

# Swift API Design Guidelines

## The Grand Renaming

Session 403

Doug Gregor Swift Engineer

Michael Ilseman Swift Engineer

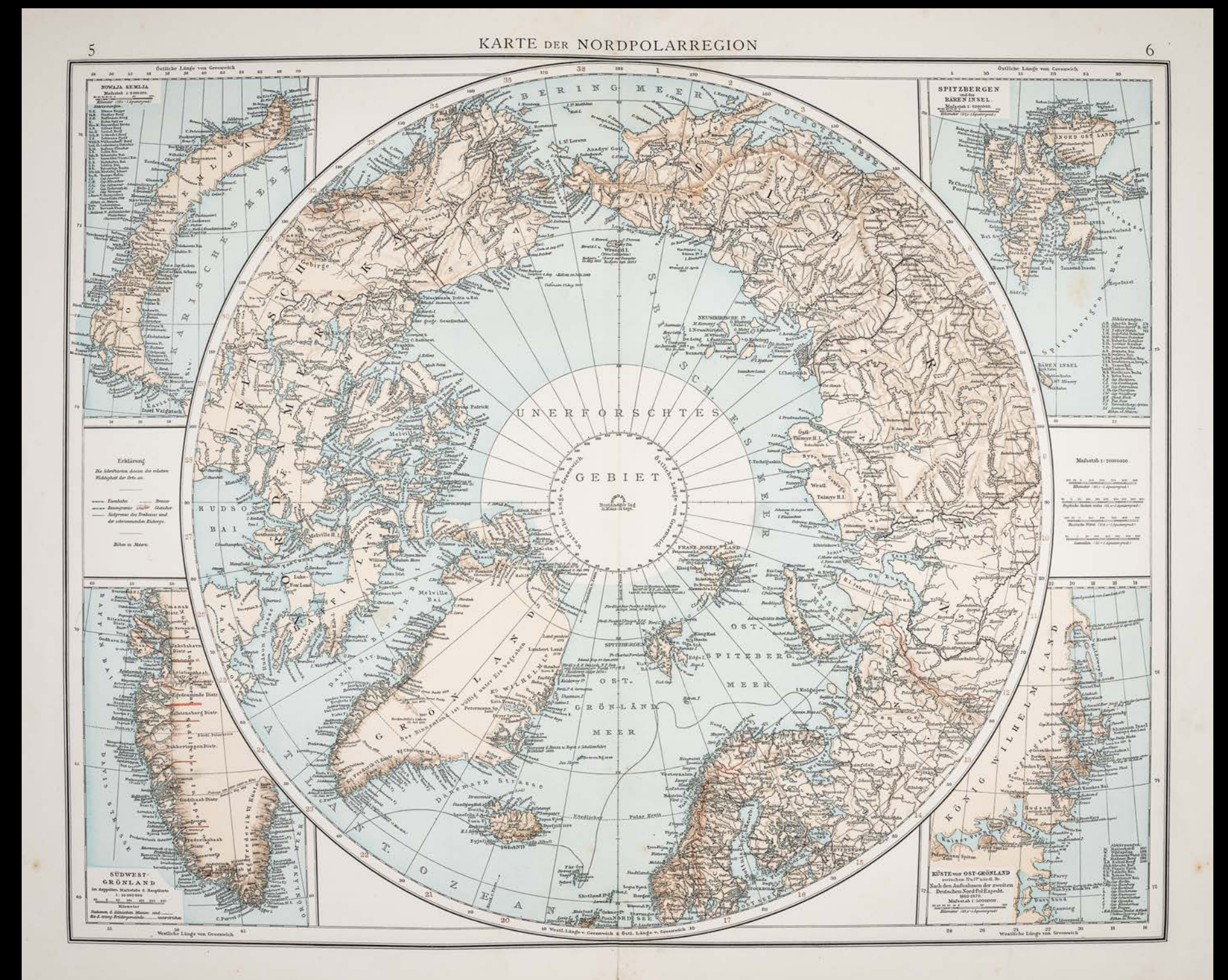


# Roadmap

Swift API Design Guidelines

The Grand Renaming

Mapping Objective-C APIs into Swift





Why?



# Languages Have Character

Every language has its own feel





# Languages Have Character

Every language has its own feel

```
DispatchQueue.main.async {  
    self.listDocumentsViewController?.present(signedOutController, animated: true)  
}
```

# Languages Have Character

Every language has its own feel

Feel of everyday APIs

```
DispatchQueue.main.async {  
    self.listDocumentsViewController?.present(signedOutController, animated: true)  
}
```



**Introduction**

- ▶ **Code Naming Basics**
- ▶ **Naming Methods**
  - Naming Functions**
- ▶ **Naming Properties and Data Types**
  - Acceptable Abbreviations and Acronyms**
- ▶ **Tips and Techniques for Framework Developers**
- Revision History**

[Next](#)

# Introduction to Coding Guidelines for Cocoa

Developing a [Cocoa framework](#), plug-in, or other executable with a public API requires some approaches and conventions that are different from those used in application development. The primary clients of your product are developers, and it is important that they are not mystified by your programmatic interface. This is where API naming conventions come in handy, for they help you to make your interfaces consistent and clear. There are also programming techniques that are special to—or of greater importance with—frameworks, such as versioning, binary compatibility, error-handling, and [memory management](#). This topic includes information on both Cocoa naming conventions and recommended programming practices for frameworks.

## Organization of This Document

The articles contained in this topic fall into two general types. The first and larger group presents naming [conventions](#) for programmatic interfaces. These are the same conventions (with some minor exceptions) that Apple uses for its own Cocoa frameworks. These articles on naming conventions include the following:

- [Code Naming Basics](#)
- [Naming Methods](#)
- [Naming Functions](#)
- [Naming Properties and Data Types](#)
- [Acceptable Abbreviations and Acronyms](#)

The second group (currently with a membership of one) discusses aspects of framework programming:

- [Tips and Techniques for Framework Developers](#)

[Next](#)



[▼ Table of Contents](#)**Introduction**

- ▶ **Code Naming Basics**
- ▶ **Naming Methods**
  - Naming Functions**
- ▶ **Naming Properties and Data Types**
- ▶ **Acceptable Abbreviations and Acronyms**
- ▶ **Tips and Techniques for Framework Developers**
- Revision History**

# Introduction to Coding Guidelines for Cocoa

Developing a [Cocoa framework](#), plug-in, or other executable with a public API requires some a that are different from those used in application development. The primary clients of your pro is important that they are not mystified by your programmatic interface. This is where API nar handy, for they help you to make your interfaces consistent and clear. There are also programming special to—or of greater importance with—frameworks, such as versioning, binary compatibility, error-handling, and [memory management](#). This topic includes information on both Cocoa naming conventions and recommended programming practices for frameworks.

## Organization of This Document

The articles contained in this topic fall into two general types. The first and larger group presents naming [conventions](#) for programmatic interfaces. These are the same conventions (with some minor exceptions) that Apple uses for its own Cocoa frameworks. These articles on naming conventions include the following:

- [Code Naming Basics](#)
- [Naming Methods](#)
- [Naming Functions](#)
- [Naming Properties and Data Types](#)
- [Acceptable Abbreviations and Acronyms](#)

The second group (currently with a membership of one) discusses aspects of framework programming:

- [Tips and Techniques for Framework Developers](#)

[Next](#)

For  
Objective-C



What is “Swifty?”



