What’s New in Swift

Ted Kremenek, Swift Team
Anna Zaks, Swift Team
Released in March

Swift 5
Xcode 10.2

Developer Preview

Swift 5.1
Xcode 11
APIs
Shared Swift Runtime for Apps
Binary Frameworks
Apple Swift-Only Frameworks
Package Manager in Xcode
More API Expressibility
Library Evolution
Shared Swift Runtime for Apps
Binary Frameworks
Apple Swift-Only Frameworks
Package Manager in Xcode
More API Expressibility
Library Evolution
ABI and Module Stability
What is ABI?

ABI = “Application Binary Interface”

Specifies details of a program’s representation at runtime

Compatible ABIs allows separately compiled code to interact at runtime
Running process
Before ABI stability these both need to be built with the same compiler!
ABI stability in Swift 5
Program and framework can be built with different compilers (Swift 5 or later)
What is Module Stability?

Swift libraries and the APIs they export are known as “modules”

Module files are created (and used) by the compiler
Before module stability these both need to be built with the **same compiler!**

```swift
import MyFramework

func doSomething()
```

Foo.swift

.swiftmodule

exec
Swift 5.1 introduces a stable and textual module interface file

```swift
import MyFramework

func doSomething()
```

Swift 5.1 introduces a stable and textual module interface file.

import MyFramework

func doSomething()
Module + ABI stability
= Binary frameworks
ABI Stability and More

FEBRUARY 7, 2019  Jordan Rose

It has been a longstanding goal to stabilize Swift’s ABI on macOS, iOS, watchOS, and tvOS. While a stable ABI is an important milestone for the maturity of any language, the ultimate benefit to the Swift ecosystem was to enable binary compatibility for apps and libraries. This post describes what binary compatibility means in Swift 5 and how it will evolve in future releases of Swift.

You may ask: what about other platforms? ABI stability is implemented for each operating system that it compiles and runs on. Swift’s ABI is currently declared stable for Swift 5 on Apple platforms. As development of Swift on Linux, Windows, and other platforms matures, the Swift Core Team will evaluate stabilizing the ABI on those platforms.

Swift 5 provides binary compatibility for apps: a guarantee that going forward, an app built with one version of the Swift compiler will be able to talk to a library built with another version. This applies even when using the compatibility mode with older language versions (-swift-version 4.2).
ABI Stability and More

FEBRUARY 7, 2019  Jordan Rose

It has been a longstanding goal to stabilize Swift’s ABI on macOS, iOS, watchOS, and tvOS. While a stable ABI is an important milestone for the maturity of any language, the ultimate benefit to the Swift ecosystem was to enable binary compatibility for apps and libraries. This post describes what binary compatibility means in Swift 5 and how it will evolve in future releases of Swift.

You may already know that the ABI is implemented for each operating system. In Swift, the ABI currently declared stable for Swift 5 on Apple platforms. As development of Swift on Linux, Windows, and other platforms matures, the Swift Core Team will evaluate stabilizing the ABI on those platforms.

Swift 5 provides binary compatibility for apps: a guarantee that going forward, an app built with one version of the Swift compiler will be able to talk to a library built with another version. This applies even when using the compatibility mode with older language versions (~swift-version 4.2).
Binary Frameworks in Swift

Thursday, 3:00 PM
Adopting Swift Packages in Xcode

Creating Swift Packages
Performance
# Shared Swift Runtime for Apps in the OS

<table>
<thead>
<tr>
<th>macOS</th>
<th>iOS</th>
<th>tvOS</th>
<th>watchOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.14.4</td>
<td>12.2</td>
<td>12.2</td>
<td>5.2</td>
</tr>
</tbody>
</table>
Apps and the Shared Swift Runtime in the OS

Apps use the runtime from OS when it is available
Apps and the Shared Swift Runtime in the OS

Apps use the runtime from OS when it is available

Apps that backward deploy contain a runtime copy

• Embedded runtime copy is inert on OSs with shared runtime
• iOS App Store thins out runtime copy when downloading to iOS 12.2 or later
Launch time overhead with Swift 4.2
5%
Launch time overhead with Swift 4.2
5%
Launch time overhead with Swift 5
Launch time overhead with Swift 5: 0%
Code Size Reductions
10% Smaller Code Size
15% Smaller Code Size when using ‘Optimize for Size’
Faster Bridging Between Swift and Objective-C
Swift Bridging of Currency Types Used in Cocoa

Objective-C

- (BOOL)unlockWithPassword:(NSString *)password;

Swift

func unlock(withPassword password: String) -> Bool
1.6x

NSDictionary to Dictionary Bridging
15x

NSString operations when performed on bridged Swift Strings
String
UTF-8 String

MARCH 20, 2019  Michael Iseman

Swift 5 switches the preferred encoding of strings from UTF-16 to UTF-8 while preserving efficient Objective-C-interoperability. Because the String type abstracts away these low-level concerns, no source-code changes from developers should be necessary*, but it’s worth highlighting some of the benefits this move gives us now and in the future.

Switching to UTF-8 fulfills one of String’s long-term goals to enable high-performance processing, which is the most passionate request from performance-sensitive developers. It also lays the groundwork for providing even more performant APIs in the future. String’s preferred encoding is baked into Swift’s ABI for performance, so it was imperative that this switch happen in time for ABI stability in Swift 5.

