here when you're finished. Don't worry, we'll wait!

## Create the Pod

Open Terminal in the **`ThreeRingControl`** directory.

Run the following command:

```
pod spec create ThreeRingControl
```

This creates the file **ThreeRingControl.podspec** in the current directory. It's a template that describes the pod and how to build it. Open it in a text editor.

The template contains plenty of comment descriptions and suggestions for the commonly used settings.

1. Replace the **`Spec Metadata`** section with:

   ```
   s.name         = "ThreeRingControl"
   s.version      = "1.0.0"
   s.summary      = "A three-ring control like the Activity status bars"
   s.description  = "The three-ring is a completely customizable widget that can be used in
   any iOS app. It also plays a little victory fanfare."
   s.homepage     = "http://raywenderlich.com"
   ```

   Normally, the description would be a little more descriptive, and the homepage would point to a project page for the framework.

2. Replace the **`Spec License`** section with the below code, as this iOS frameworks tutorial code uses an MIT License:

   ```
   s.license      = "MIT"
   ```

3. You can keep the **`Author Metadata`** section as is, or set it u with how you'd lke to be credited and contacted.

4. Replace the **`Platform Specifics`** section with the below code, because this is an iOS-only framework.:

   ```
   s.platform     = :ios, "9.0"
   ```

5. Remove the comments from **`Source Location`**, but keep the stub **`git`** link for **`s.source`**. When you're ready to share the pod, this will be a link to the GitHub repository and the commit tag for this version.

6. Replace the **`Source Code`** section with:

   ```
   s.source_files = "ThreeRingControl", "ThreeRingControl/**/*.{h,m,swift}"
   ```

7. And the **`Resources`** section with:

   ```
   s.resources    = "ThreeRingControl/*.mp3"
   ```

   This will include the audio resources into the pod's framework bundle.

8. Remove the **`Project Linking`** and **`Project Settings`** sections.

9. Remove all the comments — the lines that start with **`#`**.

You now have a workable development Podspec.

**Note:** If you run **`pod spec lint`** to verify the **Podspec** in Terminal, it'll show an error because the **`source`** was not set to a valid URL. If you push the project to GitHub and fix that link, it will pass. However, having the linter pass is not necessary for local pod development. The **Publish the Pod** section below covers this.

## Use the Pod

At this point, you've got a pod ready to rock and roll. Test it out by implementing it in the **Phonercise** app.

Back in Terminal, navigate up to the **`Phonercise`** directory, and then run the following command:

```
pod init
```

This steathily creates a new file named **Podfile** that lists all pods that the app uses, along with their versions and optional configuration information.

Open **Podfile** in a text editor.

As suggested by the Podfile comments, uncomment the **`use_frameworks!`** line to have the pod build as an iOS framework.

Next, add the following line between **`target 'Phonercise' do`** and **`end`**:

```
pod 'ThreeRingControl', :path => '../ThreeRingControl'
```

Save the file.

Run this in Terminal:

```
pod install
```

With this command, you're searching the CocoaPods repository and downloading any new or updated pods that match the Podfile criteria. It also resolves any dependencies, updates the Xcode project files so it knows how to build and link the pods, and performs any other required configuration.

Finally, it creates a **Phonercise.xcworkspace** file. Use this file from now on — instead of the `xcodeproj` — because it has the reference to the Pods project and the app project.

