

Swift 6

A Guide to Concurrency

INTRO

Kanagasabapathy Rajkumar

Senior iOS Developer

Apple Platforms Developer since 2014

Experienced in Swift, UIKit, SwiftUI, and Protocol-Oriented Programming

Specialized in Concurrency and Swift Package Management



Prepare to Say Goodbye to

- 🚫 Callback hell
- 🚫 Error-Handling Headaches
- 🚫 Thread-Safety Issues
- 🚫 Complicated Synchronization
- 🚫 Unreadable Asynchronous Code

Thanks to Swift 5.5 and Enhanced in Swift 6

Embrace...

-  Async/Await
-  Structured Concurrency
-  Actors Isolation
-  Tasks
-  Cleaner Code
-  Strict Concurrency

Thanks to Swift 5.5 and Enhanced in Swift 6

Evolution of Concurrency APIs



NSThread - Manual thread management

GCD - Task management with queues

Operation Queues - Task dependencies and ordering

Combine - Reactive programming model

Swift 5 - Async/Await & Structured Concurrency

Swift 6 - Complete Currency by default and Data Race Safety

Concurrency Roadmap

 Features

 Version History

 Advantages and Disadvantages

 Practical Examples

Topics on the Agenda

 Async/Await

 Structured Concurrency

 Tasks

 Actors

 Sendable

Async/Await

Basic Syntactic Building Block

 **Suspension Points:** Mark functions as `async` to enable the use of suspension points within the function.

 **Awaiting Results:** The `await` keyword introduces a suspension point, where the function pauses until the asynchronous operation completes.

 **Error Handling:** Integrates seamlessly with `do-catch` for handling errors in asynchronous functions.

Async/Await Flow



Async/Await Practical Examples

```
// Define a function that simulates an asynchronous task
func performTask(taskNumber: Int) async {
    print("Task \(taskNumber) started.")
    do {
        try await Task.sleep(nanoseconds:
1_000_000_000) // Simulate some asynchronous work (1 second)
        print("Task \(taskNumber) completed.")
    } catch {
        print(error)
    }
}
```

```
// Async-await Networking Example with Generics
func sendRequest<T>(urlStr: String) async throws -> T where T : Decodable {
    guard let urlStr = urlStr as String?, let url = URL(string: urlStr) as URL? else {
        throw NetworkError.invalidURL
    }
    let (data, response) = try await URLSession.shared.data(from: url)
    guard let response = response as? HTTPURLResponse, 200...299 ~=
response.statusCode else {
        throw NetworkError.unexpectedStatusCode
    }
    guard let data = data as Data? else {
        throw NetworkError.unknown
    }
    guard let decodedResponse = try? JSONDecoder().decode(T.self, from: data) else {
        throw NetworkError.decode
    }
    return decodedResponse
}
```

Pros & Cons

Simplifying Asynchronous Programming

Streamlining Concurrency and Alternative to Callback/Closures

- Introduced: iOS 15

Changes

- Swift 5: Not available
- Swift 5.5: Introduction of Async/Await - **Proposal SE-0296**
- Swift 6: Continued improvements and complementary use with GCD, OperationQueue, and Combine.

Advantages

- Readability, Error Handling and alternative to Completion Handler/Closures
- Reduces complexity associated with Callback Hell or nested closures
- Reduces chance of race conditions or deadlocks

Disadvantages

- Limited backward compatibility
- Complex Integrations when interacting with existing GCD or OperationQueue



**CALLBACK
HELL**

ASYNC/AWAIT

Tasks

Fundamental Unit of Concurrent Work

 Sequential, Asynchronous and Self-contained

 **Task Groups:** Use TaskGroup to manage collections of tasks that run concurrently and complete as a group.

 **Task Prioritization:** Assign priorities to tasks (e.g., .userInitiated, .background) to optimize performance.

 **Task Cancellation:** Easily cancel tasks using the cancel() method, propagating cancellation to child tasks.

```
Task(priority: .|, operation: () async → _)
```

- M high
- M background
- M low
- M medium
- M userInitiated
- M utility
- K none
- K some()

high: TaskPriority

Tasks

```
Task(priority: .high) {  
    print("high Task Started")  
    try? await Task.sleep(nanoseconds: 1_000_000_000)  
    print("high Task Completed")  
}  
  
Task(priority: .userInitiated) {  
    print("userInitiated Task Started")  
    try? await Task.sleep(nanoseconds: 1_000_000_000)  
    print("userInitiated Task Completed")  
}  
  
Task(priority: .medium) {  
    print("medium Task Started")  
    try? await Task.sleep(nanoseconds: 1_000_000_000)  
    print("medium Task Completed")  
}  
  
Task(priority: .low) {  
    print("low Task Started")  
    try? await Task.sleep(nanoseconds: 1_000_000_000)  
    print("low Task Completed")  
}  
  
Task(priority: .utility) {  
    print("utility Task Started")  
    try? await Task.sleep(nanoseconds: 1_000_000_000)  
    print("utility Task Completed")  
}  
  
Task(priority: .background) {  
    print("Background Task Started")  
    try? await Task.sleep(nanoseconds: 1_000_000_000)  
    print("Background Task Completed")  
}
```

```
// Independent - Useful when you start the task that isn't tied  
// to current context.  
Task.detached {  
    print("Detached Task started at \(Date())")  
    try? await Task.sleep(nanoseconds: 2 * 1_000_000_000)  
    print("Detached Task completed at \(Date())")  
}  
  
// Group Tasks  
func stucturedConcurrencyWithTaskGroup() async {  
    await withTaskGroup(of: String.self) { group in  
        for i in 1...3 {  
            group.addTask {  
                try? await Task.sleep(nanoseconds: UInt64(i) *  
1_000_000_000)  
                return "Group Task \(i) completed"  
            }  
        }  
        for await result in group {  
            print(result)  
        }  
    }  
}  
Task {  
    await stucturedConcurrencyWithTaskGroup()  
}
```

```
// Task Cancellation  
func taskWithCancellation() async {  
    let task = Task {  
        for i in 1...3 {  
            try? await Task.sleep(nanoseconds:  
1_000_000_000)  
            try await Task.checkCancellation()  
            print("Task iteration \(i) completed")  
        }  
        return "Task completed"  
    }  
    DispatchQueue.global().asyncAfter(deadline:  
.now() + 2) {  
        task.cancel()  
    }  
    do {  
        let result = try await task.value  
        print(result)  
    } catch {  
        print("Task was canceled")  
    }  
}  
Task {  
    await taskWithCancellation()  
}  
Thread.sleep(forTimeInterval: 7)
```

Pros & Cons

The Building Block of Swift Concurrency

Fundamental Unit of Concurrent Work

- Introduced: iOS 15

Changes

- *Swift 5*: Not available
- *Swift 5.5*: Introduction of Tasks - **Proposal [SE-0304](#)**
- *Swift 6*: Improvements in Task Management and Task Isolation

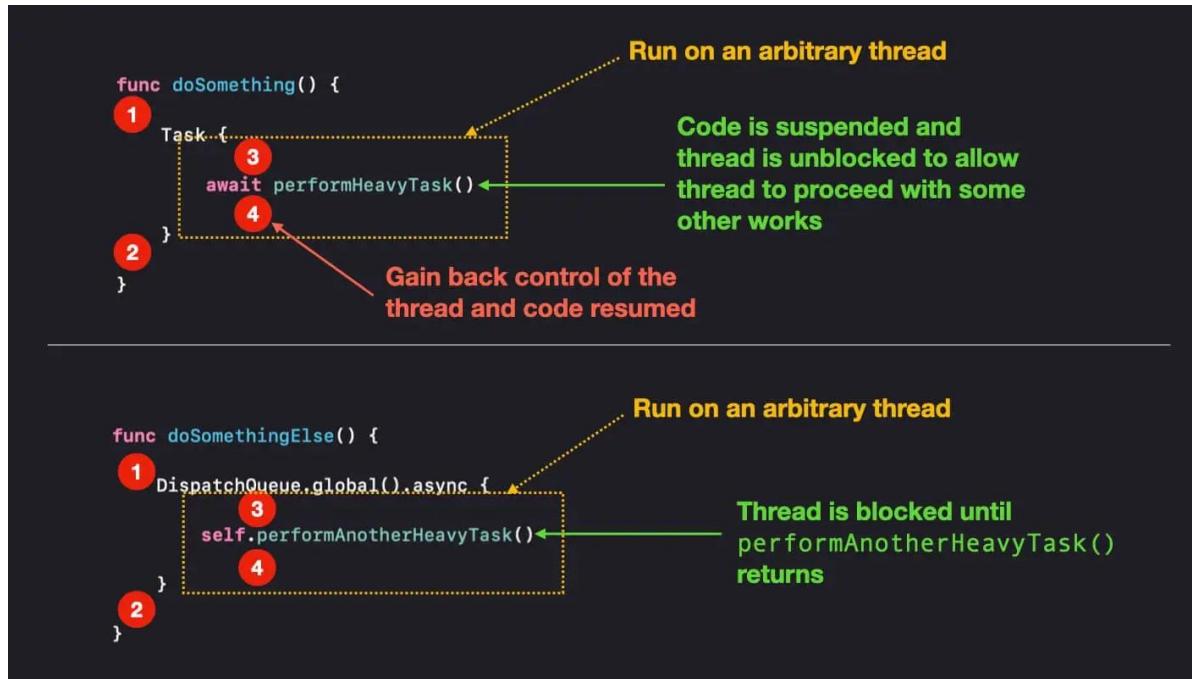
Advantages

- Built-in Cancellation
- Works well with `async-await`
- Reducing common concurrency issues such as Thread explosion and so on

