

USE YOUR LOAF

START HERE

ARTICLES

MODERN AUTO LAYOUT

NEWSLETTER

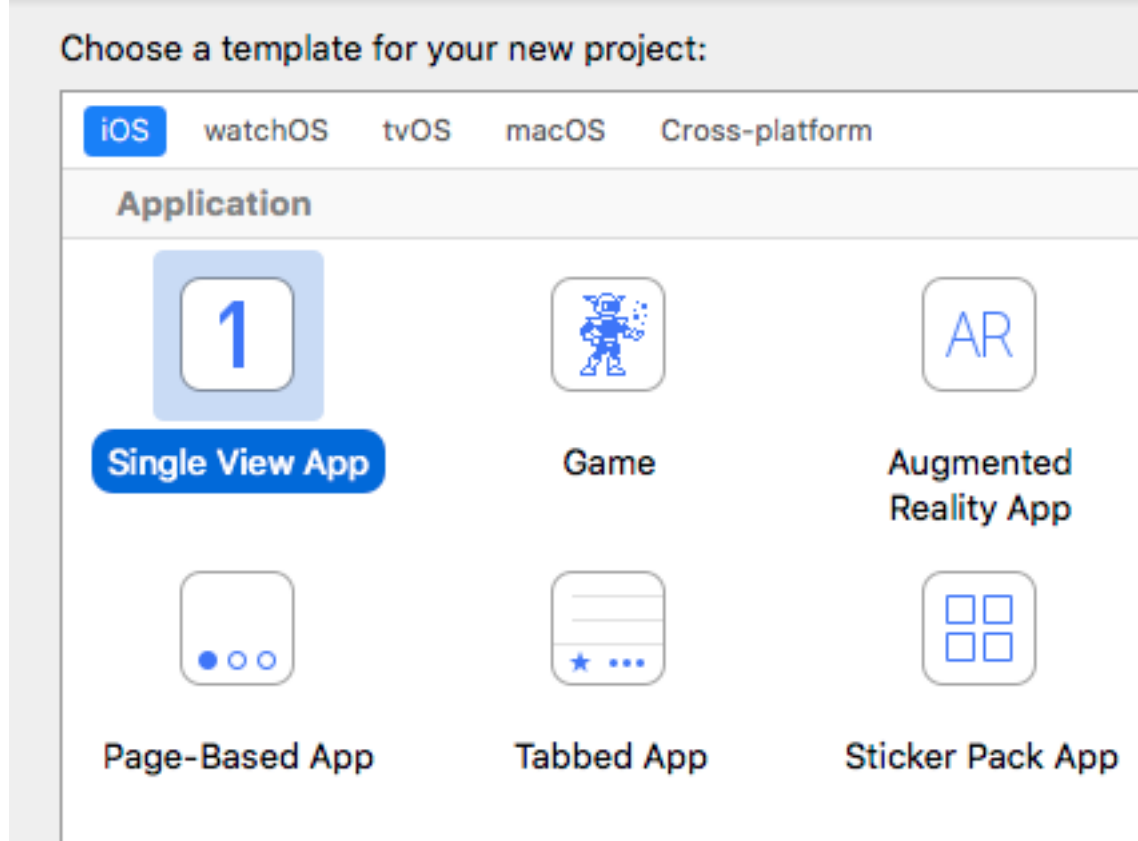
Creating Custom Xcode Project Templates

Jan 29, 2018 · 7 minute read

Xcode

I recently wanted to play with some projects that did not use a main Storyboard. Starting from the single view app template it soon gets tiring to clean things up and add the boilerplate to the app delegate to create the window. A better approach is to create your own custom Xcode project template. Here are my notes for future reference.

