



FREE eBook

LEARNING

Xcode

Free unaffiliated eBook created from
Stack Overflow contributors.

#xcode

Table of Contents

About.....	1
Chapter 1: Getting started with Xcode.....	2
Remarks.....	2
Versions.....	2
Examples.....	3
Get Started.....	3
Use multiple versions of Xcode.....	4
Changing The Color Scheme.....	6
Pro Tip.....	8
Chapter 2: Certificates, Provisioning Profiles, & Code Signing.....	10
Examples.....	10
Choose the right code signing approach.....	10
Using Xcode's code signing feature.....	10
Xcode 7 and lower.....	10
Xcode 8 and up.....	10
Manually.....	11
Using fastlane match.....	11
Chapter 3: Command Line Tools.....	12
Examples.....	12
Running Tests.....	12
List available targets, schemes and build configurations.....	12
Compile and sign schema.....	13
Access any command line tool in Xcode app bundle (xcrun).....	14
Switching command line tools with xcode-select.....	14
Chapter 4: Creating Custom Controls in Interface Builder with @IBDesignable.....	16
Remarks.....	16
Examples.....	16
A Live-Rendered Rounded View.....	16
Chapter 5: Cross-Platform Development.....	19
Examples.....	19

TargetConditionals.....	19
Chapter 6: Customizing Xcode IDE.....	21
Introduction.....	21
Examples.....	21
Open Terminal in current Xcode project folder.....	21
Clear derived data with hotkey.....	24
Chapter 7: Debugging.....	26
Examples.....	26
Breakpoints.....	26
Wireless Debugging in Xcode-9.....	29
Chapter 8: Playgrounds.....	31
Examples.....	31
Getting Started with Playground.....	31
Latest Value, Value History and Graph.....	32
Adding Images, Static Data, Sounds, etc. to a Playground.....	33
Chapter 9: Projects & Workspaces.....	34
Examples.....	34
Projects overview.....	34
Create a project.....	34
Working with projects.....	34
Build, Run, Profile, Test, and Analyze your project.....	36
Adjust workspace to your needs and freely navigate it.....	37
Chapter 10: Xcode 8 features.....	39
Remarks.....	39
Examples.....	39
Color and image literals.....	39
Chapter 11: Xcode Tips.....	40
Examples.....	40
Reuse code snippets in Xcode.....	40
Install Plugins on Xcode 7.....	41
Installation.....	41

Recommendations	41
Usage	42
Hide strange unwanted and extra Xcode 8 logs.....	43
Credits	45

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [xcode](#)

It is an unofficial and free Xcode ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Xcode.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with Xcode

Remarks



Xcode is an [integrated development environment](#) for macOS which supports the development of native apps for macOS, iOS, watchOS, and tvOS. Xcode is the successor to NeXT's [Project Builder](#) and PBX. (In fact, Xcode's project manifest files are still named with the `.pbxproj` extension.)

Xcode releases include stable versions of the [clang](#) C/C++/Obj-C compiler, the [Swift](#) compiler, the [LLDB](#) debugger, and iOS/watchOS/tvOS simulators. Xcode also includes [Interface Builder](#), as well as tools for viewing and editing 3D models and scenes, image assets, and more.

Versions

Version	Release Date
1.0	2003-09-28
2.0	2005-04-04
3.0	2007-10-26
4.0	2011-03-14
5.0	2013-09-18
6.0	2014-09-17
7.0	2015-09-16
7.1.1	2015-11-09
7.2	2015-12-08
7.2.1	2016-02-03
7.3	2016-03-21
7.3.1	2016-05-03
8.0	2016-09-13

Version	Release Date
8.1	2016-10-27
8.2	2016-12-12
8.2.1	2016-12-19
8.3	2017-03-27
8.3.1	2017-04-06
8.3.2	2017-04-18
8.3.3	2017-06-05

Examples

Get Started

- [Download Xcode](#) from the Mac App Store.
- Click to create a new project or playground:



Welcome to Xcode

Version 7.3 (7D175)



Get started with a playground

Explore new ideas quickly and easily.



Create a new Xcode project

Start building a new iPhone, iPad or Mac OS project.



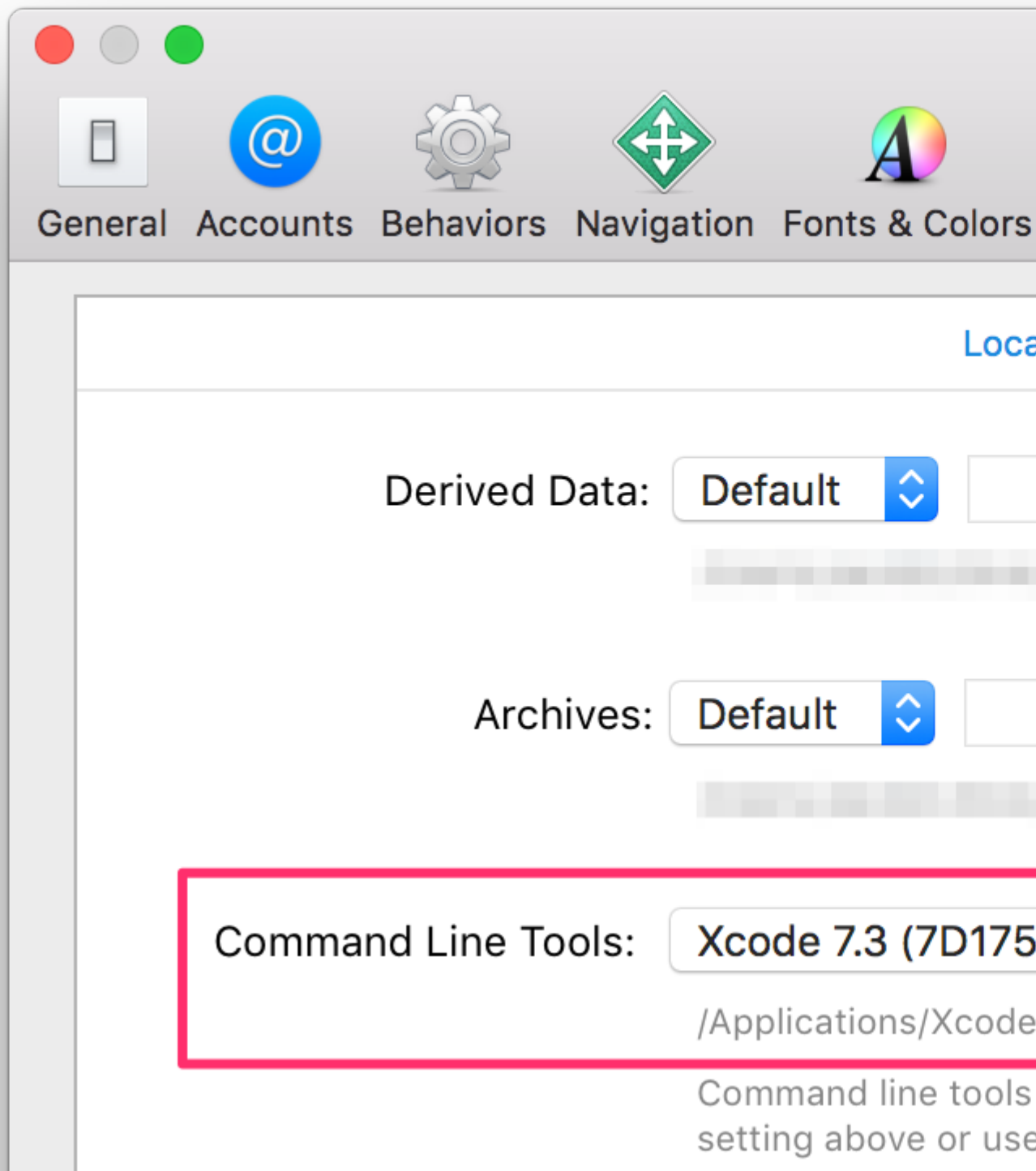
Check out an existing project

Start working on something from an SCM repository.

an existing version on your machine. You can also install Xcode from a [direct download](#) to get more control over which versions you have.

Each copy of Xcode includes command line tools (`clang`, `xcodebuild`, etc.). You can choose which ones are invoked by the commands in `/usr/bin`.

In Xcode's preferences, under the Locations tab, choose a version of Xcode:



Or you can manage versions from the command line using `xcode-select`:

```

# Print the currently selected version
$ xcode-select --print-path
/Applications/Xcode.app/Contents/Developer

$ clang --version
Apple LLVM version 7.3.0 (clang-703.0.29)
Target: x86_64-apple-darwin15.4.0
Thread model: posix
InstalledDir:
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin

# Find all installed versions using Spotlight
$ mdfind 'kMDItemCFBundleIdentifier = "com.apple.dt.Xcode"'
/Applications/Xcode.app
/Applications/Xcode72.app

# Check their version numbers
$ mdfind 'kMDItemCFBundleIdentifier = "com.apple.dt.Xcode"' | xargs mdls -name kMDItemVersion
kMDItemVersion = "7.3"
kMDItemVersion = "7.2.1"

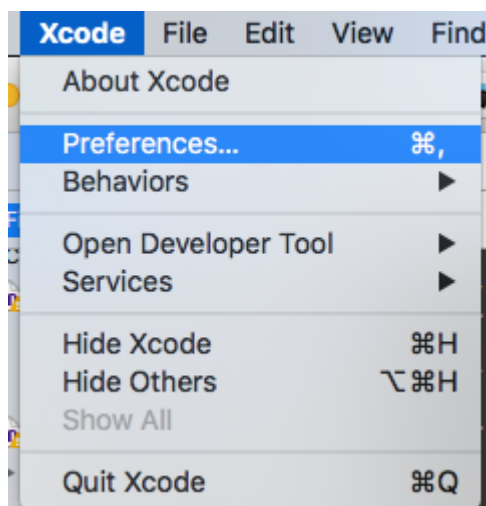
# Switch to a different version
$ sudo xcode-select --switch /Applications/Xcode72.app

$ clang --version
Apple LLVM version 7.0.2 (clang-700.1.81)
Target: x86_64-apple-darwin15.4.0
Thread model: posix

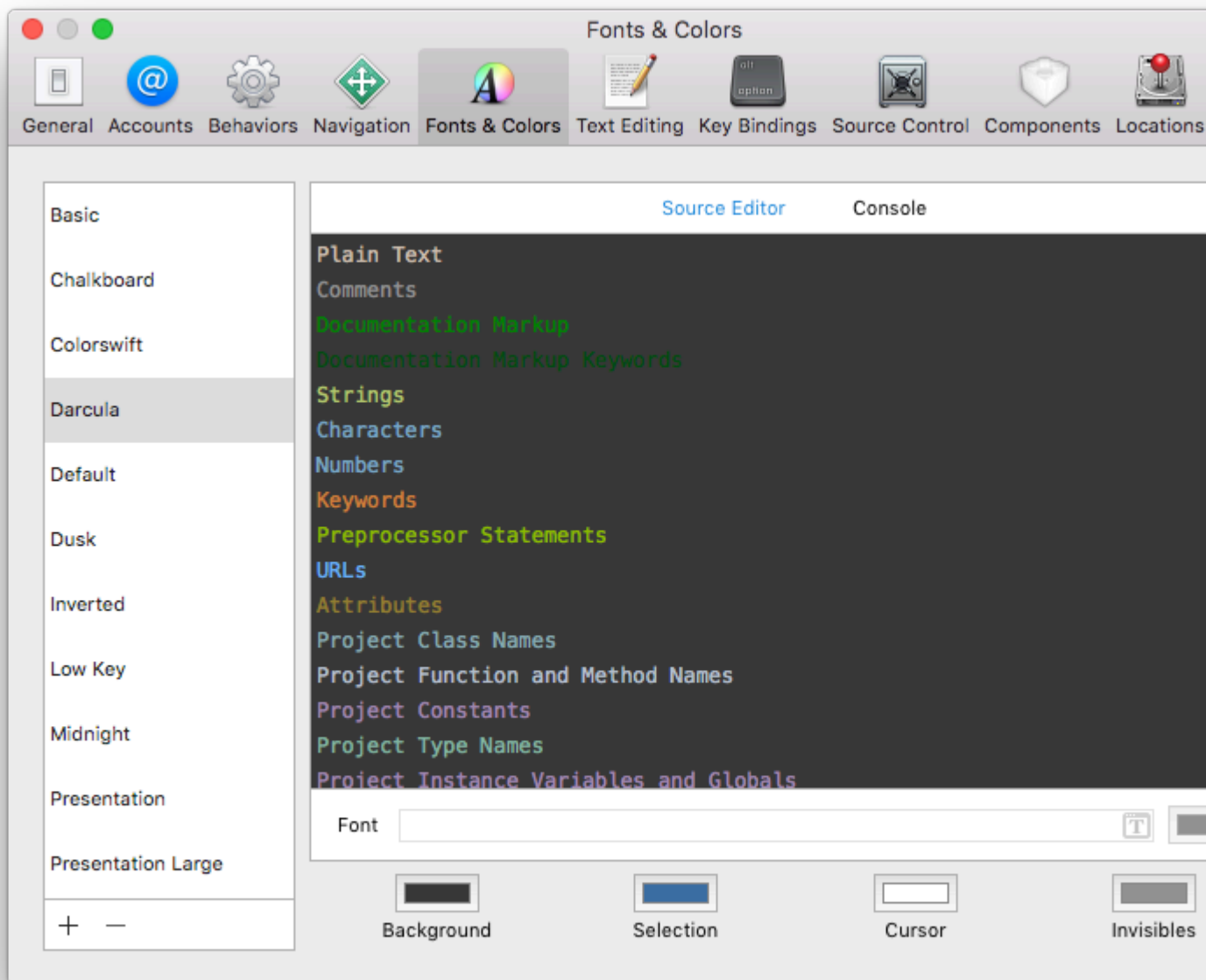
```

