

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)  
}
```

▼ 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**

[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 conventions that are different from those used in application development. The primary clients of your program are likely to be other programs, so 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 conventions 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.

For
Objective-C

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](#)

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..<end]
    let serializedList: [[String: AnyObject]] = shortList.map {
        (ingredient: Ingredient) -> [String: AnyObject] in return ingredient.asDictionary
    }

    return serializedList
}
```

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..    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(_)
```

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.backgroundTitle(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
```



[ABOUT SWIFT](#)

[BLOG](#)

[DOWNLOAD](#)

[GETTING STARTED](#)

[DOCUMENTATION](#)

[SOURCE CODE](#)

[COMMUNITY](#)

[CONTRIBUTING](#)

[CONTINUOUS
INTEGRATION](#)

[PROJECTS](#)

[COMPILER AND
STANDARD LIBRARY](#)

[PACKAGE MANAGER](#)

[CORE LIBRARIES](#)

[REPL AND DEBUGGER](#)

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 | \ˈgrand\br/>*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 Lister > Lister WatchKit App > Lister WatchKit Extension > Glance > GlanceBadge.swift

```

96  INIT(totalItemCount: Int, completeItemCount: Int) {
97    self.totalItemCount = totalItemCount
98
99    self.completeItemCount = completeItemCount
100
101   incompleteItemCount = totalItemCount - completeItemCount
102
103   percentage = totalItemCount > 0 ? Double(completeItemCount) /
104     Double(totalItemCount) : 0
105
106 // MARK: Drawing
107
108 // Draw the text containing the number of complete items.
109 func drawCompleteItemsCountInCurrentContext() {
110   let center = CGPoint(x: groupBackgroundImageSize.width / 2.0,
111                         y: groupBackgroundImageSize.height / 2.0)
112
113   let itemsCompleteText = "\(completeItemCount)"
114   let completeAttributes = [
115     NSFontAttributeName: UIFont.systemFontOfSize(36),
116     NSForegroundColorAttributeName: completeTextPathColor
117   ]
118   let completeSize = itemsCompleteText.sizeWithAttributes(
119     completeAttributes)
120
121   // Build and gather information about the done string.
122   let doneText = NSLocalizedString("Done", comment: "")
123   let doneAttributes = [
124     NSFontAttributeName: UIFont.systemFontOfSize(16),
125     NSForegroundColorAttributeName: UIColor.darkGrayColor()
126   ]
127   let doneSize = doneText.sizeWithAttributes(doneAttributes)
128
129   let completeRect = CGRect(x: center.x - 0.5 * completeSize.
130                             width, y: center.y - 0.5 * completeSize.height - 0.5 *
131                             doneSize.height, width: completeSize.width, height:
132                             completeSize.height)
133
134   let doneRect = CGRect(x: center.x - 0.5 * doneSize.width, y:
135                         center.y + 0.125 * doneSize.height, width: doneSize.width,
136                         height: doneSize.height)
137
138   itemsCompleteText.drawString(CGRectIntegral(completeRect),
139                               withAttributes: completeAttributes)
140
141   doneText.drawString(CGRectIntegral(doneRect), withAttributes:
142                      doneAttributes)
143
144 }
145
146 }

```

File GlanceBadge.swift (Before Conversion)

File 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])  
    }  
}
```



NEW

#selector for Property Getters/Setters

```
class Artist : NSObject {  
    var name: String  
  
    func releaseAlbum(_ album: Album) {  
        switch album {  
            case .TheGoldExperience:  
                self.perform(#selector(setter: Artist.name)), with: "FFFF", 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)  
    }  
}
```



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 forEvent:(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 interface with a project named "ListerKit (watchOS)". The left sidebar displays two groups of issues:

- ListerKit (watchOS) 1 issue**:
 - Swift Compiler Warning: Result of call to 'unsafeToggleListItem' is unused. Item is unused. (in file 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: Rename to 'application(_:open:sourceApplication:annotation:)' to satisfy this requirement.
 - Instance method 'application(application:openURL:sourceApplication:annotation:)' nearly matches optional requirement 'application(_:open:sourceApplication:annotation:)' of protocol 'UIApplicationDelegate'. Fix-it: Make 'application(application:openURL:sourceApplication:annotation:)' private to silence this warning.
 - Instance method 'application(application:openURL:sourceApplication:annotation:)' nearly matches optional requirement 'application(_:open:sourceApplication:annotation:)' of protocol 'UIApplicationDelegate'. Fix-it: Add '@nonobjc' to silence this warning.
 - Expression of type '[UIViewController]' is unused. (in file ListDocumentsViewController...)

The main editor area shows Swift code for a `ListDocumentsViewController` class. A specific warning is highlighted in yellow:

```
func application(_ application: UIApplication, open url: URL, sourceApplication: String?, annotation: AnyObject) -> Bool {
```

A tooltip for this warning states: "Instance method 'application(application:openURL:sourceApplication:annotation:)' nearly matches optional requirement 'application(_:open:sourceApplication:annotation:)' of protocol 'UIApplicationDelegate'". Below the code, three "Fix-it" suggestions are listed:

- 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

The code itself includes comments and other methods:

```
    } // Make sure that URL opening is handled after the app sandbox is extended. See `application(_:, willFinishLaunchingWithOptions:)` above.
    appDelegateQueue.async {
        listDocumentsViewController.configureViewControllerWithLaunchContext(launchContext)
    }

    return true
}
```

Objective-C APIs and Swift

Michael Ilseman
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: 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 = 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, "†")
```

Computed Properties

Swift use

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

Computed Properties

Swift use

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

Computed Properties

Swift use

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

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



A large pile of ripe yellow bananas on a black background.

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 16