# API Design Guidelines

SE-0023



# Principles

Clarity at the point of use

Clarity is more important than brevity

Concise code is a consequence of using contextual cues



# Clarity at the Point of Use

Design APIs to make uses clear and concise

Uses of APIs always have surrounding context



# Clarity at the Point of Use

Design APIs to make uses clear and concise

Uses of APIs always have surrounding context

```
if let completedPosition = tasks.index(of: completed) {  
    tasks.remove(at: completedPosition)  
}
```

# Clarity at the Point of Use

Design APIs to make uses clear and concise

Uses of APIs always have surrounding context

```
if let c = a.index(of: b) {  
    a.remove(at: c)  
}
```





# Clarity at the Point of Use

Design APIs to make uses clear and concise

Uses of APIs always have surrounding context

Don't optimize for bad code

```
if let c = a.index(of: b) {  
    a.remove(at: c)  
}
```



# Clarity at the Point of Use

Design APIs to make uses clear and concise

Uses of APIs always have surrounding context

Don't optimize for bad code

```
if let c = a.index(of: b) {  
    a.remove(at: c)  
}
```





# Clarity at the Point of Use

Design APIs to make uses clear and concise

Uses of APIs always have surrounding context

Don't optimize for bad code

```
if let completedPosition = tasks.index(of: completed) {  
    tasks.remove(at: completedPosition)  
}
```

# Strive for Clear Usage

`removeItem`



# Strive for Clear Usage

```
friends.removeItem(ted)
```

# Strive for Clear Usage

```
friends.removeItem(ted)
```



# Strive for Clear Usage

```
friends.removeItem(ted)
```

```
friends.removeObject(ted)
```

# Strive for Clear Usage

```
friends.removeItem(ted)
```

```
friends.removeObject(ted)
```

```
friends.removeElement(ted)
```

# Strive for Clear Usage

```
friends.removeItem(ted)
```

```
friends.removeObject(ted)
```

```
friends.removeElement(ted)
```

```
organicCompounds.removeElement(caffeine)
```



# Strive for Clear Usage

```
friends.removePerson(ted)
```

# Strive for Clear Usage

```
friends.removePerson(ted)  
primes.removeNumber(currentMultiple)  
activeViews.removeView(closedView)
```

# Omit Needless Words

```
friends.remove(ted)
```



# Omit Needless Words

```
friends.remove(ted)
```



# Omit Needless Words

```
friends.remove(ted)
```



# Omit Needless Words

```
friends.remove(caffeine)
```



Error: cannot convert value of type 'Compound' to expected argument type 'Person'



# Clarity Is More Important Than Brevity

Brevity itself is not a worthwhile goal

# Clarity Is More Important Than Brevity

Brevity itself is not a worthwhile goal

Concise code is a consequence of using contextual cues

# Clarity Is More Important Than Brevity

Brevity itself is not a worthwhile goal

Concise code is a consequence of using contextual cues

```
func glanceList(list: [Ingredient]) -> [[String: AnyObject]] {
    if list.isEmpty { return [] }
    let end: [Ingredient].Index = list.index(list.startIndex, offsetBy: 3,
                                             limitedBy: list.endIndex)!
    let shortList: ArraySlice<Ingredient> = list[0..
```

# Clarity Is More Important Than Brevity

Brevity itself is not a worthwhile goal

Concise code is a consequence of using contextual cues

```
func glanceList(list: [Ingredient]) -> [[String: AnyObject]] {
    if list.isEmpty { return [] }
    let end: [Ingredient].Index = list.index(list.startIndex, offsetBy: 3,
                                             limitedBy: list.endIndex)!
    let shortList: ArraySlice<Ingredient> = list[0..
```



# Clarity Is More Important Than Brevity

Brevity itself is not a worthwhile goal

Concise code is a consequence of using contextual cues

```
func glanceList(list: [Ingredient]) -> [[String: AnyObject]] {  
    if list.isEmpty { return [] }  
    let end = list.index(list.startIndex, offsetBy: 3,  
                        limitedBy: list endIndex)!  
    let shortList = list[0..  
end]  
    let serializedList = shortList.map {  
        ingredient in return ingredient.asDictionary  
    }  
  
    return serializedList  
}
```

# Clarity Is More Important Than Brevity

Brevity itself is not a worthwhile goal

Concise code is a side effect of contextual cues

```
func glanceList(list: [Ingredient]) -> [[String: AnyObject]] {
    if list.isEmpty { return [] }
    let end = list.index(list.startIndex, offsetBy: 3,
                        limitedBy: list endIndex)!
    let shortList = list[0..
```

# Clarity Is More Important Than Brevity

Brevity itself is not a worthwhile goal

Concise code is a side effect of contextual cues

```
func glanceList(list: [Ingredient]) -> [[String: AnyObject]] {  
    if list.isEmpty { return [] }  
    let end = list.index(list.startIndex, offsetBy: 3,  
                        limitedBy: list endIndex)!  
    let shortList = list[0..  
    let serializedList = shortList.map {  
        let serializedList: [[String: AnyObject]]  
  
        return serializedList  
    }  
}
```

# Which Words Are Needed?

Write out a use case

```
mainView.addChild(sideBar, atPoint: origin)
```

Does each word contribute to understanding?

# Which Words Are Needed?

Write out a use case

```
mainView.addChild(sideBar, atPoint: origin)
```

Does each word contribute to understanding?

- Clarify parameter *role*



# Which Words Are Needed?

Write out a use case

```
mainView.addChild(sideBar, atPoint: origin)
```

Does each word contribute to understanding?

- Clarify parameter *role*
- Don't restate type information

# Which Words Are Needed?

Write out a use case

```
mainView.addChild(sideBar, at: origin)
```

Does each word contribute to understanding?

- Clarify parameter *role*
- Don't restate type information

# Make Uses of Your APIs Read Grammatically

```
friends.remove(ted)
```

# Make Uses of Your APIs Read Grammatically

```
friends.remove(ted)
```

# Make Uses of Your APIs Read Grammatically

```
friends.remove(positionOfFormerFriend)
```





# Make Uses of Your APIs Read Grammatically

```
friends.remove(at: positionOfFormerFriend)
```



# Compound Names

Different APIs can be distinguished by argument label alone

```
friends.remove(ted)           // remove(_:)
friends.remove(at: positionOfTed) // remove(at:)
```

# Compound Names

Different APIs can be distinguished by argument label alone

```
friends.remove(ted) // remove(_:)  
friends.remove(at: positionOfTed) // remove(at:)
```

Two APIs should share a *compound name* if they have the same semantics

```
text.append(aCharacter) // append(_:)  
text.append(aString)
```

# First Argument Should Read Grammatically

If the first argument is part of a prepositional phrase, give it a label

```
truck.removeBoxes(withLabel: "WWDC 2016")
```

# First Argument Should Read Grammatically

If the first argument is part of a prepositional phrase, give it a label

```
truck.removeBoxes(withLabel: "WWDC 2016")
```

If the first argument is *not* part of a grammatical phrase, give it a label

```
viewController.dismiss(true)
```





# First Argument Should Read Grammatically

If the first argument is part of a prepositional phrase, give it a label

```
truck.removeBoxes(withLabel: "WWDC 2016")
```

If the first argument is *not* part of a grammatical phrase, give it a label

```
viewController.dismiss(animated: true)
```



# First Argument Should Read Grammatically

If the first argument is part of a prepositional phrase, give it a label

```
truck.removeBoxes(withLabel: "WWDC 2016")
```

If the first argument is *not* part of a grammatical phrase, give it a label

```
viewController.dismiss(animated: true)
```

# First Argument Should Read Grammatically

If the first argument is part of a prepositional phrase, give it a label

```
truck.removeBoxes(withLabel: "WWDC 2016")
```

If the first argument is *not* part of a grammatical phrase, give it a label

```
viewController.dismiss(animated: true)
```

Otherwise, don't use a first argument label

```
friends.insert(michael, at: friends.startIndex)
```

# Name Methods Based on Their Side Effects

Use a verb to describe the side effect

```
friends.reverse()
```

```
viewController.present(animated: true)
```

```
organicCompounds.append(caffeine)
```

# Name Methods Based on Their Side Effects

Use a verb to describe the side effect

```
friends.reverse()
```

```
viewController.present(animated: true)
```

```
organicCompounds.append(caffeine)
```

Use a noun to describe the result

```
button.backgroundColor(for: .disabled)
```

```
friends.suffix(3)
```

# Mutating/Non-Mutating Pairs

The “ed/ing” rule



# Mutating/Non-Mutating Pairs

## The “ed/ing” rule

“ed” rule

```
x.reverse() // mutating  
let y = x.reversed() // non-mutating
```

# Mutating/Non-Mutating Pairs

## The “ed/ing” rule


### “ed” rule

```
x.reverse() // mutating  
let y = x.reversed() // non-mutating
```

### “ing” rule

```
documentDirectory.appendPathComponent(".list") // mutating  
let documentFile = documentDirectory.appendingPathComponent(".list") // non-mutating
```