## Check it Out

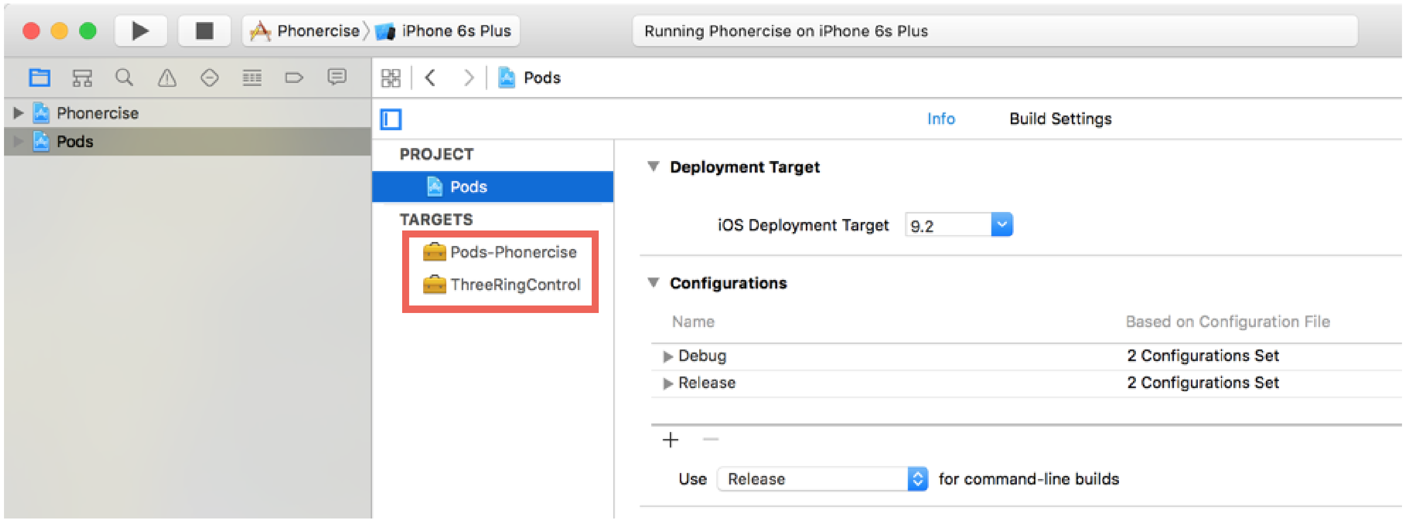
Close the `Phonercise` and `ThreeRingControl` projects if they are open, and then open **Phonercise.xcworkspace**.

Build and run. Like magic, the app should work exactly the same before. This ease of use is brought to you by these two facts:

1. You already did the work to separate the `ThreeRingControl` files and use them as a framework, e.g. adding import statements.
2. CocoaPods does the heavy lifting of building and packaging those files; it also takes care of all the business around embedding and linking.

## Pod Organization

Take a look at the **Pods** project, and you'll notice two targets:



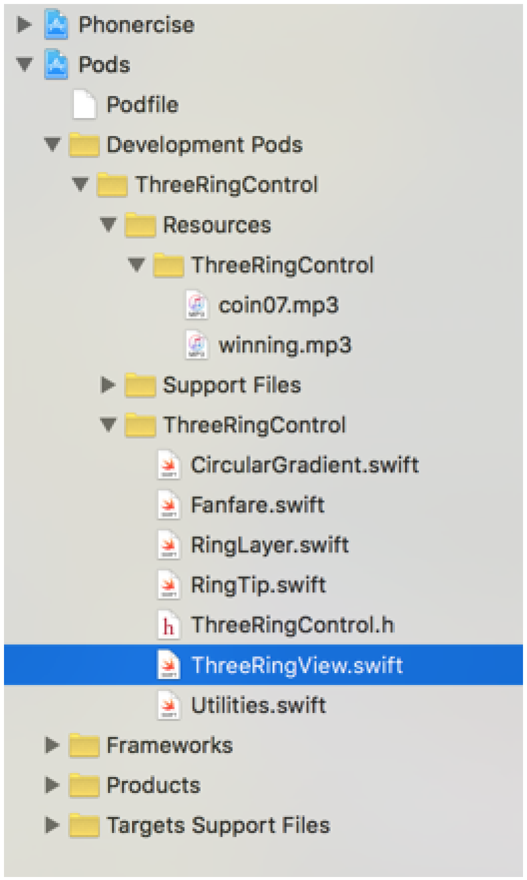
iOS Deployment Target 9.2

| Name | Based on Configuration File |
|---|---|
| Debug | 2 Configurations Set |
| Release | 2 Configurations Set |

Use Release for command-line builds

- **Pods-phonercise**: a pod project builds all the individual pods as their own framework, and then combines them into one single framework: `Pods-Phonercise`.
- **ThreeRingControl**: this replicates the same framework logic used for building it on its own. It even adds the music files as framework resources.

