
Swift 5.0

Who is this guy?

Who is this guy?

- Richard L Zarth III
- Software Engineer and iOS Developer
- Website: rlziii.com
- Email: rlziii@icloud.com

Who is this guy?

- Started programming in 2016 with iOS 9 and Swift 2.2
- UCF
  - Major: Computer Science
  - Minor: Secure Computing and Networks

What We'll Cover

- Swift Evolution
- ABI Stability
- Changes in Swift 5.0
- Getting Started with Swift 5.0

Swift Evolution

Swift Evolution

- Swift started development in 2010
- Swift was revealed at WWDC 2014
- Chris Lattner was the project lead
- Ted Kremenek became the project lead in early 2017

Open Source

- Swift became open source in late 2015
- Swift development lives on GitHub
- Swift Evolution repo keeps track of proposals
- Swift Forums available for discussions and pre-proposals

Proposal Process

- Swift Core Team has the final say regarding proposals
- A formal proposal is submitted
- A review time is allocated and revisions made as necessary
- Active Review → Accepted → Implemented

Commonly Rejected Changes

- Replace brackets with Python-style indentation
- Replace logical operators with keywords (e.g. && → AND)
- Use garbage collection instead of automatic reference counting
- Rewrite Swift compiler in Swift

What We'll Cover

- Swift Evolution
- ABI Stability
- Changes in Swift 5.0
- Getting Started with Swift 5.0

What We'll Cover

- ~~Swift Evolution~~
- ABI Stability
- Changes in Swift 5.0
- Getting Started with Swift 5.0

ABI Stability

ABI Stability

- Application Binary Interface
- Primary focus of Swift 5.0
  - Source compatibility across future Swift versions
  - Binary and runtime compatibility

Source Compatibility

- Newer compiler can compile older Swift code
- Reduce migration pains
- Removes "version lock"
- Maintain single code base across multiple Swift versions

Binary & Runtime Compatibility

- Distribute frameworks in binary form across Swift versions
- Code to link together and interoperate at a runtime level
- Enables module format stability
- Enables ABI stability

Module Format Stability

- Communicates source-level information about a framework
- Compiler's representation of the public interface of a framework
- Shared library provides compiled implementation to the runtime
- This will not be finalized until a Swift 5.x release

What is ABI?

- The API of binary Swift programs
- Binary entities must agree on many low-level details
- Must conform to be linked together and executed
- ABI is per-platform (architecture and operating system)

What is ABI Stability?

- Lock down ABI to enable future binary conformance
- Tends to persist for the rest of the platform's lifetime
- Decisions about the ABI tend to have long-term ramifications
- New and orthogonal changes are called ABI-additive changes

What We'll Cover

- ~~Swift Evolution~~
- ABI Stability
- Changes in Swift 5.0
- Getting Started with Swift 5.0

What We'll Cover

- ~~Swift Evolution~~
- ~~ABI Stability~~
- Changes in Swift 5.0
- Getting Started with Swift 5.0

Changes in Swift 5.0

Changes in Swift 5.0

- Strong source compatibility with Swift 4.2
- Drops support for Swift 3 compatibility
- Final branching happened on November 16, 2018
- 22 proposals implemented in Swift 5.0

Handling Future Enum Cases

- SE-0192 Handling Future Enum Cases
- Switch statements must be exhaustive
  - However, what if a case is added to an API at a later time?
  - Without a **`default`** case this would break code

Handling Future Enum Cases

- **`@unknown default`** accommodates this problem
- Display warning if switch is not exhaustive without **`default`** case
- Can only have **`@unknown`** attribute on a **`default`** case
- Cannot have both **`default`** and **`@unknown default`** cases

