The App Delegate Source File

The AppDelegate.swift source file has two primary functions:

* It defines your AppDelegate class. The [app delegate](https://developer.apple.com/library/content/referencelibrary/GettingStarted/DevelopiOSAppsSwift/GlossaryDefinitions.html#//apple_ref/doc/uid/TP40015214-CH12-SW27) creates the window where your app’s content is drawn and provides a place to respond to state transitions within the app.
* It creates the [entry point](https://developer.apple.com/library/content/referencelibrary/GettingStarted/DevelopiOSAppsSwift/GlossaryDefinitions.html#//apple_ref/doc/uid/TP40015214-CH12-SW37) to your app and a [run loop](https://developer.apple.com/library/content/referencelibrary/GettingStarted/DevelopiOSAppsSwift/GlossaryDefinitions.html#//apple_ref/doc/uid/TP40015214-CH12-SW61) that delivers input events to your app. This work is done by the UIApplicationMain attribute (@UIApplicationMain), which appears toward the top of the file.  Using the UIApplicationMain attribute is equivalent to calling the UIApplicationMain function and passing your AppDelegate class’s name as the name of the delegate class. In response, the system creates an [application object](https://developer.apple.com/library/content/referencelibrary/GettingStarted/DevelopiOSAppsSwift/GlossaryDefinitions.html#//apple_ref/doc/uid/TP40015214-CH12-SW75). The application object is responsible for managing the life cycle of the app. The system also creates an instance of your AppDelegate class, and assigns it to the application object. Finally, the system launches your app.

The AppDelegate class is automatically created whenever you create a new project. Unless you are doing something highly unusual, you should use this class provided by Xcode to initialize your app and respond to app-level events. The AppDelegate class adopts the UIApplicationDelegate protocol. This protocol defines a number of methods you use to set up your app, to respond to the app’s state changes, and to handle other app-level events.

The AppDelegate class contains a single property: window.

* var window: a href="" UIWindow /a ?

This property stores a reference to the app’s window. This window represents the root of your app’s view hierarchy. It is where all of your app content is drawn. Note that the window property is an [optional](https://developer.apple.com/library/content/referencelibrary/GettingStarted/DevelopiOSAppsSwift/GlossaryDefinitions.html#//apple_ref/doc/uid/TP40015214-CH12-SW11), which means it may have no value (be [nil](https://developer.apple.com/library/content/referencelibrary/GettingStarted/DevelopiOSAppsSwift/GlossaryDefinitions.html#//apple_ref/doc/uid/TP40015214-CH12-SW5)) at some point.

The AppDelegate class also contains stub implementations of the following delegate [methods](https://developer.apple.com/library/content/referencelibrary/GettingStarted/DevelopiOSAppsSwift/GlossaryDefinitions.html#//apple_ref/doc/uid/TP40015214-CH12-SW12):

* func application(\_ application: a href="" UIApplication /a , didFinishLaunchingWithOptions launchOptions: [ a href="" UIApplicationLaunchOptionsKey /a : Any]?) -> a href="" Bool /a
* func applicationWillResignActive(\_ application: a href="" UIApplication /a )
* func applicationDidEnterBackground(\_ application: a href="" UIApplication /a )
* func applicationWillEnterForeground(\_ application: a href="" UIApplication /a )
* func applicationDidBecomeActive(\_ application: a href="" UIApplication /a )
* func applicationWillTerminate(\_ application: a href="" UIApplication /a )

