Tools

#WWDC14

Swift Interoperability in Depth

Session 407

Doug Gregor
Engineer, Swift Compiler Team

Introduction

Swift is a new language for Cocoa

Seamless interoperability with Objective-C

Focus on language-level interoperability

Roadmap

Working with Cocoa

- How Objective-C APIs look and feel in Swift
- id and AnyObject

Bridging Core Cocoa Types

Subclassing Objective-C Classes

CF Interoperability

Working with Cocoa

Swift View of Objective-C APIs

Swift provides seamless access to Objective-C APIs

Swift View of Objective-C APIs

Swift provides seamless access to Objective-C APIs

Swift view of an Objective-C API is *different* from Objective-C

Swift View of Objective-C APIs

Swift provides seamless access to Objective-C APIs

Swift view of an Objective-C API is *different* from Objective-C

Same Cocoa conventions and idioms

Our Example: UIDocument

```
typedef NS_ENUM(NSInteger, UIDocumentSaveOperation) {
    UIDocumentSaveForCreating,
    UIDocumentSaveForOverwriting
};

@interface UIDocument : NSObject

@property NSDate *fileModificationDate;

- (instancetype)initWithFileURL:(NSURL *)url;

- (NSString *)fileNameExtensionForType:(NSString *)typeName
                saveOperation:(UIDocumentSaveOperation)saveOperation;

@end
```

Properties

Objective-C

```
@property NSDate *fileModificationDate;
```

Properties

Objective-C

```
@property NSDate *fileModificationDate;
```

Swift

```
var fileModificationDate: NSDate!
```

Implicitly Unwrapped Optionals

```
var fileModificationDate: NSDate!
```

Implicitly Unwrapped Optionals

```
var fileModificationDate: NSDate!
```

A value of class type in Swift is never `nil`

Implicitly Unwrapped Optionals

```
var fileModificationDate: NSDate!
```

A value of class type in Swift is never `nil`

Optional types generalize the notion of `nil`

Implicitly Unwrapped Optionals

```
var fileModificationDate: NSDate!
```

A value of class type in Swift is never `nil`

Optional types generalize the notion of `nil`

Objective-C does not have a notion of a “never-`nil`” pointer

Implicitly Unwrapped Optionals

```
var fileModificationDate: NSDate!
```

A value of class type in Swift is never `nil`

Optional types generalize the notion of `nil`

Objective-C does not have a notion of a “never-`nil`” pointer

‘!’ is an *implicitly unwrapped optional*

- Can be tested explicitly for `nil`
- Can directly access properties/methods of the underlying value
- Can be implicitly converted to its underlying value (e.g., `NSDate`)

- Intermediate Swift — Presidio — Wednesday 2:00PM

Mapping Objective-C Types to Swift

Objective-C

```
@property (readonly) NSString *fileType;
```

Mapping Objective-C Types to Swift

Objective-C

```
@property (readonly) NSString *fileType;
```

Swift

```
var fileType: String! { get }
```

Objective-C Types in Swift

Objective-C Type	Swift Equivalent
`BOOL`	`Bool`
`NSInteger`	`Int`
`SEL`	`Selector`
`id`	`AnyObject!`
`Class`	`AnyClass!`
`NSString *`	`String!`
`NSArray *`	`AnyObject[]!`

Methods

Objective-C

```
- (NSString *)fileNameExtensionForType:(NSString *)typeName
                  saveOperation:(UIDocumentSaveOperation)saveOperation;
```

Methods

Objective-C

```
- (NSString *)fileNameExtensionForType:(NSString *)typeName
                  saveOperation:(UIDocumentSaveOperation)saveOperation;
```

Swift

```
func fileNameExtensionForType(typeName: String!,
          saveOperation: UIDocumentSaveOperation) -> String!
```

Methods

Argument labels

Objective-C

```
- (NSString *)fileNameExtensionForType:(NSString *)typeName
                  saveOperation:(UIDocumentSaveOperation)saveOperation;
```

Swift

```
func fileNameExtensionForType(typeName: String!,
         saveOperation: UIDocumentSaveOperation) -> String!
```

Methods

Argument labels in calls

Objective-C

```
- (NSString *)fileNameExtensionForType:(NSString *)typeName
                  saveOperation:(UIDocumentSaveOperation)saveOperation;
```

Swift

```
func fileNameExtensionForType(typeName: String!,
           saveOperation: UIDocumentSaveOperation) -> String!
```

Usage

```
let ext = document.fileNameExtensionForType("public.presentation",
                         saveOperation: UIDocumentSaveOperation.ForCreating)
```

Methods

Objective-C

```
- (void)saveToURL:(NSURL *)url
        forSaveOperation:(UIDocumentSaveOperation)saveOperation
        completionHandler:(void (^)(BOOL success))completionHandler;
```

Methods

Objective-C

```
- (void)saveToURL:(NSURL *)url
        forSaveOperation:(UIDocumentSaveOperation)saveOperation
        completionHandler:(void (^)(BOOL success))completionHandler;
```

Swift

```
func saveToURL(url: NSURL!,
               forSaveOperation saveOperation: UIDocumentSaveOperation,
               completionHandler: ((success: Bool) -> Void)!)
```

Methods

Argument labels and internal parameter names

Objective-C

```
- (void)saveToURL:(NSURL *)url
        forSaveOperation:(UIDocumentSaveOperation)saveOperation
        completionHandler:(void (^)(BOOL success))completionHandler;
```

Swift

```
func saveToURL(url: NSURL!,
               forSaveOperation saveOperation: UIDocumentSaveOperation,
               completionHandler: ((success: Bool) -> Void)!)
```

Methods

Argument labels and internal parameter names

Objective-C

```
- (void)saveToURL:(NSURL *)url
        forSaveOperation:(UIDocumentSaveOperation)saveOperation
        completionHandler:(void (^)(BOOL success))completionHandler;
```

Swift

```
func saveToURL(url: NSURL!,
               forSaveOperation saveOperation: UIDocumentSaveOperation,
               completionHandler: ((success: Bool) -> Void)!)
```

Methods

Blocks and closures

Objective-C

```
- (void)saveToURL:(NSURL *)url
        forSaveOperation:(UIDocumentSaveOperation)saveOperation
        completionHandler:(void (^)(BOOL success))completionHandler;
```

Swift

```
func saveToURL(url: NSURL!,
               forSaveOperation saveOperation: UIDocumentSaveOperation,
               completionHandler: ((success: Bool) -> Void)!)
```

Methods

Swift

```
func saveToURL(url: NSURL!,
               forSaveOperation saveOperation: UIDocumentSaveOperation,
               completionHandler: ((success: Bool) -> Void)!)
```

Methods

Trailing closure syntax

Swift

```
func saveToURL(url: NSURL!,
               forSaveOperation saveOperation: UIDocumentSaveOperation,
               completionHandler: ((success: Bool) -> Void)!)
```

Methods

Trailing closure syntax

Swift

```
func saveToURL(url: NSURL!,
               forSaveOperation saveOperation: UIDocumentSaveOperation,
               completionHandler: ((success: Bool) -> Void)!)
```

Usage

```
document.saveToURL(documentURL,
                   saveOperation: UIDocumentSaveOperation.ForCreating) {
    success in
        if success { … } else { … }
}
```

Initializers

Objective-C

```
- (instancetype)initWithFileURL:(NSURL *)url;
```

Initializers

Objective-C

```
- (instancetype)initWithFileURL:(NSURL *)url;
```

Swift

```
init(fileURL url: NSURL!)
```

Initializers

Objective-C

```
- (instancetype)initWithFileURL:(NSURL *)url;
```

Swift

```
init(fileURL url: NSURL!)
```

Initializers

Objective-C

```
- (instancetype)initWithFileURL:(NSURL *)url;
```

Swift

```
init(fileURL url: NSURL!)
```

Initializers

Objective-C

```
- (instancetype)initWithFileURL:(NSURL *)url;
```

Swift

```
init(fileURL url: NSURL!)
```

Creating Objects

Objective-C

```
- (instancetype)initWithFileURL:(NSURL *)url;

UIDocument *document = [[UIDocument alloc] initWithFileURL:documentURL];
```