Inside the project organizer, you'll see several groups. `ThreeRingControl` is under `Development Pods`. This is a **development pod** because you defined the pod with `:path` link in the app's **Podfile**. These files are symlinked, so you can edit and develop this code side-by-side with the main app code.



Pods that come from a repository, such as those from a third party, are copied into the **Pods** directory and listed in a **Pods** group. Any modifications you make are not pushed to the repository and are overwritten whenever you update the pods.

Congratulations! You've now created and deployed a CocoaPod — and you're probably thinking about what to pack into a pod first.

You're welcome to stop here, congratulate yourself and move on to **Where to Go From Here**, but if you do, you'll miss out on learning an advanced maneuver.

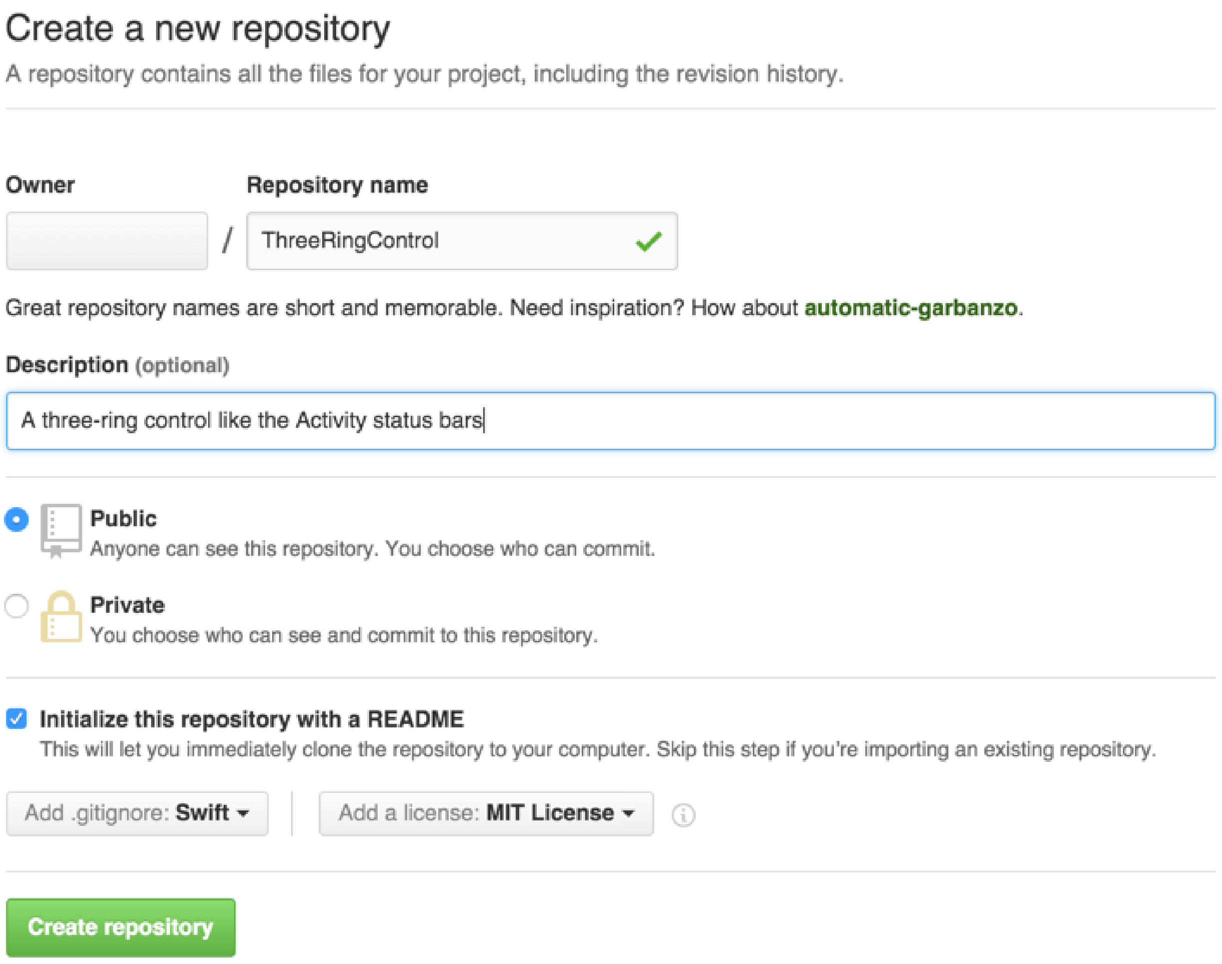
## Publish the Pod

This section walks you through the natural next step of publishing the pod to GitHub and using it like a third party framework.

## Create a Repository

If you don't already have a GitHub account, create one.

Now create a new repository to host the pod. **ThreeRingControl** is the obvious best fit for the name, but you can name it whatever you want. Be sure to select **Swift** as the **`.gitignore`** language and **MIT** as the license.


Create a new repository
A repository contains all the files for your project, including the revision history.
Owner
Repository name
ThreeRingControl