Changing The Color Scheme

Many developers like to customize the font, text, and background color of their IDE's. You can do this in Xcode by opening the app preference pane, either by going to XCODE->Preferences, or by pressing ','

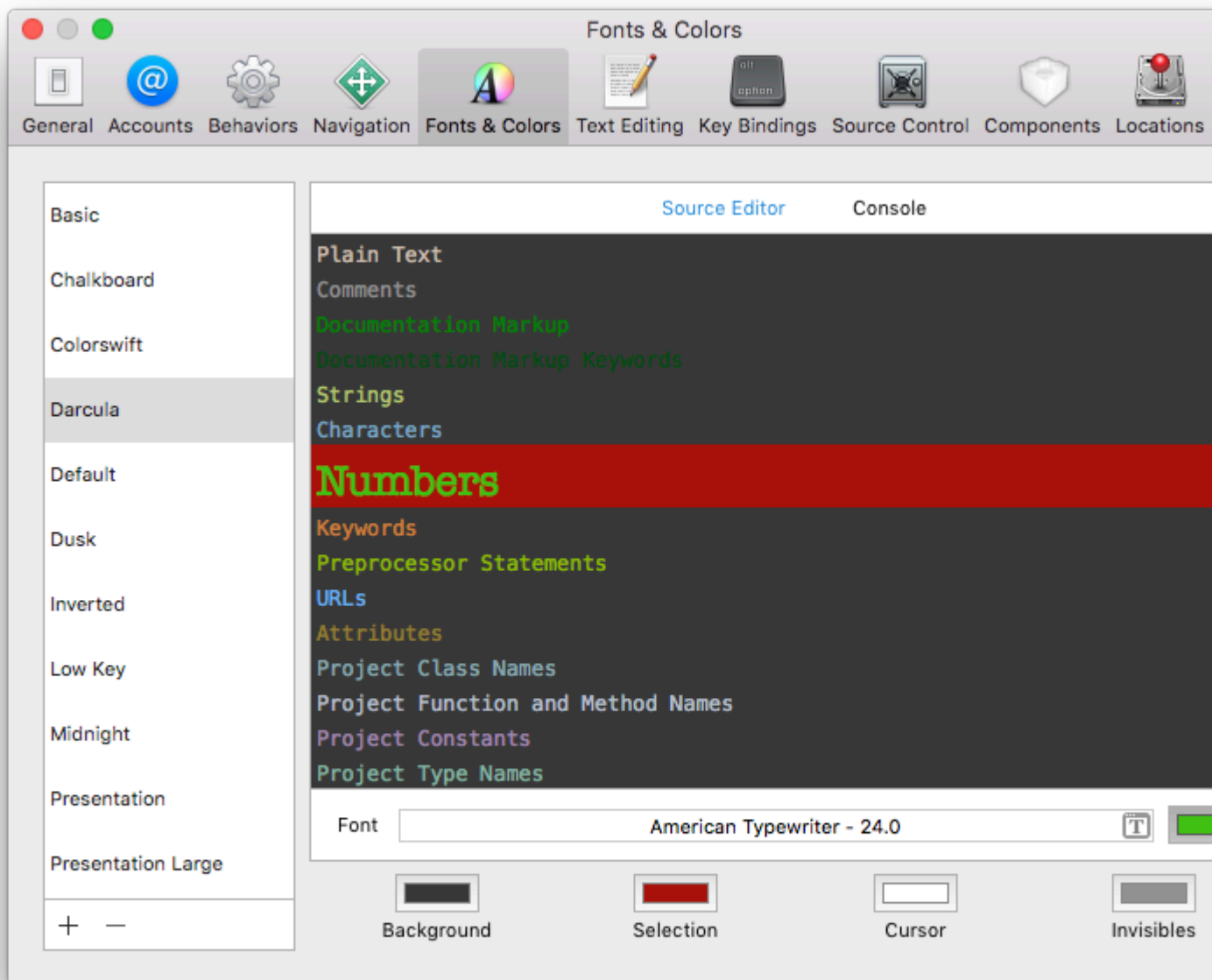


With the preference pane open you can click on the 'Fonts and Colors' tab.

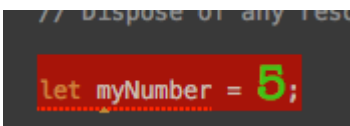


From here you can change the source AND console background and font colors. There are many pre-made color and font schemes provided with Xcode. You choose these from the list on the left (Basic, Chalkboard, etc). You can find and download more online (like [here](#) for example).

To further customize any theme, you can customize any of the types listed in the right pane (Plain Text, Comments, Documentation Markup, etc). For example, say I really want my 'Numbers' to show up in my code. So I change the font to 'American Typewriter' at 24 px, the color to a greenish color, and set the line highlighting to red:



Now in my text editing, I can really see my numbers:



Now you can customize the look and feel of the 'Source Editor' and 'Console' to your hearts delight!

Pro Tip

Many developers like to theme their IDEs dark (light text, dark background). In Xcode, you can only do this for the 'Source Editor' and the 'Console'. However, the Navigation (left side), Debug (bottom), and Utility (far right) sections are non-customizable. There are two work arounds to this.

First (kind of tricky, is to leave the IDE light themed (Light background, dark text) then invert the screen colors all together. This will make everything dark, but colors in the simulator and in the rest of the system are now wonky. The second work around is to hide The Navigation, Debug, and Utility areas when not in use. You can toggle these areas quickly using the following commands:

Navigator : 0

Debug Area : Y

Utility : 0

Read [Getting started with Xcode online](https://riptutorial.com/xcode/topic/294/getting-started-with-xcode): <https://riptutorial.com/xcode/topic/294/getting-started-with-xcode>

Chapter 2: Certificates, Provisioning Profiles, & Code Signing

Examples

Choose the right code signing approach

If you are just starting a new project, it's important to think about how you want to handle code signing.

If you are new to code signing, check out the [WWDC session](#) that describes the fundamentals of code signing in Xcode.

To properly code-sign your app, you have to have the following resources on your local machine:

- The private key (.p12 file)
- The certificate (.cer file), matching the private key
- The provisioning profile (.mobileprovision file), matching the certificate and private key installed locally

On the Apple Developer Portal it's also required to have a valid App ID associated with your provisioning profile.

Using Xcode's code signing feature

Occasionally the `Automatic` setting as the provisioning profile doesn't work reliably as it will just select the most recently updated provisioning profile, no matter if the certificate is installed.

That's why it is recommended to specify a specific provisioning profile somehow:

Xcode 7 and lower

You should avoid clicking the `Fix Issue` button (There is an [Xcode plugin](#) that disables the button), as it sometimes revokes existing certificates, and with it the provisioning profiles.

Unfortunately you can't specify the name of the provisioning profile in Xcode 7. Instead you can specify the UUID of the profile, which changes every time the profile gets re-generated (e.g. when you add a new device).

To work around this issue, check out [XcodeProject.md](#) on how to pass a provisioning profile to Xcode when building your app.

Xcode 8 and up

Apple improved code signing a lot with the release of Xcode 8, the following has changed:

- No more `Fix Issue` button, instead all code signing processes run in the background and show the log right in Xcode
- You can now specify the provisioning profile by name, instead of the UUID (Check out [XcodeProject.md](#) for more information)
- Improved error messages when something goes wrong. If you run into code signing errors you should always try building and signing with Xcode to get more detailed error information. (Check out [Troubleshooting.md](#) for more information)

Manually

You can always manually create and manage your certificates and provisioning profiles using the Apple Developer Portal. Make sure to store the private key (`.p12`) of your certificates in a safe place, as they can't be restored if you lose them.

You can always download the certificate (`.cer`) and provisioning profile (`.mobileprovision`) from the Apple Developer Portal.

If you revoke your certificate or it expires, all associated provisioning profiles will be invalid.

Using fastlane match

The concept of `match` is described in the [codesigning guide](#) and is the recommended code signing approach if you use `fastlane`

With `match` you store your private keys and certificates in a git repo to sync them across machines. This makes it easy to onboard new team-members and set up new Mac machines. This approach [is secure](#) and uses technology you already use.

Getting started with `match` requires you to revoke your existing certificates.

Read [Certificates, Provisioning Profiles, & Code Signing](#) online:

<https://riptutorial.com/xcode/topic/3711/certificates--provisioning-profiles---code-signing>

Chapter 3: Command Line Tools

Examples

Running Tests

To run your unit tests in the simulator using `xcodebuild` use

If you have a workspace (e.g. when using [CocoaPods](#))

```
xcodebuild \  
-workspace MyApp.xcworkspace \  
-scheme "MyScheme" \  
-sdk iphonesimulator \  
-destination 'platform=iOS Simulator,name=iPhone 6,OS=9.1' \  
test
```

If you have a project file

```
xcodebuild \  
-project MyApp.xcproj \  
-scheme "MyScheme" \  
-sdk iphonesimulator \  
-destination 'platform=iOS Simulator,name=iPhone 6,OS=9.1' \  
test
```

Alternative `destination` values are

```
-destination 'platform=iOS,id=REAL_DEVICE_UDID' \  
-destination 'platform=iOS,name=IPHONE NAME'
```

List available targets, schemes and build configurations

To list all available schemes for the project in your current directory

```
xcodebuild -list
```

Optionally you can pass a path to a project or workspace file

```
xcodebuild -list -workspace ./MyApp.xcworkspace \  
xcodebuild -list -project ./MyApp.xcodeproj
```

Example output

```
Information about project "Themoji":  
  Targets:  
    Themoji  
    ThemojiUITests
```


Unit

Build Configurations:

Debug

Release

If no build configuration is specified and `-scheme` is not passed then "Release" is used.

Schemes:

Themoji

ThemojiUITests

Units

Compile and sign schema

Cleaning and compiling code for iPhone, on project MyProject for schema Qa:

```
xcrun xcodebuild clean \  
-workspace "MyProject.xcworkspace" \  
-scheme "YourScheme" \  
-sdk iphoneos \  
-configuration Debug \  
archive \  
-archivePath builds/MyProject.xcarchive
```

Configuration can be either `Debug` or `Release`.

Signing the previously compiled code:

```
xcrun xcodebuild -exportArchive \  
-archivePath builds/MyProject-Qa.xcarchive \  
-exportOptionsPlist config.plist \  
-exportPath builds
```

`config.plist` contains the information about how to package and sign the application, for development builds use:

```
<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">  
<plist version="1.0">  
<dict>  
  <key>method</key>  
  <string>development</string>  
  <key>uploadSymbols</key>  
  <true/>  
</dict>  
</plist>
```

An App Store release plist should contain something like:

```
<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
```

```
<plist version="1.0">
<dict>
  <key>teamID</key>
  <string>xxxxxxxxxxxx</string>
  <key>method</key>
  <string>app-store</string>
  <key>uploadSymbols</key>
  <true/>
</dict>
</plist>
```

Where the Team ID can be obtained from your keychain.

All available parameters

- compileBitcode
- embedOnDemandResourcesAssetPacksInBundle
- iCloudContainerEnvironment
- manifest
- method
- onDemandResourcesAssetPacksBaseURL
- teamID
- thinning
- uploadBitcode
- uploadSymbols

To get a more information about each of the parameters run `xcodebuild --help`

Access any command line tool in Xcode app bundle (xcrun)

`xcrun` uses the system default Xcode version (set via `xcode-select`) to locate and execute command line tools from the Xcode application bundle, e.g., `llvm-cov`.

```
# Generate code coverage reports via llvm-cov
# /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin
xcrun llvm-cov [parameters]

# Execute xcodebuild
# /Applications/Xcode.app/Contents/Developer/usr/bin
xcrun xcodebuild [parameters]

# Use Xcode's version of git, e.g., if you have installed a newer version
# /Applications/Xcode.app/Contents/Developer/usr/bin
xcrun git [parameters]
```

Switching command line tools with xcode-select

Print the path to the active developer directory (selected Xcode)

```
xcode-select -p
```

Select a different version of Xcode, e.g. Beta

```
sudo xcode-select -s /Applications/Xcode-beta.app
```

Reset to the default version of Xcode

```
sudo xcode-select -r
```

This is equivalent to running `sudo xcode-select -s /Applications/Xcode.app`

For more details: `man xcode-select`

Read Command Line Tools online: <https://riptutorial.com/xcode/topic/2158/command-line-tools>

Chapter 4: Creating Custom Controls in Interface Builder with @IBDesignable

Remarks

It became much easier to create custom controls in Interface Builder with the introduction of the `@IBDesignable` and `@IBInspectable` directives in Swift. Developers can now build rich, complex, fully animated controls using just a few extra lines of code. I'm surprised by how many developers have yet to fully embrace this feature, and I frequently find that adding just a few of lines of code to existing classes can make them so much easier to work with.