# API Design Guidelines

 To facilitate use as a quick reference, the details of many guidelines can be expanded individually. Details are never hidden when this page is printed. [Expand all details now](#)

## Table of Contents

- [Fundamentals](#)
- [Naming](#)
  - [Promote Clear Usage](#)
  - [Strive for Fluent Usage](#)
  - [Use Terminology Well](#)
- [Conventions](#)
  - [General Conventions](#)
  - [Parameters](#)
  - [Argument Labels](#)
- [Special Instructions](#)

# The Grand Renaming

**grand** | \ˈɡrænd\  
*adjective*

1. large or ambitious in scope or scale

# The Grand Renaming

Bring APIs into adherence with the Swift API Design Guidelines *in Swift*

Impacts a huge number of APIs

- The Swift standard library
- All Cocoa and Cocoa Touch frameworks
- Core Graphics and Grand Central Dispatch get “Swifty” makeover

SE-0005

SE-0006

SE-0086

SE-0088



Review changes:

File Name	Line	Code (Before Conversion)	Code (After Conversion)
AllListItemsPresenter.swift	96	<code>init(totalItemCount: Int, completeItemCount: Int) {</code>	<code>init(totalItemCount: Int, completeItemCount: Int) {</code>
AppConfiguration.swift	97	<code>self.totalItemCount = totalItemCount</code>	<code>self.totalItemCount = totalItemCount</code>
AppDelegate.swift	98	<code>self.completeItemCount = completeItemCount</code>	<code>self.completeItemCount = completeItemCount</code>
AppLaunchContext.swift	99	<code>incompleteItemCount = totalItemCount - completeItemCount</code>	<code>incompleteItemCount = totalItemCount - completeItemCount</code>
AppLaunchContextTests.swift	100	<code>percentage = totalItemCount &gt; 0 ? Double(completeItemCount) /</code>	<code>percentage = totalItemCount &gt; 0 ? Double(completeItemCount) /</code>
CheckBox.swift	101	<code>Double(totalItemCount) : 0</code>	<code>Double(totalItemCount) : 0</code>
CheckBoxLayer.swift	102	<code>}</code>	<code>}</code>
CloudListCoordinator.swift	103	<code>// MARK: Drawing</code>	<code>// MARK: Drawing</code>
ColoredTextViewController.swift	104	<code>/// Draw the text containing the number of complete items.</code>	<code>/// Draw the text containing the number of complete items.</code>
DirectoryMonitor.swift	105	<code>func drawCompleteItemsCountInCurrentContext() {</code>	<code>func drawCompleteItemsCountInCurrentContext() {</code>
GlanceBadge.swift	106	<code>let center = CGPoint(x: groupBackgroundImageSize.width / 2.0,</code>	<code>let center = CGPoint(x: groupBackgroundImageSize.width / 2.0,</code>
GlanceInterfaceViewController.swift	107	<code>y: groupBackgroundImageSize.height / 2.0)</code>	<code>y: groupBackgroundImageSize.height / 2.0)</code>
IncompleteListItemsPresenter.swift	108	<code>let itemsCompleteText = "\(completeItemCount)"</code>	<code>let itemsCompleteText = "\(completeItemCount)"</code>
List.swift	109	<code>let completeAttributes = [</code>	<code>let completeAttributes = [</code>
ListCell.swift	110	<code>NSFontAttributeName: UIFont.systemFont(ofSize: 36),</code>	<code>NSFontAttributeName: UIFont.systemFont(ofSize: 36),</code>
ListColorCell.swift	111	<code>NSForegroundColorAttributeName: completeTextPathColor</code>	<code>NSForegroundColorAttributeName: completeTextPathColor</code>
ListColorUI.swift	112	<code>]</code>	<code>]</code>
ListCoordinator.swift	113	<code>let completeSize = itemsCompleteText.sizeWithAttributes</code>	<code>let completeSize = itemsCompleteText.size(attributes:</code>
ListDocument.swift	114	<code>(completeAttributes)</code>	<code>completeAttributes)</code>
ListDocumentViewController.swift	115	<code>// Build and gather information about the done string.</code>	<code>// Build and gather information about the done string.</code>
ListInfo.swift	116	<code>let doneText = NSLocalizedString("Done", comment: "")</code>	<code>let doneText = NSLocalizedString("Done", comment: "")</code>
ListInterfaceViewController.swift	117	<code>let doneAttributes = [</code>	<code>let doneAttributes = [</code>
ListItem.swift	118	<code>NSFontAttributeName: UIFont.systemFont(ofSize: 16),</code>	<code>NSFontAttributeName: UIFont.systemFont(ofSize: 16),</code>
ListItemCell.swift	119	<code>NSForegroundColorAttributeName: UIColor.darkGrayColor()</code>	<code>NSForegroundColorAttributeName: UIColor.darkGrayColor()</code>
ListItemRowController.swift	120	<code>]</code>	<code>]</code>
ListPresenterAlgorithms.swift	121	<code>let doneSize = doneText.sizeWithAttributes(doneAttributes)</code>	<code>let doneSize = doneText.size(attributes: doneAttributes)</code>
ListPresenterDelegate.swift	122	<code>let completeRect = CGRect(x: center.x - 0.5 * completeSize.</code>	<code>let completeRect = CGRect(x: center.x - 0.5 * completeSize.</code>
ListPresenterType.swift	123	<code>width, y: center.y - 0.5 * completeSize.height - 0.5 *</code>	<code>width, y: center.y - 0.5 * completeSize.height - 0.5 *</code>
ListPresenterUtilities.swift	124	<code>doneSize.height, width: completeSize.width, height:</code>	<code>doneSize.height, width: completeSize.width, height:</code>
ListsController.swift	125	<code>completeSize.height)</code>	<code>completeSize.height)</code>
ListsInterfaceViewController.swift	126	<code>let doneRect = CGRect(x: center.x - 0.5 * doneSize.width, y:</code>	<code>let doneRect = CGRect(x: center.x - 0.5 * doneSize.width, y:</code>
ListUtilities.swift	127	<code>center.y + 0.125 * doneSize.height, width: doneSize.width,</code>	<code>center.y + 0.125 * doneSize.height, width: doneSize.width,</code>
ListViewController.swift	128	<code>height: doneSize.height)</code>	<code>height: doneSize.height)</code>
LocalListCoordinator.swift	129	<code>itemsCompleteText.drawInRect(CGRectIntegral(completeRect),</code>	<code>itemsCompleteText.draw(in: completeRect.integral,</code>
	130	<code>withAttributes: completeAttributes)</code>	<code>withAttributes: completeAttributes)</code>
	131	<code>doneText.drawInRect(CGRectIntegral(doneRect), withAttributes:</code>	<code>doneText.draw(in: doneRect.integral, withAttributes:</code>
	132	<code>doneAttributes)</code>	<code>doneAttributes)</code>
	133	<code>}</code>	<code>}</code>
	134	<code>}</code>	<code>}</code>
	135	<code>}</code>	<code>}</code>
	136	<code>}</code>	<code>}</code>

GlanceBadge.swift (Before Conversion)

GlanceBadge.swift (After Conversion)

Cancel

Previous

Save



One API, Two Names

# Objective-C Names in Swift

```
extension MyController {  
    func handleDrag(sender: UIControl, for event: UIEvent) { }  
  
    func setupForDrag() {  
        control.addTarget(self, action: Selector("???"),  
                           for: [.touchDragInside, .touchDragOutside])  
    }  
}
```



# Objective-C Names in Swift

```
extension MyController {  
    func handleDrag(sender: UIControl, for event: UIEvent) { }  
  
    func setupForDrag() {  
        control.addTarget(self, action: Selector("???"),  
                           for: [.touchDragInside, .touchDragOutside])  
    }  
}
```

# Objective-C Names in Swift

```
extension MyController {  
    func handleDrag(sender: UIControl, for event: UIEvent) { }  
  
    func setupForDrag() {  
        control.addTarget(self, action: Selector("handleDragWithSender:for:"),  
                           for: [.touchDragInside, .touchDragOutside])  
    }  
}
```