Disadvantages

- Managing multiple tasks
- Limited backward compatibility
- Complex scenarios such as cancellation and task groups

Task vs DispatchQueue



Task vs DispatchQueue

Grand Central Dispatch

```
var isCancelled: Bool = false  
DispatchQueue.global().async { [weak self] in  
    Thread.sleep(forTimeInterval: 30)  
    guard let self else { return }  
    guard !self.isCancelled else { return }  
    self.continueWithMyLogic()  
}  
  
isCancelled = true
```



Manual state management

Non-throwing suspension

No automatic cancellation support

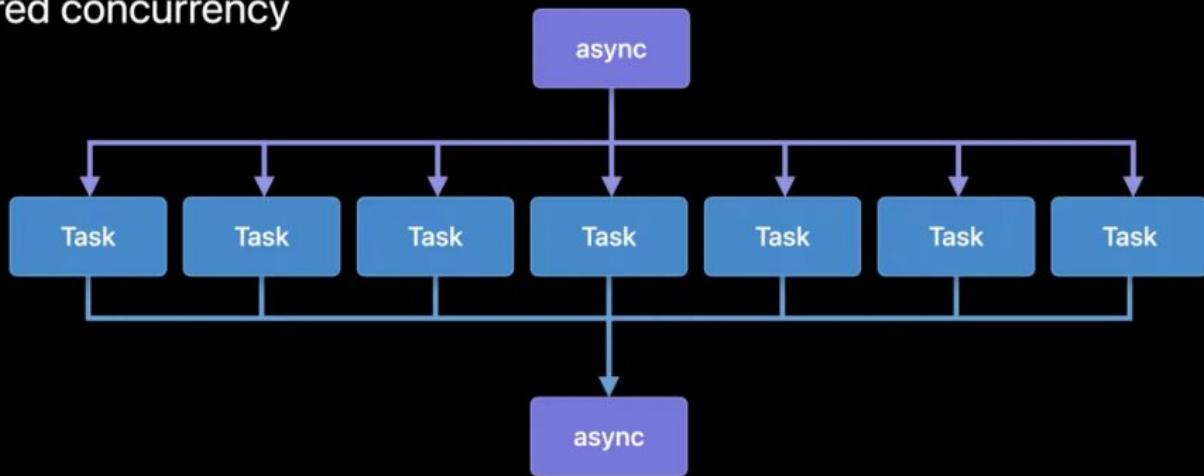
Structured Concurrency

Organized and Efficient Task Management

-  **Thread Safety:** Structured concurrency ensures that all spawned tasks are properly handled within their scope.
-  **Task Lifespan Management:** Automatically handle the lifecycle of tasks, cleaning up resources when tasks complete.
-  **Error Propagation:** Any error in child tasks is automatically propagated up to the parent task, ensuring robust error handling.

Swift Concurrency

Structured concurrency



Structured Concurrency

```
private let baseURL = "https://picsum.photos/"
private(set) var images: [UIImage] = [UIImage]()
private func fetchImage(imageUrl: String) async throws -> UIImage {
    do {
        guard let url = URL(string: baseURL + imageUrl) else {
            throw URLError(.badURL)
        }
        print("Val: \(imageUrl)")
        let (data, _) = try await URLSession.shared.data(from: url, delegate: nil)
        if let image = UIImage(data: data) {
            print(image)
            return image
        } else {
            throw URLError(.badURL)
        }
    } catch {
        if let error = error as? URLError, error.code == URLError.cancelled {
            print(error)
        } else {
            throw error
        }
    }
    return UIImage()
}
```

// Output:
Val: 104
Val: 101
Val: 103
Val: 100
Val: 102

```
func fetchMultipleImagesArray() async throws {
    async let image100 = fetchImage(imageUrl:"100")
    async let image101 = fetchImage(imageUrl:"101")
    async let image102 = fetchImage(imageUrl:"102")
    async let image103 = fetchImage(imageUrl:"103")
    async let image104 = fetchImage(imageUrl:"104")
    let asyncLetResult = try await
([image100,image101,image102,image103,image104])
        await MainActor.run {
            images.append(contentsOf: asyncLetResult)
        }
}
Task {
    do {
        try await fetchMultipleImagesArray()
    } catch {
        print("Failed to download images: \(error)")
    }
}
```

Image: <UIImage:0x6000030018c0 anonymous {104, 104} renderingMode=automatic(original)>
Image: <UIImage:0x600003001830 anonymous {101, 101} renderingMode=automatic(original)>
Image: <UIImage:0x600003009170 anonymous {102, 102} renderingMode=automatic(original)>
Image: <UIImage:0x600003009200 anonymous {100, 100} renderingMode=automatic(original)>
Image: <UIImage:0x6000030017a0 anonymous {103, 103} renderingMode=automatic(original)>

Pros & Cons

Organizing Concurrent Tasks

Ensuring Safe and Predictable Task Management

- Introduced: iOS 15

Changes

- Swift 5: Not Available
- Swift 5.5: Introduction of Structured Concurrency - Proposal [SE-0304](#)
- Swift 6: Enhanced support and refinements for structured concurrency.

Advantages

- Safety and Predictability
- Integration with Async/Await, Async-let - Proposal [SE-0317](#)
- Error Handling and Isolation of concurrent operations

Disadvantages

- Limited backward compatibility
- Complexity in Large codebases

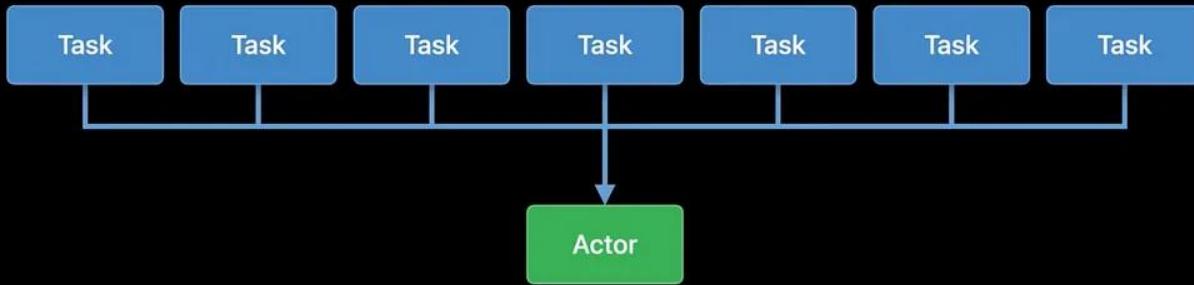
Actors

Coordination and Safety in Concurrency

-  **Task Coordination:** Orchestrate/Coordinate multiple tasks with controlled access to shared data.
-  **Data Isolation:** Isolates internal state, allowing only one task at a time to modify it.
-  **Concurrency Safety:** Prevents data races by managing concurrent mutations effectively.