Where Are The Xcode Templates



The built-in file and project templates are in the Xcode application bundle. Right-click on the Xcode app in the `/Applications` folder and show package contents. The iOS templates are in sub-folders here:

```
Contents/Developer/Platforms/iPhoneOS.platform/  
Developer/Library/Xcode/Templates/Project Templates
```

They make use of a number of shared base templates stored here:

```
Contents/Developer/Library/Xcode/Templates/Project
```

You add your own templates by placing them in the folder:

```
~/Library/Developer/Xcode/Templates/
```

I followed Apple's direction and created sub-folders for the file and project templates:

```
$ mkdir -p ~/Library/Developer/Xcode/Templates/File  
$ mkdir -p ~/Library/Developer/Xcode/Templates/Project
```

How Do They Work

I cannot find any official Apple guides on how the templates work (some useful links at the end of this post). The easiest way seems to be to start from one of the built-in templates. I am going to use the single view application iOS template which is in a folder named `Single View App.xctemplate`. It has just four files:



Main.storyboard



TemplateIcon.png



TemplateIcon@2x.png

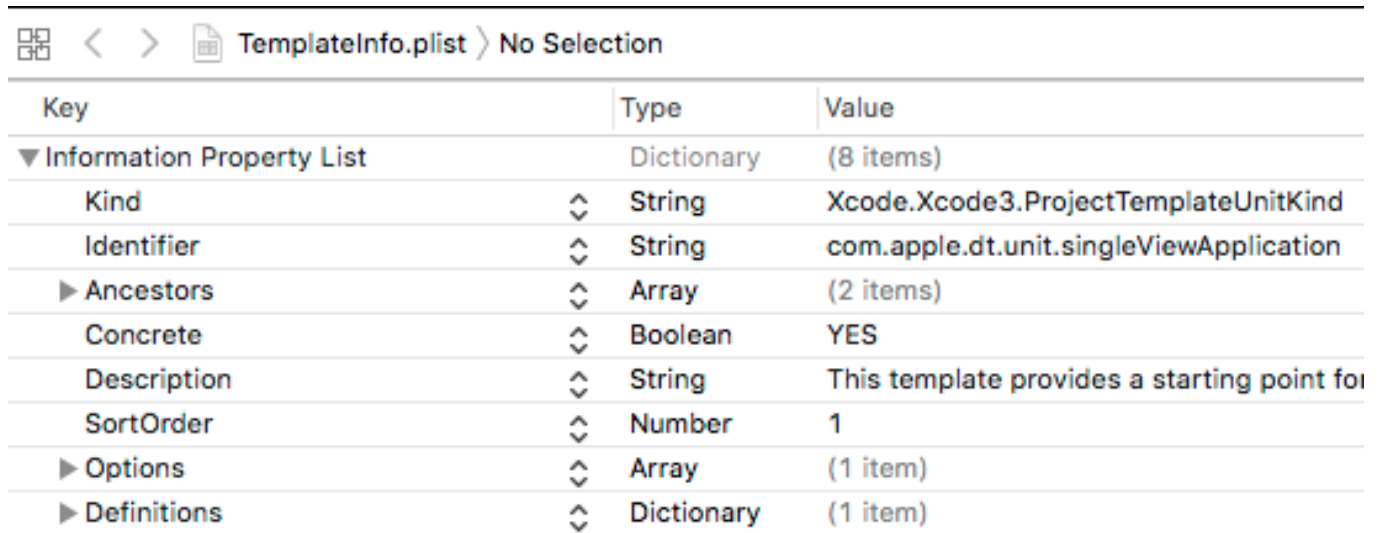


TemplateInfo.plist

- `Main.storyboard` - the storyboard with the initial view controller.
- `TemplateIcon.png` - 48x48 icon

- `TemplateIcon@2x.png` - 96x96 icon
- `TemplateInfo.plist` - the configuration file for the template.

The interesting file is the `TemplateInfo.plist` file. Here is how it looks at the top level:



Key	Type	Value
▼ Information Property List	Dictionary	(8 items)
Kind	String	Xcode.Xcode3.ProjectTemplateUnitKind
Identifier	String	com.apple.dt.unit.singleViewApplication
▶ Ancestors	Array	(2 items)
Concrete	Boolean	YES
Description	String	This template provides a starting point for
SortOrder	Number	1
▶ Options	Array	(1 item)
▶ Definitions	Dictionary	(1 item)

Kind

Set to `Xcode.Xcode3.ProjectTemplateUnitKind` for project templates.

Identifier

A unique identifier for this template. Used when other templates inherit from this template.

Ancestors

The template system allows you to inherit from and override other templates. The `Ancestors` field is an array of identifiers for templates that this template inherits from.