# Objective-C Names in Swift

```
extension MyController {  
    func handleDrag(sender: UIControl, for event: UIEvent) { }  
  
    func setupForDrag() {  
        control.addTarget(self, action: Selector("handleDragWithSender:for:"),  
                           for: [.touchDragInside, .touchDragOutside])  
    }  
}
```



# Use #selector for Objective-C Selectors

```
extension MyController {  
    func handleDrag(sender: UIControl, for event: UIEvent) { }  
  
    func setupForDrag() {  
        control.addTarget(self, action: #selector(handleDrag(sender:for:)),  
                           for: [.touchDragInside, .touchDragOutside])  
    }  
}
```

# Use #selector for Objective-C Selectors

```
extension MyController {  
    func handleDrag(sender: UIControl, for event: UIEvent) { }  
  
    func setupForDrag() {  
        control.addTarget(self, action: #selector(handleDrag(sender:for:)),  
                           for: [.touchDragInside, .touchDragOutside])  
    }  
}
```



# #selector for Property Getters/Setters

NEW

```
class Artist : NSObject {
    var name: String

    func releaseAlbum(_ album: Album) {
        switch album {
            case .TheGoldExperience:
                self.perform(#selector(setter: Artist.name), with: "🎵", afterDelay: 60.0)
                // ...
            }
        }
    }
}
```

SE-0064



# Key Paths

```
class Artist : NSObject {
    dynamic var name: String
}

class Album : NSObject {
    dynamic var artist: Artist
}

class MyController : NSObject {
    func monitorNameChanges(album: Album) {
        album.addObserver(self, forKeyPath: "artist.name",
                          options: .new, context: &artistNameContext)
    }
}
```

# Key Paths

```
class Artist : NSObject {  
    dynamic var name: String  
}  
  
class Album : NSObject {  
    dynamic var artist: Artist  
}  
  
class MyController : NSObject {  
    func monitorNameChanges(album: Album) {  
        album.addObserver(self, forKeyPath: "artist.name",  
                           options: .new, context: &artistNameContext)  
    }  
}
```



# Key Paths via #keyPath

NEW

```
class Artist : NSObject {
    dynamic var name: String
}

class Album : NSObject {
    dynamic var artist: Artist
}

class MyController : NSObject {
    func monitorNameChanges(album: Album) {
        album.addObserver(self, forKeyPath: #keyPath(Album.artist.name),
                          options: .new, context: &artistNameContext)
    }
}
```

SE-0062

# Key Paths via #keyPath

NEW

```
class Artist : NSObject {
    dynamic var name: String
}

class Album : NSObject {
    dynamic var artist: Artist
}

class MyController : NSObject {
    func monitorNameChanges(album: Album) {
        album.addObserver(self, forKeyPath: #keyPath(Album.artist.name),
                           options: .new, context: &artistNameContext)
    }
}
```



SE-0062

# Controlling Objective-C Names

The Objective-C name of a Swift entity sometimes matters

- Objective-C code in mix-and-match projects
- External tools

```
extension MyController {  
  
    func handleDrag(sender: UIControl, for event: UIEvent) { } // handleDrag(sender:for:)  
}
```

```
// Generated Objective-C  
@interface MyController ()  
- (void)handleDragWithSender:(UIControl *)sender for:(UIEvent *)event;  
@end
```

# Controlling Objective-C Names

The Objective-C name of a Swift entity sometimes matters

- Objective-C code in mix-and-match projects
- External tools

```
extension MyController {  
  
    func handleDrag(sender: UIControl, for event: UIEvent) { } // handleDrag(sender:for:)  
}
```

```
// Generated Objective-C  
@interface MyController ()  
- (void)handleDragWithSender:(UIControl *)sender for:(UIEvent *)event;  
@end
```



# Controlling Objective-C Names

The Objective-C name of a Swift entity sometimes matters

- Objective-C code in mix-and-match projects
- External tools

```
extension MyController {  
    @objc(handleDrag:forEvent:)  
    func handleDrag(sender: UIControl, for event: UIEvent) { } // handleDrag(sender:for:)  
}
```

```
// Generated Objective-C  
@interface MyController ()  
- (void)handleDragWithSender:(UIControl *)sender for:(UIEvent *)event;  
@end
```

# Controlling Objective-C Names

The Objective-C name of a Swift entity sometimes matters

- Objective-C code in mix-and-match projects
- External tools

```
extension MyController {  
    @objc(handleDrag:forEvent:)  
    func handleDrag(sender: UIControl, for event: UIEvent) { } // handleDrag(sender:for:)  
}
```

```
// Generated Objective-C  
@interface MyController ()  
- (void)handleDrag:(UIControl *)sender forKey:(UIEvent *)event;  
@end
```

# Tool Support for the Grand Renaming

I get by with a little help from my tools

Swift abstracts away the need to reason about Objective-C names

# Tool Support for the Grand Renaming

I get by with a little help from my tools

Swift abstracts away the need to reason about Objective-C names

Swift 3 migrator migrates Swift 2.x code to Swift 3 names



# Tool Support for the Grand Renaming

I get by with a little help from my tools

Swift abstracts away the need to reason about Objective-C names

Swift 3 migrator migrates Swift 2.x code to Swift 3 names

Swift 3 compiler helps with renaming-related problems

The screenshot shows the Xcode IDE with a Swift file open. The left sidebar displays a list of issues for 'ListerKit (watchOS)'. The main editor shows Swift code with a warning on line 159. A fix-it menu is open, offering three options to resolve the warning.

```
150     restorationHandler([ListDocumentsViewController])
151     }
152
153     return true
154 }
155
156     return false
157 }
158
159 func application(_ application: UIApplication, open url: URL, sourceApplication: String?, annotation: AnyObject) -> Bool {
    // Make sure that URL opening is handled after the app sandbox is extended. See `application(_, willFinishLaunchingWithOptions:)` above.
    appDelegateQueue.async {
        listDocumentsViewController.configureViewControllerWithLaunchContext(launchContext)
    }
    return true
}
```

**Issues:**

- ListerKit (watchOS) 1 issue
  - Swift Compiler Warning
    - Result of call to 'unsafeToggleListItem' is unused (AllListItemsPresenter.swift)
- Lister 4 issues
  - Swift Compiler Warning
    - Instance method 'application(application:openURL:sourceApplication:annotation:)' nearly matches optional requirement 'application(\_:open:sourceApplication:annotation:)' of protocol 'UIApplicationDelegate'

**Fix-it options:**

- Rename to 'application(\_:open:sourceApplication:annotation:)' to satisfy this requirement
- Make 'application(application:openURL:sourceApplication:annotation:)' private to silence this warning
- Add '@nonobjc' to silence this warning

# Objective-C APIs and Swift

Michael Iseman  
Swift Engineer



# Swift 2

```
extension UIDocument {  
    func saveToURL(_ url: NSURL, forSaveOperation saveOperation: UIDocumentSaveOperation,  
                  completionHandler: ((Bool) -> Void)?)  
    func revertToContentsOfURL(_ url: NSURL, completionHandler: ((Bool) -> Void)?)  
}
```

# Automatic Translation of Objective-C APIs

NEW

```
extension UIDocument {  
    func saveToURL(_ url: NSURL, forSaveOperation saveOperation: UIDocumentSaveOperation,  
                  completionHandler: ((Bool) -> Void)?)  
    func revertToContentsOfURL(_ url: NSURL, completionHandler: ((Bool) -> Void)?)  
}
```

SE-0005

SE-0069

# Automatic Translation of Objective-C APIs

NEW

```
extension UIDocument {  
    func saveToURL(_ url: NSURL, forSaveOperation saveOperation: UIDocumentSaveOperation,  
                  completionHandler: ((Bool) -> Void)?)  
    func revertToContentsOfURL(_ url: NSURL, completionHandler: ((Bool) -> Void)?)  
}
```

Identify first argument labels

SE-0005

SE-0069

# Automatic Translation of Objective-C APIs

NEW

```
extension UIDocument {  
    func save(toURL url: NSURL, forSaveOperation saveOperation: UIDocumentSaveOperation,  
             completionHandler: ((Bool) -> Void)?)  
    func revert(toContentsOfURL url: NSURL, completionHandler: ((Bool) -> Void)?)  
}
```