Swift Concurrency

Actors



Class vs Actors

```
class Counter {  
    var value = 0  
    func increment() -> Int {  
        let currentValue = value  
        Thread.sleep(forTimeInterval: 0.1)  
        value = currentValue + 1  
        return value  
    }  
}  
  
let counter = Counter()  
Task.detached {  
    print("Class 🎾 \\"(counter.increment())\"")  
}  
Task.detached {  
    print("Class ⚾ \\"(counter.increment())\"")  
}
```



```
actor SafeCounter {  
    var value = 0  
    func increment() -> Int {  
        let currentValue = value  
        Thread.sleep(forTimeInterval: 0.1)  
        value = currentValue + 1  
        return value  
    }  
}  
  
let safeCounter = SafeCounter()  
Task.detached {  
    print("Actor ⚽ \\"await safeCounter.increment()\"")  
}  
Task.detached {  
    print("Actor ⚾ \\"await safeCounter.increment()\"")  
}
```



Pros & Cons

Safeguarding Shared State

Isolating State to Prevent Data Races

- Introduced: iOS 15

Changes

- Swift 5: Not available
- Swift 5.5: Introduction of Actors - **Proposal [SE-0306](#)**
- Swift 6: Actor Isolation

Advantages

- Data safety and simplicity
- Reference type that helps in concurrency environment
- Integration with async/await and Task and helps in Isolation

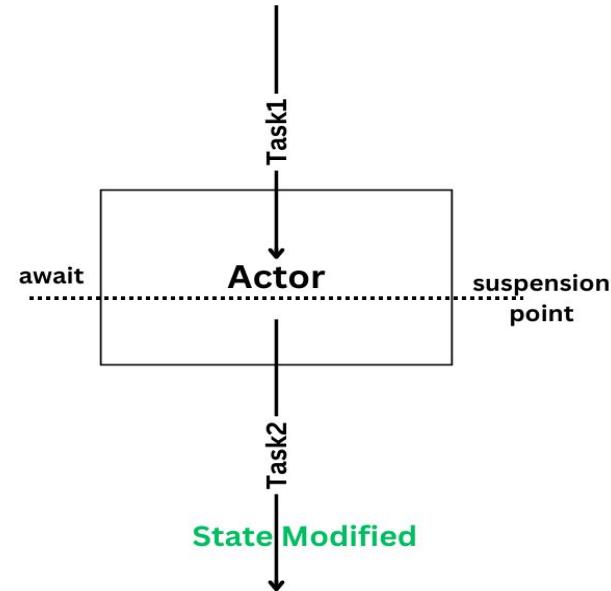
Disadvantages

- Actor Reentrancy
- Limited backward compatibility
- Complexity in Large codebases

Actor Reentrancy

- An actor is suspended during a task
- Another task enters the actor while the first task is suspended
- Causes by actor's reentrant behavior

Actor Reentrancy



Sendable

A *thread-safe type whose values can be shared across arbitrary concurrent contexts without introducing a risk of data races.*

Sendable and Sendable Closures [SE-0302](#)

Sendable == Thread Safety

1. Introduced in Swift 5.7
2. Sendable is just a **Protocol**
3. Most base types of Swift Foundation are **Sendable**

```
@available(macOS 10.10, iOS 8.0, watchOS 2.0, tvOS 9.0, *)  
public struct Date : Comparable, Hashable, Equatable, Sendable {
```



Files



main



Go to file



> proposal-templates

proposals

0001-keywords-as-argument-la...

0002-remove-currying.md

0003-remove-var-parameters.md

0004-remove-pre-post-inc-decr...

0005-objective-c-name-translat...

0006-apply-api-guidelines-to-th...

0007-remove-c-style-for-loops....

0008-lazy-flatmap-for-optionals...

0009-require-self-for-accessing...

0010-add-staticstring-unicodes...

0011-replace-typealias-associat...

0012-add-noescape-to-public-li...

0013-remove-partial-application...

0014-constrained-AnySequence...

0015-tuple-comparison-operato...

0016-initializers-for-converting...

swift-evolution / proposals / 0302-concurrent-value-and-concurrent-closures.md



shahmishal Update the URLs from apple/swift-evolution to swiftlang/swift-evolution

821970a · 2 months ago



Preview

Code

Blame

674 lines (467 loc) · 43.3 KB

Raw



Sendable and @Sendable closures

- Proposal: [SE-0302](#)
- Authors: [Chris Lattner](#), [Doug Gregor](#)
- Review Manager: [John McCall](#)
- Status: **Implemented (Swift 5.7)**
- Implementation: [apple/swift#35264](#)
- Major Contributors: Dave Abrahams, Paul Cantrell, Matthew Johnson, John McCall
- Review: ([first review](#)) ([revision announcement](#)) ([second review](#)) ([acceptance](#))

Contents

- [Introduction](#)
- [Motivation](#)
 - [Swift Value Semantics](#)
 - [Value Semantic Composition](#)
 - [Higher Order Functional Programming](#)
 - [Immutable Classes](#)
 - [Internally Synchronized Reference Types](#)

Sendable

```
// Value Types
struct User: Codable, Sendable {
    let name: String
    let age: Int
}

/// Reference types that internally
/// manage access to their state
actor Account {
    private var balance: Int = 0
    func deposit(amount: Int) {
        balance += amount
    }
    func getBalance() -> Int {
        return balance
    }
}
```

```
// Functions and Closures- @Sendable
func fetchData(endPoint: String, resultHandler: @Sendable
    @escaping (Result<User, Error>) -> Void) {
    resultHandler(.success(User(name: "Saba", age: 1)))
}

/// Reference types with no mutable storage
final class Address: Sendable {
    let houseNo: Int
    let streetName: String
    let cityName: String
    init(houseNo: Int, streetName: String, cityName: String) {
        self.houseNo = houseNo
        self.streetName = streetName
        self.cityName = cityName
    }
}
```

What is Isolation

- Mechanism - used to make **data races impossible**
- Deals with accessing mutable state in **unsafe ways** **not all** data races
- Isolation can change when you use **await** keyword
- Isolation applies to **variables, functions, closures, protocols**

Isolation Flavours

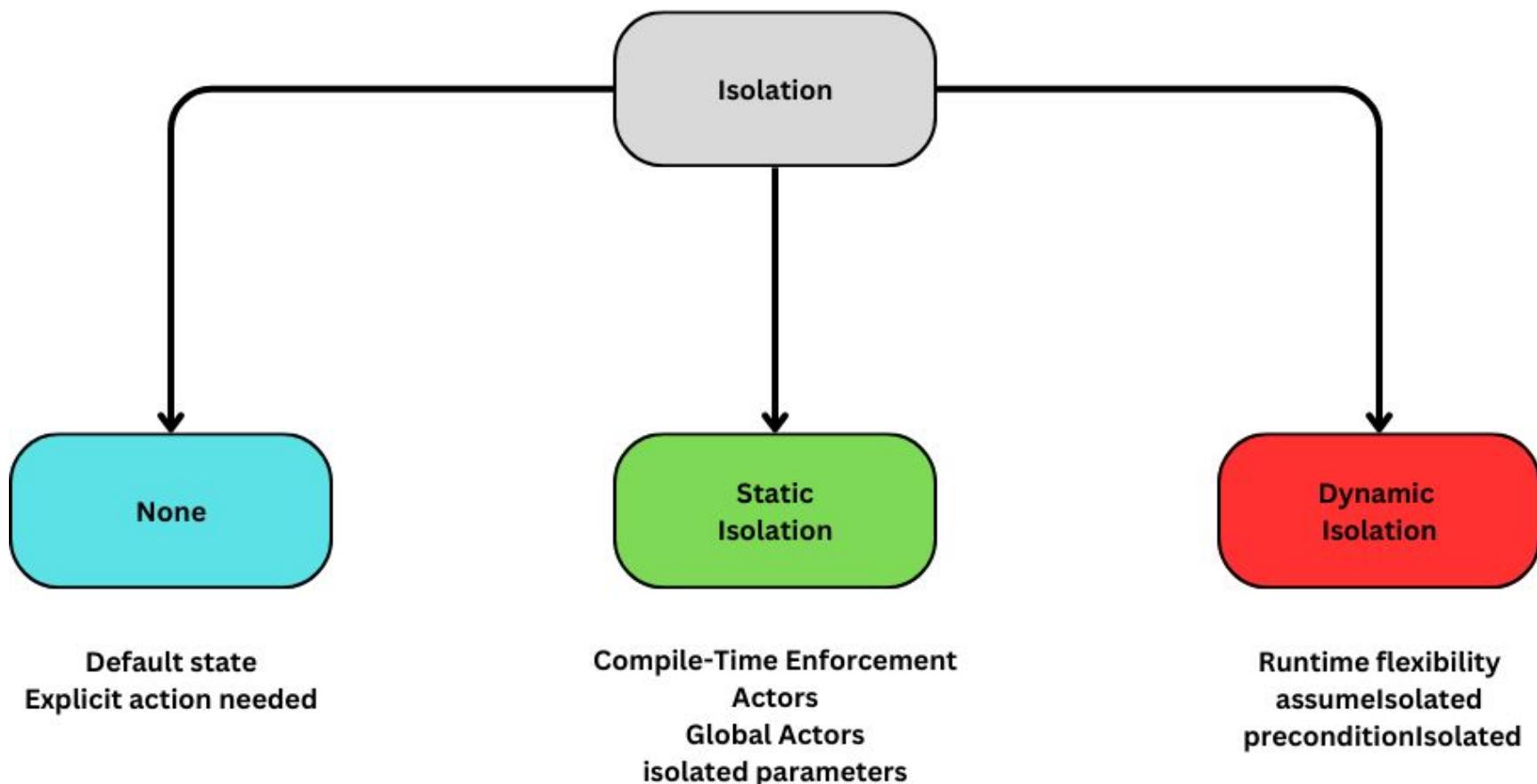
Isolation Flavours:

- **None** - By default, Everything is **non-isolated**. *Explicit action needed!* - Like actors or await
- **Static** - *Actors, global actors* and *isolated* parameters - *compile-time enforcement* of Isolation
- **Dynamic** - *assumes isolated* and *precondition isolated* attribute - allows at runtime

Pro Tip: Prefer **static** isolation for compile-time safety and dynamic only when reqd.