Creating Objects

Objective-C

```
- (instancetype)initWithFileURL:(NSURL *)url;

UIDocument *document = [[UIDocument alloc] initWithFileURL:documentURL];
```

Swift

```
init(fileURL url: NSURL!)

let document = UIDocument(fileURL: documentURL)
```

Factory Methods

Objective-C

```
+ (UIColor *)colorWithRed:(CGFloat)red green:(CGFloat)green
                     blue:(CGFloat)blue alpha:(CGFloat)alpha;
```

Factory Methods

Objective-C

```
+ (UIColor *)colorWithRed:(CGFloat)red green:(CGFloat)green
                     blue:(CGFloat)blue alpha:(CGFloat)alpha;

UIColor *color = [UIColor colorWithRed:1 green:0.67 blue:0.04 alpha:0];
```

Factory Methods

Objective-C

```
+ (UIColor *)colorWithRed:(CGFloat)red green:(CGFloat)green
                     blue:(CGFloat)blue alpha:(CGFloat)alpha;

UIColor *color = [UIColor colorWithRed:1 green:0.67 blue:0.04 alpha:0];
```

Swift

```
class func colorWithRed(red: CGFloat, green: CGFloat, blue: CGFloat,
                        alpha: CGFloat) -> UIColor!
```

Factory Methods

Objective-C

```
+ (UIColor *)colorWithRed:(CGFloat)red green:(CGFloat)green
                     blue:(CGFloat)blue alpha:(CGFloat)alpha;

UIColor *color = [UIColor colorWithRed:1 green:0.67 blue:0.04 alpha:0];
```

Swift

```
class func colorWithRed(red: CGFloat, green: CGFloat, blue: CGFloat,
                        alpha: CGFloat) -> UIColor!

let color = UIColor.colorWithRed(1, green: 0.67, blue: 0.04, alpha: 0)
```

Factory Methods as Initializers

Objective-C

```
+ (UIColor *)colorWithRed:(CGFloat)red green:(CGFloat)green
                     blue:(CGFloat)blue alpha:(CGFloat)alpha;

UIColor *color = [UIColor colorWithRed:1 green:0.67 blue:0.04 alpha:0];
```

Swift

```
init(red: CGFloat, green: CGFloat, blue: CGFloat, alpha: CGFloat)
```

Factory Methods as Initializers

Objective-C

```
+ (UIColor *)colorWithRed:(CGFloat)red green:(CGFloat)green
                     blue:(CGFloat)blue alpha:(CGFloat)alpha;

UIColor *color = [UIColor colorWithRed:1 green:0.67 blue:0.04 alpha:0];
```

Swift

```
init(red: CGFloat, green: CGFloat, blue: CGFloat, alpha: CGFloat)

let color = UIColor(red: 1, green: 0.67, blue: 0.04, alpha: 0)
```

Enums

Objective-C

```
typedef NS_ENUM(NSInteger, UIDocumentSaveOperation) {
    UIDocumentSaveForCreating,
    UIDocumentSaveForOverwriting
};
```

Enums

Objective-C

```
typedef NS_ENUM(NSInteger, UIDocumentSaveOperation) {
    UIDocumentSaveForCreating,
    UIDocumentSaveForOverwriting
};
```

Enums

Objective-C

```
typedef NS_ENUM(NSInteger, UIDocumentSaveOperation) {
    UIDocumentSaveForCreating,
    UIDocumentSaveForOverwriting
};
```

Swift

```
enum UIDocumentSaveOperation : Int {
    case ForCreating
    case ForOverwriting
}
```

Swift

```
enum UIDocumentSaveOperation : Int {
    case ForCreating
    case ForOverwriting
}
```

Usage

```
let ext = document.fileNameExtensionForType("public.presentation",
                saveOperation:
```

Using Enums

Swift

```
enum UIDocumentSaveOperation : Int {
    case ForCreating
    case ForOverwriting
}
```

Usage

```
let ext = document.fileNameExtensionForType("public.presentation",
                saveOperation: UIDocumentSaveOperation .ForCreating)
```

Using Enums

Swift

```
enum UIDocumentSaveOperation : Int {
    case ForCreating
    case ForOverwriting
}
```

Usage

```
let ext = document.fileNameExtensionForType("public.presentation",
                saveOperation: .ForCreating)
```

NSError

Objective-C

```
- (id)contentsForType:(NSString *)typeName error:(NSError **)outError;
```

NSError

Objective-C

```
- (id)contentsForType:(NSString *)typeName error:(NSError **)outError;
```

Swift

```
func contentsForType(typeName: String!, error: NSErrorPointer) -> AnyObject!
```

Swift

```
func contentsForType(typeName: String!, error: NSErrorPointer) -> AnyObject!
```

Using NSError

Swift

```
func contentsForType(typeName: String!, error: NSErrorPointer) -> AnyObject!
```

Using NSError

Swift

```
func contentsForType(typeName: String!, error: NSErrorPointer) -> AnyObject!
```

Usage

```
var error: NSError?
if let contents = document.contentsForType("public.presentation",
                                           error: &error) {
    // use the contents
}
```

Using NSError

Swift

```
func contentsForType(typeName: String!, error: NSErrorPointer) -> AnyObject!
```

Usage

```
var error: NSError?
if let contents = document.contentsForType("public.presentation",
                                           error: &error) {
    // use the contents
} else if let actualError = error {
    // handle error
}
```

```
35 class UIDocument : NSObject, NSFilePresenter, NSObjectProtocol {
36
37     // The designated initializer. Passing an empty URL will cause this method to t
38     init(fileURL url: NSURL!)
39
40     // UIKit may call these methods on background threads, so subclasses that overr
41     // These values will be set by UIKit before the completion handlers to the open
42     // Clients that wish to access these properties outside of an open, save, or re
43
44     var fileURL: NSURL! { get }
45     var localizedName: String! { get } // The default implementation derives the na
46     var fileType: String! { get } // The file's UTI. Derived from the fileURL by de
47     var fileModificationDate: NSDate! // The last known modification date of the do
48
49     var documentState: UIDocumentState { get }
50
51     // Subclassing this method without calling super should be avoided. Subclassers
52     // Open the document located by the fileURL.  This will call readFromURL:error:
53     func openWithCompletionHandler(completionHandler: ((Bool) -> Void)!)
54
55     // Close the document. The default implementation calls [self autosaveWithCompl
56     func closeWithCompletionHandler(completionHandler: ((Bool) -> Void)!)
```

Modernizing Your Objective-C

These rules apply to all Objective-C APIs imported into Swift

Swift benefits greatly from "modern" Objective-C:

- Properties
- instancetype
- NS_ENUM/NS_OPTIONS
- NS_DESIGNATED_INITIALIZER

Modernizing Your Objective-C

NEW

These rules apply to all Objective-C APIs imported into Swift

Swift benefits greatly from "modern" Objective-C:

- Properties
- instancetype
- NS_ENUM/NS_OPTIONS
- NS_DESIGNATED_INITIALIZER

Modernizing Your Objective-C

NEW

These rules apply to all Objective-C APIs imported into Swift

Swift benefits greatly from "modern" Objective-C:

- Properties
- `instancetype`
- `NS_ENUM/NS_OPTIONS`
- `NS_DESIGNATED_INITIALIZER`

- What's New in LLVM — Marina — Wednesday 9:00AM

id and AnyObject

id in Objective-C

Upcasts

```
id object = [[NSURL alloc] initWithString:@"http://developer.apple.com"];
object = view.superview;
```

id in Objective-C

Upcasts

```
id object = [[NSURL alloc] initWithString:@"http://developer.apple.com"];
object = view.superview;
```