Identify first argument labels

SE-0005

SE-0069

# Automatic Translation of Objective-C APIs

NEW

```
extension UIDocument {  
    func save(toURL url: NSURL, forSaveOperation saveOperation: UIDocumentSaveOperation,  
            completionHandler: ((Bool) -> Void)?)  
    func revert(toContentsOfURL url: NSURL, completionHandler: ((Bool) -> Void)?)  
}
```

Identify first argument labels

Remove words that restate type information

SE-0005

SE-0069

# Automatic Translation of Objective-C APIs

NEW

```
extension UIDocument {  
    func save(to url: NSURL, for saveOperation: UIDocumentSaveOperation,  
             completionHandler: ((Bool) -> Void)?)  
    func revert(toContentsOf url: NSURL, completionHandler: ((Bool) -> Void)?)  
}
```

Identify first argument labels

Remove words that restate type information

SE-0005

SE-0069



# Automatic Translation of Objective-C APIs

NEW

```
extension UIDocument {  
    func save(to url: NSURL, for saveOperation: UIDocumentSaveOperation,  
             completionHandler: ((Bool) -> Void)?  
    func revert(toContentsOf url: NSURL, completionHandler: ((Bool) -> Void)?  
}
```

Identify first argument labels

Remove words that restate type information

SE-0005

SE-0069

# Automatic Translation of Objective-C APIs

NEW

```
extension UIDocument {  
    func save(to url: NSURL, for saveOperation: UIDocumentSaveOperation,  
             completionHandler: ((Bool) -> Void)? = nil)  
    func revert(toContentsOf url: NSURL, completionHandler: ((Bool) -> Void)? = nil)  
}
```

Identify first argument labels

Remove words that restate type information

SE-0005

SE-0069

# Automatic Translation of Objective-C APIs

NEW

```
extension UIDocument {  
    func save(to url: NSURL, for saveOperation: UIDocumentSaveOperation,  
             completionHandler: ((Bool) -> Void)? = nil)  
    func revert(toContentsOf url: NSURL, completionHandler: ((Bool) -> Void)? = nil)  
}
```

Identify first argument labels

Remove words that restate type information

Introduce default arguments

SE-0005

SE-0069

# Automatic Translation of Objective-C APIs

NEW

```
extension UIDocument {  
    func save(to url: NSURL, for saveOperation: UIDocumentSaveOperation,  
             completionHandler: ((Bool) -> Void)? = nil)  
    func revert(toContentsOf url: NSURL, completionHandler: ((Bool) -> Void)? = nil)  
}
```

Identify first argument labels

Remove words that restate type information

Introduce default arguments

SE-0005

SE-0069

# Automatic Translation of Objective-C APIs

NEW

```
extension UIDocument {  
    func save(to url: URL, for saveOperation: UIDocumentSaveOperation,  
             completionHandler: ((Bool) -> Void)? = nil)  
    func revert(toContentsOf url: URL, completionHandler: ((Bool) -> Void)? = nil)  
}
```

Identify first argument labels

Remove words that restate type information

Introduce default arguments

Use bridged value types

SE-0005

SE-0069

# Automatic Translation of Objective-C APIs

```
extension UIDocument {  
    func save(to url: URL, for saveOperation: UIDocumentSaveOperation,  
             completionHandler: ((Bool) -> Void)? = nil)  
    func revert(toContentsOf url: URL, completionHandler: ((Bool) -> Void)? = nil)  
}
```

Identify first argument labels

Remove words that restate type information

Introduce default arguments

Use bridged value types



# Choosing Your Own Names

## Objective-C

- (`NSLayoutConstraint *`)constraintEqualToAnchor:  
    (`NSLayoutAnchor<AnchorType> *`)anchor `NS_SWIFT_NAME(constraint(equalTo:))`;
- (`NSLayoutConstraint *`)constraintGreaterThanOrEqualToAnchor:  
    (`NSLayoutAnchor<AnchorType> *`)anchor `NS_SWIFT_NAME(constraint(greaterThanOrEqualTo:))`;

## Generated Swift Interface

```
func constraint(equalTo: NSLayoutAnchor) -> NSLayoutConstraint  
func constraint(greaterThanOrEqualTo: NSLayoutAnchor) -> NSLayoutConstraint
```

# Types

Names don't go far enough

```
let cal = NSCalendar(calendarIdentifier: NSCalendarIdentifierGregorian)
```

# Types

Names don't go far enough

```
let cal = NSCalendar(calendarIdentifier: "gregorian")
```



# Types

Names don't go far enough

Objective-C

```
extern NSString * NSCalendarIdentifierGregorian;
```

Generated Swift Interface

```
let NSCalendarIdentifierGregorian: String
```

# Types

Names don't go far enough

Objective-C

```
typedef NSString * NSCalendarIdentifier;  
NSCalendarIdentifier NSCalendarIdentifierGregorian;
```

Generated Swift Interface

```
 typealias NSCalendarIdentifier = NSString  
 let NSCalendarIdentifierGregorian: NSCalendarIdentifier
```

# Put a Struct on It

NEW

```
typedef NSString * NSCalendarIdentifier NS_EXTENSIBLE_STRING_ENUM;
```

SE-0033

# Stringly Typed

## Objective-C

```
typedef NSString * NSCalendarIdentifier NS_EXTENSIBLE_STRING_ENUM;  
NSCalendarIdentifier NSCalendarIdentifierGregorian;
```

## Generated Swift Interface

```
 typealias NSCalendarIdentifier = NSString  
 let NSCalendarIdentifierGregorian: NSCalendarIdentifier
```



# Stringly Typed

## Objective-C

```
typedef NSString * NSCalendarIdentifier NS_EXTENSIBLE_STRING_ENUM;  
NSCalendarIdentifier NSCalendarIdentifierGregorian;
```

## Generated Swift Interface

```
 typealias NSCalendarIdentifier = NSString  
 let NSCalendarIdentifierGregorian: NSCalendarIdentifier
```

# Stringly Typed

## Objective-C

```
typedef NSString * NSCalendarIdentifier NS_EXTENSIBLE_STRING_ENUM;  
NSCalendarIdentifier NSCalendarIdentifierGregorian;
```

## Generated Swift Interface

```
 typealias NSCalendarIdentifier = NSString  
 let NSCalendarIdentifierGregorian: NSCalendarIdentifier
```

# Stringly Typed

## Objective-C

```
typedef NSString * NSCalendarIdentifier NS_EXTENSIBLE_STRING_ENUM;  
NSCalendarIdentifier NSCalendarIdentifierGregorian;
```

## Generated Swift Interface

```
struct NSCalendarIdentifier : RawRepresentable {  
  
    static let gregorian: NSCalendarIdentifier  
  
}
```

# ~~Stringly Typed~~ Strongly Typed

## Objective-C

```
typedef NSString * NSCalendarIdentifier NS_EXTENSIBLE_STRING_ENUM;  
NSCalendarIdentifier NSCalendarIdentifierGregorian;
```

## Generated Swift Interface

```
struct NSCalendarIdentifier : RawRepresentable {  
    init(_ rawValue: String);  
    var rawValue: String { get }  
    static let gregorian: NSCalendarIdentifier  
}
```

# Better Types=Clearer Use Site

```
let cal = NSCalendar(calendarIdentifier: NSCalendarIdentifierGregorian)
```

# Better Types=Clearer Use Site

```
let cal = NSCalendar(identifier: .gregorian)
```

# Better Types=Clearer Use Site

```
let cal = NSCalendar(identifier: .gregorian)
```



# Better Types=Clearer Use Site

```
let cal = Calendar(identifier: .gregorian)
```











C APIs

```
// Drawing with Core Graphics in Swift 2

// Get a transform that will rotate around a given offset
func rotationAround(offset: CGPoint, angle: CGFloat,
                   transform: CGAffineTransform = CGAffineTransformIdentity) -> CGAffineTransform {
    var result = CGAffineTransformTranslate(transform, offset.x, offset.y)
    result = CGAffineTransformRotate(result, angle)
    return CGAffineTransformTranslate(result, -offset.x, -offset.y)
}

// Trace a path in red
func trace(in context: CGContext, path: CGPath) {
    let red = CGColorCreateGenericRGB(1, 0, 0, 1)
    CGContextSaveGState(context)
    CGContextAddPath(context, path)
    CGContextSetStrokeColorWithColor(context, red)
    CGContextStrokePath(context)
    CGContextRestoreGState(context)
}
```