Consideration: Dynamic offers flexibility but introduce data races

Isolation in Swift Concurrency



None Isolation

None Isolation

- By default, Everything is **non-isolated**.
- Means there is no specific isolation enforced.
- This is the default for synchronous code

```
// Example of None Isolation
extension Island {
    nonisolated func meetTheFlock() async {
        let flockNames = await flock.map { $0.name }
        print("Meet our fabulous flock: \(flockNames)")
    }
}
```

Static Isolation

Actor Isolation

- Only one task can execute on an actor at a time
 - ◆ This prevents data races
- Isolate - mutable state to prevent concurrent access
 - ◆ No other task can modify the state concurrently
- Other task must wait their turn
 - ◆ Queued **but not strictly FIFO**
- Sendable checks enter or exit an actor
 - ◆ Ensures data is safe for concurrent use
- Actors are implicitly **Sendable**
 - ◆ By default actors conform to Sendable protocol

Actor Isolation

```
actor Island {  
    var flock: [Chicken]  
    var food: [Pineapple]  
    func advanceTime() {  
        let totalSlices = food.indices.reduce(0) { (total, nextIndex) in  
            total + food[nextIndex].slice()  
        }  
        Task {  
            flock.map(Chicken.produce)  
        }  
        Task.detached {  
            let ripePineapples = await food.filter { $0.ripeness ==  
                .perfect }  
            print("There are \(ripePineapples.count) ripe pineapples on  
the island")  
        }  
    }  
}
```



```
actor Island {  
    var flock: [Chicken]  
    var food: [Pineapple]  
    func advanceTime() {  
        let totalSlices = food.indices.reduce(0) { (total,  
nextIndex) in  
            total + food[nextIndex].slice()  
        }  
        Task {  
            flock.map(Chicken.produce)  
        }  
    }  
}
```



@MainActor Isolation

- **@MainActor:** Main Thread
- Isolates UI-related code to prevent concurrency issues
- Lot of UI Framework code and UI apps - run on it
- But only run a single job at a time
- Tasks shuttle b/w @MainActor(UI) & other actors(background)

Sendable Classes

To satisfy the requirements of the Sendable protocol, a class must:

- Be marked `final`
- Contain only stored properties that are immutable and sendable
- Have no superclass or have `NSObject` as the superclass

Classes marked with `@MainActor` are implicitly sendable, because the main actor coordinates all access to its state. These classes can have stored properties that are mutable and nonsendable.

@MainActor Isolation

```
@MainActor func updateView() { ... }
Task { @MainActor in
    // ...
    view.selectedChicken = lily
}
```

isolated vs nonisolated

Both are part of adding actor isolated control - properties and methods

isolated

- Default actor's behavior: Actors are automatically isolated
- **Explicit Control:** Mark properties or parameters as isolated

nonisolated

- Opt-Out of actor's isolation: Bypass actor isolation
- **Ideal** for accessing immutable values or when conforming to **protocol requirement**.
- Mark properties or methods as non-isolated

isolated vs nonisolated

```
actor BankAccount {  
    let accountHolder: String  
    var balance: Int  
    init(accountHolder: String, balance: Int = 0) {  
        self.accountHolder = accountHolder  
        self.balance = balance  
    }  
    func deposit(amount: Int) {  
        balance += amount  
    }  
    func transfer(amount: Int, to: isolated  
BankAccount) {  
        to.deposit(amount: amount)  
    }  
}
```

Actor-isolated property 'description' cannot be used to satisfy nonisolated protocol requirement
'description' declared here (Swift.CustomStringConvertible) Show
Add '@preconcurrency' to the 'CustomStringConvertible' conformance to defer isolation checking to run time Fix

```
extension BankAccount: CustomStringConvertible {  
    var description: String {  
        "bank \(balance)"  
    }  
}
```

```
extension BankAccount: CustomStringConvertible {  
    nonisolated var description: String {  
        "bank \(accountHolder)"  
    }  
}
```



isolated vs nonisolated

The screenshot shows a GitHub repository interface for the `swiftlang / swift-evolution` project. The repository has 51 pull requests. The current view is on the `Code` tab, showing the file `0313-actor-isolation-control.md`.

The file content is as follows:

Improved control over actor isolation

- Proposal: [SE-0313](#)
- Authors: [Doug Gregor](#), [Chris Lattner](#)
- Review Manager: [Ted Kremeneck](#)
- Status: **Implemented (Swift 5.5)**
- Previous revision: [1](#)
- Implementation: Partially available in [recent main snapshots](#) behind the flag `-Xfrontend -enable-experimental-concurrency`

Table of Contents

- [Introduction](#)
- [Motivation](#)
- [Proposed design](#)
 - [Actor-isolated parameters](#)
 - [Non-isolated declarations](#)
 - [Protocol conformances](#)
 - [Pre-async asynchronous protocols](#)
- [Source compatibility](#)

Isolation Boundary

```
/// Isolation Boundary
actor Account {
    private var balance: Int = 0
    func deposit(amount: Int) {
        balance += amount
    }
    func getBalance() -> Int {
        return balance
    }
}
```

- Account is **Actor**
- Internal state - *balance* is **isolated**
- *deposit* and *getBalance* accessible within actor's boundary
- Safe modification even accessing concurrently by different tasks

Dynamic Isolation

assumeIsolated

Assume that the current task is executing on this actor's serial executor, or stop program execution otherwise.

- Synchronous
- Being able to assert isolation for non-task code

```
nonisolated public func caffeineLevel(at level: Double) {  
    MainActor.assumeIsolated {  
        if level < minimumCaffeine {  
            // TODO: alert user to drink more coffee!  
        }  
    }  
}
```

preconditionIsolated

Stops program execution if the current task is not executing on this actor's serial executor.

- Assert isolation at runtime
- Adds a runtime check to ensure correctness

```
actor MyActor {  
    var value: Int = 0  
    func updateValue() {  
        preconditionIsolated(self) // Ensures that `self` is isolated  
        value += 1  
    }  
}
```

Task Isolation

- Tasks are isolated independently executing async operations
- Sendable checks - Ensure tasks remain isolated
- Use Sendable constraints wherever values need to be shared

[Swift](#) / [Swift Standard Library](#) / [Concurrency](#) / [Task](#)

Structure

Task

A unit of asynchronous work.

iOS 13.0+ | iPadOS 13.0+ | Mac Catalyst 13.0+ | macOS 10.15+ | tvOS 13.0+ | visionOS 1.0+ | watchOS 6.0+

```
@frozen
struct Task<Success, Failure> where Success : Sendable, Failure : Error
```

Challenges of Isolation

- **Performance:** Context switching and actor can introduce delay
- **Complexity:** Sequential execution can impact the performance
- Limited **Flexibility:** @unchecked Sendable can lead to bugs
- **Debugging** Race conditions can be challenging
- API and Tooling **Limitations:** Difficulty in integrating with the legacy API or libraries.

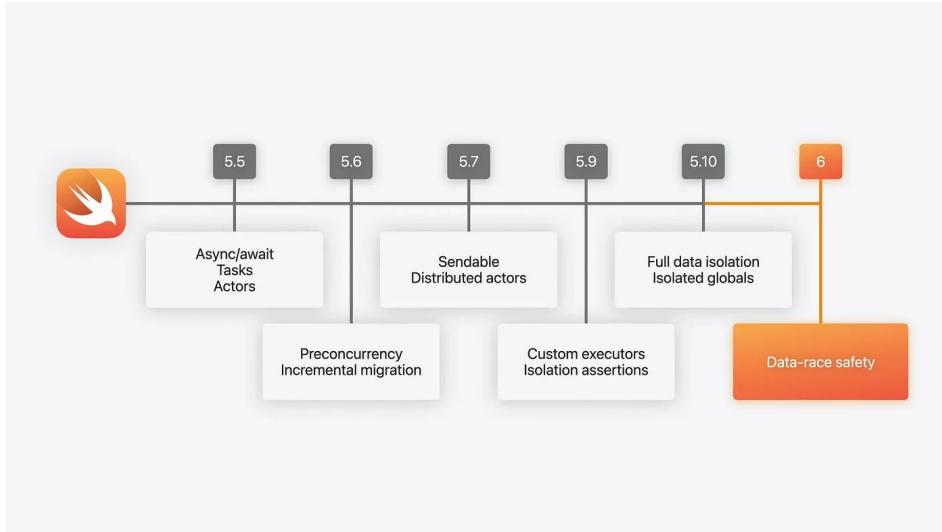
WHEN YOU FINALLY UNDERSTAND ASYNC/AWAIT..