* See "Use of String.Index.encodedOffset Considered Harmful" below for a potential change in behavior if misused

Background

A Change in Structure

Even though the String type is technically a struct, it can exist in many forms. You can think of String as an artisanal enum, hand-crafted using traditional bit-twiddling techniques in...
Native Swift Strings now interoperate with C APIs without overhead
Native Swift Strings now interoperate with C APIs without overhead

Small Strings now include Unicode characters in addition to ASCII
Native Swift Strings now interoperate with C APIs without overhead.

Small Strings now include Unicode characters in addition to ASCII.

Fast Interoperability with NSString.
SwiftNIO is a cross-platform asynchronous event-driven network application framework for rapid development of maintainable high performance protocol servers & clients.

It's like Netty, but written for Swift.

Repository organization

The SwiftNIO project is split across multiple repositories:

<table>
<thead>
<tr>
<th>Repository</th>
<th>NIO 2 (Swift 5)</th>
<th>NIO 1 (Swift 4+)</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="https://github.com/apple/swift-nio">https://github.com/apple/swift-nio</a></td>
<td>from: &quot;2.0.0&quot;</td>
<td>from: &quot;1.0.0&quot;</td>
</tr>
<tr>
<td>SwiftNIO core</td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="https://github.com/apple/swift-nio-ssl">https://github.com/apple/swift-nio-ssl</a></td>
<td>from: &quot;2.0.0&quot;</td>
<td>from: &quot;1.0.0&quot;</td>
</tr>
<tr>
<td>TLS (SSL) support</td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="https://github.com/apple/swift-nio-http2">https://github.com/apple/swift-nio-http2</a></td>
<td>from: &quot;1.0.0&quot;</td>
<td>from: &quot;0.1.0&quot;</td>
</tr>
<tr>
<td>HTTP/2 support</td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="https://github.com/apple/swift-nio-extras">https://github.com/apple/swift-nio-extras</a></td>
<td>from: &quot;1.0.0&quot;</td>
<td>from: &quot;0.1.0&quot;</td>
</tr>
<tr>
<td>useful additions around SwiftNIO</td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="https://github.com/apple/swift-nio-transport-services">https://github.com/apple/swift-nio-transport-services</a></td>
<td>from: &quot;1.0.0&quot;</td>
<td>from: &quot;0.1.0&quot;</td>
</tr>
<tr>
<td>first-class support for macOS, iOS, and tvOS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Supported Platforms

SwiftNIO aims to support all of the platforms where Swift is supported. Currently, it is developed and tested on macOS and Linux, but may support other platforms upon the availability of Swift on those platforms.
SwiftNIO is a cross-platform asynchronous event-driven network application framework for rapid development of maintainable high performance protocol servers & clients.

It's like Netty, but written for Swift.

<table>
<thead>
<tr>
<th>Repository organization</th>
<th>NIO 1 (Swift 4+)</th>
<th>NIO 2 (Swift 5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SwiftNIO core</td>
<td>from &quot;1.0.0&quot;</td>
<td>from &quot;0.1.0&quot;</td>
</tr>
<tr>
<td></td>
<td>HTTP2 support</td>
<td></td>
</tr>
<tr>
<td></td>
<td>first-class support for macOS, iOS, and tvOS</td>
<td>useful additions around SwiftNIO</td>
</tr>
<tr>
<td></td>
<td>from &quot;1.0.0&quot;</td>
<td>from &quot;0.1.0&quot;</td>
</tr>
</tbody>
</table>

Supported Platforms

SwiftNIO aims to support all of the platforms where Swift is supported. Currently, it is developed and tested on macOS and Linux, but support for other platforms is planned.

“SwiftNIO is a cross-platform asynchronous event-driven network application framework for rapid development of maintainable high performance protocol servers & clients.”
20%  
More requests per second on HTTP/1.1 server built on SwiftNIO
Swift Tooling and Open Source
SwiftNIO is a cross-platform asynchronous event-driven network application framework for rapid development of maintainable high performance protocol servers & clients.
$ docker pull swift
Using default tag: latest
latest: Pulling from library/swift
6abc03819f3e: Downloading 13.91MB/28.86MB
05731e63f211: Download complete
0bd67c50d6be: Download complete
372e66a40b8a: Downloading 18.7MB/116.7MB
763e107b8a65: Downloading 18.25MB/317.4MB
$ docker run -it -rm -v ~/swift-nio:/swift-nio swift
# cd /swift-nio
# swift build
[8/36] Compiling Swift Module 'NIO' (55 sources)
$ docker run -it -rm -v ~/swift-nio:/swift-nio swift

# cd /swift-nio
# swift build
[8/36] Compiling Swift Module 'NIO' (55 sources)
Code completion
Jump-to-definition
Refactoring

SourceKit
Introducing the sourcekitd Stress Tester

February 6, 2019  Nathan Hawes

Sourcekitd provides the data backing key editor features like code completion, semantic highlighting, and refactoring for Swift files in both Xcode and the recently announced SourceKit-LSP. To help improve its robustness, we’re introducing a new tool, the sourcekitd stress tester, that over the past few months has helped find 91 reproducible sourcekitd crashes, assertion failures, and hangs. This post covers the stress tester’s implementation, its deployment in Swift’s CI and PR testing, and how Swift developers can run it over their own projects to help improve the Swift editing experience for everyone.