# Import as Member

Reconstructive surgery with NS\_SWIFT\_NAME

# Properties

C

```
CFStringRef kCGColorWhite;
```

Generated Swift Interface

```
let kCGColorWhite: CFString
```

Swift use

```
let color = kCGColorWhite
```



# Properties

C

```
CFStringRef kCGColorWhite NS_SWIFT_NAME(CGColor.white);
```

Generated Swift Interface

```
let kCGColorWhite: CFString
```

Swift use

```
let color = kCGColorWhite
```

# Properties

C

```
CFStringRef kCGColorWhite NS_SWIFT_NAME(CGColor.white);
```

Generated Swift Interface

```
let kCGColorWhite: CFString
```

Swift use

```
let color = kCGColorWhite
```

# Properties

C

```
CFStringRef kCGColorWhite NS_SWIFT_NAME(CGColor.white);
```

Generated Swift Interface

```
let kCGColorWhite: CFStringRef
```

Swift use

```
let color = kCGColorWhite
```

# Properties

C

```
CFStringRef kCGColorWhite NS_SWIFT_NAME(CGColor.white);
```

Generated Swift Interface

```
extension CGColor { static let white: CFString }
```

Swift use

```
let color = kCGColorWhite
```

# Properties

C

```
CFStringRef kCGColorWhite NS_SWIFT_NAME(CGColor.white);
```

Generated Swift Interface

```
extension CGColor { static let white: CFString }
```

Swift use

```
let color = kCGColorWhite
```

# Properties

C

```
CFStringRef kCGColorWhite NS_SWIFT_NAME(CGColor.white);
```

Generated Swift Interface

```
extension CGColor { static let white: CFString }
```

Swift use

```
let color = CGColor.white
```



# Properties

C

```
CFStringRef kCGColorWhite NS_SWIFT_NAME(CGColor.white);
```

Generated Swift Interface

```
extension CGColor { static let white: CFString }
```

Swift use

```
let color = CGColor.white
```

# Initializers

C

```
CGAffineTransform CGAffineTransformMakeTranslation(CGFloat tx, CGFloat ty)
    NS_SWIFT_NAME(CGAffineTransform.init(translationX:y:));
```

Generated Swift Interface

```
func CGAffineTransformMakeTranslation(_: CGFloat, _: CGFloat) -> CGAffineTransform
```

Swift use

```
let translate = CGAffineTransformMakeTranslation(1.0, 0.5)
```

# Initializers

C

```
CGAffineTransform CGAffineTransformMakeTranslation(CGFloat tx, CGFloat ty)
    NS_SWIFT_NAME(CGAffineTransform.init(translationX:y:));
```

Generated Swift Interface

```
func CGAffineTransformMakeTranslation(_: CGFloat, _: CGFloat) -> CGAffineTransform
```

Swift use

```
let translate = CGAffineTransformMakeTranslation(1.0, 0.5)
```

# Initializers

C

```
CGAffineTransform CGAffineTransformMakeTranslation(CGFloat tx, CGFloat ty)
    NS_SWIFT_NAME(CGAffineTransform.init(translationX:y:));
```

Generated Swift Interface

```
func CGAffineTransformMakeTranslation(_: CGFloat, _: CGFloat) -> CGAffineTransform
```

Swift use

```
let translate = CGAffineTransformMakeTranslation(1.0, 0.5)
```

# Initializers

C

```
CGAffineTransform CGAffineTransformMakeTranslation(CGFloat tx, CGFloat ty)
    NS_SWIFT_NAME(CGAffineTransform.init(translationX:y:));
```

Generated Swift Interface

```
extension CGAffineTransform { init(translationX: CGFloat, y: CGFloat) }
```

Swift use

```
let translate = CGAffineTransformMakeTranslation(1.0, 0.5)
```

# Initializers

C

```
CGAffineTransform CGAffineTransformMakeTranslation(CGFloat tx, CGFloat ty)
    NS_SWIFT_NAME(CGAffineTransform.init(translationX:y:));
```

Generated Swift Interface

```
extension CGAffineTransform { init(translationX: CGFloat, y: CGFloat) }
```

Swift use

```
let translate = CGAffineTransformMakeTranslation(1.0, 0.5)
```



# Initializers

C

```
CGAffineTransform CGAffineTransformMakeTranslation(CGFloat tx, CGFloat ty)  
    NS_SWIFT_NAME(CGAffineTransform.init(translationX:y:));
```

Generated Swift Interface

```
extension CGAffineTransform { init(translationX: CGFloat, y: CGFloat) }
```

Swift use

```
let translate = CGAffineTransform(translationX: 1.0, y: 0.5)
```

# Initializers

C

```
CGAffineTransform CGAffineTransformMakeTranslation(CGFloat tx, CGFloat ty)
    NS_SWIFT_NAME(CGAffineTransform.init(translationX:y:));
```

Generated Swift Interface

```
extension CGAffineTransform { init(translationX: CGFloat, y: CGFloat) }
```

Swift use

```
let translate = CGAffineTransform(translationX: 1.0, y: 0.5)
```

# Methods

C

```
void CGContextFillPath(CGContextRef) NS_SWIFT_NAME(CGContext.fillPath(self:));
```

Generated Swift Interface

```
func CGContextFillPath(_: CGContext)
```

Swift use

```
CGContextFillPath(context)
```

# Methods

C

```
void CGContextFillPath(CGContextRef) NS_SWIFT_NAME(CGContext.fillPath(self:));
```

Generated Swift Interface

```
func CGContextFillPath(_: CGContext)
```

Swift use

```
CGContextFillPath(context)
```

# Methods

C

```
void CGContextFillPath(CGContextRef) NS_SWIFT_NAME(CGContext.fillPath(self:));
```

Generated Swift Interface

```
func CGContextFillPath(_: CGContext)
```

Swift use

```
CGContextFillPath(context)
```

# Methods

C

```
void CGContextFillPath(CGContextRef) NS_SWIFT_NAME(CGContext.fillPath(self:));
```

Generated Swift Interface

```
extension CGContext { func fillPath() }
```

Swift use

```
CGContextFillPath(context)
```



# Methods

C

```
void CGContextFillPath(CGContextRef) NS_SWIFT_NAME(CGContext.fillPath(self:));
```

Generated Swift Interface

```
extension CGContext { func fillPath() }
```

Swift use

```
CGContextFillPath(context)
```

# Methods

C

```
void CGContextFillPath(CGContextRef) NS_SWIFT_NAME(CGContext.fillPath(self:));
```

Generated Swift Interface

```
extension CGContext { func fillPath() }
```

Swift use

```
context.fillPath()
```

# Methods

C

```
void CGContextFillPath(CGContextRef) NS_SWIFT_NAME(CGContext.fillPath(self:));
```

Generated Swift Interface

```
extension CGContext { func fillPath() }
```

Swift use

```
context.fillPath()
```

# Computed Properties

C

```
CFStringRef ArtistGetName(ArtistRef) NS_SWIFT_NAME(getter:Artist.name(self:));  
void ArtistSetName(ArtistRef, CFStringRef)  
    NS_SWIFT_NAME(setter:Artist.name(self:newValue:));
```

Generated Swift Interface

```
func ArtistGetName(_: Artist) -> CFString  
func ArtistSetName(_: Artist, _: CFString)
```

# Computed Properties

C

```
CFStringRef ArtistGetName(ArtistRef) NS_SWIFT_NAME(getter:Artist.name(self:));  
void ArtistSetName(ArtistRef, CFStringRef)  
    NS_SWIFT_NAME(setter:Artist.name(self:newValue:));
```

Generated Swift Interface

```
func ArtistGetName(_: Artist) -> CFString  
func ArtistSetName(_: Artist, _: CFString)
```

# Computed Properties

C

```
CFStringRef ArtistGetName(ArtistRef) NS_SWIFT_NAME(getter:Artist.name(self:));  
void ArtistSetName(ArtistRef, CFStringRef)  
    NS_SWIFT_NAME(setter:Artist.name(self:newValue:));
```