Swift 6

Swift 6

- Compiler supports from **Swift 4.0**
- **Data-race Safety**
- Everything should be **Sendable**
- **Actors** strictly not **FIFO**
- Non-isolated(**unsafe**)
- **@preconcurrency** and **@unchecked** with careful consideration



What are Data Races?

- Two threads concurrently access at the same time
- Write access
- Caused by a shared mutable state

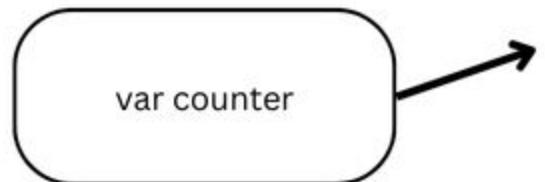
```
var counter = 0
Task.detached {
    counter += 1
}
Task.detached {
    counter += 1
}
print(counter) // The result could be 1 or 2 due to the data race
```

Data Race

Thread A



Thread B



Shared
State

Compiler support

- Backward Compatibility
- Supports from swift 4.0
- Migration technique to leverage new features and improvements
- **Async/Await** Integration - Transition
- Large old codebases can be migrated
- Brings Stability of existing code

Everything should be Sendable

- Sendable = sign of having too much isolation boundaries
- Cause of this - introducing more actors to the code

`() -> Void`

More !

`@escaping
() -> Void`

More !

`@Sendable
@escaping () -> Void`

More !

`@MainActor @Sendable
@escaping () -> Void`

More !

`@autoclosure
@MainActor @Sendable
@escaping () -> Void`

More !



Files

-  main + ↗
-  Go to file t
-  0410-atomics.md
-  0411-isolated-default-values.md
-  0412-strict-concurrency-for-glo...
-  0413-typed-throws.md
-  0414-region-based-isolation.md
-  0415-function-body-macros.md
-  0416-keypath-function-subtypin...
-  0417-task-executor-preference....
-  0418-inferring-sendable-for-me...
-  0419-backtrace-api.md
-  0420-inheritance-of-actor-isolat...
-  0421-generalize-async-sequenc...
-  0422-caller-side-default-argum...
-  0423-dynamic-actor-isolation.md
-  0424-custom-isolation-checkin...
-  0425-int128.md
-  0426-bitwise-copyable.md
-  0427-noncopyable-generics.md
-  0428-receiving-distributed-actor...

swift-evolution / proposals / 0418-inferring-sendable-for-methods.md

**hborla**Mark a bunch of proposals as implemented in Swift 6.0. (#2496) ...3a56140 · 2 months ago  History**Preview****Code****Blame**

421 lines (280 loc) · 19.7 KB

Raw

Inferring Sendable for methods and key path literals

- Proposal: [SE-0418](#)
- Authors: [Angela Laar](#), [Kavon Farvardin](#), [Pavel Yaskevich](#)
- Review Manager: [Becca Royal-Gordon](#)
- Status: **Implemented (Swift 6.0)**
- Upcoming Feature Flag: `InferSendableFromCaptures`
- Review: ([pitch](#)) ([review](#)) ([acceptance](#))

Introduction

This proposal is focused on a few corner cases in the language surrounding functions as values and key path literals when using concurrency. We propose Sendability should be inferred for partial and unapplied methods. We also propose to lift a Sendability restriction placed on key path literals in [SE-0302](#) by allowing the developers to control whether key path literal is Sendable or not. The goal is to improve flexibility, simplicity, and ergonomics without significant changes to Swift.

Motivation

The partial application of methods and other first-class uses of functions have a few rough edges when combined with concurrency.

Actors are not strictly FIFO

Key difference from Serial Dispatch Queue

- Actors prioritize highest priority work first
- Eliminates priority inversion

Order of Execution

Tasks -> Run code in **each** task in the **specified** order. The order between tasks **isn't guaranteed**

AsyncStreams -> Deliver elements in order

@preconcurrency

The @preconcurrency declaration can be added to:

- functions
- types
- protocols
- Imports

`@preconcurrency import MyModule`

Can import old modules into new code

Allow usage of new code in legacy codebases

```
extension HomeViewController: @preconcurrency CBCentralManagerDelegate {
```

nonisolated(unsafe)

The nonisolated(unsafe) declaration

- can be used to annotate the global variable
- disable static checking of data isolation for the global variable
- Thread Sanitizer could still identify failures

```
class NearByKit{  
  
    nonisolated(unsafe) static let sharedManager = NearByKit()  
}
```

@unchecked

Bypass certain compiler checks for types and operations

Use Cases:

- Type conversion
- Data access
- Performance Optimization

```
class Services: NSObject, URLSessionDelegate, @unchecked Sendable {  
    var baseUrl = "http://52.63.35.48:8080/test/"
```

@preconcurrency, nonisolated(unsafe) & @unchecked

Attribute	Purpose	Use Cases	Unsafe Opt Outs
@preconcurrency	Codebase doesn't have Swift concurrency	Older APIs to silence concurrency-related warnings	@preconcurrency Sendable considered “unsafe opt-outs” Careful consideration is must
nonisolated(unsafe)	Bypasses actor isolation without concurrency safety	Performance reasons when actor isolation not required	Potential risk
@unchecked	Bypasses compiler checks for flexibility	Type casting when compiler is too restrictive	@unchecked Sendable considered “unsafe opt-outs” Careful consideration is must

Caution: Using these can lead to data races.

Proposals related to Concurrency in Swift 6

[**SE-0337**](#) - Added Feature Flag - ***StrictConcurrency*** Incremental migration to concurrency checking

[**SE-0411**](#) - Isolated default value expressions

[**SE-0414**](#) - Enhancing Concurrency with Region-based Isolation

[**SE-0417**](#) - Task Executor Preference

[**SE-0418**](#) - Inferring Sendable for methods and key path literals

[**SE-0420**](#) - Inheritance of actor isolation

Proposals related to Concurrency in Swift 6

SE-0423 - Improves concurrency support when needing to operate with Objective-C frameworks.

SE-0424 - Custom isolation checking for SerialExecutor

SE-0430 - Adds a new *Sending* parameter

SE-0431 - *@isolated(any)* Function Types

SE-0434 - Usability of global-actor-isolated types

SE-0337 - Incremental migration to concurrency checking

Challenge

Adopting Swift's Concurrency Model in Legacy Codebases

Fix

Incremental Migration Support

Upcoming Feature Flag: **StrictConcurrency** (Implemented in **Swift 6.0**) (Enabled in Swift 6 language mode)

Impact

Smoother Transition to Concurrency Safety

We will look into depth in upcoming slides

Why it's Important?

Facilitates Gradual Adoption

- Main actor-isolated property 'persistentContainer' can not be referenced from a nonisolated context

- Add '@MainActor' to make instance method 'getAuditScreenByData()' part of global actor 'MainActor'

Fix

SE-0414 - Region-based Isolation

Challenge

Swift 5.10 banned passing all non-Sendable values across isolation boundaries

Fix

Introduces a new mechanism to enforce memory isolation within regions of code

Impact

It's just a beginning. Makes your code safer by reducing the risk of data conflicts in concurrent environments

Why it's Important?

Helps improve the reliability and safety of your code when working with concurrency



Passing argument of non-sendable type 'NonSendable' into main actor-isolated context may introduce data races

```
class NonSendable {}  
func doSomething(with: NonSendable) async {}  
@MainActor  
func modifyOnMainActor(_ x: NonSendable) async {  
    let value = NonSendable()  
    // WARNING: Passing argument of non-sendable type 'NonSendable' outside  
    // of main actor-isolated context may introduce data races  
    await doSomething(with: x)  
}
```

```
struct Animal: Sendable {}  
class NonSendable {  
    var animal: Animal  
}  
func doSomething(with x: NonSendable) async {  
    x.animal = Animal()  
}  
@MainActor  
func modifyOnMainActor(_ x: NonSendable) async {  
    let value = NonSendable(animal: Animal())  
    await doSomething(with: x)  
    _ = x.animal // Doesn't produce ERROR - could race with doSomeThing  
}
```

SE-0417 - Task Executor Preference

Challenge

Tasks in Swift might not always run on the best executor, which can slow things down.

Fix

Allows you to choose the preferred executor for tasks, giving you more control over how they run.

Impact

Helps tasks run more efficiently by using the right resources.