Some background on sourcekitd

Sourcekitd is designed to work as a service and uses a request-response model to communicate with Xcode and other clients about a set of Swift source files. Before diving into how sourcekitd is being stress tested, it’s helpful to understand the types of requests sourcekitd supports, the information they return, and what client features typically rely on that information. This is summarized for the subset of requests currently exercised by the stress tester in the table below:
“To help improve its robustness, we’re introducing a new tool, the sourcekitd stress tester, that over the past few months has helped find 91 reproducible sourcekitd crashes, assertion failures, and hangs.”
Language Server Protocol
Language Server Protocol implementation for Swift and C-based languages

Manage topics

- 279 commits
- 3 branches
- 33 releases
- 19 contributors

Branch: master

<table>
<thead>
<tr>
<th>Editors</th>
<th>Sources</th>
<th>Tests</th>
<th>Utilities</th>
<th>.gitignore</th>
<th>CODEOWNERS</th>
<th>CONTRIBUTING.md</th>
<th>LICENSE.txt</th>
<th>Package.swift</th>
<th>README.md</th>
<th>overrides.xcconfig</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add how to add code cmd in the terminal</td>
<td>Use manifest cache for faster startup</td>
<td>[test] Fix rare failure in LineTable editing test</td>
<td>Remove Package.resolved and move to HEAD</td>
<td>Move sourcekit-lsp off of SwiftPM's POSIX</td>
<td>Import SourceKit-LSP sources</td>
<td>Import SourceKit-LSP sources</td>
<td>Import SourceKit-LSP sources</td>
<td>Remove Package.resolved and move to HEAD</td>
<td>Must include full path to &quot;Block.h&quot;</td>
<td>[build] Update xcconfig overrides with workarounds from SR-9292</td>
</tr>
</tbody>
</table>

Latest commit 0:1a342 7 days ago
let x = Int.rand

random(in: ClosedRange<Int>)
random(in: ClosedRange<Int>, using: &RandomNumberGenerator)
random(in: Range<Int>)
random(in: Range<Int>, using: &RandomNumberGenerator)
Language and Standard Library

Anna Zaks, Swift Team
<table>
<thead>
<tr>
<th>Implicit returns</th>
<th>Implicit return from single expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>(SE-0255)</td>
<td>(SE-0255)</td>
</tr>
<tr>
<td>Generic math functions</td>
<td>Key path expressions as functions</td>
</tr>
<tr>
<td>(SE-0246)</td>
<td>(SE-0249)</td>
</tr>
<tr>
<td>Default values for memberwise initializer</td>
<td>Literal initialization via coercion</td>
</tr>
<tr>
<td>(SE-0242)</td>
<td>(SE-0213)</td>
</tr>
<tr>
<td>Static and class subscripts</td>
<td>Identity key path</td>
</tr>
<tr>
<td>(SE-0254)</td>
<td>(SE-0227)</td>
</tr>
<tr>
<td>Ordered Collection Diffing</td>
<td>Result type</td>
</tr>
<tr>
<td>(SE-240)</td>
<td>(SE-0235)</td>
</tr>
<tr>
<td>SIMD refinements</td>
<td></td>
</tr>
<tr>
<td>(SE-0229)</td>
<td></td>
</tr>
</tbody>
</table>
Implicit returns
(SE-0255)

Generic math functions
(SE-0246)

Default values for
memberwise initializer
(SE-0242)

Static and class subscripts
(SE-0254)

Ordered Collection Diffing
(SE-240)

SIMD refinements
(SE-0229)

Property wrappers
(SE-0258)

Key path member lookup
(SE-0252)

Dynamically "callable" types
(SE-0216)

Library evolution for
stable ABIs
(SE-0260)

String interpolation
improvements
(SE-0228)

Opaque result types
(SE-0244)

Implicit return from
single expressions
(SE-0255)

Key path expressions
as functions
(SE-0249)

Literal initialization via coercion
(SE-0213)

Identity key path
(SE-0227)

Result type
(SE-0235)
<table>
<thead>
<tr>
<th>Proposal</th>
<th>Title</th>
<th>Author</th>
<th>Review Manager</th>
<th>Implemented in</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SE-0255</td>
<td>Implicit returns from single-expression functions</td>
<td>Nate Chandler</td>
<td>Ben Cohen</td>
<td>Swift 5.1</td>
<td>swift#23251</td>
</tr>
<tr>
<td>SE-0254</td>
<td>Static and class subscripts</td>
<td>Brent Royal-Gordon</td>
<td>Doug Gregor</td>
<td>Swift 5.1</td>
<td>swift#23358</td>
</tr>
<tr>
<td>SE-0252</td>
<td>If-let, guard-let, guard-match</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Swind Evolution

11 proposals

**Implemented**
**SE-0255** Implicit returns from single-expression functions
Author: Nate Evans
Review Manager: Ben Cohen
Implementation: swift#23251

**Implemented**
**SE-0254** Static and class subscripts
Author: Brent Royal-Gordon
Review Manager: Doug Gregor
Implementation: Swift 5.1, swift#23358