Great repository names are short and memorable. Need inspiration? How about automatic-garbanzo.
Description (optional)
A three-ring control like the Activity status bars
Public
Anyone can see this repository. You choose who can commit.
Private
You choose who can see and commit to this repository.
Initialize this repository with a README
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.
Add .gitignore: Swift
Add a license: MIT License
Create repository

Click **Create Repository**. From the dashboard page that follows, copy the **HTTPS** link.

## Clone the Repository

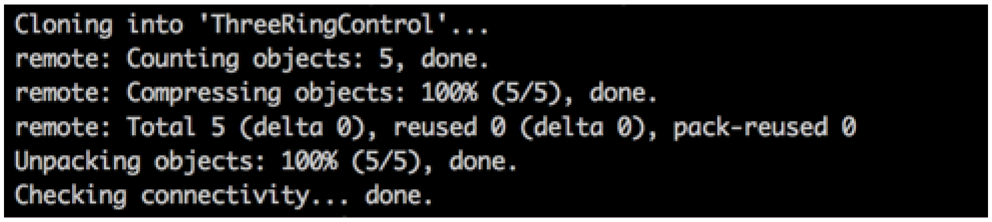
Go back to Terminal and create a new directory. The following commands will create a **repo** directory from the project's folder.

```
mkdir repo
cd repo
```

From there, **clone** the GitHub repository. Replace the **URL** below with the **HTTPS** link from the GitHub page.

```
git clone URL
```

This will set up a Git folder and go on to copy the pre-created **README** and **LICENSE** files.



```
Cloning into 'ThreeRingControl'...
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (5/5), done.
Checking connectivity... done.
```

## Add the Code to the Repository

Next, copy the files from the previous **ThreeRingControl** directory to the **repo/ThreeRingControl** directory.

Open the copied version of **ThreeRingControl.podspec**, and update the **`s.source`** line to:

```
s.source       = { :git => "URL", :tag => "1.0.0" }
```

Set the **URL** to be the link to your repository.