Note that these features are also available in Objective-C and are a great way of breathing life into old classes. The syntactic equivalents in Objective-C are `IB_DESIGNABLE` and `IBInspectable`, but for now I'll be concentrating on examples in Swift.

Examples

A Live-Rendered Rounded View

This is such a common requirement in iOS development, and it was always something that had to be done purely in code (or using images - yuck!). Now it's incredibly easy to preview this kind of thing in Interface Builder, there's absolutely no excuse for not using it.

Here's the code:-

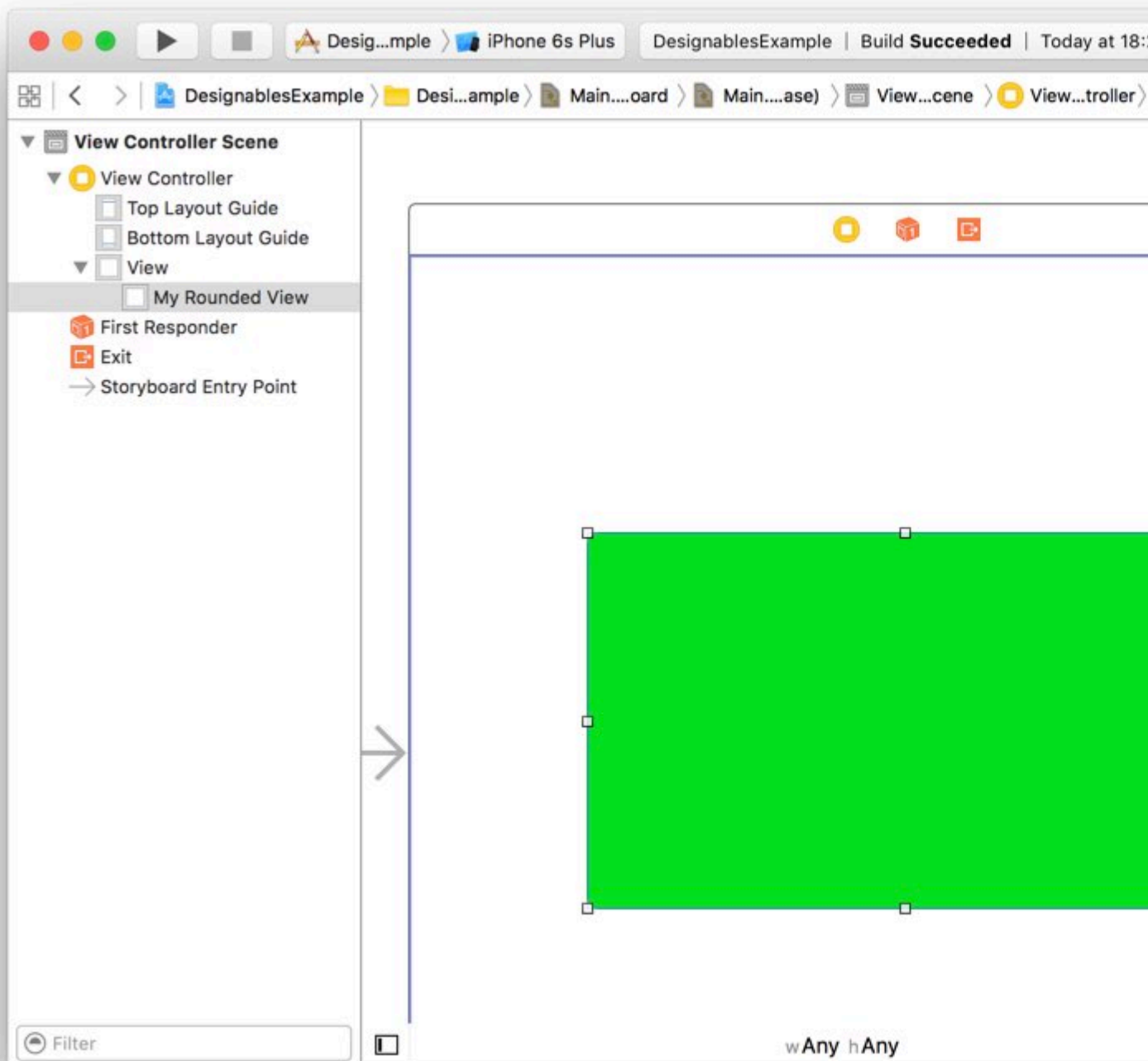
```
import UIKit

@IBDesignable
class MyRoundedView: UIView {

    @IBInspectable var radius: CGFloat = 8 {
        didSet {
            self.layer.cornerRadius = radius
        }
    }

    override func awakeFromNib() {
        self.layer.cornerRadius = self.radius
        self.layer.masksToBounds = true
    }
}
```

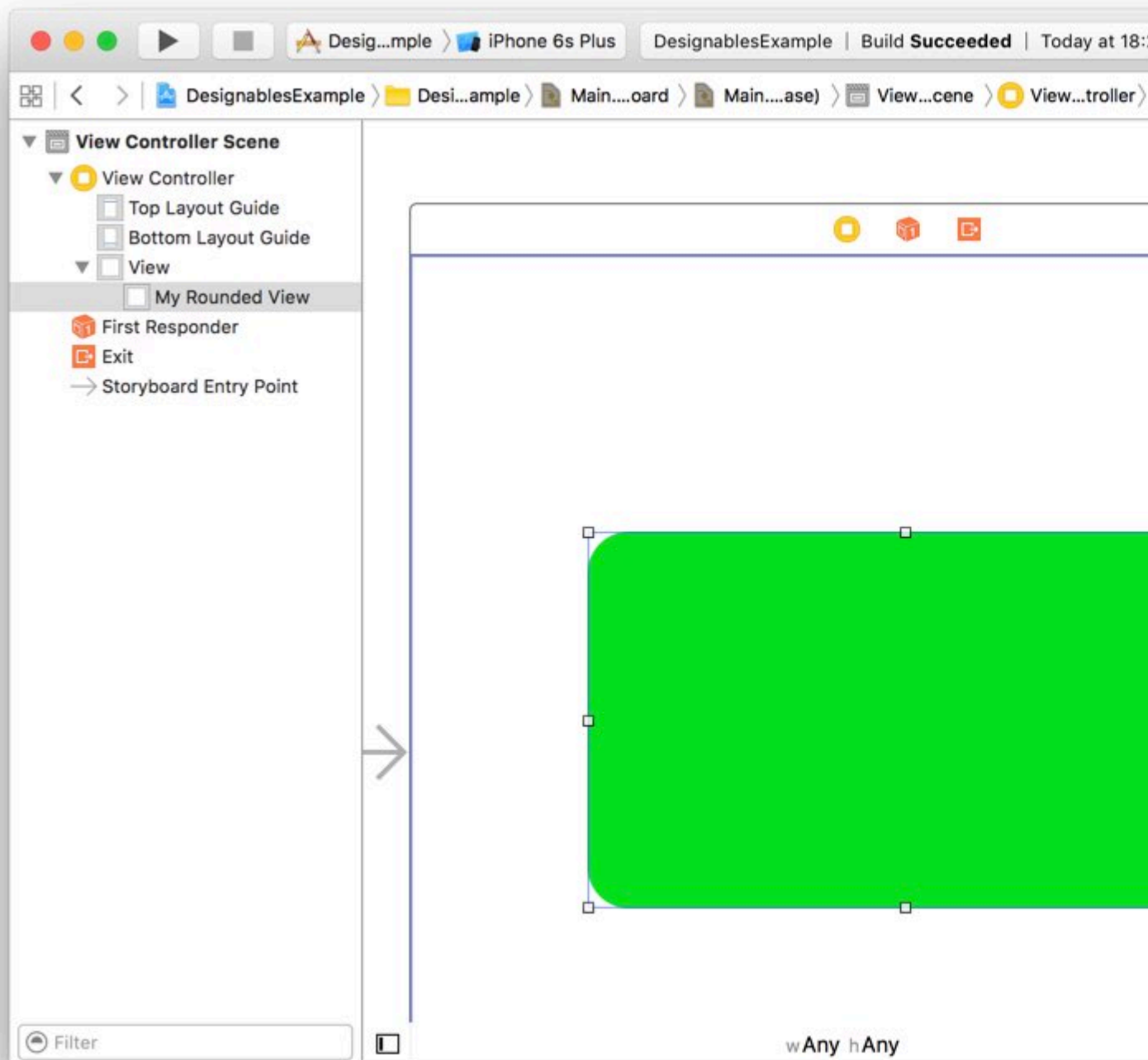
To use this class, add it to your project and then open the storyboard in IB and create a normal `UIView` of a decent size. Give it a background colour so you can see it, then navigate to the Identity Inspector in the right-hand Utilities panel and change the class type in the drop-down to `MyRoundedView`.



When you do this you should see a third label appear beneath "Class" and "Module" that says "Designables", and it should say "Updating" for a moment before changing to "Up to date". This means that Xcode has recompiled your code for `MyRoundedView` successfully.

Now you can open the Attributes Inspector and you should see (maybe after a short pause) a new section at the top of the pane with the heading "My Rounded View" and a new attribute labelled "Radius" with the value 8 (because that is the initial value we set in the code). This has appeared in the Attributes Inspector because we marked it as `@IBInspectable`.

You can now change this to another number and you should see the rounded view's corner radius update in real-time!



Read [Creating Custom Controls in Interface Builder with @IBDesignable](https://riptutorial.com/xcode/topic/6193/creating-custom-controls-in-interface-builder-with-ibdesignable) online:
<https://riptutorial.com/xcode/topic/6193/creating-custom-controls-in-interface-builder-with-ibdesignable>

Chapter 5: Cross-Platform Development

Examples

TargetConditionals

The system header `TargetConditionals.h` defines several macros which you can use from C and Objective-C to determine which platform you're using.

```
#import <TargetConditionals.h> // imported automatically with Foundation

- (void)doSomethingPlatformSpecific {
    #if TARGET_OS_IOS
        // code that is compiled for iPhone / iPhone Simulator
    #elif TARGET_OS_MAC && !TARGET_OS_IPHONE
        // code that is compiled for OS X only
    #else
        // code that is compiled for other platforms
    #endif
}
```

The values of the macros are:

7.0

When using the iOS 9.1, tvOS 9.0, watchOS 2.0, OS X 10.11 or newer SDKs:

Macro	Mac	iOS	iOS simulator	Watch	Watch simulator	TV	TV simulator
TARGET_OS_MAC	1	1	1	1	1	1	1
TARGET_OS_IPHONE	0	1	1	1	1	1	1
TARGET_OS_IOS	0	1	1	0	0	0	0
TARGET_OS_WATCH	0	0	0	1	1	0	0
TARGET_OS_TV	0	0	0	0	0	1	1
TARGET_OS_SIMULATOR	0	0	1	0	1	0	1
TARGET_OS_EMBEDDED	0	1	0	1	0	1	0
TARGET_IPHONE_SIMULATOR	0	0	1	0	1	0	1

7.0

When using the iOS 8.4, OS X 10.10, or older SDKs:

Macro	Mac	iOS	iOS simulator
TARGET_OS_MAC	1	1	1
TARGET_OS_IPHONE	0	1	1
TARGET_OS_EMBEDDED	0	1	0
TARGET_IPHONE_SIMULATOR	0	0	1

Read Cross-Platform Development online: <https://riptutorial.com/xcode/topic/358/cross-platform-development>

Chapter 6: Customizing Xcode IDE

Introduction

This is collection of different tips and tricks, to customize and improve your Xcode IDE

Examples

Open Terminal in current Xcode project folder

Xcode have ability to run any script with hotkey.

Here is example how to assign hotkey `⌘+⌘+^+⌘+T` to open Terminal app in current project folder.