Message sends

```
[object removeFromSuperview];
```

id in Objective-C

Upcasts

```
id object = [[NSURL alloc] initWithString:@"http://developer.apple.com"];
object = view.superview;
```

Message sends

```
[object removeFromSuperview];
```

Subscripting

```
id date = object[@"date"];
```

AnyObject: An Object of Any Type

Upcasts

```
id object = [[NSURL alloc] initWithString:@"http://developer.apple.com"];
object = view.superview;
```

Message sends

```
[object removeFromSuperview];
```

Subscripting

```
id date = object[@"date"];
```

AnyObject: An Object of Any Type

Upcasts

```
var object: AnyObject = NSURL(string: "http://developer.apple.com")
object = view.superview
```

Message sends

```
[object removeFromSuperview];
```

Subscripting

```
id date = object[@"date"];
```

AnyObject: An Object of Any Type

Upcasts

```
var object: AnyObject = NSURL(string: "http://developer.apple.com")
object = view.superview
```

Message sends

```
object.removeFromSuperview()
```

Subscripting

```
id date = object[@"date"];
```

AnyObject: An Object of Any Type

Upcasts

```
var object: AnyObject = NSURL(string: "http://developer.apple.com")
object = view.superview
```

Message sends

```
object.removeFromSuperview()
```

Subscripting

```
let date = object["date"]
```

respondsToSelector Idiom

Messaging `id` or `AnyObject` can result in “unrecognized selector” failures

```
[object removeFromSuperview];
```

respondsToSelector Idiom

Messaging `id` or `AnyObject` can result in "unrecognized selector" failures

```
[object removeFromSuperview];
```

`respondsToSelector` idiom to test the presence of a method

```
if ([object respondsToSelector:@selector(removeFromSuperview)]) {
    [object removeFromSuperview];
}
```

Checking the Presence of a Method

A method of `AnyObject` is "optional"

```
object.removeFromSuperview()
```

Checking the Presence of a Method

A method of `AnyObject` is "optional"

```
object.removeFromSuperview?()
```

Chaining ? folds the `respondsToSelector` check into the call

Downcasting AnyObject

`AnyObject` does not implicitly downcast

```
let view: UIView = object
```

Downcasting AnyObject

`AnyObject` does not implicitly downcast

```
let view: UIView = object // error: ‘AnyObject’ cannot be implicitly downcast
```

Downcasting AnyObject

`AnyObject` does not implicitly downcast

```
let view: UIView = object // error: 'AnyObject' cannot be implicitly downcast
```

"`as`" operator forces the downcast

```
let view = object as UIView
```

Downcasting AnyObject

`AnyObject` does not implicitly downcast

```
let view: UIView = object // error: 'AnyObject' cannot be implicitly downcast
```

"`as`" operator forces the downcast

```
let view = object as UIView
```

"`as?`" operator performs a conditional downcast

```
if let view = object as? UIView {
    // view is a UIView
}
```

Protocols

Objective-C

```
@protocol UITableViewDataSource<NSObject>
@optional
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView;
@required
- (NSInteger)tableView:(UITableView *)tableView
             numberOfRowsInSection:(NSInteger)section;
@end
```

Protocols

Objective-C

```
@protocol UITableViewDataSource<NSObject>
@optional
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView;
@required
- (NSInteger)tableView:(UITableView *)tableView
             numberOfRowsInSection:(NSInteger)section;
@end
```

Swift

```
@objc protocol UITableViewDataSource : NSObjectProtocol {
  func tableView(tableView: UITableView, numberOfRowsInSection: Int) -> Int
  @optional func numberOfSectionsInTableView(tableView: UITableView) -> Int
}
```

Protocols

Objective-C

```
@protocol UITableViewDataSource<NSObject>
@optional
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView;
@required
- (NSInteger)tableView:(UITableView *)tableView
             numberOfRowsInSection:(NSInteger)section;
@end
```

Swift

```
@objc protocol UITableViewDataSource : NSObjectProtocol {
  func tableView(tableView: UITableView, numberOfRowsInSection: Int) -> Int
  @optional func numberOfSectionsInTableView(tableView: UITableView) -> Int
}
```

Protocols

Objective-C

```
@protocol UITableViewDataSource<NSObject>
@optional
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView;
@required
- (NSInteger)tableView:(UITableView *)tableView
             numberOfRowsInSection:(NSInteger)section;
@end
```

Swift

```
@objc protocol UITableViewDataSource : NSObjectProtocol {
  func tableView(tableView: UITableView, numberOfRowsInSection: Int) -> Int
  @optional func numberOfSectionsInTableView(tableView: UITableView) -> Int
}
```

Protocol Types

Objective-C

```
@property id <UITableViewDataSource> dataSource;
```

Protocol Types

Objective-C

```
@property id <UITableViewDataSource> dataSource;
```

Swift

```
var dataSource: UITableViewDataSource!
```

Testing Protocol Conformance

Downcast via “as?” to protocol type

```
if let dataSource = object as? UITableViewDataSource {
  let rowsInFirstSection = dataSource.tableView(tableView,
                                                numberOfRowsInSection: 0)
}
```

Protocol Types

Objective-C

```
@property id <UITableViewDataSource> dataSource;
@property id <UINavigationControllerDelegate,
              UIImagePickerControllerDelegate> delegate;
```

Swift

```
var dataSource: UITableViewDataSource!
var delegate: protocol<UINavigationControllerDelegate,
                       UIImagePickerControllerDelegate>
```

Number of Rows in the Last Section

```
if let dataSource = object as? UITableViewDataSource {
  var lastSection = 0
  let numSections = dataSource.numberOfSectionsInTableView(tableView)
  lastSection = numSections - 1

  let rowsInLastSection = dataSource.tableView(tableView,
                                               numberOfRowsInSection: lastSection)
}
```

Optional Methods in Protocols

```
if let dataSource = object as? UITableViewDataSource {
  var lastSection = 0
  let numSections = dataSource.numberOfSectionsInTableView(tableView)
  lastSection = numSections - 1

  let rowsInLastSection = dataSource.tableView(tableView,
                                               numberOfRowsInSection: lastSection)
}
```

Optional Methods in Protocols

```
if let dataSource = object as? UITableViewDataSource {
  var lastSection = 0
  let numSections = dataSource.numberOfSectionsInTableView(tableView)
  lastSection = numSections - 1    // error: method is optional

  let rowsInLastSection = dataSource.tableView(tableView,
                                               numberOfRowsInSection: lastSection)
}
```

Optional Methods in Protocols

Use the chaining ? operator

```
if let dataSource = object as? UITableViewDataSource {
  var lastSection = 0
  let numSections = dataSource.numberOfSectionsInTableView(tableView)
  lastSection = numSections - 1

  let rowsInLastSection = dataSource.tableView(tableView,
                                               numberOfRowsInSection: lastSection)
}
```

Optional Methods in Protocols

Use the chaining ? operator

```
if let dataSource = object as? UITableViewDataSource {
  var lastSection = 0
  if let numSections = dataSource.numberOfSectionsInTableView?(tableView){
      lastSection = numSections - 1
  }
  let rowsInLastSection = dataSource.tableView(tableView,
                                               numberOfRowsInSection: lastSection)
}
```

Optional Methods in Protocols

Use the chaining ? operator

```
if let dataSource = object as? UITableViewDataSource {
  var lastSection = 0
  if let numSections = dataSource.numberOfSectionsInTableView?(tableView){
      lastSection = numSections - 1
  }
  let rowsInLastSection = dataSource.tableView(tableView,
                                               numberOfRowsInSection: lastSection)
}
```

Optionals and Safety

Optionals and Safety

`AnyObject` is Swift's equivalent to `id`

- Similar functionality, more safe by default
- `AnyClass` is Swift's equivalent to `Class`