The single view application template has two entries:

- `com.apple.dt.unit.storyboardApplication` - adds the `UIMainStoryboardFile` key to the `Info.plist` file.
- `com.apple.dt.unit.coreDataCocoaTouchApplication` - adds the Core Data boilerplate to the app delegate if you select the checkbox

The storyboard and core data templates these identifiers reference both inherit from a Cocoa Touch Application Base template:

- `com.apple.dt.unit.cocoaTouchApplicationBase` - adds unit test options, the application delegate boilerplate code and some `Info.plist` settings.

This template inherits from a number of others including an iOS App Base template to build a complex setup that takes some unpicking.

Concrete

Must be `YES` for Xcode to show the template. If missing Xcode treats the template as an abstract base template that other templates will inherit from.

Description

A text description that Xcode used to display when

browsing the templates. Xcode 9 does not seem to show it.

SortOrder

Set to override the default alphabetical sort order in the template browser.

Nodes

An array of items to create in the new project. These can be code snippets, project and build settings or whole files. The value of each item is often an identifier to a definition stored in the `Definitions` section of this template or an inherited template.

Definitions

A dictionary of definitions for the items in `Nodes` . You can override a definition from a previous template. You can include an external file using a path dictionary. This is how the template adds the main storyboard file:

▼ Definitions	↕	Dictionary	(1 item)
▼ Base.lproj/Main.storyboard		Dictionary	(2 items)
Path		String	Main.storyboard
SortOrder		Number	99

Options

Options allow for user input to customize the template. The language choice between Swift and Objective-C is a menu with two choices listed in a `Units` dictionary:

▼ Options	↕	Array	(1 item)
▼ Item 0		Dictionary	(2 items)
Identifier		String	languageChoice
▼ Units		Dictionary	(2 items)
▼ Objective-C		Dictionary	(1 item)
► Nodes		Array	(10 items)
▼ Swift	⊕ ⊖	Dictionary	↕ (1 item)
▼ Nodes		Array	(7 items)
Item 0		String	ViewController.swift:comments
Item 1		String	ViewController.swift:imports:importCocoa
Item 2		String	ViewController.swift:implementation(___FILEBASENAME__: UIViewController)
Item 3		String	ViewController.swift:implementation:methods:viewDidLoad(override func viewDidLoad())
Item 4		String	ViewController.swift:implementation:methods:viewDidLoad:super
Item 5		String	ViewController.swift:implementation:methods:didReceiveMemoryWarning(override func didReceiveMemoryWarning())
Item...	⊕ ⊖	String	ViewController.swift:implementation:methods:didReceiveMemoryWarning:super

The language choice determines which set of **Nodes** Xcode adds to the project. In this case it builds the Swift or Objective-C versions of the **ViewController** class. The **Unit** dictionary can also contain **Definitions**. You can use any of the Xcode placeholder strings (**___FILEBASENAME___**) in the values. Xcode replaces them when creating the project.

Creating A New Project Template

My aim is to create a new version of the single view application template that manually creates the window and root view hierarchies without using a main Storyboard.

To get started create a folder under "Project Templates" to store the template. The folder name shows up as the section name in the Xcode template browser.

```
$ mkdir -p ~/Library/Developer/Xcode/Templates/Proj
```

Then copy the single view app template to have a starting

point:

```
$ APP_TEMPLATES='/Applications/Xcode.app/Contents/Developer/Library/Xcode/Templates/Project Templates/iOS/Application'
```

```
$ cd ~/Library/Developer/Xcode/Templates/Project\ Templates/Single\ View\ App.xctemplate
```

```
$ cp -r "$APP_TEMPLATES/Single View App.xctemplate" "Manual Single View App.xctemplate"
```

To make changes to the template we need to edit the `TemplateInfo.plist` file. Double-click to open it in Xcode or your favourite plist editor.

Change The Identifier

The first thing to do is change the identifier to something unique. I used

```
com.useyourloaf.dt.unit.manual.singleViewApplication
```

I also updated the description.