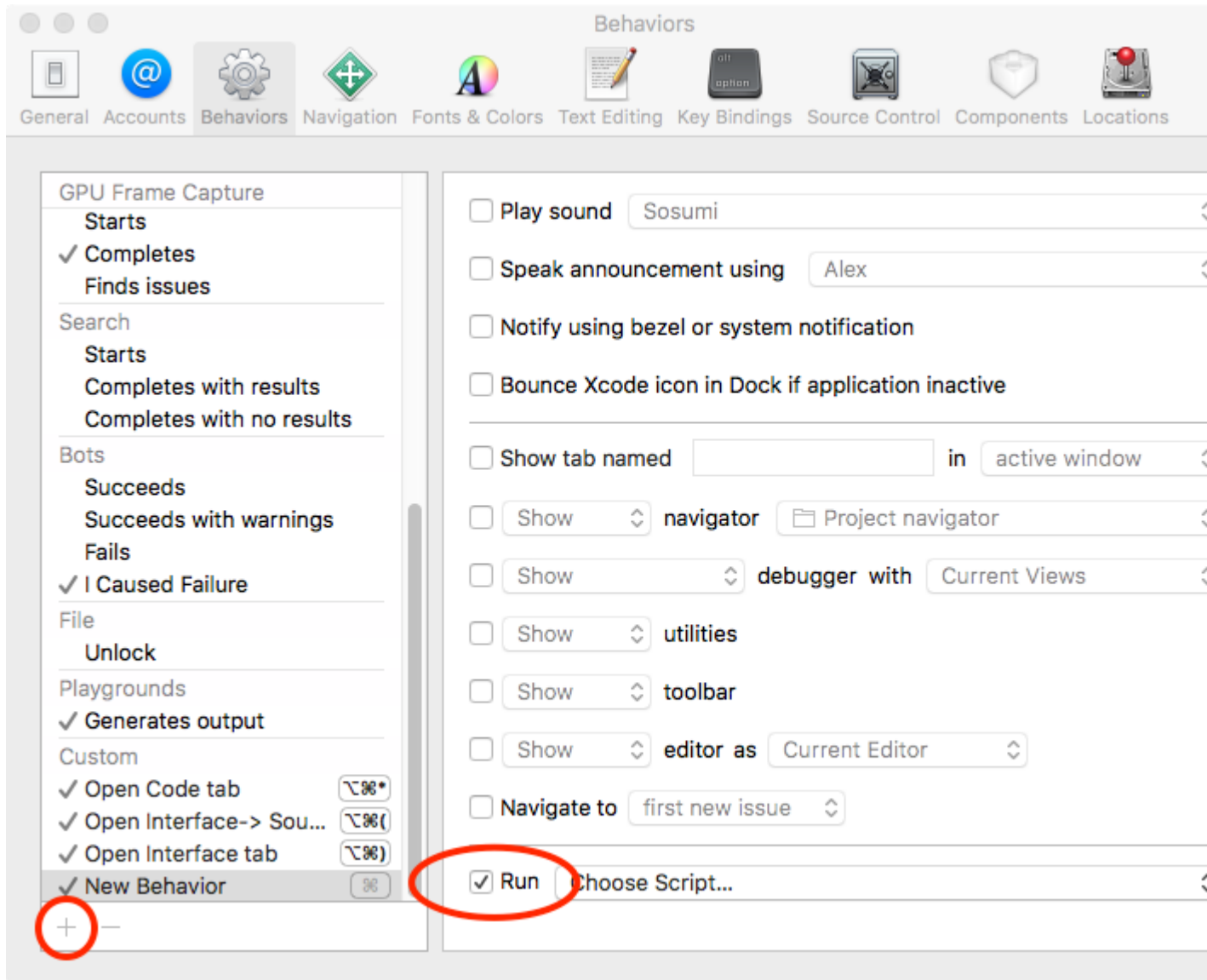
1. Create bash script and save it in some folder

```
#!/bin/bash

# Project Name:  $XcodeProject
# Project Dir:   $XcodeProjectPath
# Workspace Dir: $XcodeWorkspacePath

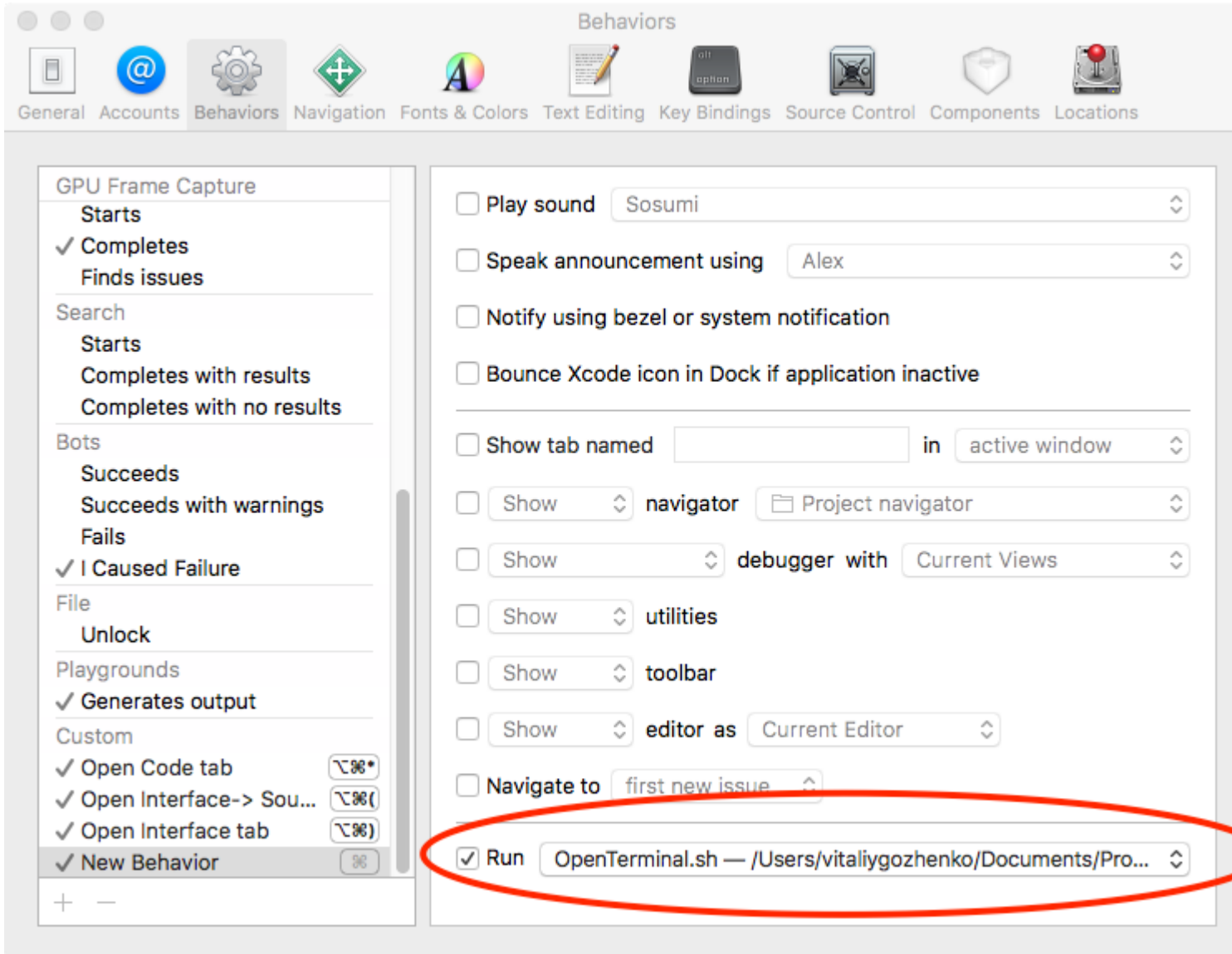
open -a Terminal "$(dirname $XcodeProjectPath)"
```

2. Make script executable: open Terminal at script folder and run `chmod +x your_script_name.sh`
3. Open Xcode Preferences at Behaviors tab
4. Add new custom behavior by tapping + in the bottom left corner
5. Check `Run` action at the right

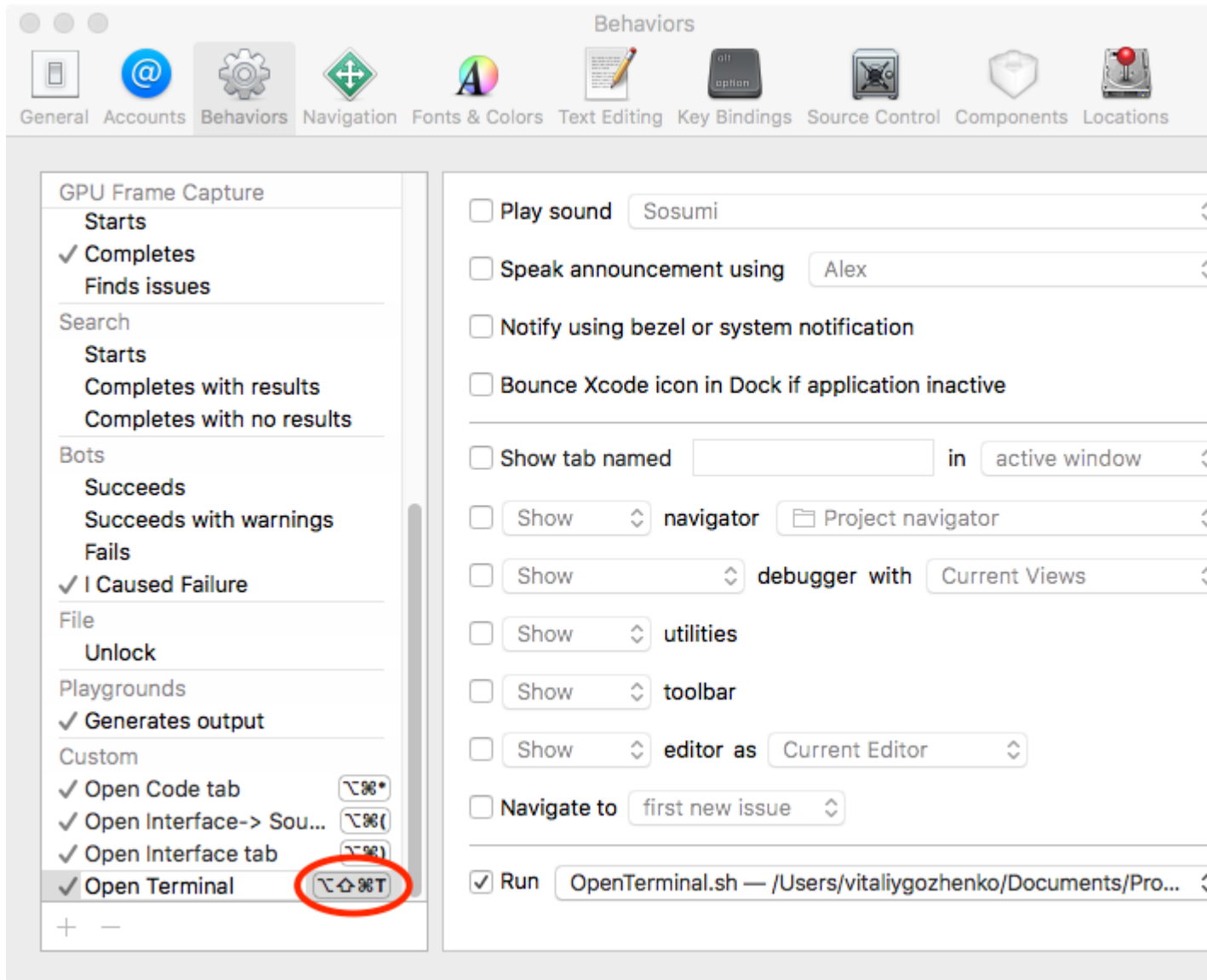


6. Choose script, which you create previously by clicking at the `Choose Script...` twice.

If your script is grayed, be sure, that you run `chmod +x` on your script file



7. Assign hotkey (for example $\square + \square + \wedge + \square + \text{T}$) to your behavior and rename it



Now you can open terminal in project folder with one hotkey.

This is only one example of using Xcode behaviors, but you can create any script and launch any app with it.

Bash script author: <http://mattorb.com/xcode-behaviors-for-fun-and-profit/>

Clear derived data with hotkey

In the same way as in `Open Terminal in current Xcode project folder` example, you can add clear of derived data folder with hotkey.

Create custom behavior (follow the steps in [Open Terminal in current Xcode project folder](#)). But use another script.

Script text:

```
#!/bin/bash  
  
rm -rf $HOME"/Library/Developer/Xcode/DerivedData/"
```

Read Customizing Xcode IDE online: <https://riptutorial.com/xcode/topic/8260/customizing-xcode-ide>

Chapter 7: Debugging

Examples

Breakpoints

In xcode developers can pause/break the execution of running app and can examine the state of program.

Here's how to pause running programs:

Just open any file in which we want to put breakpoint and click on the line on gutter at left side where we want to pause execution.

Debug gauges

The screenshot shows the Xcode IDE interface with the following components:

- Debug gauges (left):** Displays system metrics for the running application 'Jogr' (PID 2753, Paused). Metrics include CPU (0%), Memory (227.6 MB), Disk (16 KB/s), and Network (Zero KB/s). A thread list shows 'Thread 84' (Queue: Graph serial queue (serial)) is selected, with a sub-list of threads including '0 _35-[GraphView _plotAccel...' and '1 _dispatch_call_block_and_r...'.
- Source code (middle):** Shows the implementation of the `GraphView` class. A breakpoint is set on line 21, and another is set on line 38. The code includes properties for `_routeProvider`, `_graphImage`, `_routeDescription`, `_routeImage`, `_routeStartLocation`, `_velocityDataLock`, and `_velocityData`. The `_plotAccelerationCurve` method is shown, which uses a serial queue to plot acceleration data points.
- Variable inspector (bottom):** Shows the current state of variables at the breakpoint on line 38. The variables are: `self` (GraphView *) 0x7fe67f6250f0, `currentPoint` (CGPoint) (x=0, y=0), `path` (UIBezierPath *) 0x7fe67f18a1e0, `_graphSerialQueue` (dispatch_queue_t) 0x7fe67c727e00, and `velocityDataLock` (NSLock *) 0x7fe67c727d90.