Handling Future Enum Cases

- Enums can be declared as `@_frozen`
  - These enums will never get new cases
  - e.g. `Optional`, `FloatingPointSign`
  - Not for general use by application developers

```swift
enum UserType {
    case regular
    case admin
}

func doSomething(userType: UserType) {
    switch userType {
    case .regular:
        print("This is a regular user.")
    case .admin:
        print("This is an admin.")
}
```

```swift
enum UserType {
    case regular
    case admin
}

func doSomething(userType: UserType) {
    switch userType {
    case .regular:
        print("This is a regular user.")
    case .admin:
        print("This is an admin.")
    default:
        print("User has unknown type.")
    }
}
```

```swift
enum UserType {
    case regular
    case admin
}

func doSomething(userType: UserType) {
    switch userType {
    case .regular:
        print("This is a regular user.")
    case .admin:
        print("This is an admin.")
    @unknown default:
        print("User has unknown type.")
    }
}
```

```swift
enum UserType {
    case regular
    case admin
    case moderator
}

func doSomething(userType: UserType) {
    switch userType {
    case .regular:
        print("This is a regular user.")
    case .admin:
        print("This is an admin.")
    @unknown default:
        print("User has unknown type.")
    }
}
```

```swift
enum UserType {
    case regular
    case admin
    case moderator
}

func doSomething(userType: UserType) {
    switch userType {
    case .regular:
        print("This is a regular user.")
    case .admin:
        print("This is an admin.")
    case .moderator:
        print("This is a moderator.")
    @unknown default:
        print("User has unknown type.")
    }
}
```

Enhancing String Literal Delimiters

- SE-0200 Enhancing String Literals Delimiters to Support Raw Text
- Adding a **#** symbol before the first " symbol will change the delimiter
  - The string terminator " now becomes **"#**
  - The escape delimiter \ now becomes **\#**

Enhancing String Literal Delimiters

- Can be used in multiline strings as well using **#"""** and **"""#**
- Can be especially useful when writing regular expressions
- Adding multiple **#** symbols will further change the delimiters
  - Using **##** makes the string terminator **"##** and escape delimiter **\##**

```
print("Is \"swifty\" even a word?")
```

```
print(#"Is "swifty" even a word?"#)
```

```
let kirbyDance = """
\\(^_^)/
<(^_^)>
/(^_^)\\
"""
```

```
let kirbyDance = #"""
\(^_^)/
<(^_^)>
/(^_^)\
"""#
```

Unicode and Character Properties

- Two proposals:
  - SE-0211 Add Unicode Properties to Unicode.Scalar
  - SE-0221 Character Properties
- Adds many convenience properties for Unicode and Character

Unicode.Scalar.Properties

- Boolean properties (e.g. **`isAlphabetic`**, **`isMath`**, **`isEmoji`**)
- Case mappings (e.g. **`lowercaseMapping`**, **`uppercaseMapping`**)
- Identification and classification (e.g. **`age`**, **`name`**)
- Numerics (e.g. **`numericType`**, **`numericValue`**)

```swift
let chick = Unicode.Scalar("🐥")
print(chick.properties.isEmoji)      // true

let a = Unicode.Scalar("a")
print(a.properties.uppercaseMapping) // A
print(a.properties.name!)            // LATIN SMALL LETTER A
print(a.properties.age!)             // (major: 1, minor: 1)

let one = Unicode.Scalar("1")
print(one.properties.numericType!)   // decimal
print(one.properties.numericValue!)  // 1.0
```

Character Properties

- Boolean properties (e.g. **`isASCII`**, **`isCurrencySymbol`**, **`isLetter`**)
- Case methods (e.g. **`lowercased()`**, **`uppercased()`**)
- ASCII properties (e.g. **`asciiValue`**)
- Numeric properties (e.g. **`hexDigitValue`**, **`wholeNumberValue`**)

```swift
let a = Character("a")
print(a.isASCII)                // true
print(a.asciiValue!)            // 97
print(a.uppercased())           // A

let yen = Character("¥")
print(yen.isCurrencySymbol)  // true

let yon = Character("四")
print(yon.wholeNumberValue!) // 4
```

Dynamically "Callable" Types

- Followup to the **`@dynamicMemberLookup`** attribute (Swift 4.2)
- Adds **`@dynamicCallable`** attribute
- Allows for elegant interoperation with dynamic languages
  - e.g. Python, JavaScript, Pearl, Ruby