[apple.github.io/swift-evolution](apple.github.io/swift-evolution)
// Implicit return from single expressions
// Swift Evolution: SE-0255

struct Rectangle {
    var width = 0.0, height = 0.0
    var area: Double {
        return width * height
    }
}
// Implicit return from single expressions
// Swift Evolution: SE-0255

struct Rectangle {
    var width = 0.0, height = 0.0
    var area: Double { width * height }
}

// Synthesized default values for the memberwise initializer
// Proposal and implementation by an open source contributor Alejandro Alonso
// Swift Evolution: SE-0242

struct Dog {
    var name = "Generic dog name"
    var age = 0
}

// Synthesized default values for the memberwise initializer
// Proposal and implementation by an open source contributor Alejandro Alonso
// Swift Evolution: SE-0242

struct Dog {
    var name = "Generic dog name"
    var age = 0
}

let boltNewborn = Dog() ✓

let daisyNewborn = Dog(name: "Daisy", age: 0) ✓

let benjiNewborn = Dog(name: "Benji") × cannot invoke initializer for type 'Dog' with an argument list of type '(name: String)'
// Synthesized default values for the memberwise initializer
// Proposal and implementation by an open source contributor Alejandro Alonso
// Swift Evolution: SE-0242

struct Dog {
    var name = "Generic dog name"
    var age = 0
}

let boltNewborn = Dog()

let daisyNewborn = Dog(name: "Daisy", age: 0)

let benjiNewborn = Dog(name: "Benji")
Types for fixed-size SIMD vectors and matrices:

\[ \text{SIMD2}\langle T \rangle, \text{SIMD3}\langle T \rangle, \text{SIMD4}\langle T \rangle, \text{SIMD8}\langle T \rangle, \text{SIMD16}\langle T \rangle, \text{SIMD32}\langle T \rangle, \text{SIMD64}\langle T \rangle \]

Most standard integer and floating-point types can be used as element types
// SIMD Vectors API

// Initialize from array literals
let x: SIMD4<Int> = [1,2,3,4]
let y: SIMD4<Int> = [3,2,1,0]

// Pointwise equality, inequality, and ordered comparisons
// Return SIMDMask type
let gr = x .> y
// gr : SIMDMask<SIMD4<Int>>(false, false, true, true)

// Boolean operations on SIMDMasks
let lteq = .!gr
// lteq : SIMDMask<SIMD4<Int>>(true, true, false, false)
// SIMD Vectors API

// Initialize from array literals
let x: SIMD4<Int> = [1,2,3,4]
let y: SIMD4<Int> = [3,2,1,0]

// Pointwise equality, inequality, and ordered comparisons
// Return SIMDMask type
let gr = x .> y
// gr : SIMDMask<SIMD4<Int>>(false, false, true, true)

// Boolean operations on SIMDMasks
let lteq = .!gr
// lteq : SIMDMask<SIMD4<Int>>(true, true, false, false)
// SIMD Vectors API

// Initialize from array literals
let x: SIMD4<Int> = [1,2,3,4]
let y: SIMD4<Int> = [3,2,1,0]

// Pointwise equality, inequality, and ordered comparisons
// Return SIMDMask type
let gr = x .> y
// gr : SIMDMask<SIMD4<Int>>(false, false, true, true)

// Boolean operations on SIMDMasks
let lteq = .!gr
// lteq : SIMDMask<SIMD4<Int>>(true, true, false, false)
// SIMD Vectors API

// Initialize from array literals
let x: SIMD4<Int> = [1,2,3,4]
let y: SIMD4<Int> = [3,2,1,0]

// Pointwise equality, inequality, and ordered comparisons
// Return SIMDMask type
let gr = x .> y
// gr : SIMDMask<SIMD4<Int>>(false, false, true, true)

// Boolean operations on SIMDMasks
let lteq = .!gr
// lteq : SIMDMask<SIMD4<Int>> (true, true, false, false)
// SIMD Vectors API

// Initialize from array literals
let x: SIMD4<Int> = [1,2,3,4]
let y: SIMD4<Int> = [3,2,1,0]

// Pointwise equality, inequality, and ordered comparisons
// Return SIMDMask type
let gr = x .> y
// gr : SIMDMask<SIMD4<Int>>(false, false, true, true)

// Boolean operations on SIMDMasks
let lteq = .!gr
// lteq : SIMDMask<SIMD4<Int>>(true, true, false, false)
New Design for String Interpolation
Swift Evolution: SE-0228

Up to 1.7x faster than Swift 4.2’s design

Extend `String` with new interpolation behaviors

Support in your own types by conforming to `ExpressibleByStringInterpolation` protocol
let quantity = 10
label.text = "You have \(\text{quantity}\) apples"
let quantity = 10

label.text = NSLocalizedString(
    "You have \(quantity) apples",
    comment: "Number of apples"
)
let quantity = 10
let formatString = NSLocalizedString(
    "You have %lld apples",
    comment: "Number of apples"
)
label.text = String(format: formatString, quantity)
let quantity = 10
let formatString = NSLocalizedString(
    "You have %lld apples",
    comment: "Number of apples"
)
label.text = String(format: formatString, quantity)
let quantity = 10

let formatString = NSLocalizedString(
    "You have %lld apples",
    comment: "Number of apples"
)

label.text = String(format: formatString, quantity)
let quantity = 10
let formatString = NSLocalizedString(
    "You have %lld apples",
    comment: "Number of apples"
)
label.text = String(format: formatString, quantity)
let quantity = 10
return Text(
    "You have \(quantity\) apples",
    comment: "Number of apples"
)
let quantity = 10
return Text(
   "You have \(quantity) apples",
   comment: "Number of apples"
)
let quantity = 10
return Text(
    "You have \(quantity) apples",
    comment: "Number of apples"
).