Debug navigator

Breakpoint

Debug bar

Variable

So here we placed breakpoints on line no 21 and 38; when execution reaches at line 38 Xcode

paused execution and shown green line on that line.

Debug Gauges gives us an glimpse of CPU usage, Memory usage and at bottom the execution stack with Threads and function names. We can know which stack or sequence of functions lead execution to this line of break.

Variables View gives all the details of states and values of all variables in the scope of breaded line. We can see their values, memory addresses, properties of instances and their details.

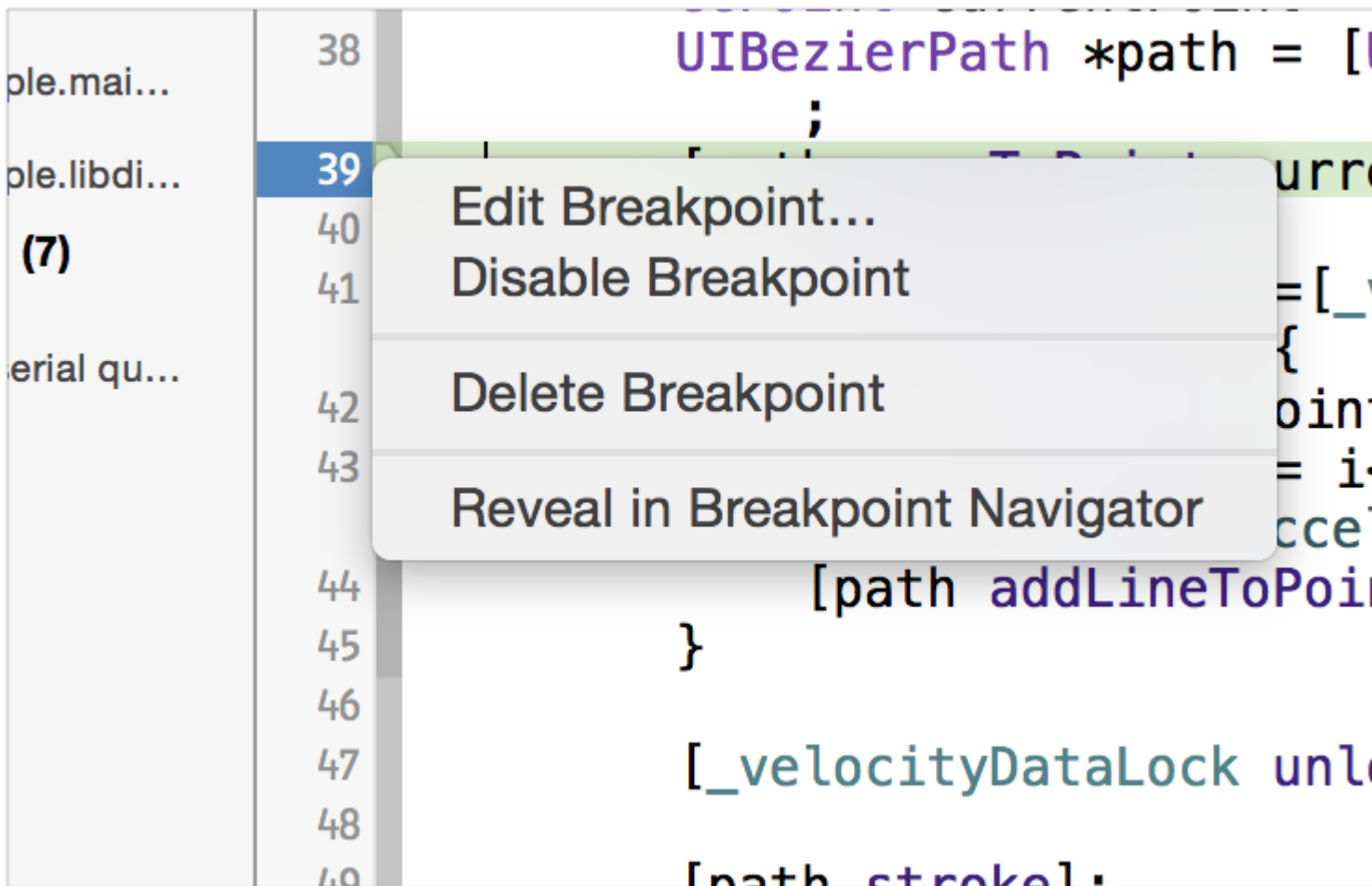
Console can be used to print value of any variable that is in scope. Using `PO` command we can achieve this.

Debug Bar has controls for breakpoints.

- First button is to enable/disable paused breakpoint.
- Second button used to pause/resume the execution of programs
- Third one is Step-Over button used to execute to next line
- Fourth button in Step-In used to enter inside currently executing function
- Fifth one is Step-Out button for coming out of current function

Configure Breakpoint:

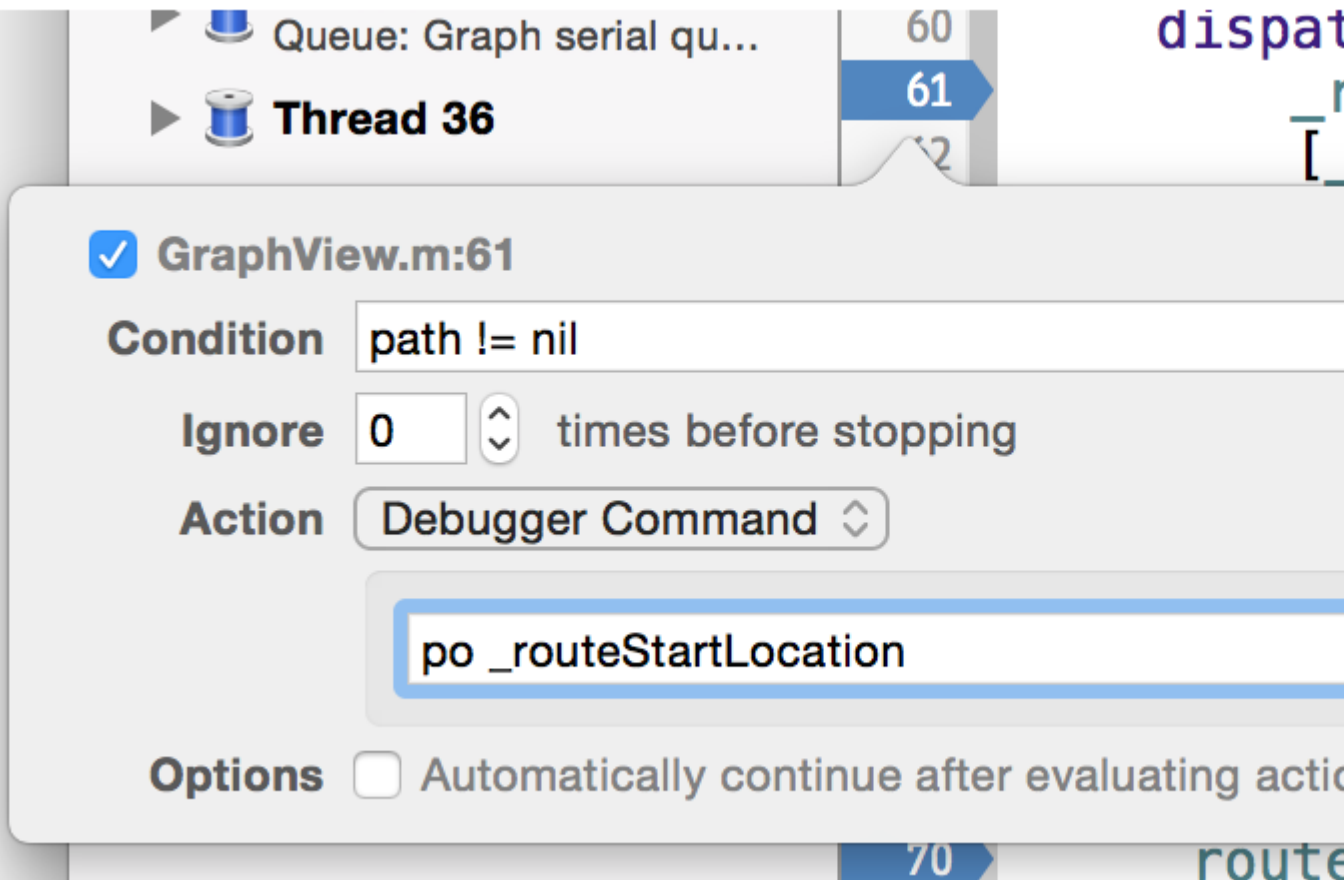
We can even have more control on breakpoints.



Delete and Disable are straightforward functions.

Reveal in Navigator takes us to Breakpoint navigator where all the breakpoints from project are listed as File Navigator.

Edit Breakpoint is something we should use more often for detailed debugging. We can configure breakpoints using this function. We can add conditions and actions to breakpoints as:



As shown in image, that breakpoint will be paused only if `path != nil`. If this condition is true then `po _routeStartLocation` action is executed and mentioned earlier `po` will print value of `_routeStartLocation` on console.

For detailed explanation, [follow this detailed link](#).

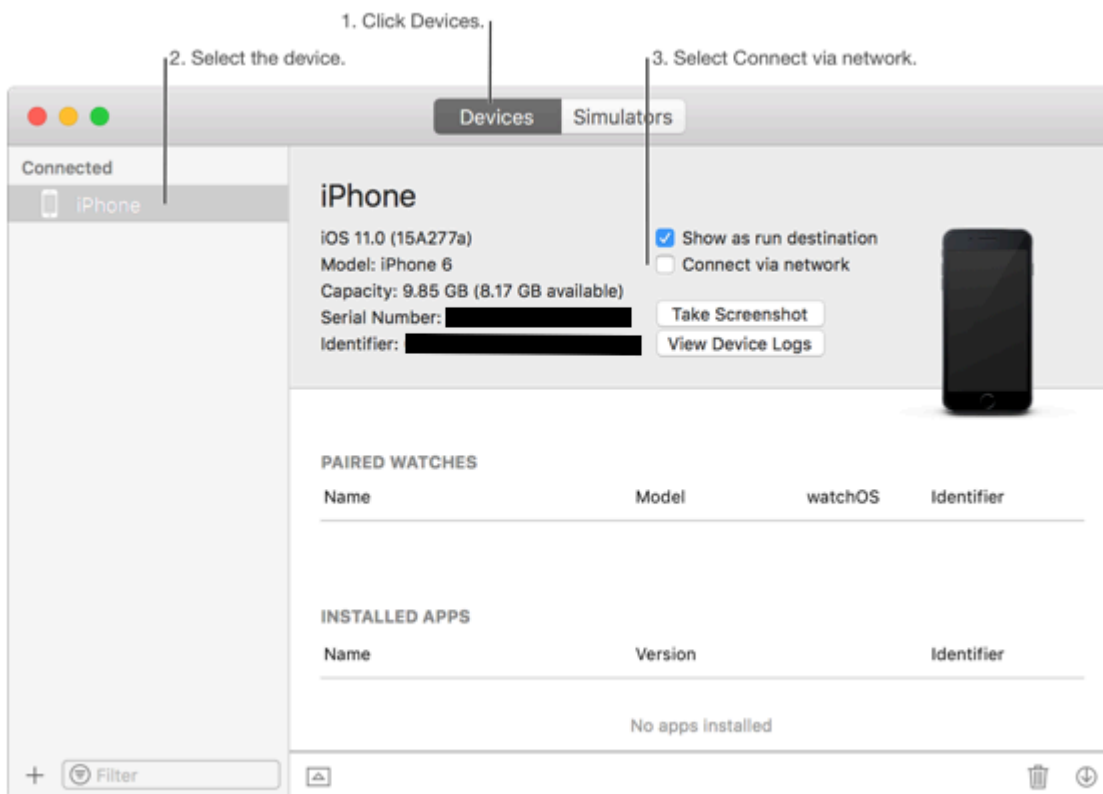
Wireless Debugging in Xcode-9

As recently Apple rolled out iOS11 and Xcode-9, we can now debug apps on devices without connecting devices to Xcode through USB.

We can take advantage of wireless debugging feature added to this Xcode-9.

To enable wireless debugging, we have to configure some steps in Xcode.

- 1 First connect the device running iOS11 to the Xcode-9.
- 2 Go to Window > Devices and Simulators in Xcode menus and under Devices tab connected device will be listed.
- 3 Then check the checkbox named Connect via network as in this picture:



(Image courtesy: [Surjeets' SO post](#))

4 Then disconnect your device from USB cord, make sure iPhone/iPad/iPod device and Mac running Xcode are in same wireless network.

5 In Xcode you will see this devices listed and you can directly run your app on that device.

We can perform all the operations with Xcode on that device same as if it is connected using USB; except that we **can not see logs if** app is run using Xcode, put it in background and suspended in background state and we launch it again. This is possible with USB debugging.