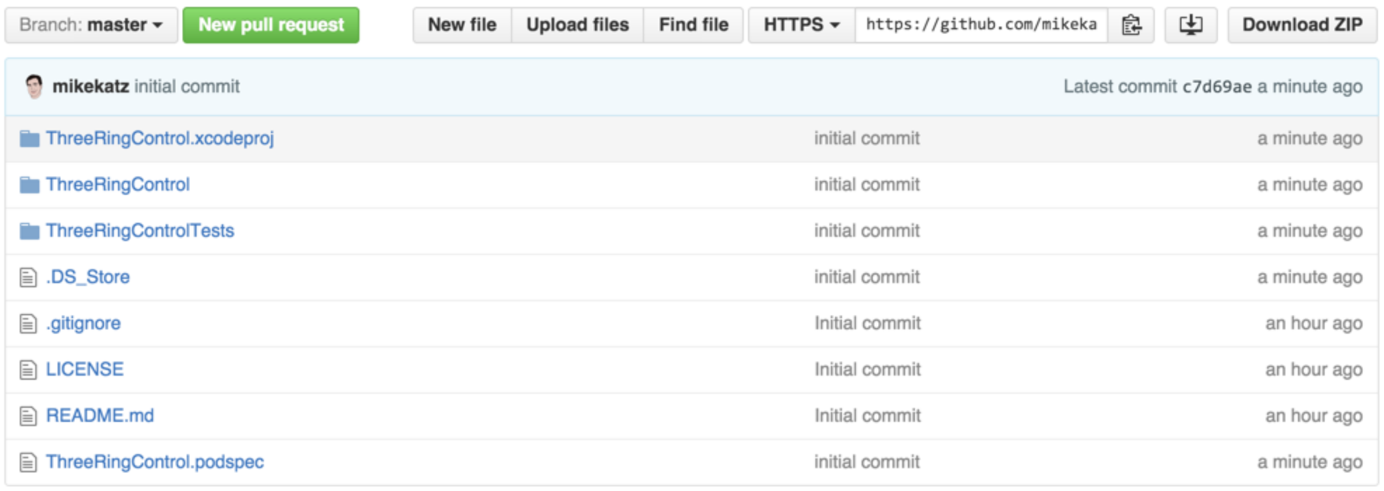
## Make the Commitment

Now it gets real. In this step, you'll commit and push the code to GitHub. Your little pod is about to enter the big kid's pool.

Run the following commands in Terminal to commit those files to the repository and push them back to the server.

```
cd ThreeRingControl/
git add .
git commit -m "initial commit"
git push
```

Visit the GitHub page and refresh it to see all the files.



mikekatz initial commit — Latest commit c7d69ae a minute ago

| | | |
|---|---|---|
| ThreeRingControl.xcodeproj | initial commit | a minute ago |
| ThreeRingControl | initial commit | a minute ago |
| ThreeRingControlTests | initial commit | a minute ago |
| .DS_Store | initial commit | a minute ago |
| .gitignore | Initial commit | an hour ago |
| LICENSE | Initial commit | an hour ago |
| README.md | Initial commit | an hour ago |
| ThreeRingControl.podspec | initial commit | a minute ago |

## Tag It

You need to tag the repository so it matches the Podspec. Run this command to set the tags:

```
git tag 1.0.0
git push --tags
```

Check your handiwork by running:

```
pod spec lint
```

The response you're looking for is **`ThreeRingControl.podspec passed validation.`**

## Update the Podfile

Change the **Podfile** in the **Phonercise** directory. Replace the existing **ThreeRingControl** line with:

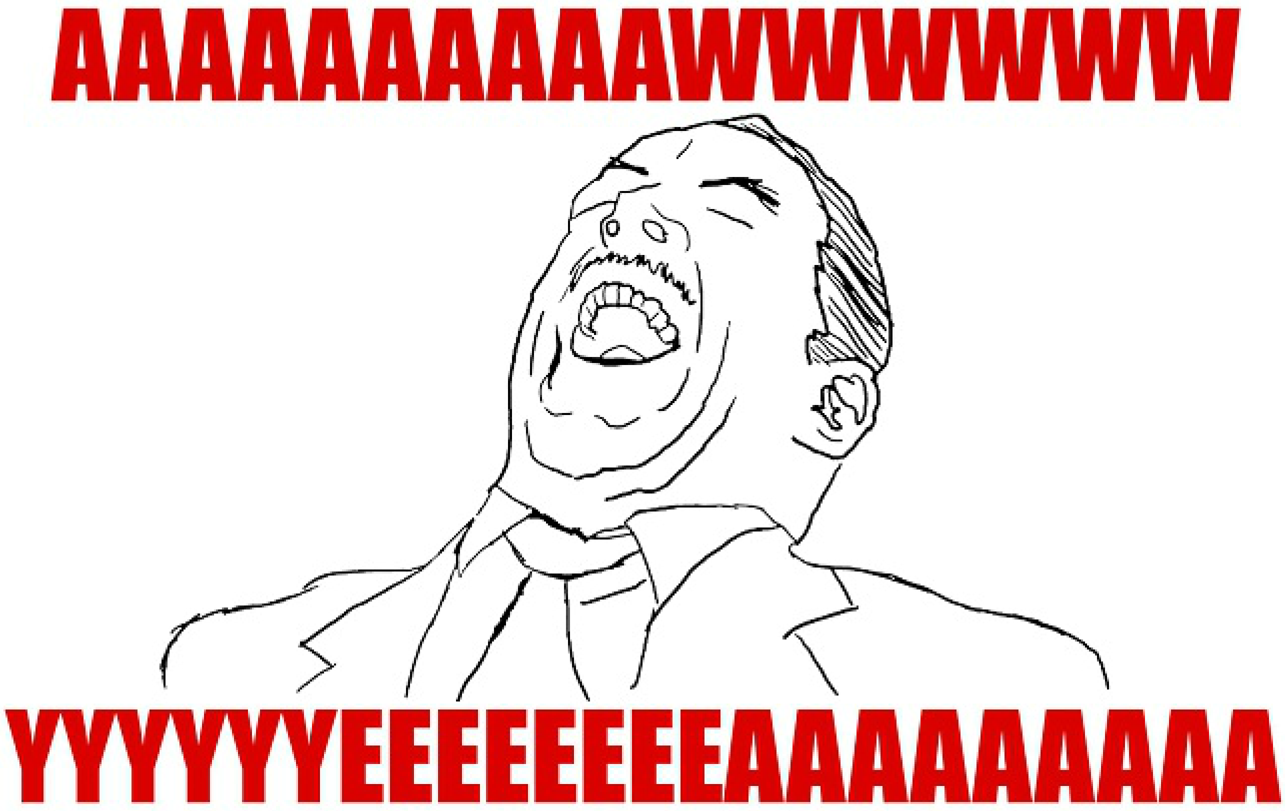
```
pod 'ThreeRingControl', :git => 'URL', :tag => '1.0.0'
```

Replace **URL** with your GitHub link.

From Terminal, run this in the **Phonercise** directory:

```
pod update
```

Now the code will pull the framework from the GitHub repository, and it is no longer be a development pod!



## Where to Go From Here?

In this iOS frameworks tutorial, you've made a framework from scratch that contains code and resources, you've reimported that framework back into your app, and even turned it into a CocoaPod. Nice work!

You can download the final project here. It doesn't include the "advanced maneuver" steps, as the steps to set up a GitHub repository depends on your personal account information.

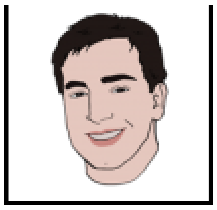
Hats off to Sam Davies for developing the `ThreeRingControl`. You might remember him from such videos as Introducing Custom Controls, where you can learn more about custom controls and custom frameworks.

The team here has put together a lot of tutorials about CocoaPods, and now that you've gone through a crash course, you're ready to learn more. Here are a couple that you might like:

- How to Use CocoaPods with Swift
- How to create a CocoaPod in Swift

Spend some time at CocoaPods.org and be sure to check out how to submit to the public pod repository and the dozens of configuration flags.

What did you learn from this? Any lingering questions? Want to share something that happened along the way? Let's talk about it in the forums. See you there!



## Michael Katz

*Michael Katz envisions a world where mobile apps always work, respect users' privacy, and integrate well with their users' life. When not coding, he can be found with his family playing board games, brewing, gardening, and watching the Yankees.*