// In SwiftUI.framework
public struct Text {
    public init(
        _ key: LocalizedStringKey,
        tableName: String? = nil,
        bundle: Bundle? = nil,
        comment: StaticString? = nil
    )
}
let quantity = 10
return Text(
    "You have \(quantity) apples",
    comment: "Number of apples"
)
```swift
let quantity = 10
return Text(
    "You have \(quantity) apples",
    comment: "Number of apples"
)

// Generated by the Swift compiler
var builder = LocalizedStringKey.StringInterpolation(
    literalCapacity: 16, interpolationCount: 1
)

builder.appendLiteral("You have ")
builder.appendInterpolation(quantity)
builder.appendLiteral(" apples")
LocalizedStringKey(stringInterpolation: builder)
```
let quantity = 10
return Text(
    "You have \(quantity) apples",
    comment: "Number of apples"
)

// Generated by the Swift compiler
var builder = LocalizedStringKey.StringInterpolation(
    literalCapacity: 16, interpolationCount: 1
)

builder.appendLiteral("You have ")
builder.appendInterpolation(quantity)
builder.appendLiteral(" apples")

LocalizedStringKey(stringInterpolation: builder)
let quantity = 10
return Text(
    "You have \(quantity) apples",
    comment: "Number of apples"
)

// Generated by the Swift compiler
var builder = LocalizedStringKey.StringInterpolation(
    literalCapacity: 16, interpolationCount: 1
)

builder.appendLiteral("You have ")
builder.appendInterpolation(quantity)
builder.appendLiteral(" apples")

LocalizedStringKey(stringInterpolation: builder)
let quantity = 10
return Text(
    "You have \(quantity) apples",
    comment: "Number of apples"
)

// Generated by the Swift compiler
var builder = LocalizedStringKey.StringInterpolation(
    literalCapacity: 16, interpolationCount: 1
)

builder.appendLiteral("You have ")
builder.appendInterpolation(quantity)
builder.appendLiteral(" apples")

LocalizedStringKey(stringInterpolation: builder)
let quantity = 10
return Text(
    "You have \(quantity) apples",
    comment: "Number of apples"
)

// Generated by the Swift compiler
var builder = LocalizedStringKey.StringInterpolation(
    literalCapacity: 16, interpolationCount: 1
)

builder.appendLiteral("You have ")
builder.appendInterpolation(quantity)
builder.appendLiteral(" apples")

LocalizedStringKey(stringInterpolation: builder)
let quantity = 10
return Text(
    "You have \(quantity) apples",
    comment: "Number of apples"
)

// Generated by the Swift compiler
var builder = LocalizedStringKey.StringInterpolation(
    literalCapacity: 16, interpolationCount: 1
)

builder.appendLiteral("You have ")
builder.appendInterpolation(quantity)
builder.appendLiteral(" apples")

LocalizedStringKey(stringInterpolation: builder)
let quantity = 10
return Text(
    "You have \(quantity) apples",
    comment: "Number of apples"
)
let quantity = 10
return Text(
    "You have \(quantity) apples",
    comment: "Number of apples"
)
New Design for String Interpolation
Swift Evolution: SE-0228

Framework and package authors are already at work

Read the `ExpressibleByStringInterpolation` documentation to get started
Reasons to Abstract a Return Type
Reasons to Abstract a Return Type

API returns different types conforming to the same protocol
Reasons to Abstract a Return Type

API returns different types conforming to the same protocol

API returns the same type but is leaking implementation details
// Shapes Example

protocol Shape { /* ... */ }
struct Square: Shape { /* ... */ }
struct Circle: Shape { /* ... */ }
struct Oval: Shape { /* ... */ }

struct Union<A: Shape, B: Shape>: Shape { /* ... */ }
struct Transformed<S: Shape>: Shape { /* ... */ }
// API returns different types conforming to the same protocol
// Use Protocol

struct FaceShape {
    ...  
    var shape: Shape {
        switch faceType {
        case .round:  
            return Circle()
        case .square:  
            return Square()
        case .diamond:  
            return Transformed(Square(), by: .fortyFiveDegrees))
        default:  
            return Oval()
        }
    }
}
// API returns different types conforming to the same protocol
// Use Protocol

struct FaceShape {
    ...
    var shape: Shape {
        switch faceType {
        case .round:
            return Circle()
        case .square:
            return Square()
        case .diamond:
            return Transformed(Square(), by: .fortyFiveDegrees)
        default:
            return Oval()
        }
    }
}
// API returns different types conforming to the same protocol
// Use Protocol

struct FaceShape {
    ...
    var shape: Shape {
        switch faceType {
            case .round:
                return Circle()
            case .square:
                return Square()
            case .diamond:
                return Transformed(Square(), by: .fortyFiveDegrees)
            default:
                return Oval()
        }
    }
}
// API returns the same type but is leaking implementation details

struct EightPointedStar {
    ...
    var shape: Union<Square, Transformed<Square>> {
        return Union(Square(), Transformed(Square(), by: .fortyFiveDegrees))
    }
}
// API returns the same type but is leaking implementation details

struct EightPointedStar {
    ...
    var shape: Union<Square, Transformed<Square>> {
        return Union(Square(), Transformed(Square(), by: .fortyFiveDegrees))
    }
}
// API returns the same type but is leaking implementation details
// Protocol type?