Remove The Storyboard

I do not want this template to inherit from the storyboard template so I replaced it in the ancestors array with the Cocoa Touch Application Base template that it inherits from. I left the Core Data template:

```
<key>Ancestors</key>
```



```
<array>
  <string>com.apple.dt.unit.cocoaTouchApplicationBa
  <string>com.apple.dt.unit.coreDataCocoaTouchAppli
</array>
```

I also removed the storyboard definition entry and the `Main.storyboard` file.

The Root View Controller

I added an item to the `Options` to allow the user to choose the name of the root view controller when creating the project:

▼ Options	⇅	Array	(3 items)
▼ Item 0		Dictionary	(6 items)
Identifier		String	rootClassName
Required		Boolean	YES
Name		String	Root View Controller Class:
Description		String	Class name of root view controller
Type		String	text
Default		String	RootViewController

It defaults to `RootViewController`. We can use the value elsewhere in the template with the variable placeholder `___VARIABLE_rootClassName___`.

To create the root view controller class I modified the nodes for both language choices. This was mostly just replacing the hardcoded `ViewController` with our new `rootClassName` variable:

▼ Item 1	Dictionary	(2 items)
Identifier	String	languageChoice
▼ Units	Dictionary	(2 items)
▼ Objective-C	Dictionary	(1 item)
▼ Nodes	Array	(9 items)
Item 0	String	AppDelegate.m:imports:importHeader:___VARIABLE_rootClassName___.h
Item 1	String	___VARIABLE_rootClassName___.h:comments
Item 2	String	___VARIABLE_rootClassName___.h:imports:importCocoa
Item 3	String	___VARIABLE_rootClassName___.h:interface(___FILEBASENAME___ : UIViewController)
Item 4	String	___VARIABLE_rootClassName___.m:comments
Item 5	String	___VARIABLE_rootClassName___.m:imports:importHeader:___VARIABLE_rootClassName___.h
Item 6	String	___VARIABLE_rootClassName___.m:extension
Item 7	String	___VARIABLE_rootClassName___.m:implementation:methods:viewDidLoad(- (void))viewDidLoad()
Item 8	String	___VARIABLE_rootClassName___.m:implementation:methods:viewDidLoad:super
▼ Swift	Dictionary	(1 item)
▼ Nodes	Array	(5 items)
Item 0	String	___VARIABLE_rootClassName___.swift:comments
Item 1	String	___VARIABLE_rootClassName___.swift:imports:importCocoa
Item 2	String	___VARIABLE_rootClassName___.swift:implementation(___FILEBASENAME___ : UIViewController)
Item 3	String	___VARIABLE_rootClassName___.swift:implementation:methods:viewDidLoad(override func viewDidLoad())
Item 4	String	___VARIABLE_rootClassName___.swift:implementation:methods:viewDidLoad:super

Note that for the Objective-C version of the project I also include the view controller header file in the app delegate as we will need it shortly:

```
AppDelegate.m:imports:importHeader:___VARIABLE_rootClassName___.h
```

Change the App Delegate

The app delegate needs code to create the main window and root view controller. For some extra polish I decided to also add a user option to embed the view controller in a navigation controller. Note the `Type` is `checkbox` and defaults to `false` (a string not a boolean):

▼ Item 2	Dictionary	(6 items)
Identifier	String	navigationController
Name	String	Embed in Navigation Controller
Description	String	Is the root view controller embedded in a navigation controller
Type	String	checkbox
Default	String	false
▼ Units	Dictionary	(2 items)
▶ false	Array	(2 items)
▶ true	Array	(2 items)

The `Units` dictionary for a checkbox has entries for `true` and `false`. If you only wanted to add items for the `true` case you can omit `false`. In my case I am going to generate a different app delegate in both cases. I also need to make the result dependent on the language choice. The `false` case looks like this with an item for Objective-C and an item for Swift:

▼ Units	⊕ ⊖ Dictionary	(2 items)
▼ false	Array	(2 items)
▼ Item 0	Dictionary	(2 items)
▼ RequiredOptions	Dictionary	(1 item)
languageChoice	String	Objective-C
▼ Definitions	Dictionary	(1 item)
AppDelegate.m:implementation:met...	String	self.window = [[UIWindow alloc] initWithFrame:[UIScreen mainScreen bounds]];
▼ Item 1	Dictionary	(2 items)
▼ RequiredOptions	Dictionary	(1 item)
languageChoice	String	Swift
▼ Definitions	Dictionary	(1 item)
AppDelegate.swift:implementation:...	String	window = UIWindow(frame: UIScreen.main.bounds)