Dynamically "Callable" Types

- This proposal is purely syntactic sugar
- Written by Chris Lattner and Dan Zheng
  - Both work on TensorFlow
- Particularly useful for server-side Swift and machine learning tools

```swift
// Swift 4.2
let file = Python.open.call(with: filename)

// Swift 5.0
let file = Python.open(filename)
```

```
// Import DogModule.Dog as Dog.
let Dog = Python.import.call(with: "DogModule.Dog")

// Python: dog = Dog("Brianna")
let dog = Dog.call(with: "Brianna")

// Python: dog.add_trick("Roll over")
dog.add_trick.call(with: "Roll over")

// Python: dog2 = Dog("Kaylee").add_trick("snore")
let dog2 = Dog.call(with: "Kaylee").add_trick.call(with: "snore")
```

```
// Import DogModule.Dog as Dog.
let Dog = Python.import("DogModule.Dog")

// Python: dog = Dog("Brianna")
let dog = Dog("Brianna")

// Python: dog.add_trick("Roll over")
dog.add_trick("Roll over")

// Python: dog2 = Dog("Kaylee").add_trick("snore")
let dog2 = Dog("Kaylee").add_trick("snore")
```

Dictionary.compactMapValue

- Adds a combined **`filter`** and **`map`** operation to dictionaries
- Corresponds to **`Sequence.compactMap()`**
- Keys stay intact while values are transformed
- Results are unwrapped and **`nil`** values are discarded

```
let releaseYears = [
    "iPhone": "2007",
    "iPhone 5": "2012",
    "iPhone 6 Plus": "2014",
    "iPhone 9": "Right around the corner",
    "iPhone XS Max": "2018",
    "iPhone X Double-S Max Plus Deluxe Extreme": "TBD"
]

let currentReleases = releaseYears.compactMapValues { Int($0) }
```

```
let releaseYears = [
    "iPhone": "2007",
    "iPhone 5": "2012",
    "iPhone 6 Plus": "2014",
    "iPhone 9": "Right around the corner",
    "iPhone XS Max": "2018",
    "iPhone X Double-S Max Plus Deluxe Extreme": "TBD"
]

let currentReleases = releaseYears.compactMapValues { Int($0) }
```

```
let screenSizes = [
    "original": 3.5,
    "tall": 4,
    "hd": 4.7,
    "plus": 5.5,
    "x": 5.8,
    "xr": 6.1,
    "max": 6.5,
    "xdsmpde": nil
]

let currentSizes = screenSizes.compactMapValues { $0 }
```

```swift
let screenSizes = [
    "original": 3.5,
    "tall": 4,
    "hd": 4.7,
    "plus": 5.5,
    "x": 5.8,
    "xr": 6.1,
    "max": 6.5,
    "xdsmpde": nil
]

let currentSizes = screenSizes.compactMapValues { $0 }
```

Sequence.count(where:)

- Counts the number of elements in a **Sequence** that pass some test
- Combines what would normally be two steps: **`filter`** then **`count`**
- Less wasteful because there is no need for an intermediate array
- This proposal solves a common problem that programmers face

```
let result = [-3, -2, -1, 0, 1, 2, 3].filter({ $0 > 0 }).count
```

```
let result = [-3, -2, -1, 0, 1, 2, 3].count { $0 > 0 }
```

BinaryInteger.isMultiple(of:)

- **`BinaryInteger`** is the base protocol for **`Int`**, **`Int16`**, **`UInt32`**, etc.
- Adds a simple way of checking multiplicity
- Limits the need for remainder operator in many cases
- Primary reason for proposal was readability and new programmers

```
if (x % 2 == 0) {
    // Do something for even numbers only.
}
```

```
if x.isMultiple(of: 2) {
    // Do something for even numbers only.
}
```

```swift
// Fizz-Buzzin' every day.
for n in 1...100 {
    switch (n.isMultiple(of: 3), n.isMultiple(of: 5)) {
    case (true, false):
        print("Fizz")
    case (false, true):
        print("Buzz")
    case (true, true):
        print("FizzBuzz")
    default:
        print(n)
    }
}
```