NOTES:

1 We have to use Xcode-9, iOS 11 running on device

2 Both device and Mac should on same wireless network

Read Debugging online: <https://riptutorial.com/xcode/topic/10459/debugging>

Chapter 8: Playgrounds

Examples

Getting Started with Playground

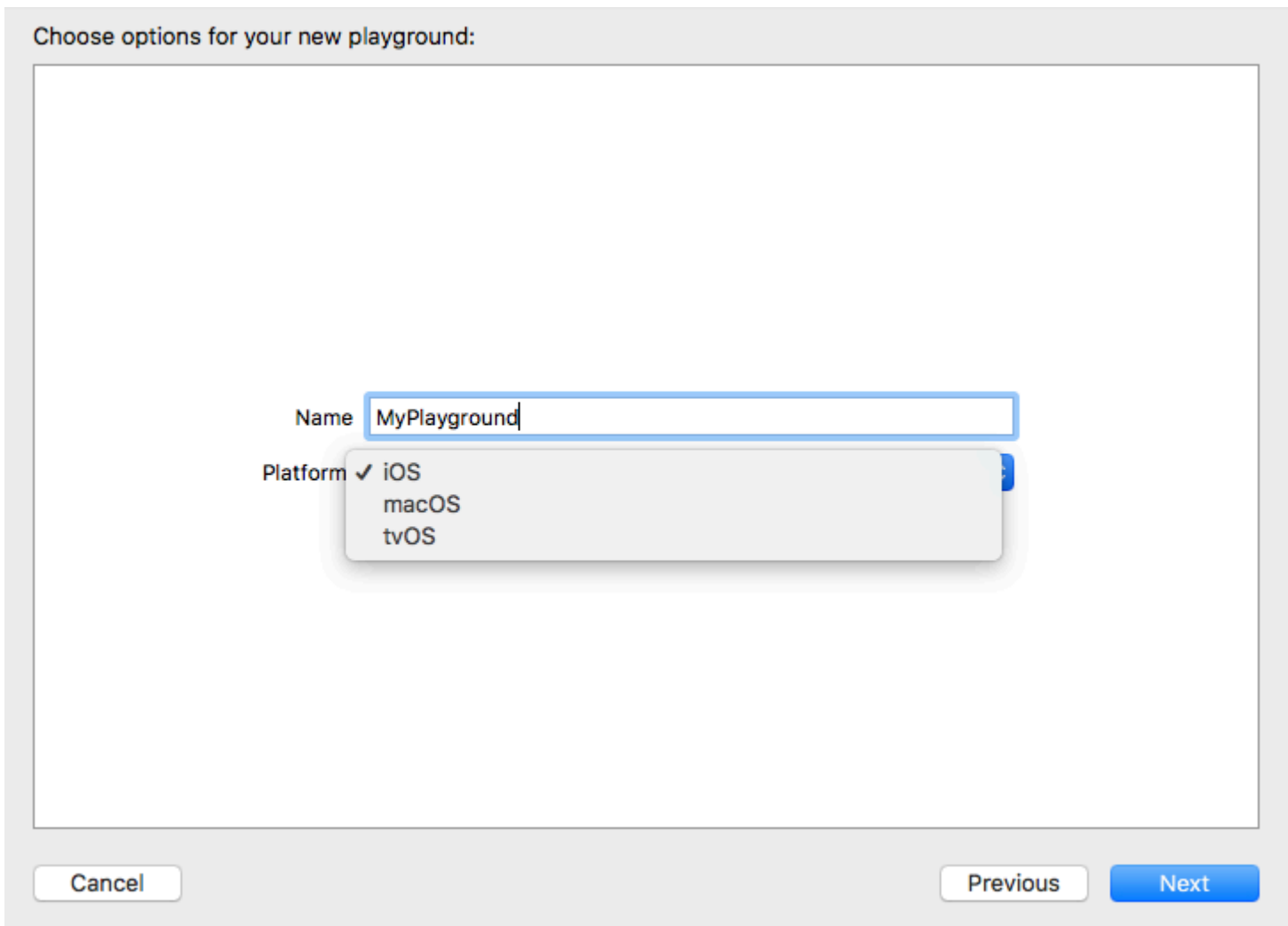
1. Create a new playground file:

- First option: From Xcode welcome screen, select the first option (**Get started with a playground**).



- Second option: From menu select **File** → **New** → **Playground (N)**.

2. Name your playground and select the platform (iOS/macOS/tvOS), then click **Next**.



3. On the next screen, choose where you want to save your playground, then click **Create**.

Latest Value, Value History and Graph

Using Playground it is easy to see that happens inside loops or objects while the change is happening.

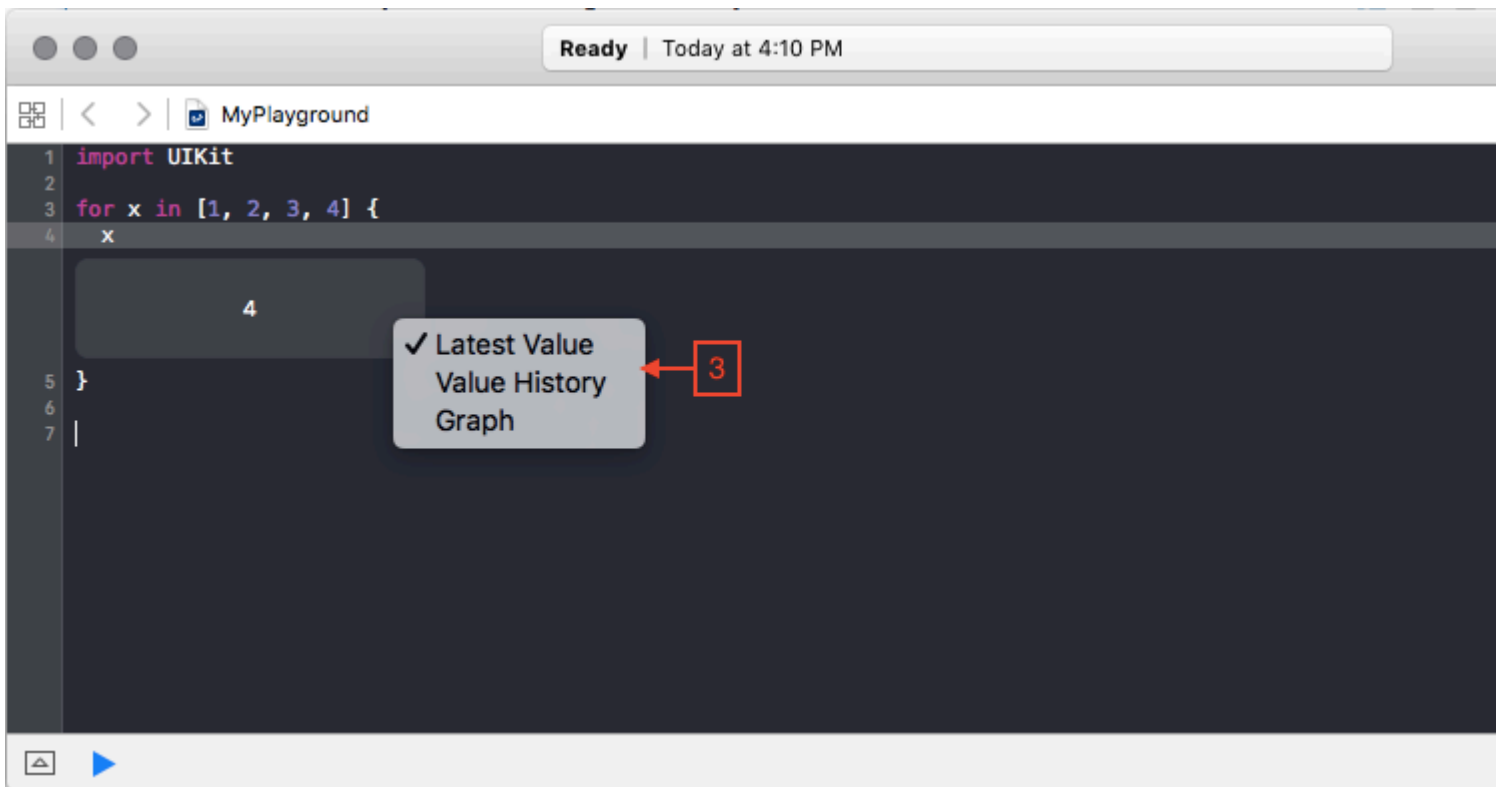
For example, in the code below, the value of `x` will change from 1 to 4.

```
import UIKit
for x in [1, 2, 3, 4] {
    x
}
```

(1) Clicking on the eye symbol on the right will give us a quick look.

(2) Clicking on the circle next to it will open show the *Latest Value* below the line.

(3) Right click on the view added will show a drop down menu with **Latest Value, Value History and Graph**



Adding Images, Static Data, Sounds, etc. to a Playground

Images, static data, sounds, etc. are resources in a Playground.

1. If the Project Navigator is hidden, choose View > Navigators > Show Project Navigator (1)
2. There are several ways to add files
 - Drag your resources to the `Resources` folder or
 - Select the Resources folder and choose File > Add Files to "Resources" or
 - control-click the Resources folder and choose Add Files to "Resources"
3. Use your resource. For example `let i = UIImage(named: "tacos.jpg")`

Read Playgrounds online: <https://riptutorial.com/xcode/topic/1236/playgrounds>

Chapter 9: Projects & Workspaces

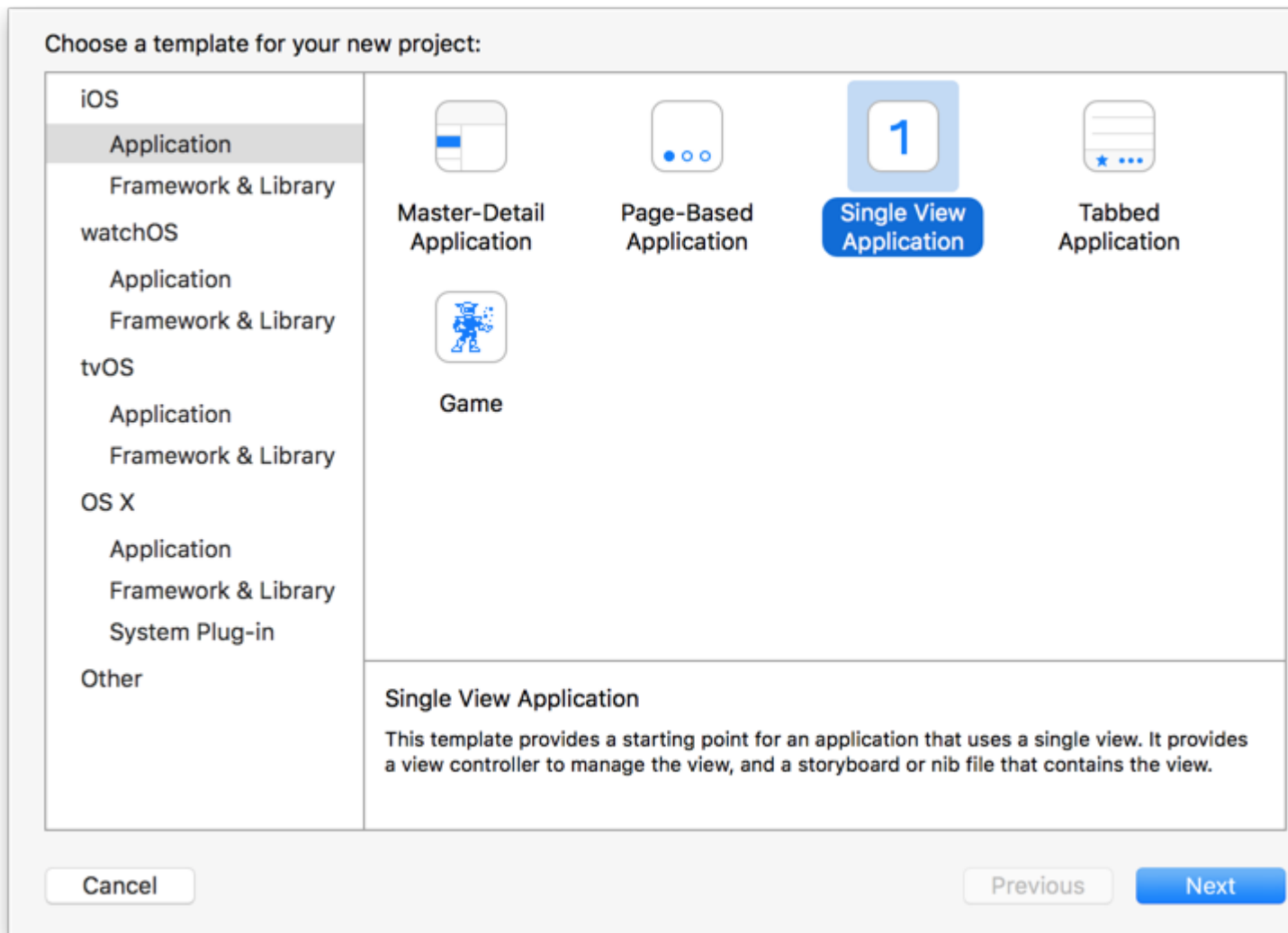
Examples

Projects overview

Xcode **projects** are used to organize source files, library dependencies, and other resources, as well as the settings and steps required to build the project's products. **Workspaces** are groups of projects and other files.