Why it's Important?

Gives developers better control, leading to faster and smoother performance in Swift apps.

The SerialExecutor guarantees mutual exclusion, and the TaskExecutor provides a source of threads.

```
nonisolated func doSomething() async {  
    // ...  
}  
Task(executorPreference: preferredExecutor) {  
    // doSomething body would execute on 'preferredExecutor'  
    await doSomething()  
}  
// doSomething body would execute on 'default global concurrent  
executor'  
await doSomething()
```

SE-0418 - Inferring Sendable for methods and key path literals

Challenge

Manually declaring `Sendable` for methods and key paths was tedious and prone to mistakes.

Fix

Swift now automatically infers `Sendable` for methods and key paths, reducing manual effort.

Impact

Makes code cleaner and safer by ensuring methods and key paths are correctly handled in concurrent contexts.

Why it's Important?

Eases the burden of concurrency management, making your code more reliable.

```
struct MyStruct: Sendable {  
    var value: Int  
    func update() -> Int {  
        return value  
    }  
}  
@MainActor let keyPath: KeyPath<MyStruct, Int> = \.value
```

SE-0420: Inheritance of actor isolation

Challenge

Complexity in Actor Inheritance

Fix

Inheritance of Actor Isolation

Impact

Consistency in Isolation. Isolation parameters prevent suspension at the call site.

Why it's Important?

Ensures Safe Concurrency in Subclasses

```
func doStuff(isolatedTo actor: isolated (any Actor)?) async {  
}  
// this is non-isolated  
await doStuff(isolatedTo: nil)
```

```
func doStuff(isolatedTo actor: isolated (any Actor)? = #isolation)  
async {  
}  
@MainActor  
class MyClass {  
    func doLotsOfStuff() {  
        await doStuff()  
        await doStuff()  
        await doStuff()  
    }  
}
```

SE-0423 - Dynamic Isolation enforcement from non-strict-concurrency contexts

Challenge

Ensuring Safe Actor Interaction in Non-Strict Concurrency Contexts

Fix

Dynamic actor isolation enforcement - using @preconcurrency

Impact

Increased Safety in Mixed Concurrent codebases

Why it's Important?

Bridges the gap between the Legacy and Modern Concurrency by using **@preconcurrency**

```
@MainActor
class MyViewController: @preconcurrency
ViewDelegateProtocol {
    func respondToUIEvent() {
        // no need for nonisolated!
    }
}
```

SE-0424 - Custom isolation checking for SerialExecutor

Challenge

Ensuring tasks on custom executors follow isolation rules.

Fix

Allows custom executors to define how they check for isolation.

Impact

Increases safety by letting custom executors enforce isolation, just like Swift's built-in executors.

Why it's Important?

Gives more control over task execution, making custom concurrency safer.

```
/// Proposal extends the SerialExecutor protocol
public protocol SerialExecutor : Executor {
    @available(macOS 15.0, iOS 18.0, watchOS 11.0,
    tvOS 18.0, visionOS 2.0, *)
    func checkIsolated()
}
```

```
// implementation - DispatchSerialQueue
extension DispatchSerialQueue {
    public func checkIsolated() {
        dispatchPrecondition(condition: .onQueue(self)) //
existing Dispatch API
    }
}
```

SE-0430 - Adds a new Sending parameter

Challenge

Ensuring Safe Data Transfer Across
Concurrency Boundaries

Fix

Introduction of sending Keyword

Impact

Stricter Concurrency Safety

Why it's Important?

Standard library integrated 'sending'
keyword. For eg: Task.init()

Sendable is more important. *sending* is less
important.

```
@discardableResult
init(
    priority: TaskPriority? = nil,
    operation: sending @escaping @isolated(any) () async -> Success
)
```

SE-0431 - @isolated(any) Function Types

Challenge

Inconsistent Isolation Control

Fix

Introduction of @isolated(any) Attribute

Impact

Improved Safety and Clarity

Why it's Important?

Ensures Actor-Safe Code Execution

```
// Problem
func scheduleSomeWork(_ f: @Sendable @escaping () async -> Void) {
    // We've lost all the static isolation information
    // available where f was defined.

    Task {
        await f()
    }
}

// Problem 2
func defineClosure() {
    scheduleSomeWork { @MainActor in
        print("I'm on the main actor")
    }
}
```

```
// Solution
func scheduleSomeWork(_ f: @Sendable @escaping @isolated(any) () async -> Void) {
    // closures have properties now!
    let isolation = f.isolation
    // do something with "isolation" I guess?
}
```

SE-0434 - Usability of global-actor-isolated types

Challenge

Limited Usability of
Global-Actor-Isolated Types

Fix

Improved Usability Across Contexts

Impact

Greater Flexibility

Why it's Important?

Enables Safer and More Modular Code

```
// Problem
@MainActor
struct SomeMainActorType {
    let x: Int
    var y: Int
    nonisolated func readValues() {
        print(x) // this is allowed!
        print(y) // this is not?
        // Main actor-isolated property 'y' can not be referenced
        // from a non-isolated context
    }
}
```

```
// Problem - 2
let closure = { @MainActor in  // Let 'closure' is not
    concurrency-safe because it is not either conforming to 'Sendable'
    or isolated to a global actor; this is an error in Swift 6
    // capture a bunch of non-Sendable stuff
}
Task {
    // Error: capturing non-Sendable `closure`
    await closure()
}
```

SE-0411 - Isolated default value expressions

Challenge

Default values in functions could access shared data unsafely in concurrent code.

Fix

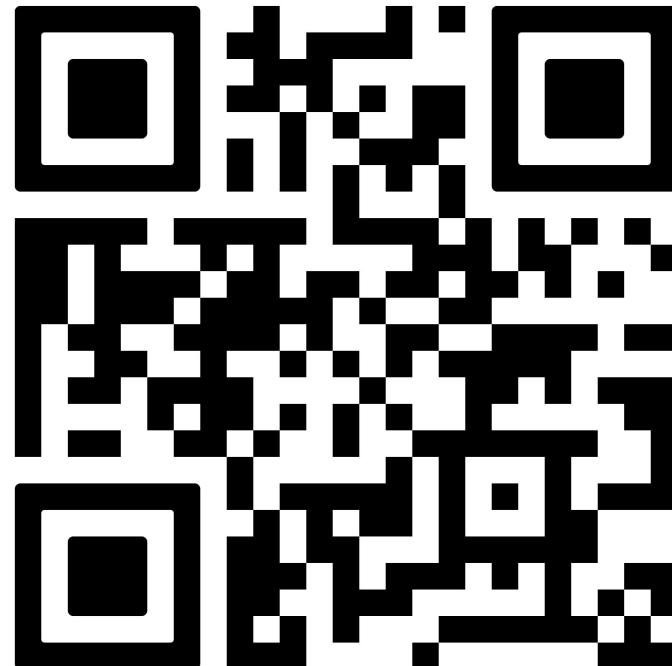
Ensures default values are executed within the correct isolation context.

Impact

Makes concurrent code safer by preventing data conflicts in default value expressions.