ExpressibleByStringInterpolation

- This protocol was deprecated in Swift 3.0 because of inefficiency
- New method is considerably more flexible and efficient
- Can be used to customize string interpolation results
- More powerful in many cases than **`CustomStringConvertable`**

```
struct Color {
    var name: String
    var tint: String
}

extension String.StringInterpolation {
    mutating func appendInterpolation(_ a: Color) {
        appendInterpolation("The color is \(a.tint) \(a.name)")
    }
}

let color = Color(name: "blue", tint: "dark")
```

```
//      Print: Color(name: "blue", tint: "dark")
print("Print: \(color)")

//      Much more expressive: The color is dark blue.
print("Much more expressive: \(color)")
```

```swift
let attuqoltuae = "Quite definitely \(42, style: .spellOut)."

// Quite definitely forty-two.
print(attuqoltuae)
```

```swift
let attuqoltuae = "Quite definitely \(42, style: .scientific)."

// Quite definitely 4.2E1.
print(attuqoltuae)
```

POP QUIZ

POP QUIZ

```swift
struct Doughnut {
    // 🍩🍩🍩
}

func optionalDoughnut() throws -> Doughnut? {
    var doughnut: Doughnut?
    // Some code goes here.
    return doughnut
}

let whatTypeAmI = try? optionalDoughnut()
```

```swift
struct Doughnut {
    // 🍩🍩🍩
}

func optionalDoughnut() throws -> Doughnut? {
    var doughnut: Doughnut?
    // Some code goes here.
    return doughnut
}

let whatTypeAmI = try? optionalDoughnut() // Doughnut??
```

Flatten Nested 'try?' Optionals

- Nested optionals are valid in Swift, but usually not intended
- Several popular workarounds for this strange behavior
  - e.g. `if let x = (try? somethingAsAny()) as? Something`
- Does not turn optionals into non-optionals

Flatten Nested 'try?' Optionals

- Nested optionals are valid in Swift, but usually not intended
- Several popular workarounds for this strange behavior
  - e.g. `if let x = (try? somethingAsAny()) as? Something` 🤔
- Does not turn optionals into non-optionals

Add Result to Standard Library

- Many individual framework and application implementations
- Offers a pragmatic compromise between present and future use
- Most helpful when dealing with asynchronous APIs
- More simple and clean than the current method

```swift
public enum Result<Value, Error: Swift.Error> {
    case success(Value)
    case failure(Error)
}
```

```swift
URLSession.shared.dataTask(with: url) { (data, response, error) in
    guard error != nil else {
        self.handleError(error!)
    }

    guard let data = data, let response = response else {
        return
    }

    handleResponse(response, data: data)
}
```

```swift
URLSession.shared.dataTask(with: url) { result in
    switch result {
    case .success(let response):
        handleResponse(response.0, data: response.1)
    case .failure(let error):
        handleError(error)
    }
}
```

What Wasn't Covered?

What Wasn't Covered?

- SE-0213 Literal initialization via coercion
- SE-0214 Renaming the DictionaryLiteral type to KeyValuePairs
- SE-0215 Conform Never to Equatable and Hashable
- SE-0219 Package Manager Dependency Mirroring

What Wasn't Covered?

- SE-0224 Support 'less than' operator in compilation conditions
- SE-0227 Identity key path
- SE-0229 SIMD Vectors
- SE-0232 Remove Some Collection Customization Points

What Wasn't Covered?

- SE-0233 Make Numeric Refine new AdditiveArithmetic Protocol
- SE-0234 Remove Sequence.SubSequence
- SE-0237 Introduce withContiguousStorageIfAvailable methods
- SE-0239 Add Codable conformance to Range types (after Swift 5.0)

What We'll Cover

- ~~Swift Evolution~~
- ~~ABI Stability~~
- Changes in Swift 5.0
- Getting Started with Swift 5.0

What We'll Cover

- ~~Swift Evolution~~
- ~~ABI Stability~~
- ~~Changes in Swift 5.0~~
- Getting Started with Swift 5.0

Getting Started with Swift 5.0

Getting Started with Swift 5.0

- Why try Swift 5.0 now?
  - Get a head start with new development features
  - Ensure that your current projects will not break
  - Just for FUN!

Let's try this live...