Optionals and Safety

`AnyObject` is Swift's equivalent to `id`

- Similar functionality, more safe by default
- `AnyClass` is Swift's equivalent to `Class`

Optionals used throughout the language to represent dynamic checks

- `as?` for safe downcasting, protocol conformance checking
- Optionals when referring to methods that may not be available
- `if let` and chaining `?` make optionals easy to use

Bridging Core Cocoa Types

Native Strings, Arrays, Dictionaries

One set of general-purpose native value types

- Safe by default
- Predictable performance
- Typed collections support items of any type

Bridged to Cocoa `NSString`, `NSArray`, `NSDictionary`

Native String Type

`String` is an efficient, Unicode-compliant string type

Flexible, efficient, high-level APIs for string manipulation

Value semantics

Native String Type

`String` is an efficient, Unicode-compliant string type

Flexible, efficient, high-level APIs for string manipulation

Value semantics

```
var s1 = "Hello"
var s2 = s1
s1 += " Swift”
println(s1)
Hello Swift
println(s2)
Hello
```

Characters

Iteration over a string produces characters

```
let dog = "Dog!🐶"
for c in dog {     // c is inferred as Character
    println(c)
}
```

Characters

Iteration over a string produces characters

```
let dog = "Dog!🐶"
for c in dog {     // c is inferred as Character
    println(c)
}
```

```
D
o
g
!
🐶
```

Characters and Code Points

Unicode characters cannot be efficiently encoded as fixed-width entities

Characters and Code Points

Unicode characters cannot be efficiently encoded as fixed-width entities

- Correct use of UTF-8 or UTF-16 requires deep knowledge of Unicode

Characters and Code Points

Unicode characters cannot be efficiently encoded as fixed-width entities

- Correct use of UTF-8 or UTF-16 requires deep knowledge of Unicode
- Low-level operations (`length`, `characterAtIndex`) are not provided by `String`

Characters and Code Points

Unicode characters cannot be efficiently encoded as fixed-width entities

- Correct use of UTF-8 or UTF-16 requires deep knowledge of Unicode
- Low-level operations (`length`, `characterAtIndex`) are not provided by `String`

`countElements` can be used to count the number of characters

```
let dog = "Dog!🐶"
println("There are \(countElements(dog)) characters in `\(dog)’")
```

Characters and Code Points

Unicode characters cannot be efficiently encoded as fixed-width entities

- Correct use of UTF-8 or UTF-16 requires deep knowledge of Unicode
- Low-level operations (`length`, `characterAtIndex`) are not provided by `String`

`countElements` can be used to count the number of characters

```
let dog = "Dog!🐶"
println("There are \(countElements(dog)) characters in `\(dog)’")

// There are 5 characters in `Dog!🐶’
```

Code Points

UTF-16 is available via a property

```
for codePoint in dog.utf16 { // codePoint is inferred as UInt16
  // …
}

print("There are \(countElements(dog.utf16)) UTF-16 code points in `\(dog)'")
```

Code Points

UTF-16 is available via a property

```
for codePoint in dog.utf16 { // codePoint is inferred as UInt16
  // …
}

print("There are \(countElements(dog.utf16)) UTF-16 code points in `\(dog)’")
// There are 6 UTF-16 code points in `Dog!🐶’
```

String and NSString

Foundation `NSString` APIs are available on `String`

```
let fruits = "apple;banana;cherry".componentsSeparatedByString(";")
```

String and NSString

Foundation `NSString` APIs are available on `String`

```
let fruits = "apple;banana;cherry".componentsSeparatedByString(";")
                                                    // inferred as String[]
```

String and NSString

Foundation `NSString` APIs are available on `String`

```
let fruits = "apple;banana;cherry".componentsSeparatedByString(";")
                                                    // inferred as String[]
```

Cast to `NSString` to access properties and methods on `NSString` categories

```
("Welcome to WWDC 2014" as NSString).myNSStringMethod()
```

String and NSString

Foundation `NSString` APIs are available on `String`

```
let fruits = "apple;banana;cherry".componentsSeparatedByString(";")
                                                    // inferred as String[]
```

Cast to `NSString` to access properties and methods on `NSString` categories

```
("Welcome to WWDC 2014" as NSString).myNSStringMethod()
```

Extend `String` with your method

```
extension String {
  func myStringMethod() -> String { … }
}
```

NSArray Bridges to Array of AnyObject

Objective-C

```
@property NSArray *toolbarItems;
```

NSArray Bridges to Array of AnyObject

Objective-C

```
@property NSArray *toolbarItems;
```

Swift

```
var toolbarItems: AnyObject[]!
```

Upcasting Arrays

An array `T[]` can be assigned to an `AnyObject[]`

```
let myToolbarItems: UIBarButtonItem[] = [item1, item2, item3]
controller.toolbarItems = myToolbarItems
```

Downcasting Arrays

Iteration over an `AnyObject[]` produces `AnyObject` values

```
for object: AnyObject in viewController.toolbarItems {
    let item = object as UIBarButtonItem
    // …
}
```

Downcasting Arrays

Iteration over an `AnyObject[]` produces `AnyObject` values

```
for object: AnyObject in viewController.toolbarItems {
    let item = object as UIBarButtonItem
    // …
}
```

Can downcast `AnyObject[]` to an array of a specific type

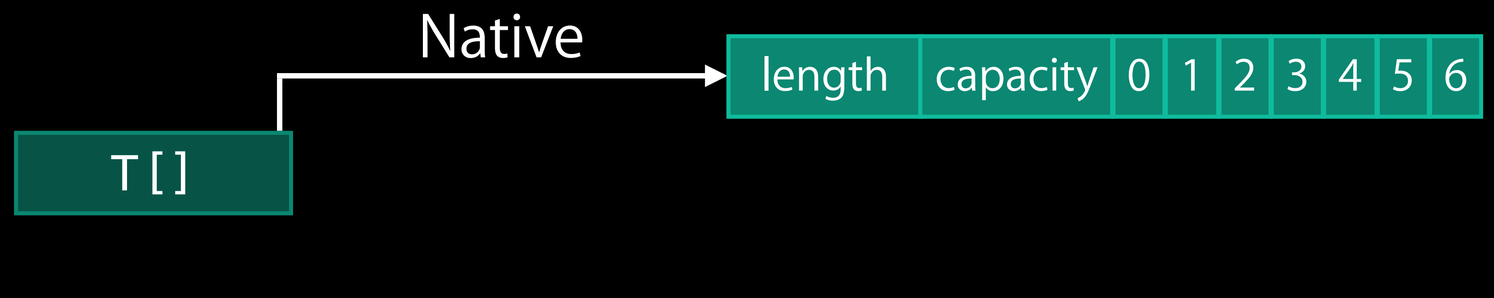
```
for item in viewController.toolbarItems as UIBarButtonItem[] {
    // …
}
```