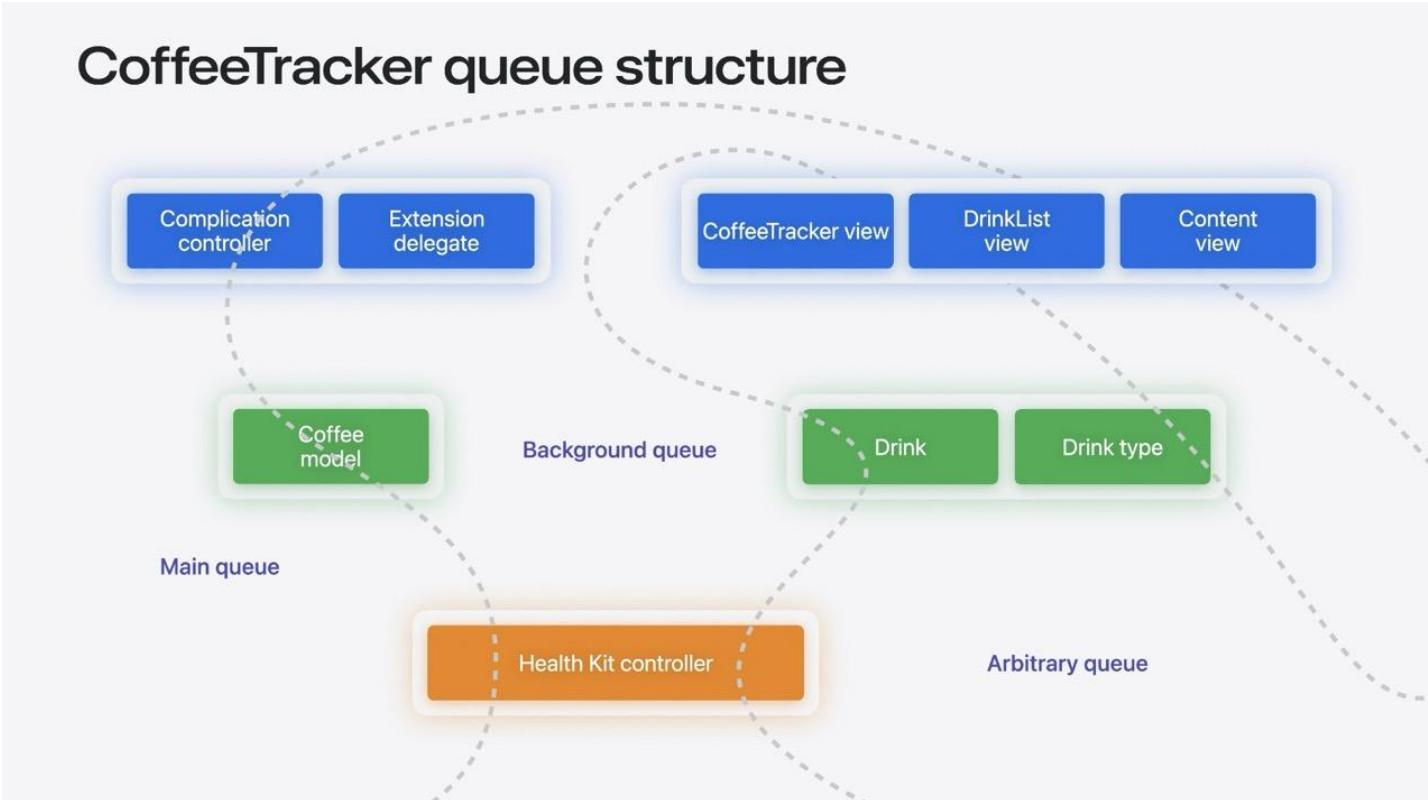
Why it's Important?

Strengthens Swift's concurrency model, making your code more reliable and secure.



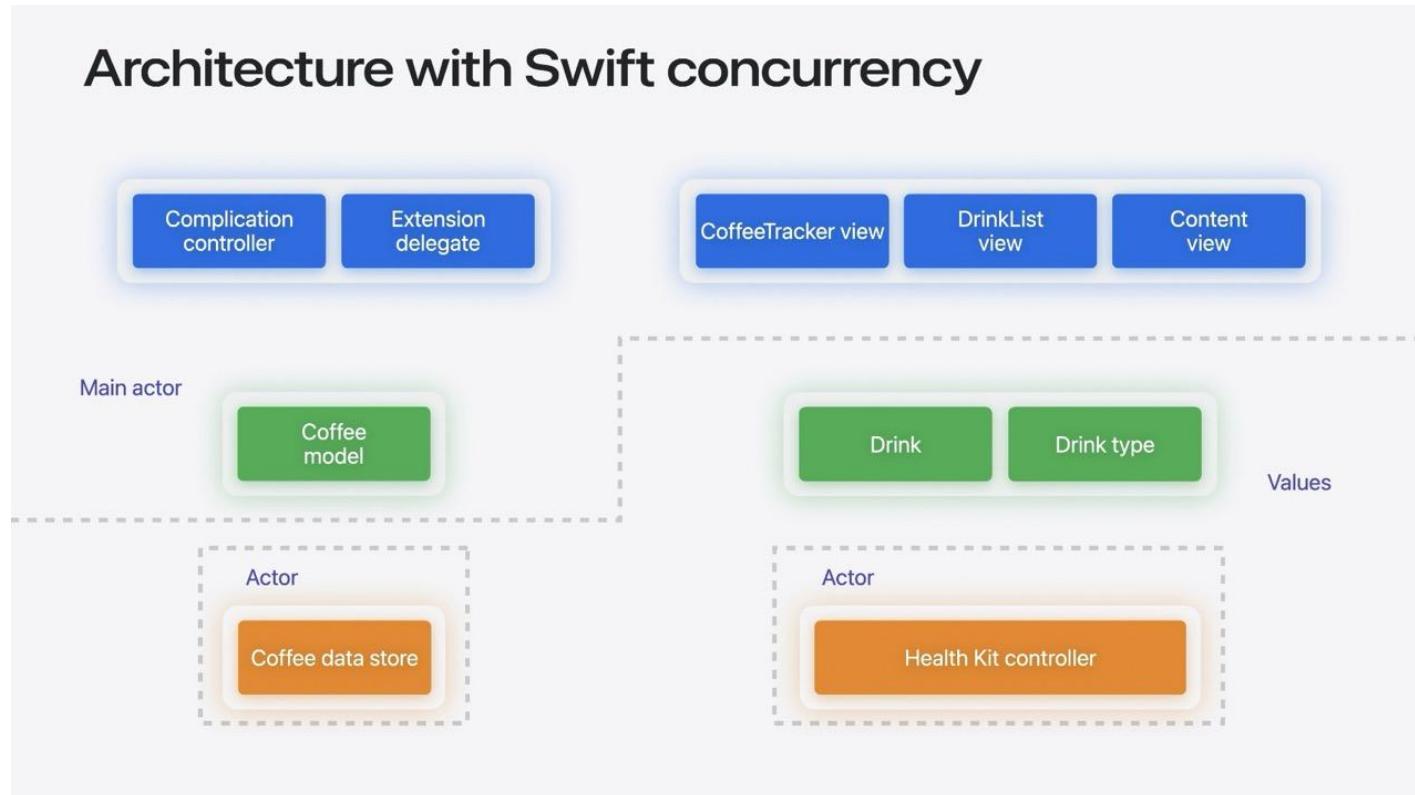
Swift 6 Migration

Pre-concurrency queue structure



Swift 6 Migration

Architecture with Swift concurrency



How to Migrate to Swift 6

- Determine the isolated part
- Enable complete concurrency checking
- Enable Swift 6 mode
- Audit any unsafe opt-outs

When to Migrate to Swift 6

- Identifying the mistakes in concurrent code is tedious
- Hard to reproduce crashes
- Integrating more concurrency
- If you have public SPM, migrate the package to Swift 6

Swift 6 - Feature Flags

-enable-upcoming-feature

- *GlobalConcurrency* - Strict concurrency for global variables
- *InferSendableFromCaptures* - Inferring Sendable for methods and key path literals
- *RegionBasedIsolation* - Region based Isolation
- *StrictConcurrency* - Incremental migration to concurrency checking
- *DisableOutwardActorInference* - Remove Actor Isolation Inference caused by Property Wrappers

Minimal Strict Concurrency Checking

Swift Compiler - Upcoming Features

Setting

Bare Slash Regex Literals	Yes ⓘ
Concise Magic File	No - \$(SWIFT_UPCOMING_FEATURE_6_0) ⓘ
Default Internal Imports	No ⓘ
Deprecate Application Main	No - \$(SWIFT_UPCOMING_FEATURE_6_0) ⓘ
Disable Outward Actor Isolation Inference	No - \$(SWIFT_UPCOMING_FEATURE_6_0) ⓘ
Forward Trailing Closures	No - \$(SWIFT_UPCOMING_FEATURE_6_0) ⓘ
Implicitly Opened Existentials	No - \$(SWIFT_UPCOMING_FEATURE_6_0) ⓘ
Import Objective-C Forward Declarations	No - \$(SWIFT_UPCOMING_FEATURE_6_0) ⓘ
Infer Sendable for Methods and Key Path Literals	No - \$(SWIFT_UPCOMING_FEATURE_6_0) ⓘ
Isolated Default Values	No - \$(SWIFT_UPCOMING_FEATURE_6_0) ⓘ
Isolated Global Variables	No - \$(SWIFT_UPCOMING_FEATURE_6_0) ⓘ
Region Based Isolation	No - \$(SWIFT_UPCOMING_FEATURE_6_0) ⓘ
Require Existential any	No ⓘ

NativeSkeltonApp

> Strict Concurrency Checking

✓ Minimal - \$(SWIFT_STRICT_CONCURRENCY_DEFAULT)

Targeted

Complete

Other...