Note how each item has the `RequiredOptions` key set to the relevant language choice. To add the code snippet to the app delegate we override the definition from the Cocoa Touch App Base template for the body for the `didFinishLaunchingWithOptions` method:

```
AppDelegate.m:implementation:methods:applicationdid
```

The Xcode plist editor is clumsy when handling multi-line values. I found it easier to cut and paste the code into the field. The swift version is as follows:

```
window = UIWindow(frame: UIScreen.main.bounds)
```

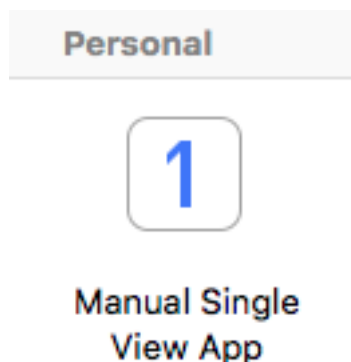
```
window?.backgroundColor = .white
window?.rootViewController = ____VARIABLE_rootClassName_
window?.makeKeyAndVisible()
```

Note that we use the variable placeholder for the name of the root view controller class. The `true` version embeds the view controller in a navigation controller:

```
window = UIWindow(frame: UIScreen.main.bounds)
window?.backgroundColor = .white
let rootViewController = ____VARIABLE_rootClassName_
let navigationController = UINavigationController(rootViewController: rootViewController)
window?.rootViewController = navigationController
window?.makeKeyAndVisible()
```

Using The Template

There is nothing more to do to activate the template. When creating a new Xcode project the template should appear in the browser:



The project settings includes our user options to change

the class name for the root view controller and the checkbox to embed it in a navigation controller:

Product Name: test

Team: None

Organization Name: Useyourloaf

Organization Identifier: com.useyourloaf

Bundle Identifier: com.useyourloaf.test

Language: Swift

Root View Controller Class: RootViewController

☒ Embed in Navigation Controller

☐ Use Core Data

☐ Include Unit Tests

☐ Include UI Tests

These options are persistent so they remember the last value used (set the `NotPersisted` boolean key to `YES` for the item to change that).

Getting The Template

The template is available in an Xcode Templates repository on GitHub here:

- [Xcode Templates](#)

To install it just copy the `Personal` folder into the `~/Library/Developer/Xcode/Templates/Project Templates` folder.

File A Bug Report For Documentation

Official Apple Xcode documentation on the file and project template system is missing. If you want to see that change please file a bug with the [Feedback Assistant](#). See mine at rdar://36908117 for an example.

Further Reading

The guide for Xcode 4 by Steffen Itterheim was great but the original link is now dead. You might still find it here via the wayback machine:

- [Xcode 4 Template Docs](#).

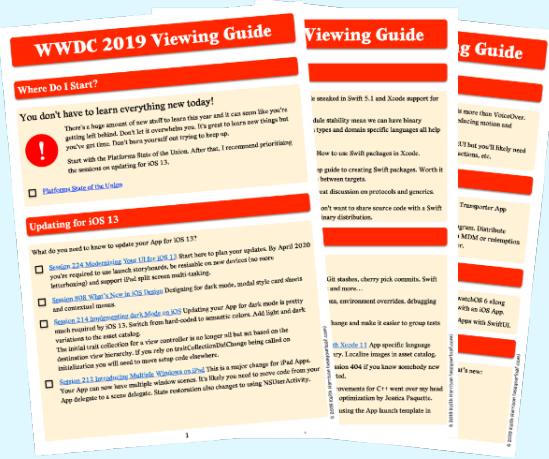
This post written by Sophia Lazaroova also helped get me started:

- [How to Create Custom Project Templates in Xcode](#)

Never Miss A Post

Sign up to get my **iOS posts and news** direct to your inbox and I'll also send you a PDF of my **WWDC Viewing Guide**

Subscribe



Unsubscribe at any time. See [privacy policy](#).

Copyright © 2020 - Keith Harrison

[Privacy](#) | [Cookies](#) | [Terms of Use](#) | [About](#)