Generated Swift Interface

```
func ArtistGetName(_: Artist) -> CFString  
func ArtistSetName(_: Artist, _: CFString)
```



# Computed Properties

C

```
CFStringRef ArtistGetName(ArtistRef) NS_SWIFT_NAME(getter:Artist.name(self:));  
void ArtistSetName(ArtistRef, CFStringRef)  
    NS_SWIFT_NAME(setter:Artist.name(self:newValue:));
```

Generated Swift Interface

```
extension Artist { var name: CFString { get set } }
```

# Computed Properties

C

```
CFStringRef ArtistGetName(ArtistRef) NS_SWIFT_NAME(getter:Artist.name(self:));  
void ArtistSetName(ArtistRef, CFStringRef)  
    NS_SWIFT_NAME(setter:Artist.name(self:newValue:));
```

Generated Swift Interface

```
extension Artist { var name: CFString { get set } }
```

# Computed Properties

Swift use

```
let formerName = ArtistGetName(myArtist)  
ArtistSetName(myArtist, "F")
```

# Computed Properties

Swift use

```
let formerName = ArtistGetName(myArtist)  
ArtistSetName(myArtist, "F")
```

# Computed Properties

Swift use

```
let formerName = myArtist.name  
myArtist.name = "F"
```

# Computed Properties

Swift use

```
let formerName = myArtist.name  
myArtist.name = "F"
```



# Better Together

```
typedef NSString * NSCalendarIdentifier NS_EXTENSIBLE_STRING_ENUM  
    NS_SWIFT_NAME(Calendar.Identifier);
```

# Better Together

```
typedef NSString * NSCalendarIdentifier NS_EXTENSIBLE_STRING_ENUM
    NS_SWIFT_NAME(Calendar.Identifier);
```

## Generated Swift Interface

```
struct Calendar.Identifier : RawRepresentable {
    init(_ rawValue: String);
    var rawValue: String { get }
    static let gregorian: Calendar.Identifier
}
```

