iOS

Ben Gavan

November 14, 2020

Contents

1 Programmatic Startup iOS 13 onward

1.1 Motivation

From the introduction of iOS 13, there has been a splitting of *AppDelegate* into *AppDelegate* and *SceneDelegate*. Since *SceneDelegate* has components only introduced and available for iOS13, all of *SceneDelegate* is no longer backwards compatible. With 95% using iOS 11 onward (and 5% still using even earlier versions), it would be recommended to still support these users. To do so, we have to make alterations in three files; *AppDelegate*, *SceneDelegate*, and *Info.plist*.

1.2 Alterations to be made in *Info.plist*

Along with removing *Main.storyboard* from the project and setting the *Main interface* in the project settings page to nothing, we also need to remove the refernece to 'Main' in the *Info.plist* file. To open the file:

- Right-click on the *Info.plist* file
- select *open as* → *Source Code*

Then change the *UIApplicationSceneManifest* section to something like this with the MainStoryboard not defined

```
<key>UIApplicationSceneManifest</key>
<dict>
  <key>UIApplicationSupportsMultipleScenes</key>
  <false/>
  <key>UISceneConfigurations</key>
  <dict>
    <key>UIWindowSceneSessionRoleApplication</key>
    <array>
      <dict>
        <key>UILaunchStoryboardName</key>
        <string>LaunchScreen</string>
        <key>UISceneConfigurationName</key>
        <string>Default Configuration</string>
        <key>UISceneDelegateClassName</key>
        <string>$(PRODUCT_MODULE_NAME).SceneDelegate</string>
    </dict>
    </array>
  </dict>
</dict>
```

1.3 *AppDelegate*

No significant changes have to be made to *AppDelegate* since everything is compatible with iOS 11.
All that is needed is to add a new variable to hold the *UIWindow* and then to initialize it how you used to (i.e. setting the root view-controller and making the window make and visible)

```
var window: UIWindow?

func application(_ application: UIApplication, didFinishLaunchingWithOptions
    launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
  // Override point for customization after application launch.

  window = window ?? UIWindow()
  window?.rootViewController = ViewController()
  window?.makeKeyAndVisible()

  return true
}
```

1.4 *SceneDelegate*

The first requirement is to add limit the *SceneDelegate* to only be used for iOS 13+. To do this, add

```
@available(iOS 13.0, *)
```

on the line directly above the class declaration.
The usual window setup is as per usual:

```
import UIKit

@available(iOS 13.0, *)
class SceneDelegate: UIResponder, UIWindowSceneDelegate {
```

```
5
6     var window: UIWindow?
7
8
9     func scene(_ scene: UIScene, willConnectTo session: UISceneSession, options
          connectionOptions: UIScene.ConnectionOptions) {
10
11      guard let windowScene = (scene as? UIWindowScene) else { return }
12
13      window = window ?? UIWindow(windowScene: windowScene)
14      window?.rootViewController = ViewController()
15      window?.makeKeyAndVisible()
16    }
17    ...
18
19  }
```

2 AppDelegate

2.1 UIWindow()

To launch the app programmatically, we need in the AppDelegate in didFinishLaunchingWithOptions:

```
1  func application(_ application: UIApplication, didFinishLaunchingWithOptions
       launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
2    // Override point for customization after application launch.
3
4    window = window ?? UIWindow()
5    window?.rootViewController = UICollectionView()
6    window?.makeKeyAndVisible()
7
8    return true
9  }
```

3 // MARK: -

3.1 Benefits of using MARKS

- consistency across files
- consistency across projects
- Keep code withing those files organized and easy to find.

3.2 Example Snippets

3.2.1 UIViewController

```
1  // MARK: - Properties
2
3  // MARK: - IBOutlets
4
5  // MARK: - Life cycle
6
7  // MARK: - Set up
8
9  // MARK: - IBActions
10
11 // MARK: - Navigation
12
13 // MARK: - Network Manager calls
14
15 // MARK: - Extensions
```

[1]

3.2.2 Models

```
  // MARK: - Attributes

  // MARK: - Initializers

  // MARK: - Parsers
```

[1]

4 View Margin

A margin specifies where a sub-view of its can be constrained up to.

The following creates two square views with one inside the other. The outer view has a margin of 20 top, 10 on the other 3 sides. When the constraints for v2 are set, we need to use the *v1.layoutMarginsGuide.*— property to access the margins to be properly constrained. [**?**, pp.42]

```
 let v1 = UIView()
v1.translatesAutoresizingMaskIntoConstraints = false
v1.backgroundColor = .blue
v1.layoutMargins = UIEdgeInsets(top: 20, left: 10, bottom: 10, right: 10)

let v2 = UIView()
v2.translatesAutoresizingMaskIntoConstraints = false
v2.backgroundColor = .red

view.addSubview(v1)

v1.centerXAnchor.constraint(equalTo: view.centerXAnchor).isActive = true
v1.centerYAnchor.constraint(equalTo: view.centerYAnchor).isActive = true
v1.heightAnchor.constraint(equalToConstant: 200).isActive = true
v1.widthAnchor.constraint(equalToConstant: 200).isActive = true

v1.addSubview(v2)

v2.topAnchor.constraint(equalTo: v1.layoutMarginsGuide.topAnchor).isActive = true
v2.leadingAnchor.constraint(equalTo: v1.layoutMarginsGuide.leadingAnchor).isActive = true
v2.heightAnchor.constraint(equalToConstant: 100).isActive = true
v2.widthAnchor.constraint(equalToConstant: 100).isActive = true
```

5 Realm

```
NSPredicate(format: "name BEGINSWITH [c]%@", searchString)
```

where [c] that follows BEGINSWITH indicates a case insensitive search.

6 Testing

To create a new test

1. Open the test panel (diamond with line through it / 6th from left)
2.

7 iOS Versions

5% ← iOS 11 → 95%
with iOS 11 being released in 19/9/2017

References

[1] Helpful iOS and Xcode Code Snippets *Matias Jurfest*. Available from: ‹https://medium.com/better-programming/helpful-code-snippets-for-ios-21aa5ef894de›
[Accessed on 15th October 2019]

[2] Programming iOS 10: Dive deep into view, view controllers, and frameworks. Matt Neuburg

8 Keyboard Shortcuts

- *shift + option + command + left*

  Minimize all code blocks

- *shift + option + command + right*

  Maximize all code blocks

- *option + command + /*

  Generates documentation

9 Notifications

The server (me) sends a request containing the device id to the APNs (Apple Push-Notification Services) which then sends the notification to the user device. The device id is sent the the server from the user device. The app the notification is being sent to has to be preregistered with APNs.

10 Sockets

11 Run Development app on REAL Device

In development machine running xCode

- Set up Signing profile

On the Real device:

- Settings → General → Profiles & Device Management →
- Select Developer app
- Press Trust xxx