Create a project

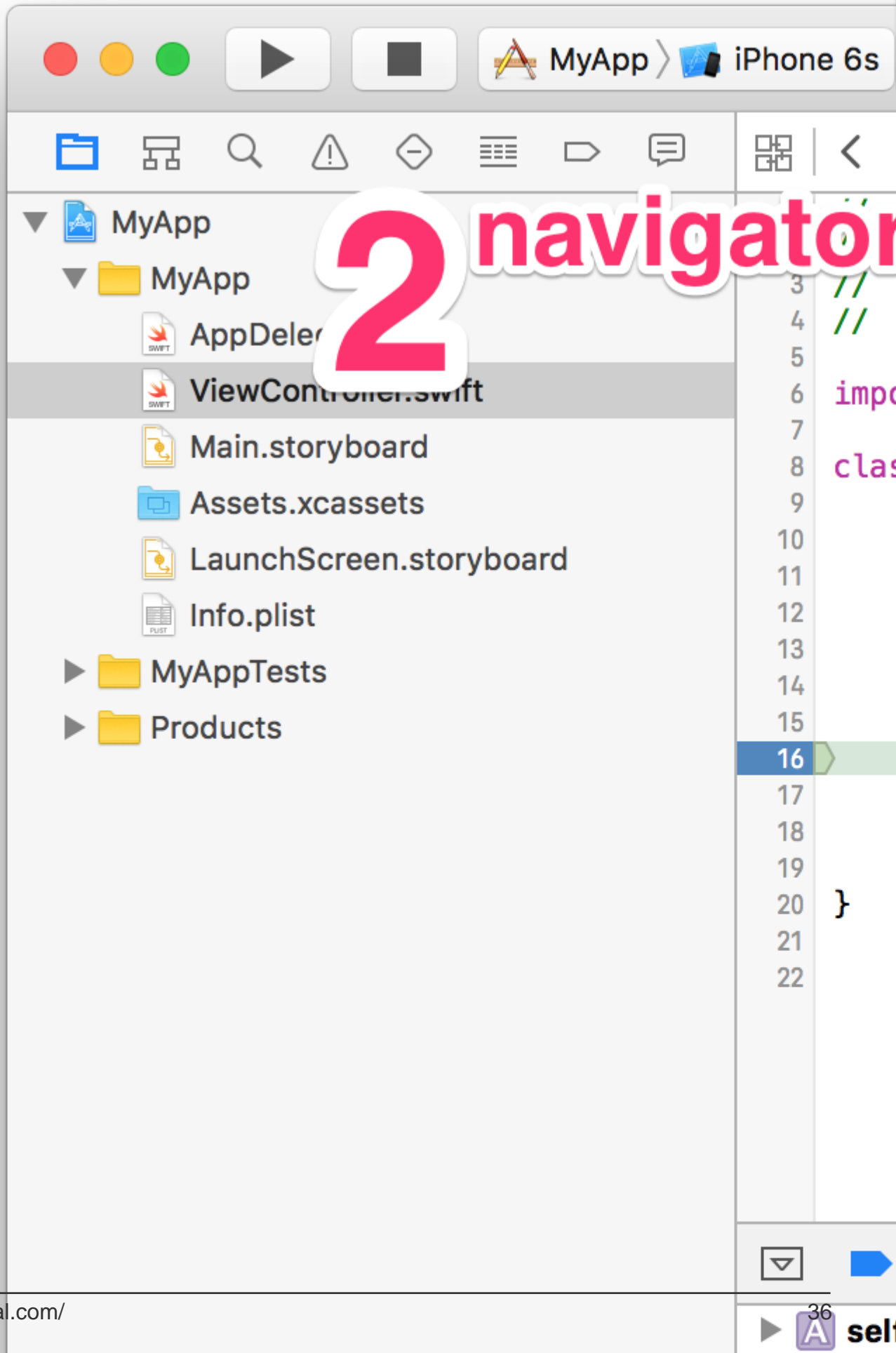
You can create a **New Project** (N) from a number of built-in templates:



Working with projects

An Xcode project window includes:

1. **Toolbar** (top)
2. **Navigator** (left)
3. **Editor** (center)
4. **Inspector** (right)
5. **Variables View** (lower-middle left)
6. **Console output** (lower-middle right)
7. **Library** (lower right)

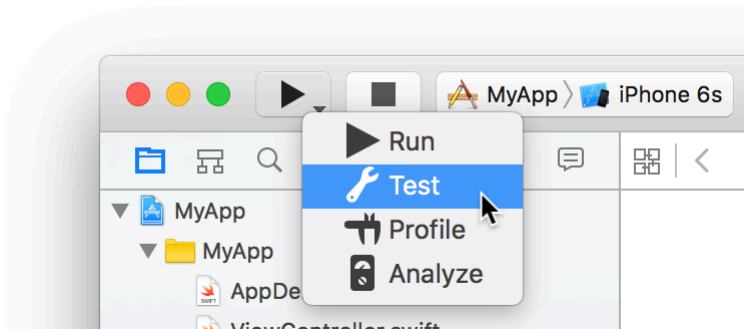


2 navigator

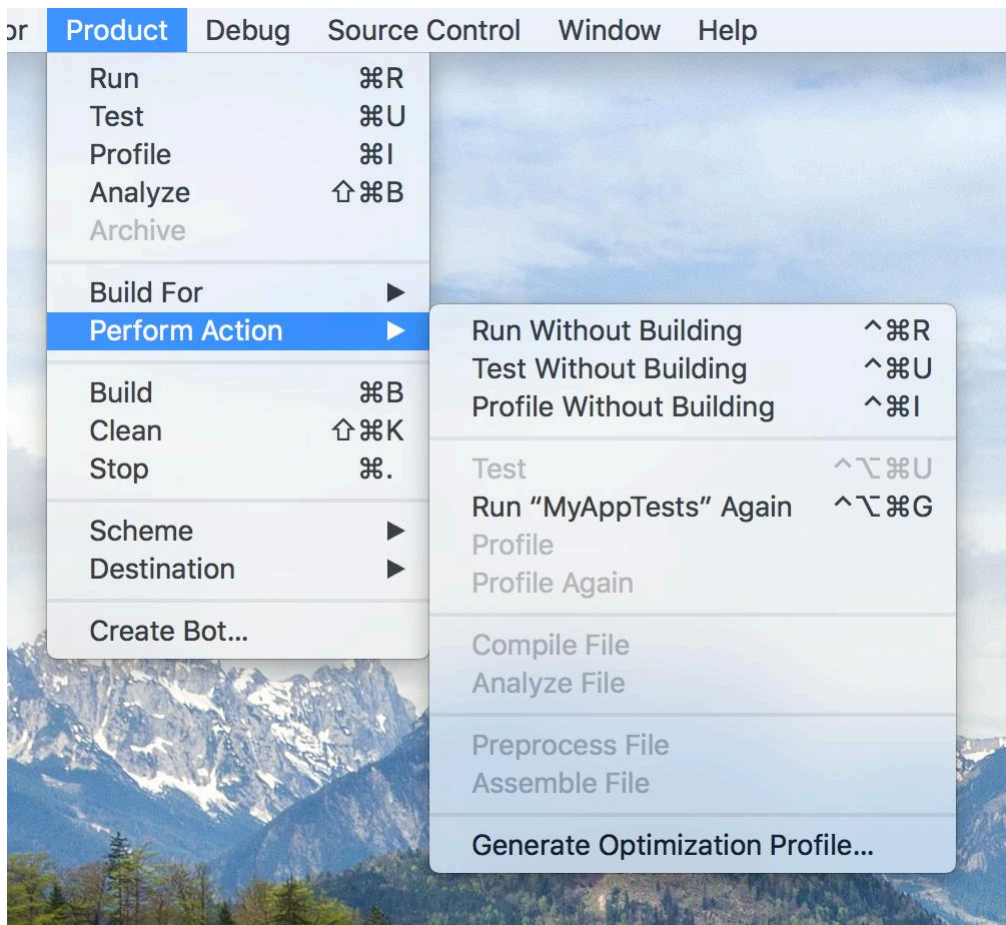
.) to stop execution.



Click & hold to see the other actions, Test (U), Profile (I), and Analyze (B). Hold down modifier keys `⌘` option, `⇧` shift, and `⌘` control for more variants.



The same actions are available in the Product menu:



Adjust workspace to your needs and freely navigate it

One of the things that can really boost your productivity while writing the code is effectively navigating the workspace. This also means making it comfortable for the moment. It's possible to achieve this by adjusting which areas of workspaces you see.

The buttons on the top of the navigation and assistant areas are not that big and are a bit hard to click with the mouse pointer. That's why there are helpful and easy to remember shortcuts that let you switch between different tabs or hide the area altogether.

You can switch between different panels in the navigation area by holding the `⌘` (Command) button and pressing on different digit keys from 1 to 8 or 0. The `⌘0` key toggles the presence of the navigator. Here's a list of shortcuts for your convenience:

1. `⌘+1` - File navigator;
2. `⌘+2` - Symbol navigator;
3. `⌘+2` - Search (also reachable through `⌘+⌘+F`);
4. `⌘+4` - Warnings, errors and static analysis messages;
5. `⌘+5` - Tests available in the project;
6. `⌘+6` - Debug session panel;
7. `⌘+7` - Breakpoints;
8. `⌘+8` - Report/action history navigator;
9. `⌘+0` - Show/Hide the navigator panel;

You can switch between different panels in the inspector area by holding `⌘` and `⌘` and pressing different digit keys from 1 to 6 (depending on the currently active editor: whether it's code, storyboard, xib or other type of resource). Pressing `⌘0` while holding these two buttons will hide the inspector area.

So if you are editing a storyboard and need more visible space just tap `⌘ + 0` and `⌘+⌘+0` to get extra pixels for the canvas.

While switching panels on either side mostly depend on the combination of `⌘` or `⌘` and `⌘`, the bottom search fields are activated by holding `⌘` and `⌘` and pressing either `⌘j` for navigation area search bar or `⌘k` for library area search bar.

Activating the navigation search area can help you narrow the list of items displayed in navigator area depending on which navigator is active (filter files in the file navigator, symbols in the symbol navigator, tests in test navigator, etc).

Activating the inspector panel will help you filter the list of either file templates, code snippets, objects or media resources. Try using this search field when you have storyboard open and you quickly need to find a `UINavigationController` or `UITableViewCell` components!

Speaking of library, you can switch between library panels (File templates, code snippets, object and media libraries) `⌘+⌘+⌘` and respective digit keys: 1 through 4.

Read Projects & Workspaces online: <https://riptutorial.com/xcode/topic/335/projects---workspaces>

Chapter 10: Xcode 8 features

Remarks

This only works with projects using Swift 3+

Examples

Color and image literals

Xcode 8 will automatically recognize any images you've got in an Asset Catalog and offer them up as a suggestion inside of a UIImage initializer.

So you could basically declare a new variable and then add an asset name that you have added to your asset catalog. For example `let img = dog.img` does now contain the image of `dog` that's in the asset catalog.

Under the hood it's creating code that looks like this: `#imageLiteral(resourceName: "dog.png")`. But inline in the source editor, you'll just see the file name of the image.

So you could do this now `imageView.image = img`.

Note that you need to click on the instellisense suggestion so that you see a thumbnail of the image in the code and then the image name.

Read Xcode 8 features online: <https://riptutorial.com/xcode/topic/7155/xcode-8-features>

Chapter 11: Xcode Tips

Examples

Reuse code snippets in Xcode

You can save your code snippets for use later simply by drag and drop. For eg: if you have an NSLog statement that used for so many places somewhere else in the project, then you can save the NSLog statements to code snippets library.