NEW

NEW

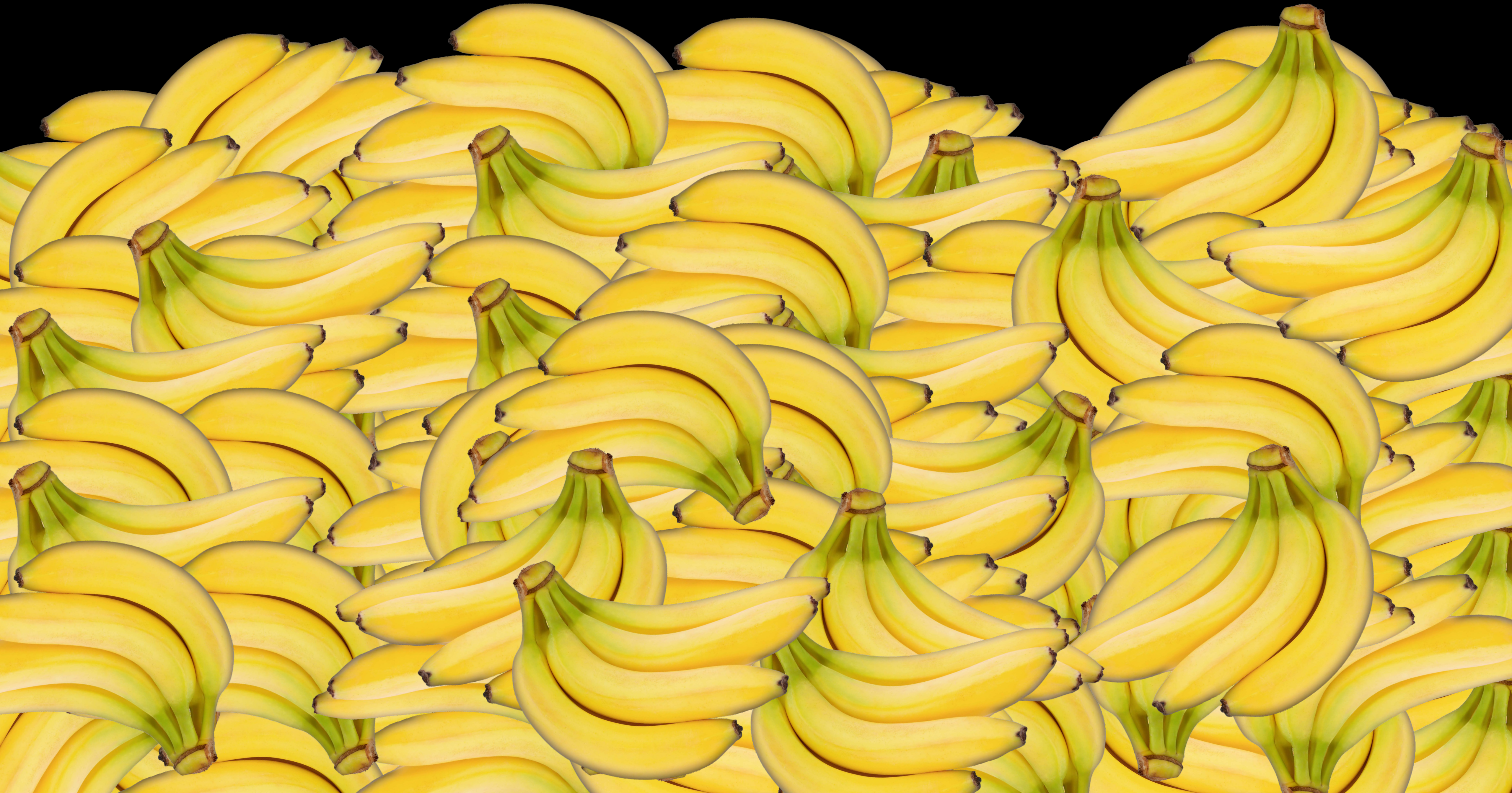


NEW





NEW





```
// Drawing with Core Graphics in Swift 2
```

```
// Get a transform that will rotate around a given offset
```

```
func rotationAround(offset: CGPoint, angle: CGFloat,  
    transform: CGAffineTransform = CGAffineTransformIdentity) -> CGAffineTransform {  
    var result = CGAffineTransformTranslate(transform, offset.x, offset.y)  
    result = CGAffineTransformRotate(result, angle)  
    return CGAffineTransformTranslate(result, -offset.x, -offset.y)  
}
```

```
// Trace a path in red
```

```
func trace(in context: CGContext, path: CGPath) {  
    let red = CGColorCreateGenericRGB(1, 0, 0, 1)  
    CGContextSaveGState(context)  
    CGContextAddPath(context, path)  
    CGContextSetStrokeColorWithColor(context, red)  
    CGContextStrokePath(context)  
    CGContextRestoreGState(context)  
}
```

```
// Drawing with Core Graphics in Swift 2
```

```
// Get a transform that will rotate around a given offset
```

```
func rotationAround(offset: CGPoint, angle: CGFloat,  
    transform: CGAffineTransform = CGAffineTransformIdentity) -> CGAffineTransform {  
    var result = CGAffineTransformTranslate(transform, offset.x, offset.y)  
    result = CGAffineTransformRotate(result, angle)  
    return CGAffineTransformTranslate(result, -offset.x, -offset.y)  
}
```

```
// Trace a path in red
```

```
func trace(in context: CGContext, path: CGPath) {  
    let red = CGColorCreateGenericRGB(1, 0, 0, 1)  
    CGContextSaveGState(context)  
    CGContextAddPath(context, path)  
    CGContextSetStrokeColorWithColor(context, red)  
    CGContextStrokePath(context)  
    CGContextRestoreGState(context)  
}
```

```
// Drawing with Core Graphics in Swift 2

// Get a transform that will rotate around a given offset
func rotationAround(offset: CGPoint, angle: CGFloat,
    transform: CGAffineTransform = .identity) -> CGAffineTransform {
    var result = CGAffineTransformTranslate(transform, offset.x, offset.y)
    result = CGAffineTransformRotate(result, angle)
    return CGAffineTransformTranslate(result, -offset.x, -offset.y)
}

// Trace a path in red
func trace(in context: CGContext, path: CGPath) {
    let red = CGColorCreateGenericRGB(1, 0, 0, 1)
    CGContextSaveGState(context)
    CGContextAddPath(context, path)
    CGContextSetStrokeColorWithColor(context, red)
    CGContextStrokePath(context)
    CGContextRestoreGState(context)
}
```

```
// Drawing with Core Graphics in Swift 2
```

```
// Get a transform that will rotate around a given offset
```

```
func rotationAround(offset: CGPoint, angle: CGFloat,  
    transform: CGAffineTransform = .identity) -> CGAffineTransform {  
    var result = CGAffineTransformTranslate(transform, offset.x, offset.y)  
    result = CGAffineTransformRotate(result, angle)  
    return CGAffineTransformTranslate(result, -offset.x, -offset.y)  
}
```

```
// Trace a path in red
```

```
func trace(in context: CGContext, path: CGPath) {  
    let red = CGColorCreateGenericRGB(1, 0, 0, 1)  
    CGContextSaveGState(context)  
    CGContextAddPath(context, path)  
    CGContextSetStrokeColorWithColor(context, red)  
    CGContextStrokePath(context)  
    CGContextRestoreGState(context)  
}
```

```
// Drawing with Core Graphics in Swift 2

// Get a transform that will rotate around a given offset
func rotationAround(offset: CGPoint, angle: CGFloat,
    transform: CGAffineTransform = .identity) -> CGAffineTransform {
    var result = transform.translateBy(x: offset.x, y: offset.y)
    result = result.rotate(angle)
    return result.translateBy(x: -offset.x, y: -offset.y)
}

// Trace a path in red
func trace(in context: CGContext, path: CGPath) {
    let red = CGColorCreateGenericRGB(1, 0, 0, 1)
    CGContextSaveGState(context)
    CGContextAddPath(context, path)
    CGContextSetStrokeColorWithColor(context, red)
    CGContextStrokePath(context)
    CGContextRestoreGState(context)
}
```

```
// Drawing with Core Graphics in Swift 2

// Get a transform that will rotate around a given offset
func rotationAround(offset: CGPoint, angle: CGFloat,
    transform: CGAffineTransform = .identity) -> CGAffineTransform {
    return transform.translateBy(x: offset.x, y: offset.y)
        .rotate(angle)
        .translateBy(x: -offset.x, y: -offset.y)
}

// Trace a path in red
func trace(in context: CGContext, path: CGPath) {
    let red = CGColorCreateGenericRGB(1, 0, 0, 1)
    CGContextSaveGState(context)
    CGContextAddPath(context, path)
    CGContextSetStrokeColorWithColor(context, red)
    CGContextStrokePath(context)
    CGContextRestoreGState(context)
}
```



```
// Drawing with Core Graphics in Swift 2

// Get a transform that will rotate around a given offset
func rotationAround(offset: CGPoint, angle: CGFloat,
    transform: CGAffineTransform = .identity) -> CGAffineTransform {
    return transform.translateBy(x: offset.x, y: offset.y)
        .rotate(angle)
        .translateBy(x: -offset.x, y: -offset.y)
}

// Trace a path in red
func trace(in context: CGContext, path: CGPath) {
    let red = CGColorCreateGenericRGB(1, 0, 0, 1)
    CGContextSaveGState(context)
    CGContextAddPath(context, path)
    CGContextSetStrokeColorWithColor(context, red)
    CGContextStrokePath(context)
    CGContextRestoreGState(context)
}
```

```
// Drawing with Core Graphics in Swift 2

// Get a transform that will rotate around a given offset
func rotationAround(offset: CGPoint, angle: CGFloat,
    transform: CGAffineTransform = .identity) -> CGAffineTransform {
    return transform.translateBy(x: offset.x, y: offset.y)
        .rotate(angle)
        .translateBy(x: -offset.x, y: -offset.y)
}

// Trace a path in red
func trace(in context: CGContext, path: CGPath) {
    let red = CGColor(red: 1, green: 0, blue: 0, alpha: 1)
    CGContextSaveGState(context)
    CGContextAddPath(context, path)
    CGContextSetStrokeColorWithColor(context, red)
    CGContextStrokePath(context)
    CGContextRestoreGState(context)
}
```

```
// Drawing with Core Graphics in Swift 2

// Get a transform that will rotate around a given offset
func rotationAround(offset: CGPoint, angle: CGFloat,
    transform: CGAffineTransform = .identity) -> CGAffineTransform {
    return transform.translateBy(x: offset.x, y: offset.y)
        .rotate(angle)
        .translateBy(x: -offset.x, y: -offset.y)
}

// Trace a path in red
func trace(in context: CGContext, path: CGPath) {
    let red = CGColor(red: 1, green: 0, blue: 0, alpha: 1)
    CGContextSaveGState(context)
    CGContextAddPath(context, path)
    CGContextSetStrokeColorWithColor(context, red)
    CGContextStrokePath(context)
    CGContextRestoreGState(context)
}
```

```
// Drawing with Core Graphics in Swift 3

// Get a transform that will rotate around a given offset
func rotationAround(offset: CGPoint, angle: CGFloat,
    transform: CGAffineTransform = .identity) -> CGAffineTransform {
    return transform.translateBy(x: offset.x, y: offset.y)
        .rotate(angle)
        .translateBy(x: -offset.x, y: -offset.y)
}

// Trace a path in red
func trace(in context: CGContext, path: CGPath) {
    let red = CGColor(red: 1, green: 0, blue: 0, alpha: 1)
    context.saveGState()
    context.addPath(path)
    context.setStrokeColor(red)
    context.strokePath()
    context.restoreGState()
}
```

# Summary

API design guidelines in Swift

Grand renaming

Crafting good Swift APIs from Objective-C

More Information

<https://developer.apple.com/wwdc16/403>



# Swift Evolution

<https://swift.org>

- 
- |         |  |
|---------|--|
| SE-0005 | Better Translation of Objective-C APIs Into Swift                                |
| SE-0006 | Apply API Guidelines to the Standard Library                                     |
| SE-0022 | Referencing the Objective-C selector of a method                                 |
| SE-0023 | API Design Guidelines  |
| SE-0033 | Import Objective-C Constants as Swift Types                                      |
| SE-0044 | Import as Member   |
| SE-0046 | Establish consistent label behavior across all parameters including first labels |
| SE-0062 | Referencing Objective-C key-paths  |
| SE-0064 | Referencing the Objective-C selector of property getters and setters             |
| SE-0086 | Drop NS Prefix in Swift Foundation   |
| SE-0088 | Modernize libdispatch for Swift 3 naming conventions                             |
-

# Related Sessions

---

What's New in Foundation for Swift

Mission

Tuesday 4:00PM

---

Concurrent Programming with GCD in Swift 3

Pacific Heights

Friday 4:00PM

---

# Labs

Swift Open Hours	Developer Tools Lab A	Tuesday 12:00PM
Swift Open Hours	Developer Tools Lab A	Tuesday 3:00PM
Swift Open Hours	Developer Tools Lab A	Wednesday 9:00AM
Swift Open Hours	Developer Tools Lab A	Wednesday 12:00PM
Swift Open Hours	Developer Tools Lab A	Wednesday 3:00PM
Swift Open Hours	Developer Tools Lab A	Thursday 9:00AM
Swift Open Hours	Developer Tools Lab A	Thursday 12:00PM
Swift Open Hours	Developer Tools Lab A	Thursday 3:00PM

# Labs

Swift Open Hours	Developer Tools Lab A	Wednesday 12:00PM
Swift Open Hours	Developer Tools Lab A	Wednesday 3:00PM
Swift Open Hours	Developer Tools Lab A	Thursday 9:00AM
Swift Open Hours	Developer Tools Lab A	Thursday 12:00PM
Swift Open Hours	Developer Tools Lab A	Thursday 3:00PM
Swift Open Hours	Developer Tools Lab A	Friday 9:00AM
Swift Open Hours	Developer Tools Lab A	Friday 12:00PM
Swift Open Hours	Developer Tools Lab A	Friday 3:00PM



W

W

D

C

1

6