Array Bridging—Under the Hood

Swift array has two representations

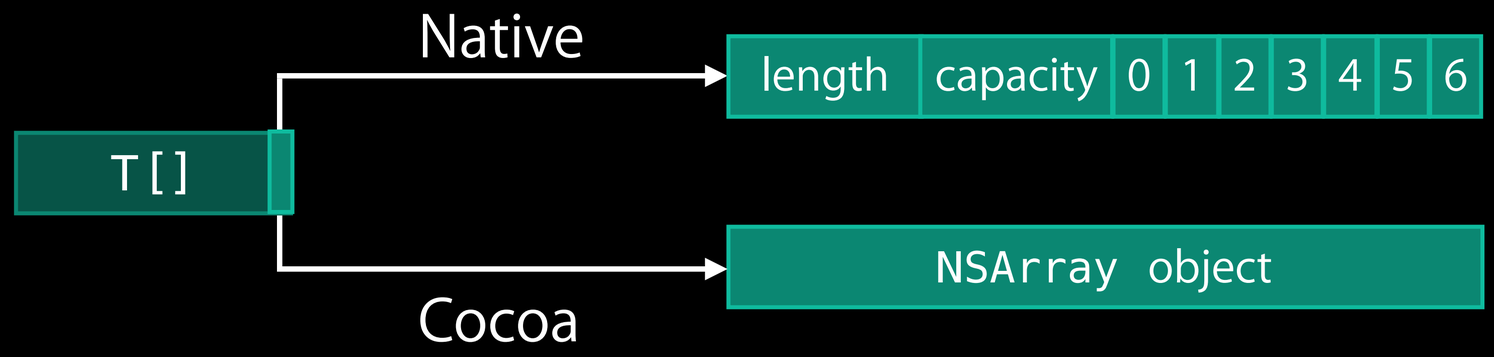
Array Bridging—Under the Hood

Swift array has two representations



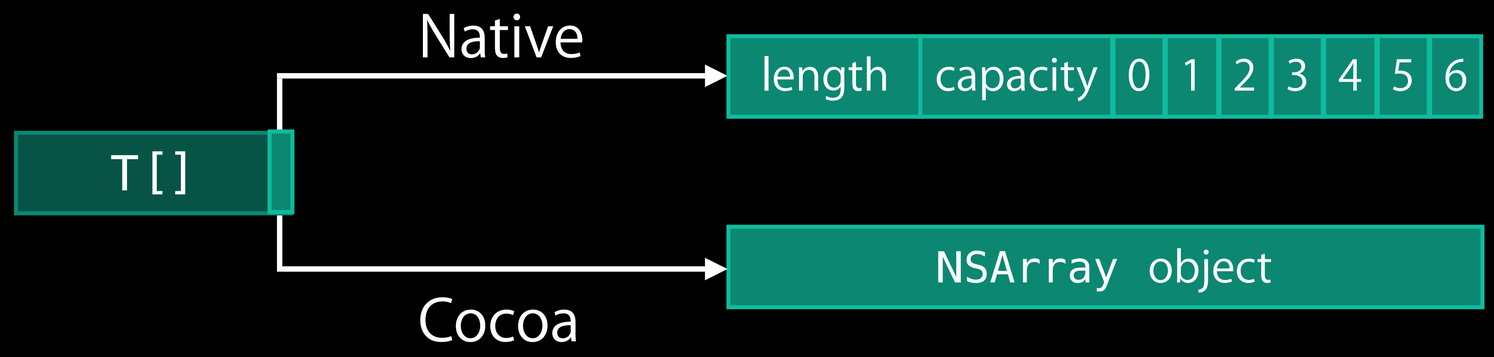
Array Bridging—Under the Hood

Swift array has two representations



Array Bridging—Under the Hood

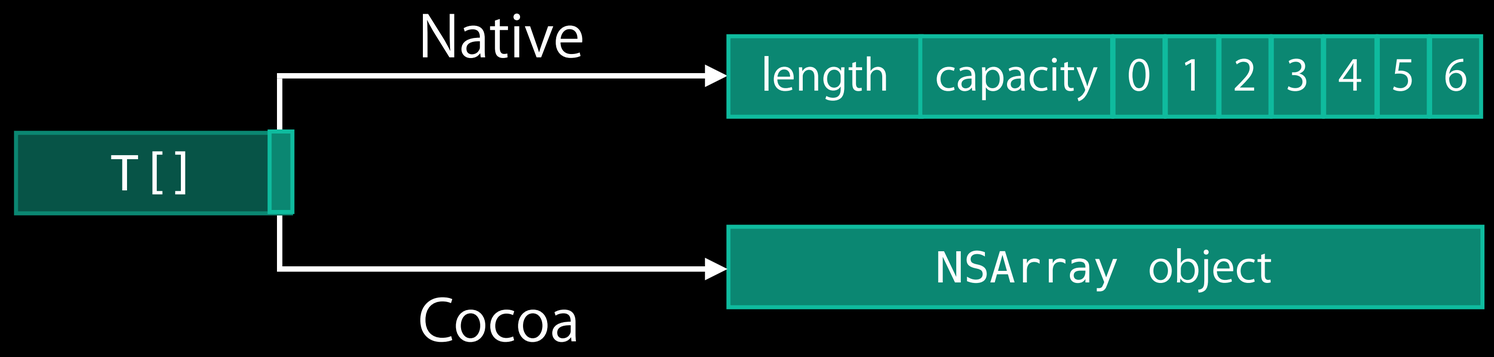
Swift array has two representations



Array methods manage the representation internally

Array Bridging—Under the Hood

Swift array has two representations



Array methods manage the representation internally

Bridging converts between `NSArray` and a Swift array

NSArray → Swift Array Bridging

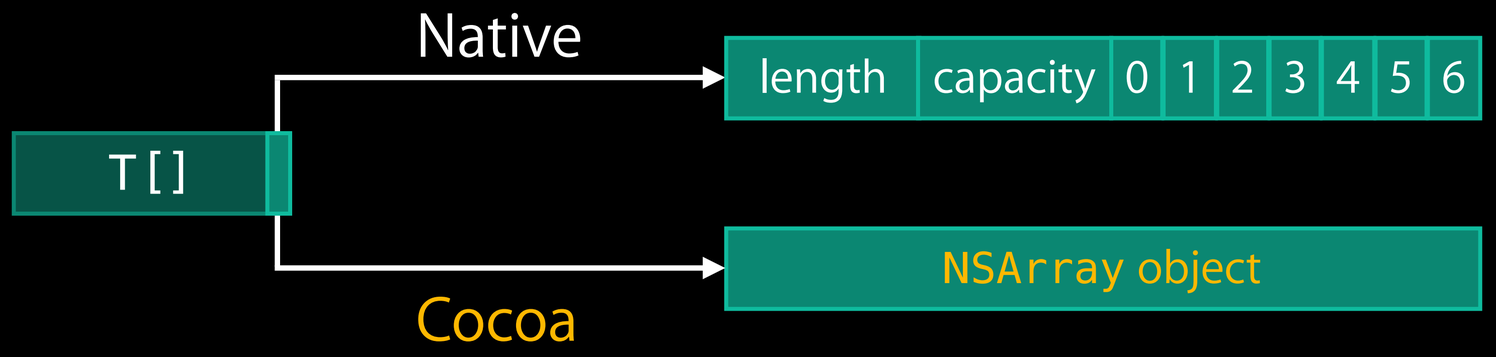
Returning an `NSArray*` from an Objective-C method to Swift

```
let items: AnyObject[] = viewController.toolbarItems
```

NSArray → Swift Array Bridging

Returning an `NSArray*` from an Objective-C method to Swift

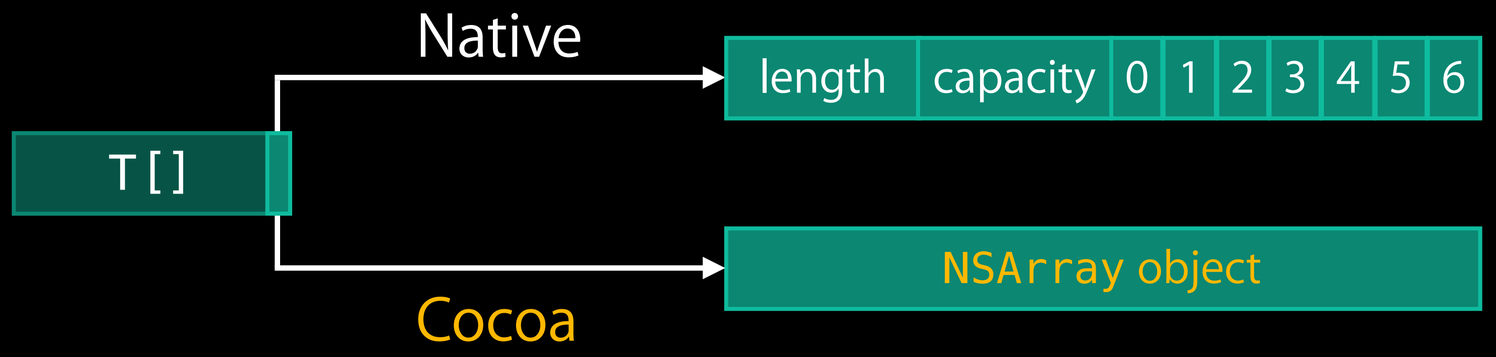
```
let items: AnyObject[] = viewController.toolbarItems
```