struct EightPointedStar {
    ...  
    var shape: Shape {
        return Union(Square(), Transformed(Square(), by: .fortyFiveDegrees))
    }
}
// API returns the same type but is leaking implementation details
// Protocol type?

```swift
struct EightPointedStar {
    ...
    var shape: Shape {
        return Union(Square(), Transformed(Square(), by: .fortyFiveDegrees))
    }
}
```
Limitations of Returning a Protocol Type

Loses type identity

Does not compose well with the generics system:

- `eightPointedStarOne == eightPointedStarTwo` (Operator ‘==’ cannot be applied)

- Returned type cannot have any associated types

- Returned type cannot have requirements that involve `Self`

Disables optimizations
// API returns the same type but is leaking implementation details
// Use Opaque Result Types

struct EightPointedStar {
    ...
    var shape: some Shape {
        return Union(Square(), Transformed(Square(), by: .fortyFiveDegrees))
    }
}
// Opaque Result Types
// Compiler enforces that the same type is returned from the implementation

struct EightPointedStar {
    var shape: some Shape { // Opaque Result Types but the return statements do not have matching types
        if symmetrical {
            return Union(Square(), Transformed(Square(), by: .fortyFiveDegrees))
        } else {
            return Transformed(Square(), by: .twentyDegrees)
        }
    }
}
Opaque Result Types
Swift Evolution: SE-0244

Use when API returns the same type but is leaking implementation details

Requires new Swift runtime support
  Available on macOS Catalina, iOS 13, tvOS 13, watchOS 6 and later
  Guard uses with availability checking when deploying to earlier OS releases
Property Wrapper Types
Reuse code for property access patterns

Lazily initialized values
Thread-local storage
Copy on write objects
Data dependencies in SwiftUI
User defaults
static var `usesTouchID`: Bool {
    get {
        return UserDefaults.standard.bool(forKey: "USES_TOUCH_ID")
    }
    set {
        UserDefaults.standard.set(newValue, forKey: "USES_TOUCH_ID")
    }
}

static var `isLoggedIn`: Bool {
    get {
        return UserDefaults.standard.bool(forKey: "LOGGED_IN")
    }
    set {
        UserDefaults.standard.set(newValue, forKey: "LOGGED_IN")
    }
}
// The purpose of property wrappers is to wrap a property, specify its access patterns

@propertyWrapper
struct UserDefault<T> {
    let key: String
    let defaultValue: T

    init(_ key: String, defaultValue: T) {
        ...
        UserDefaults.standard.register(defaults: [key: defaultValue])
    }

    var value: T {
        get {
            return UserDefaults.standard.object(forKey: key) as? T ?? defaultValue
        }
        set {
            UserDefaults.standard.set(newValue, forKey: key)
        }
    }
}
The purpose of property wrappers is to wrap a property, specify its access patterns

```swift
@propertyWrapper
struct UserDefaults<T> {
    let key: String
    let defaultValue: T

    init(_ key: String, defaultValue: T) {
        ...
        UserDefaults.standard.register(defaults: [key: defaultValue])
    }

    var value: T {
        get {
            return UserDefaults.standard.object(forKey: key) as? T ?? defaultValue
        }
        set {
            UserDefaults.standard.set(newValue, forKey: key)
        }
    }
}
```
// The purpose of property wrappers is to wrap a property, specify its access patterns

@propertyWrapper
struct UserDefault<T> {
    let key: String
    let defaultValue: T
    init(_ key: String, defaultValue: T) {
        ...
        UserDefaults.standard.register(defaults: [key: defaultValue])
    }
    var value: T {
        get {
            return UserDefaults.standard.object(forKey: key) as? T ?? defaultValue
        }
        set {
            UserDefaults.standard.set(newValue, forKey: key)
        }
    }
}
/ The purpose of property wrappers is to wrap a property, specify its access patterns

@propertyWrapper
struct UserDefaults<T> {
    let key: String
    let defaultValue: T

    init(_ key: String, defaultValue: T) {
        ...
        UserDefaults.standard.register(defaults: [key: defaultValue])
    }

    var value: T {
        get {
            return UserDefaults.standard.object(forKey: key) as? T ?? defaultValue
        }
        set {
            UserDefaults.standard.set(newValue, forKey: key)
        }
    }
}
// The purpose of property wrappers is to wrap a property, specify its access patterns