Xcode

File

Edit

View

Find

Navigate



Sample >



iPhone 6s Plus



S

▼ Sample

▼ Sample

h AppDelegate.h

m AppDelegate.m

h ViewController.h

m ViewController.m

Main.storyboard

Assets.xcassets

LaunchScreen.storyboard

Info.plist

▶ Supporting Files

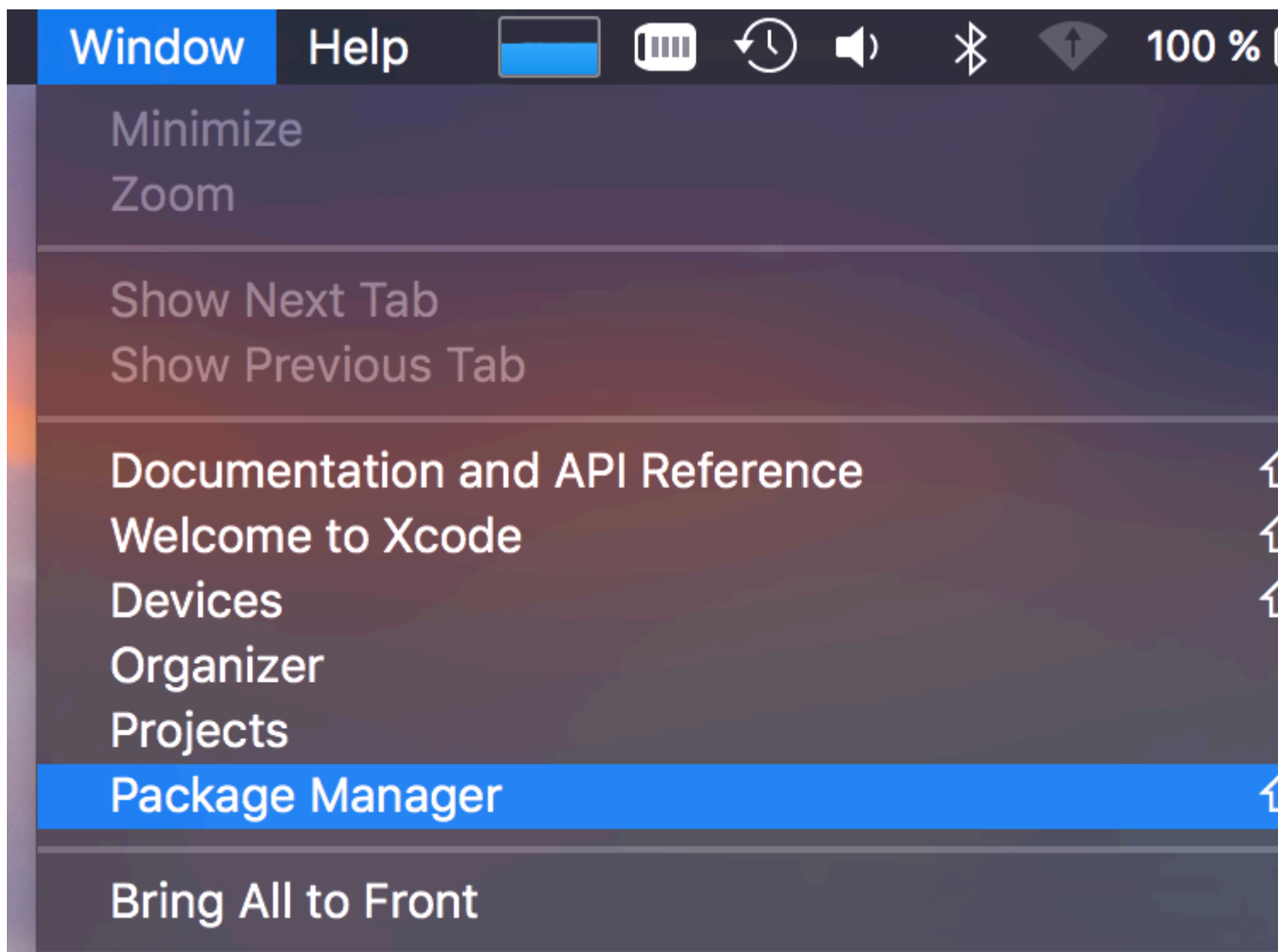
▶ Products

```
1 //
2 // ViewCon
3 // Sample
4 //
5 // Created
6 // Copyrig
7 //
8
9 #import "Vi
10
11 @interface
12
13 @end
14
15 @implementa
16
17 - (void)vie
18     [super
19     // Do a
20 }
21
22 - (void)did
23     [super
24     // Disp
25 }
26
27 - (void)unw
28     subsequ
29     NSLog(@
30
31 @end
```

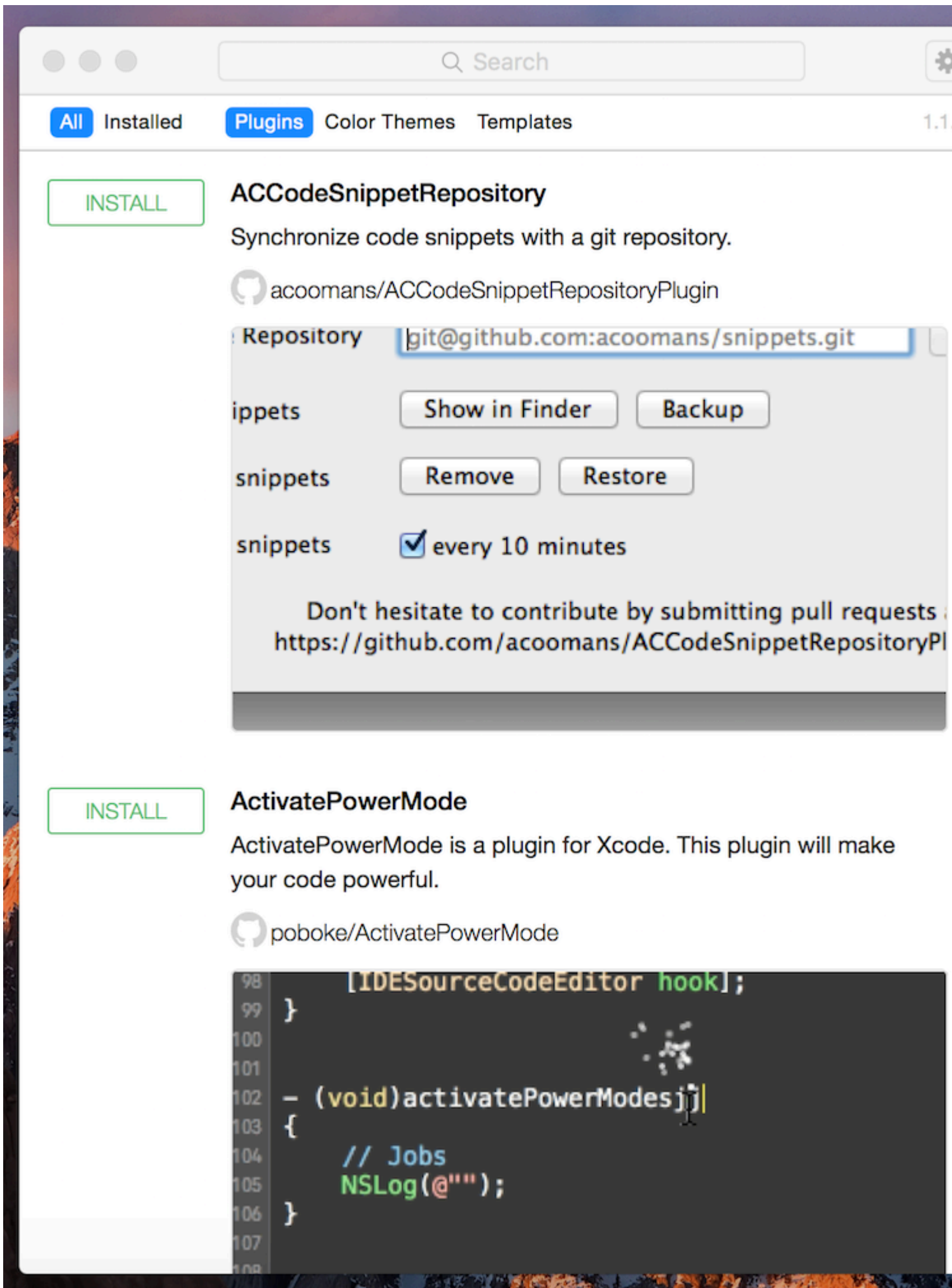
- [XcodeColors](#) - Colored console logs, e. g. using [CocoaLumberjack](#)
- [FuzzyAutocomplete](#) - Type "NSLog" and still get `NSLog` auto-completed
- [BuildTimeAnalyzer](#) - Set `-Xfrontend -debug-time-function-bodies` under `Other Swift flags` in the build settings and [optimize your Swift build time](#) in the build settings and [optimize your Swift build time](#)

Of course there are many more and some are so good, Apple already implemented them into Xcode 8 (FuzzyAutocomplete and VVDocumenter for example)

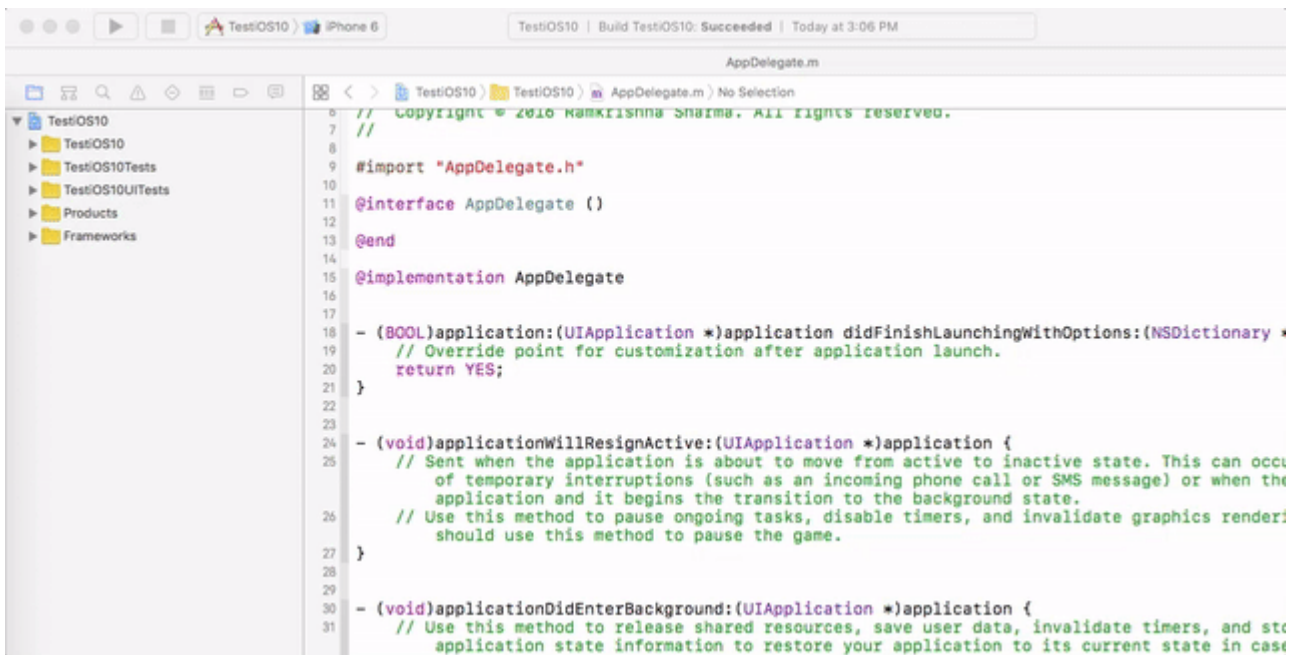
Usage



Hit `⌘ + ⌘ + 9` or use this menu to open up the Package manager.



2. On your Environment Variables set OS_ACTIVITY_MODE = disable

A screenshot of the Xcode IDE interface. The top status bar shows 'TestiOS10 | Build TestiOS10: Succeeded | Today at 3:06 PM'. The main editor window displays the AppDelegate.m file. The code is as follows:

```
0 // Copyright © 2016 Hemkrishna Sharma. All rights reserved.
1 //
2
3 #import "AppDelegate.h"
4
5 @interface AppDelegate ()
6
7 @end
8
9 @implementation AppDelegate
10
11 - (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
12     // Override point for customization after application launch.
13     return YES;
14 }
15
16 - (void)applicationWillResignActive:(UIApplication *)application {
17     // Sent when the application is about to move from active to inactive state. This can occur
18     // due to temporary interruptions (such as an incoming phone call or SMS message) or when the
19     // application is terminated. Use this method to pause ongoing tasks, disable timers, and invalidate graphics rendering.
20     // Use this method to pause ongoing tasks, disable timers, and invalidate graphics rendering.
21     // Use this method to pause ongoing tasks, disable timers, and invalidate graphics rendering.
22     // Use this method to pause ongoing tasks, disable timers, and invalidate graphics rendering.
23     // Use this method to pause ongoing tasks, disable timers, and invalidate graphics rendering.
24 }
25
26 - (void)applicationDidEnterBackground:(UIApplication *)application {
27     // Use this method to release shared resources, save user data, invalidate timers, and store
28     // application state information to restore your application to its current state in case
29     // the application is terminated.
30 }
31
```

Read Xcode Tips online: <https://riptutorial.com/xcode/topic/3349/xcode-tips>

Credits

S. No	Chapters	Contributors
1	Getting started with Xcode	Anh Pham , Community , Jbryson , jtbandes , Md. Ibrahim Hassan , Undo
2	Certificates, Provisioning Profiles, & Code Signing	KrauseFx
3	Command Line Tools	Ali Beadle , David Snabel-Caunt , Fabio , Idan , Jens Meder , KrauseFx
4	Creating Custom Controls in Interface Builder with @IBDesignable	Echelon
5	Cross-Platform Development	J F , jtbandes
6	Customizing Xcode IDE	Vitaliy Gozhenko
7	Debugging	D4ttatraya
8	Playgrounds	Anh Pham , Idan , Rob Wright
9	Projects & Workspaces	Eimantas , jtbandes
10	Xcode 8 features	Rashwan L , Sharukh Mastan
11	Xcode Tips	Anand Prem , Finn Gaida , jtbandes , user459460