NSArray → Swift Array Bridging

Returning an `NSArray*` from an Objective-C method to Swift

```
let items: AnyObject[] = viewController.toolbarItems
```

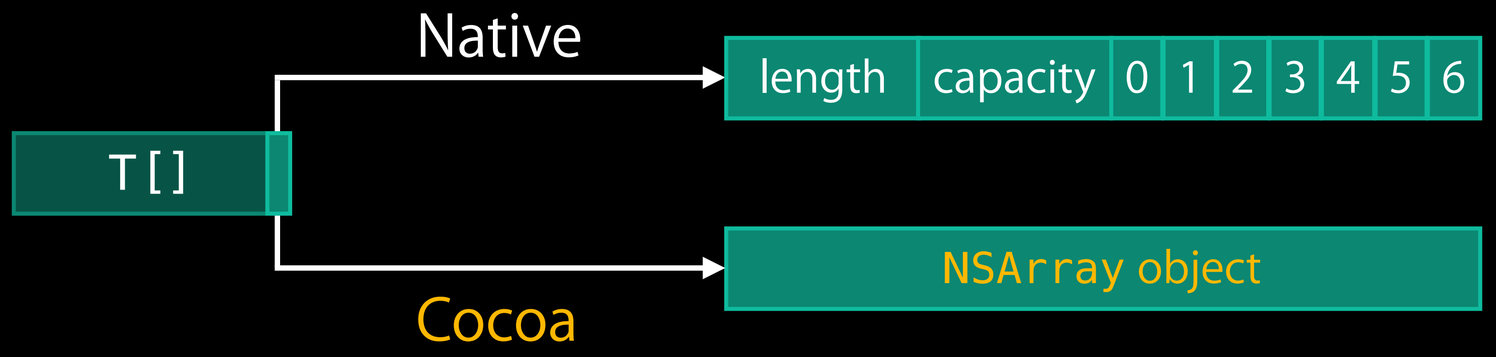


Calls `copy()` to ensure the array won't change underneath us

NSArray → Swift Array Bridging

Returning an `NSArray*` from an Objective-C method to Swift

```
let items: AnyObject[] = viewController.toolbarItems
```



Calls `copy()` to ensure the array won't change underneath us

- For immutable `NSArrays`, this operation is trivial

T[] → NSArray Bridging

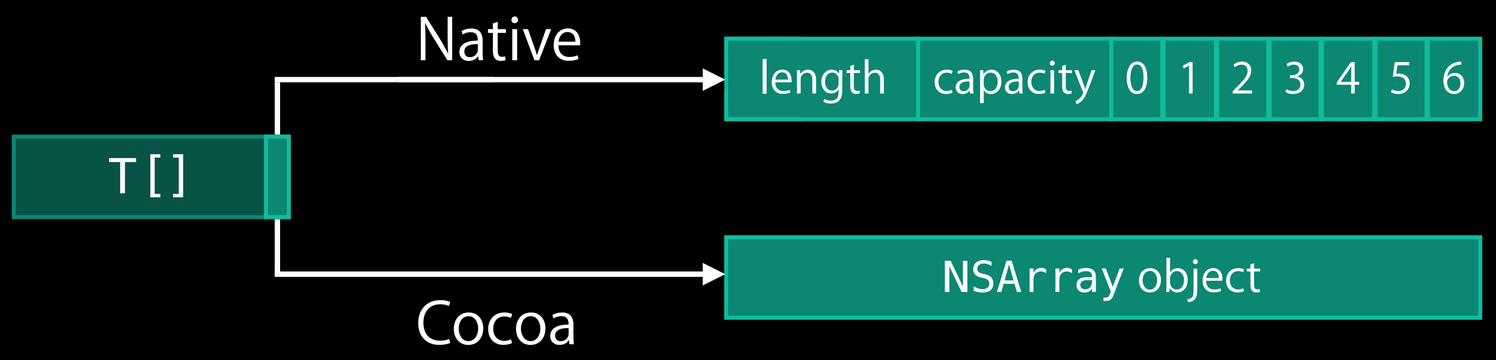
Passing a Swift array to an Objective-C method expecting an `NSArray*`

```
viewController.toolbarItems = myToolbarItems
```

T[] → NSArray Bridging

Passing a `T[]` to an Objective-C method expecting an `NSArray*`

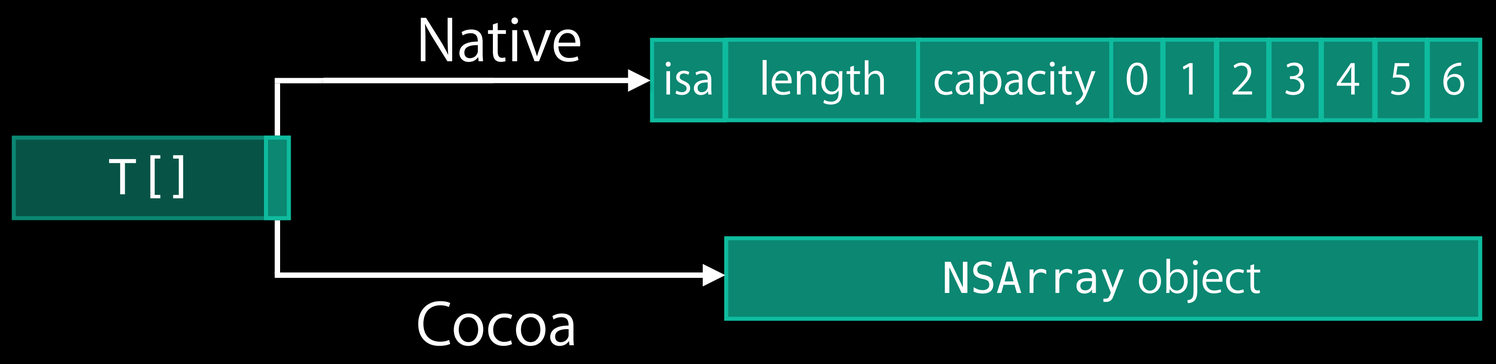
```
viewController.toolbarItems = myToolbarItems
```



T[] → NSArray Bridging

Passing a `T[]` to an Objective-C method expecting an `NSArray*`

```
viewController.toolbarItems = myToolbarItems
```



Native array representation “isa” NSArray, optimized

Subclassing Objective-C Classes

Swift Objects Are Objective-C Objects

All Swift classes are “id compatible”

- Same layout as an Objective-C class
- Same basic infrastructure (retain/release/class/etc.)

Swift Objects Are Objective-C Objects

All Swift classes are "id compatible"

- Same layout as an Objective-C class
- Same basic infrastructure (retain/release/class/etc.)