@-propertyWrapper
struct UserDefault<T> {  
    let key: String  
    let defaultValue: T  
    init(_ key: String, defaultValue: T) {  
        ...  
        UserDefaults.standard.register(defaults: [key: defaultValue])  
    }  
    var value: T {  
        get {  
            return UserDefaults.standard.object(forKey: key) as? T ?? defaultValue  
        }  
        set {  
            UserDefaults.standard.set(newValue, forKey: key)  
        }  
    }  
}

Allows writing: @UserDefault(...) var x: Bool
// Using UserDefaults property wrapper to declare and access properties

@UserDefaults("USES_TOUCH_ID", defaultValue: false)
static var usesTouchID: Bool

@UserDefaults("LOGGED_IN", defaultValue: false)
static var isLoggedIn: Bool
// Using UserDefaults property wrapper to declare and access properties

@UserDefaults("USES_TOUCH_ID", defaultValue: false)
static var usesTouchID: Bool

@UserDefaults("LOGGED_IN", defaultValue: false)
static var isLoggedIn: Bool
// Using UserDefaults property wrapper to declare and access properties

@UserDefaults("USES_TOUCH_ID", defaultValue: false)
static var usesTouchID: Bool

@UserDefaults("LOGGED_IN", defaultValue: false)
static var isLoggedIn: Bool

if !isLoggedIn && usesTouchID {
    authenticateWithTouchID()
}
Property Wrapper Types
Swift Evolution: SE-0258

Wrapper types define custom access patterns

Property can adopt by adding an attribute to its declaration
Welcome Message

Welcome to WWDC19!

{ "message": "Welcome to WWDC19!" }
<html>
<head>
    <title>Jessica’s WWDC19 Blog</title>
</head>
<body>
    <h2>Welcome to Jessica’s WWDC19 Blog!</h2>
    <br>
    Check out the talk schedule and latest news from <a href="https://developer.apple.com/wwdc19/">the source</a>
</body>
</html>
Tools Support is Vital

"""<html>
   <head>
      <title>\(name)'s WWDC19 Blog</title>
   </head>
   <body>
      <h2>Welcome to \(name)'s WWDC19 Blog!</h2>
      <br>
      Check out the talk schedule and latest news from <a href="https://developer.apple.com/wwdc19/">the source</a>
   </body>
"""
Welcome to (name)’s WWDC19 Blog!

Check out the talk schedule and latest news from the source
Welcome to \(name)’s WWDC19 Blog!

Check out the talk schedule and latest news from [the source](https://developer.apple.com/wwdc19/)!
Tools Support is Vital

Welcome to \(name\)’s WWDC19 Blog!

Check out the talk schedule and latest news from the source.

https://developer.apple.com/wwdc19/
Welcome to (name)'s WWDC19 Blog!

Check out the talk schedule and latest news from the source.
Welcome to \(name\)'s WWDC19 Blog!

Check out the talk schedule and latest news from the source.
html {
  head {
    title("(name)'s WWDC19 Blog")
  }
}
body {
  h2 { "Welcome to (name)'s WWDC19 Blog!" }
  br()
  "Check out the talk schedule and latest news from "
  a {
    "the source"
}
Define Embedded DSLs in Swift

```html
html {
    head {
        title("\(name)'s WWDC19 Blog")
    }
}

body {
    h2 { "Welcome to \(name)'s WWDC19 Blog!" }
    br()
    "Check out the talk schedule and latest news from "
    a {
        "the source"
}
}``
Welcome to \(\text{name}\)'s WWDC19 Blog!

Check out the talk schedule and latest news from the source.

https://developer.apple.com/wwdc19/
Welcome to \(name)'s WWDC19 Blog!

Check out the talk schedule and latest news from "the source"

If not logged in, log in for comments.
Welcome to \(name)'s WWDC19 Blog!

Check out the talk schedule and latest news from the source.

Log in for comments.
// Functions that construct the HTML objects

public func html(@HTMLBuilder content: () -> HTML) -> HTML {
    ...
}

public func head(@HTMLBuilder content: () -> HTML) -> HTML {
    ...
}

public func body(@HTMLBuilder content: () -> HTML) -> HTML {
    ...
}
// Functions that construct the HTML objects

public func html(@HTMLBuilder content: () -> HTML) -> HTML { ... }
public func head(@HTMLBuilder content: () -> HTML) -> HTML { ... }
public func body(@HTMLBuilder content: () -> HTML) -> HTML { ... }
// Functions that construct the HTML objects

public func html(@HTMLBuilder content: () -> HTML) -> HTML { ... }
public func head(@HTMLBuilder content: () -> HTML) -> HTML { ... }
public func body(@HTMLBuilder content: () -> HTML) -> HTML { ... }

Custom attribute backed by HTMLBuilder type
Compiler Transforming an `@HTMLBuilder` Closure

```java
head {
    meta().charset("UTF-8")
    if cond {
        title("Title 1")
    } else {
        title("Title 2")
    }
}
```
Compiler Transforming an @HTMLBuilder Closure

```javascript
head {
    meta().charset("UTF-8")
    if cond {
        title("Title 1")
    } else {
        title("Title 2")
    }
}
```
Compiler Transforming an @HTMLBuilder Closure