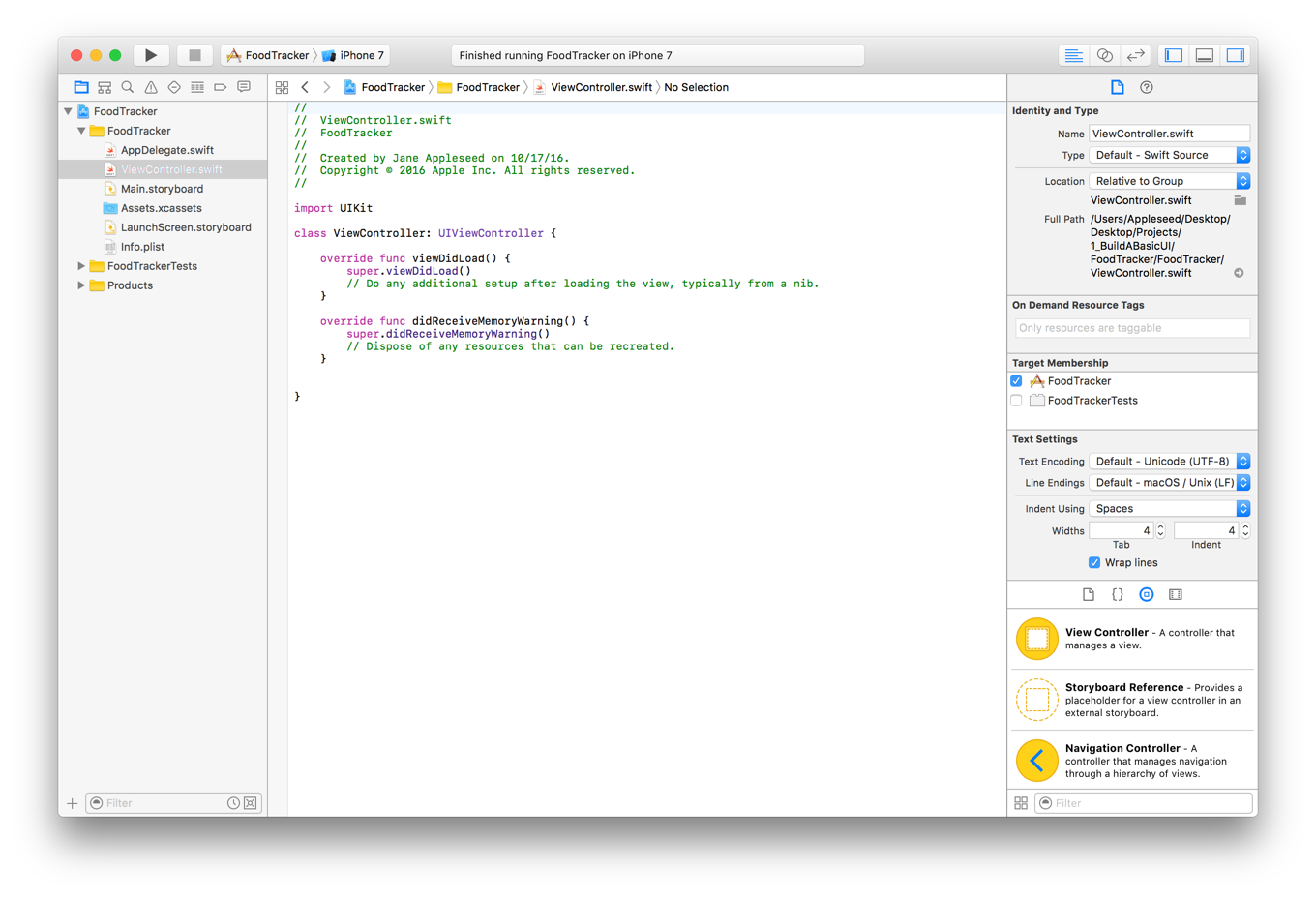
These methods let the application object communicate with the app delegate. During an app state transition—for example, app launch, transitioning to the background, and app termination—the application object calls the corresponding delegate method, giving your app an opportunity to respond. You don’t need to do anything special to make sure these methods get called at the correct time—the application object handles that job for you.

Each of the delegate methods has a default behavior. If you leave the template implementation empty or delete it from your AppDelegate class, you get the default behavior whenever that method is called. Alternatively, you can add your own code to the stub methods, defining custom behaviors that are executed when the methods are called.

The template also provides comments for each of the stub methods. These comments describe how these methods can be used by your app. You can use the stub methods and comments as a blueprint for designing many common app-level behaviors.

The View Controller Source File

The Single View Application template has another source code file: ViewController.swift. Select ViewController.swift in the project navigator to view it.



This file defines a custom [subclass](https://developer.apple.com/library/content/referencelibrary/GettingStarted/DevelopiOSAppsSwift/GlossaryDefinitions.html#//apple_ref/doc/uid/TP40015214-CH12-SW14) of UIViewController named ViewController. Right now, this class simply inherits all the behavior defined by UIViewController. To override or extend that behavior, you override the methods defined on UIViewController.

As you can see in the ViewController.swift file, the template’s implementation overrides both the viewDidLoad() and didReceiveMemoryWarning() methods; however, the template’s stub implementation doesn’t do anything yet, except call the UIViewController version of these methods. You can add your own code to customize the view controller’s response to these events.

Although the template comes with the didReceiveMemoryWarning() method, you won’t need to implement it in these lessons, so go ahead and delete it.

Adopt Auto Layout

[Auto Layout](https://developer.apple.com/library/content/referencelibrary/GettingStarted/DevelopiOSAppsSwift/GlossaryDefinitions.html#//apple_ref/doc/uid/TP40015214-CH12-SW77) is a powerful layout engine that helps you design adaptive layouts that dynamically respond to any changes to the scene’s size. You describe your layout using [constraints](https://developer.apple.com/library/content/referencelibrary/GettingStarted/DevelopiOSAppsSwift/GlossaryDefinitions.html#//apple_ref/doc/uid/TP40015214-CH12-SW78)—rules that explain where one element should be located relative to another, or what size the element should be. Auto Layout dynamically calculates the size and position of each element based on these constraints.

One of the easiest ways to define your layout is using a stack view (UIStackView). A stack view provides a streamlined interface for laying out a collection of views in either a column or a row. The stack view uses Auto Layout under the hood to calculate the size and position of all the views that it manages. This lets you easily access the full power of Auto Layout, while greatly reducing the complexity of your layout.

To adopt Auto Layout, wrap your existing interface elements in a stack view, and then add the constraints needed to position the stack view in the scene.

**Some useful instructions:**

* Choose Editor > Canvas, and make sure Show Bounds Rectangles is selected.  This setting causes Interface Builder to draw a blue bounding box around all the views in the canvas. Many views and controls have transparent backgrounds, making it difficult to see their actual size. Layout bugs occur when the system resizes a view so that it’s either larger or smaller than you anticipate. Enabling this setting helps you understand exactly what’s going on in your view hierarchy.

When a view controller is loaded from a storyboard, the system instantiates the view hierarchy and assigns the appropriate values to all the view controller’s outlets. By the time the view controller’s viewDidLoad() method is called, the system has assigned valid values to all of the controller’s outlets, and you can safely access their contents.

**-- Use stack views to create robust, flexible layouts**