Inherit from an Objective-C class to make your class directly visible in Objective-C

```
class MyDocument : UIDocument {
    var items: String[] = []
}
```

Overriding Methods

"override" keyword required when overriding a method

```
override func handleError(error: NSError!, userInteractionPermitted: Bool) {
    // customized behavior
    super.handleError(error,
                      userInteractionPermitted: userInteractionPermitted)
}
```

Overriding Properties

Override the property itself, not the getter or setter

```
override var description: String {
    return "MyDocument containing \(items)"
}
```

Overriding and NSError**

NSErrorPointer is Swift's version of NSError**

```
override func contentsForType(typeName: String!, error: NSErrorPointer)
                -> AnyObject! {
    if cannotProduceContentsForType(typeName) {
        if error {
            error.memory = NSError(domain: domain, code: code, userInfo: [:])
        }
        return nil
    }
    // …
}
```

Your Swift Class…

Swift

```
class MyDocument : UIDocument {
    var items: String[] = []

    override func handleError(error: NSError, userInteractionPermitted: Bool)

    override var description: String

    override func contentsForType(typeName: String!, error: NSErrorPointer)
                    -> AnyObject!
}
```

Your Swift Class…in Objective-C

Objective-C

```
SWIFT_CLASS("_TtC5MyApp10MyDocument")
@interface MyDocument : UIDocument

@property (nonatomic) NSArray *items;

- (void)handleError:(NSError *)error
       userInteractionPermitted:(BOOL)userInteractionPermitted;

@property (nonatomic, readonly) NSString *description;

- (id)contentsForType:(NSString *)typeName error:(NSError **)outError;
@end
```

Your Swift Class…in Objective-C

Objective-C

```
SWIFT_CLASS("_TtC5MyApp10MyDocument")
@interface MyDocument : UIDocument

@property (nonatomic) NSArray *items;

- (void)handleError:(NSError *)error
       userInteractionPermitted:(BOOL)userInteractionPermitted;

@property (nonatomic, readonly) NSString *description;

- (id)contentsForType:(NSString *)typeName error:(NSError **)outError;
@end
```

Your Swift Class…in Objective-C

Objective-C

```
SWIFT_CLASS("_TtC5MyApp10MyDocument")
@interface MyDocument : UIDocument

@property (nonatomic) NSArray *items;    // Swift: var items: String[] = []

- (void)handleError:(NSError *)error
       userInteractionPermitted:(BOOL)userInteractionPermitted;

@property (nonatomic, readonly) NSString *description;

- (id)contentsForType:(NSString *)typeName error:(NSError **)outError;
@end
```

Your Swift Class…in Objective-C

Objective-C

```
SWIFT_CLASS("_TtC5MyApp10MyDocument")
@interface MyDocument : UIDocument

@property (nonatomic) NSArray *items;

- (void)handleError:(NSError *)error
       userInteractionPermitted:(BOOL)userInteractionPermitted;

@property (nonatomic, readonly) NSString *description;

- (id)contentsForType:(NSString *)typeName error:(NSError **)outError;
@end
```

Your Swift Class…in Objective-C

Objective-C

```
SWIFT_CLASS("_TtC5MyApp10MyDocument")      // usually written MyApp.MyDocument
@interface MyDocument : UIDocument

@property (nonatomic) NSArray *items;

- (void)handleError:(NSError *)error
       userInteractionPermitted:(BOOL)userInteractionPermitted;

@property (nonatomic, readonly) NSString *description;

- (id)contentsForType:(NSString *)typeName error:(NSError **)outError;
@end
```

Limitations of Objective-C

Swift has advanced features that aren't expressible in Objective-C

- Tuples
- Generics
- Enums and structs

```
func myGenericMethod<T>(x: T) -> (String, String) { … }
```

Limitations of Objective-C

Swift has advanced features that aren't expressible in Objective-C

- Tuples
- Generics
- Enums and structs

```
@objc func myGenericMethod<T>(x: T) -> (String, String) { … }
                                    // error: not expressible in Objective-C
```

"objc" attribute verifies that the declaration can be used in Objective-C

Controlling Objective-C Names

"objc" attribute can be used to change the name of an Objective-C method

```
var enabled: Bool {
    get { … }
    set { … }
}
```

Controlling Objective-C Names

"objc" attribute can be used to change the name of an Objective-C method

```
var enabled: Bool {                                  // property is named "enabled"
    get { … }                                        // getter is named "enabled"
    set { … }                                        // setter is named "setEnabled:"
}
```

Controlling Objective-C Names

"objc" attribute can be used to change the name of an Objective-C method

```
var enabled: Bool {                                  // property is named "enabled"
    @objc(isEnabled) get { … }                       // getter is named "isEnabled"
    set { … }                                        // setter is named "setEnabled:"
}
```

Controlling Objective-C Names

"objc" attribute can be used to change the name of an Objective-C method

```
var enabled: Bool {                                  // property is named "enabled"
    @objc(isEnabled) get { … }                       // getter is named "isEnabled"
    set { … }                                        // setter is named "setEnabled:"
}
```

Or the name of a class

```
@objc(ABCMyDocument) class MyDocument : UIDocument {
    // …
}
```

CF Interoperability

CF in Objective-C

```
void drawGradientRect(CGContextRef context, CGColorRef startColor,
                      CGColorRef endColor, CGFloat width, CGFloat height) {









}
```

CF in Objective-C

```
void drawGradientRect(CGContextRef context, CGColorRef startColor,
                      CGColorRef endColor, CGFloat width, CGFloat height) {
    CGColorSpaceRef colorSpace = CGColorSpaceCreateDeviceRGB();
    NSArray *colors = @[(__bridge id)startColor, (__bridge id)endColor];
    CGFloat locations[2] = {0.0, 1.0};
    CGGradientRef gradient = CGGradientCreateWithColors(colorSpace,
                                (CFArrayRef)colors, locations);




}
```

CF in Objective-C

Bridge casts

```
void drawGradientRect(CGContextRef context, CGColorRef startColor,
                      CGColorRef endColor, CGFloat width, CGFloat height) {
    CGColorSpaceRef colorSpace = CGColorSpaceCreateDeviceRGB();
    NSArray *colors = @[(__bridge id)startColor, (__bridge id)endColor];
    CGFloat locations[2] = {0.0, 1.0};
    CGGradientRef gradient = CGGradientCreateWithColors(colorSpace,
                                (CFArrayRef)colors, locations);




}
```

CF in Objective-C

Three kinds of arrays

```
void drawGradientRect(CGContextRef context, CGColorRef startColor,
                      CGColorRef endColor, CGFloat width, CGFloat height) {
    CGColorSpaceRef colorSpace = CGColorSpaceCreateDeviceRGB();
    NSArray *colors = @[(__bridge id)startColor, (__bridge id)endColor];
    CGFloat locations[2] = {0.0, 1.0};
    CGGradientRef gradient = CGGradientCreateWithColors(colorSpace,
                                (CFArrayRef)colors, locations);




}
```

CF in Objective-C

```
void drawGradientRect(CGContextRef context, CGColorRef startColor,
                      CGColorRef endColor, CGFloat width, CGFloat height) {
    CGColorSpaceRef colorSpace = CGColorSpaceCreateDeviceRGB();
    NSArray *colors = @[(__bridge id)startColor, (__bridge id)endColor];
    CGFloat locations[2] = {0.0, 1.0};
    CGGradientRef gradient = CGGradientCreateWithColors(colorSpace,
                             (CFArrayRef)colors, locations);

    CGPoint startPoint = CGPointMake(width / 2, 0);
    CGPoint endPoint = CGPointMake(width / 2, height);
    CGContextDrawLinearGradient(context, gradient, startPoint, endPoint, 0);


}
```

CF in Objective-C

Manual memory management

```
void drawGradientRect(CGContextRef context, CGColorRef startColor,
                      CGColorRef endColor, CGFloat width, CGFloat height) {
    CGColorSpaceRef colorSpace = CGColorSpaceCreateDeviceRGB();
    NSArray *colors = @[(__bridge id)startColor, (__bridge id)endColor];
    CGFloat locations[2] = {0.0, 1.0};
    CGGradientRef gradient = CGGradientCreateWithColors(colorSpace,
                             (CFArrayRef)colors, locations);

    CGPoint startPoint = CGPointMake(width / 2, 0);
    CGPoint endPoint = CGPointMake(width / 2, height);
    CGContextDrawLinearGradient(context, gradient, startPoint, endPoint, 0);

    CGColorSpaceRelease(colorSpace);
    CGGradientRelease(gradient);
}
```