```swift
head {
    meta().charset("UTF-8")
    if cond {
        title("Title 1")
    } else {
        title("Title 2")
    }
}

head {
    let a: HTML = meta().charset("UTF-8")
    let d: HTML
    if cond {
        let b: HTML = title("Title 1")
        d = HTMLBuilder.buildEither(first: b)
    } else {
        let c: HTML = title("Title 2")
        d = HTMLBuilder.buildEither(second: c)
    }
    return HTMLBuilder.buildBlock(a, d)
}
```
Compiler Transforming an `@HTMLBuilder` Closure

```swift
head {
    meta().charset("UTF-8")
    if cond {
        title("Title 1")
    } else {
        title("Title 2")
    }
}

head {
    let a: HTML = meta().charset("UTF-8")
    let d: HTML
    if cond {
        let b: HTML = title("Title 1")
        d = HTMLBuilder.buildEither(first: b)
    } else {
        let c: HTML = title("Title 2")
        d = HTMLBuilder.buildEither(second: c)
    }
    return HTMLBuilder.buildBlock(a, d)
}
```
Compiler Transforming an `@HTMLBuilder` Closure

```swift
head {
    meta().charset("UTF-8")
    if cond {
        title("Title 1")
    } else {
        title("Title 2")
    }
}

head {
    let a: HTML = meta().charset("UTF-8")
    let d: HTML
    if cond {
        let b: HTML = title("Title 1")
        d = HTMLBuilder.buildEither(first: b)
    } else {
        let c: HTML = title("Title 2")
        d = HTMLBuilder.buildEither(second: c)
    }
    return HTMLBuilder.buildBlock(a, d)
}
```
// Embedded Swift DSL used in SwiftUI

HStack {
    VStack {
        Text("★★★★★").color(.red)
        Text("5 stars")
    }.font(.caption)

    VStack(alignment: .leading) {
        HStack {
            Text("Avocado Toast").font(.title)
            Spacer()
            Image("avocado_small")
        }
        Text("Ingredients: Avocado, Almond Butter, Bread")
            .font(.caption)
    }
}
Swift DSL is waiting for your feedback!
The Swift Forums are governed by the Swift Code of Conduct.

**Announcements**
This category is for announcements of Swift evolution proposal reviews and results, as well as other...

**Pitches**
The Pitches category is an area for pitching ideas for evolution of the Swift language prior to a formal review.

**Proposal Reviews**
This category is for posting Swift Evolution proposals for review and feedback.

**Discussion**
The Evolution Discussion category is for general discussion of the evolution of the Swift language.

---

<table>
<thead>
<tr>
<th>Topic</th>
<th>Replies</th>
<th>Views</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introducing Sequence.last?</td>
<td>6</td>
<td>65</td>
<td>1h</td>
</tr>
<tr>
<td>RFC: making Swift.org a more valuable resource for the Swift community</td>
<td>50</td>
<td>7.2k</td>
<td>16h</td>
</tr>
<tr>
<td>Adding isPowerOf2 to BinaryInteger</td>
<td>64</td>
<td>2.3k</td>
<td>17h</td>
</tr>
<tr>
<td>[Pre-pitch] removeAll(at:)</td>
<td>2</td>
<td>221</td>
<td>18h</td>
</tr>
<tr>
<td>Remove duplicate elements from a collection</td>
<td>28</td>
<td>1.1k</td>
<td>20h</td>
</tr>
<tr>
<td>Implement buffer pointer’s “MutableCollection” methods as nonmutating</td>
<td>3</td>
<td>144</td>
<td>22h</td>
</tr>
<tr>
<td>Topic</td>
<td>Views</td>
<td>Activity</td>
<td></td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>------</td>
<td>----------</td>
<td></td>
</tr>
<tr>
<td>Introducing Sequence.last?</td>
<td>6</td>
<td>1h</td>
<td></td>
</tr>
<tr>
<td>RFC: making Swift.org a more valuable resource for the Swift community</td>
<td>50</td>
<td>18h</td>
<td></td>
</tr>
<tr>
<td>Adding isPowerOf2 to BinaryInteger</td>
<td>64</td>
<td>17h</td>
<td></td>
</tr>
<tr>
<td>[Pre-pitch] removeAll(at:)</td>
<td>2</td>
<td>18h</td>
<td></td>
</tr>
<tr>
<td>Remove duplicate elements from a collection</td>
<td>28</td>
<td>20h</td>
<td></td>
</tr>
<tr>
<td>Implement buffer pointer's &quot;MutableCollection&quot; methods as nonmutating</td>
<td>3</td>
<td>22h</td>
<td></td>
</tr>
</tbody>
</table>

forums.swift.org/c/evolution
Make Your Swift APIs Better

Expressive

Clear

Easy to use
Make Your Swift APIs Better

Expressive

Clear

Easy to use

Modern Swift API Design

Thursday, 2:00PM
### More Information

[developer.apple.com/wwdc19/402](developer.apple.com/wwdc19/402)

<table>
<thead>
<tr>
<th>Title</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating Swift Packages</td>
<td>Thursday, 9:00AM</td>
</tr>
<tr>
<td>Modern Swift API Design</td>
<td>Thursday 2:00PM</td>
</tr>
<tr>
<td>Binary Frameworks in Swift</td>
<td>Thursday, 3:00PM</td>
</tr>
</tbody>
</table>