CF in Swift

```
func drawGradientRect(context: CGContext, startColor: CGColor,
                      endColor: CGColor, width: CGFloat, height: CGFloat) {









}
```

Managed CF Objects

```
func drawGradientRect(context: CGContext, startColor: CGColor,
                      endColor: CGColor, width: CGFloat, height: CGFloat) {
    let colorSpace = CGColorSpaceCreateDeviceRGB()




}
```

Managed CF Objects

```
func drawGradientRect(context: CGContext, startColor: CGColor,
                      endColor: CGColor, width: CGFloat, height: CGFloat) {
    let colorSpace = CGColorSpaceCreateDeviceRGB() // inferred as CGColorSpace




}
```

Managed CF Objects

```
func drawGradientRect(context: CGContext, startColor: CGColor,
                      endColor: CGColor, width: CGFloat, height: CGFloat) {
    let colorSpace = CGColorSpaceCreateDeviceRGB() // inferred as CGColorSpace



    // colorSpace automatically released
}
```

Toll-Free Bridging Conversions

```
func drawGradientRect(context: CGContext, startColor: CGColor,
                      endColor: CGColor, width: CGFloat, height: CGFloat) {
    let colorSpace = CGColorSpaceCreateDeviceRGB()
    let gradient = CGGradientCreateWithColors(colorSpace,
                                              [startColor, endColor],
                                              [0.0, 1.0])



}
```

C Interoperability

```
func drawGradientRect(context: CGContext, startColor: CGColor,
                      endColor: CGColor, width: CGFloat, height: CGFloat) {
    let colorSpace = CGColorSpaceCreateDeviceRGB()
    let gradient = CGGradientCreateWithColors(colorSpace,
                                              [startColor, endColor],
                                              [0.0, 1.0])


}
```

Construction of C Structs

```
func drawGradientRect(context: CGContext, startColor: CGColor,
                      endColor: CGColor, width: CGFloat, height: CGFloat) {
    let colorSpace = CGColorSpaceCreateDeviceRGB()
    let gradient = CGGradientCreateWithColors(colorSpace,
                                              [startColor, endColor],
                                              [0.0, 1.0])

    let startPoint = CGPoint(x: width / 2, y: 0)
    let endPoint = CGPoint(x: width / 2, y: height)
    CGContextDrawLinearGradient(context, gradient, startPoint, endPoint, 0)
}
```

Explicitly Bridged APIs

Some CF APIs have not been audited for implicit bridging

```
CGColorRef CGColorGetRandomColor(void);
```

Explicitly Bridged APIs

Some CF APIs have not been audited for implicit bridging

```
CGColorRef CGColorGetRandomColor(void);
```

Swift uses Unmanaged<T> when the ownership convention is unknown

```
func CGColorGetRandomColor() -> Unmanaged<CGColor>
```

Working with Unmanaged Objects

Unmanaged<T> enables manual memory management

```
struct Unmanaged<T: AnyObject> {
    func takeUnretainedValue() -> T   // for +0 returns
    func takeRetainedValue() -> T     // for +1 returns


}
```

Working with Unmanaged Objects

Unmanaged<T> enables manual memory management

```
struct Unmanaged<T: AnyObject> {
    func takeUnretainedValue() -> T   // for +0 returns
    func takeRetainedValue() -> T     // for +1 returns


}
```

Working with Unmanaged Objects

Unmanaged<T> enables manual memory management

```
struct Unmanaged<T: AnyObject> {
    func takeUnretainedValue() -> T   // for +0 returns
    func takeRetainedValue() -> T     // for +1 returns



}
```

Use it to work with unaudited CF APIs

```
let color = CGColorGetRandomColor().takeUnretainedValue()
```

Working with Unmanaged Objects

Unmanaged<T> enables manual memory management

```
struct Unmanaged<T: AnyObject> {
    func takeUnretainedValue() -> T   // for +0 returns
    func takeRetainedValue() -> T     // for +1 returns



}
```

Use it to work with unaudited CF APIs

```
let color = CGColorGetRandomColor().takeUnretainedValue()
                                                // inferred as CGColor
```

- Advanced Swift — Presidio — Thursday 11:30AM

Working with Unmanaged Objects

Unmanaged<T> enables manual memory management

```
struct Unmanaged<T: AnyObject> {
    func takeUnretainedValue() -> T   // for +0 returns
    func takeRetainedValue() -> T     // for +1 returns


    func retain() -> Unmanaged<T>
    func release()
    func autorelease() -> Unmanaged<T>
}
```

Use it to work with unaudited CF APIs

```
let color = CGColorGetRandomColor().takeUnretainedValue()
                                                  // inferred as CGColor
```

- Advanced Swift — Presidio — Thursday 11:30AM

Implicit Bridging

Audit CF APIs to ensure that conform to CF memory conventions

```
CGColorRef CGColorGetRandomColor(void);
```

Swift uses Unmanaged<T> when the ownership convention is unknown

```
func CGColorGetRandomColor() -> Unmanaged<CGColor>
```

Implicit Bridging

Audit CF APIs to ensure that conform to CF memory conventions

```
CF_IMPLICIT_BRIDGING_ENABLED
CGColorRef CGColorGetRandomColor(void);
CF_IMPLICIT_BRIDGING_DISABLED
```

Swift uses `Unmanaged<T>` when the ownership convention is unknown

```
func CGColorGetRandomColor() -> Unmanaged<CGColor>
```

Implicit Bridging

Audit CF APIs to ensure that conform to CF memory conventions

```
CF_IMPLICIT_BRIDGING_ENABLED
CGColorRef CGColorGetRandomColor(void);
CF_IMPLICIT_BRIDGING_DISABLED
```

Implicit bridging eliminates Unmanaged<T>

```
func CGColorGetRandomColor() -> CGColor
```

Summary

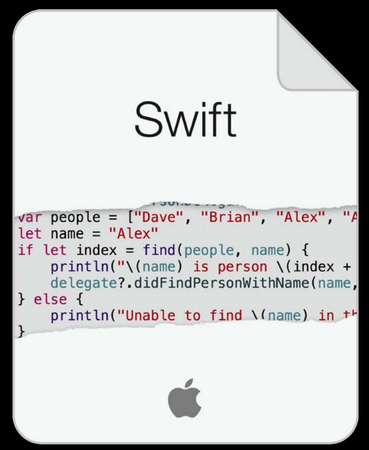
Seamless interoperability between Swift and Objective-C

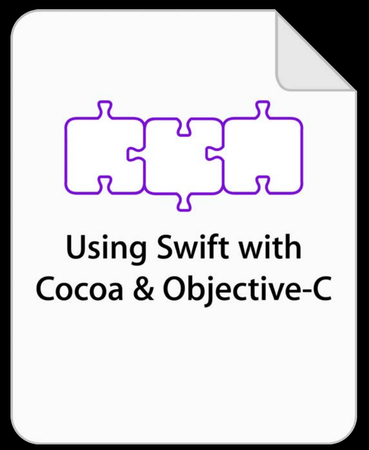
- Let the tools help you understand the relationship

Bridging of Core Cocoa types

- Prefer native strings, arrays, dictionaries

Automated CF memory management





More Information

Dave DeLong
Developer Tools Evangelist
delong@apple.com

Documentation
Using Swift with Cocoa and Objective-C
http://apple.com

Apple Developer Forums
http://devforums.apple.com

Related Sessions

• Integrating Swift with Objective-C	Presidio	Wednesday 9:00AM
• Intermediate Swift	Presidio	Wednesday 2:00PM
• Advanced Swift	Presidio	Thursday 11:30AM

Labs

• Swift	Tools Lab A	Thursday 9:00AM
• Swift	Tools Lab A	Thursday 2:00PM
• Swift	Tools Lab A	Friday 9:00AM
• Swift	Tools Lab A	Friday 2:00PM

WWDC14