Swift Compiler - Warnings Policies

Setting

Suppress Warnings	No - \$(SUPPRESS_WARNINGS) ⓘ
Treat Warnings as Errors	No ⓘ

Only Checks explicit Sendable

Minimal Strict Concurrency Checking

Actor-isolated property 'balance' can not be mutated from a nonisolated context

Mutation of this property is only permitted within the actor

Show

```
actor BankAccount {  
    var balance: Int = 0  
    func deposit(amount: Int) {  
        balance += amount  
    }  
}  
  
func performTransaction(account: BankAccount,  
amount: Int) {  
    Task {  
        account.balance += amount  
    }  
}
```



```
actor BankAccountMinimalConcurrentCheck {  
    var balance: Int = 0  
    func deposit(amount: Int) {  
        balance += amount  
    }  
}  
  
func performTransaction(account:  
BankAccountMinimalConcurrentCheck, amount: Int) {  
    Task {  
        await account.deposit(amount: amount)  
    }  
}
```



Targeted Strict Concurrency Checking

Swift Compiler - Upcoming Features

Setting

Bare Slash Regex Literals	Yes ⌂
Concise Magic File	No - \$(SWIFT_UPCOMING_FEATURE_6_0) ⌂
Default Internal Imports	No ⌂
Deprecate Application Main	No - \$(SWIFT_UPCOMING_FEATURE_6_0) ⌂
Disable Outward Actor Isolation Inference	No - \$(SWIFT_UPCOMING_FEATURE_6_0) ⌂
Forward Trailing Closures	No - \$(SWIFT_UPCOMING_FEATURE_6_0) ⌂
Implicitly Opened Existentials	No - \$(SWIFT_UPCOMING_FEATURE_6_0) ⌂
Import Objective-C Forward Declarations	No - \$(SWIFT_UPCOMING_FEATURE_6_0) ⌂
Infer Sendable for Methods and Key Path Literals	No - \$(SWIFT_UPCOMING_FEATURE_6_0) ⌂
Isolated Default Values	No - \$(SWIFT_UPCOMING_FEATURE_6_0) ⌂
Isolated Global Variables	No - \$(SWIFT_UPCOMING_FEATURE_6_0) ⌂
Region Based Isolation	No - \$(SWIFT_UPCOMING_FEATURE_6_0) ⌂
Require Existential any	No - \$(SWIFT_UPCOMING_FEATURE_6_0) ⌂

> Strict Concurrency Checking

SOLIDPrinciplesExample

Minimal

Targeted

Complete

Other...

Swift Compiler - Warnings Policies

Setting

Suppress Warnings	No - \$(SUPPRESS_WARNINGS) ⌂
Treat Warnings as Errors	No ⌂

@preconcurrency attribute can be used

Targeted Strict Concurrency Checking

⚠️ Passing argument of non-sendable type 'UserProfile' into actor-isolated context may introduce data races; this is an error in the Swift 6 language mode
 ⓘ Class 'UserProfile' does not conform to the 'Sendable' protocol

Show

```
// Non-Sendable class
class UserProfile {
    var name: String
    init(name: String) {
        self.name = name
    }
}
actor BankAccount {
    private var balance: Int = 0
    // Attempting to pass a non-Sendable type across actor boundaries
    func updateProfile(profile: UserProfile) {
        print("Updating profile for \(profile.name)")
    }
}
func performTransaction(account: BankAccount, profile: UserProfile) {
    Task {
        await account.updateProfile(profile: profile) // Error: 'UserProfile' is
not Sendable.
    }
}
```



```
struct UserProfile: Sendable {
    let name: String
}

actor BankAccount {
    private var balance: Int = 0
    func updateProfile(profile: UserProfile) {
        print("Updating profile for \(profile.name)")
    }
}
func performTransaction(account: BankAccount, profile: UserProfile) {
    Task {
        await account.updateProfile(profile: profile) // No error
because UserProfile is now Sendable.
    }
}
```



Complete Strict Concurrency Checking

Swift Compiler - Upcoming Features

Setting

- Bare Slash Regex Literals
- Concise Magic File
- Default Internal Imports
- Deprecate Application Main
- Disable Outward Actor Isolation Inference
- Forward Trailing Closures
- Implicitly Opened Existentials
- Import Objective-C Forward Declarations
- Infer Sendable for Methods and Key Path Literals
- Isolated Default Values
- Isolated Global Variables
- Region Based Isolation
- Require Existential any

 SOLIDPrinciplesExample

- Yes ⇂
- No - \$(SWIFT_UPCOMING_FEATURE_6_0) ⇂
- No ⇂
- No - \$(SWIFT_UPCOMING_FEATURE_6_0) ⇂

- Minimal
- Targeted
- Complete
- Other...

Strict Concurrency Checking

Swift Compiler - Warnings Policies

Setting

- Suppress Warnings
- Treat Warnings as Errors

 SOLIDPrinciplesExample

- No - \$(SUPPRESS_WARNINGS) ⇂
- No ⇂

Stricter concurrency checking one module at a time

Enable Swift 6 mode

The screenshot shows the Xcode build settings interface. On the left, there are two collapsed sections: "Swift Compiler - Language" and "Swift Compiler - Search Paths". The "Swift Compiler - Language" section is expanded, revealing a "Setting" row and a "Swift Language Version" dropdown menu. The dropdown menu is open, displaying five options: "Swift 4", "Swift 4.2", "Swift 5" (which is selected, indicated by a checked checkbox), "Swift 6", and "Unspecified". To the right of the dropdown, the project name "NativeSkeltonApp" is visible. The "Swift Compiler - Search Paths" section is also partially visible.

Final Step of the Migration for the Xcode Project

Requires to change the Language mode -> **Swift 6**

Go to the Building Settings and Search for ***Swift Language Version***

Compile Errors during Migration

```
✓ ⓘ AppDelegate
  > ⚡ 'UIApplicationMain' is deprecated
✓ ⓘ AppDelegateManager
  > ⚡ Static property 'appDel' is not concurrency-safe because it is nonisolated global shared mutable state
✓ ⓘ Constants
  > ⚡ Let 'defaults' is not concurrency-safe because non-'Sendable' type 'UserDefaults' may have shared mutable state
✓ ⓘ Datamodel
  > ⚡ Let 'sharedDelegate' is not concurrency-safe because non-'Sendable' type 'Datamodel' may have shared mutable state
  > ⚡ Let 'sharedDataModel' is not concurrency-safe because non-'Sendable' type 'Datamodel' may have shared mutable state
✓ ⓘ NearByKit
  > ⚡ Static property 'sharedManager' is not concurrency-safe because non-'Sendable' type 'NearByKit' may have shared mutable state
✓ ⓘ LocationManager
  > ⚡ Static property 'sharedManager' is not concurrency-safe because non-'Sendable' type 'LocationManager' may have shared mutable state
✓ ⓘ Services
  ✘ Non-final class 'Services' cannot conform to 'Sendable'; use '@unchecked Sendable'
  ✘ Stored property 'baseUrl' of 'Sendable'-conforming class 'Services' is mutable
✓ ⓘ HomeViewController
  > ⚡ Main actor-isolated instance method 'locationManager(_:didChangeAuthorization:)' cannot be used to satisfy nonisolated protocol requirement
  > ⚡ Main actor-isolated instance method 'locationManager(_:didUpdateLocations:)' cannot be used to satisfy nonisolated protocol requirement
  > ⚡ Main actor-isolated instance method 'locationManager(_:didFailWithError:)' cannot be used to satisfy nonisolated protocol requirement
  > ⚡ Main actor-isolated instance method 'centralManagerDidUpdateState' cannot be used to satisfy nonisolated protocol requirement
  > ⚡ Main actor-isolated instance method 'centralManager(_:didDiscover:advertisementData:rssi:)' cannot be used to satisfy nonisolated protocol requirement
  > ⚡ Main actor-isolated instance method 'centralManager(_:didConnect:)' cannot be used to satisfy nonisolated protocol requirement
  > ⚡ Main actor-isolated instance method 'centralManager(_:didDisconnectPeripheral:error:)' cannot be used to satisfy nonisolated protocol requirement
  > ⚡ Main actor-isolated instance method 'centralManager(_:didFailToConnect:error:)' cannot be used to satisfy nonisolated protocol requirement
  > ⚡ Main actor-isolated instance method 'peripheral(_:didDiscoverServices:)' cannot be used to satisfy nonisolated protocol requirement
  > ⚡ Main actor-isolated instance method 'peripheral(_:didDiscoverCharacteristicsFor:error:)' cannot be used to satisfy nonisolated protocol requirement
  > ⚡ Main actor-isolated instance method 'peripheral(_:didUpdateValueFor:error:)' cannot be used to satisfy nonisolated protocol requirement
✓ ⓘ LoginViewController
  > ⚡ Switch covers known cases, but 'CLAuthorizationStatus' may have additional unknown values, possibly added in future versions
  > ⚡ Main actor-isolated instance method 'locationManager(_:didUpdateLocations:)' cannot be used to satisfy nonisolated protocol requirement
✓ ⓘ BarCodeScanViewController
  > ⚡ Main actor-isolated instance method 'metadataOutput(_:didOutput:from:)' cannot be used to satisfy nonisolated protocol requirement
```

Common Error - Main actor-isolated property

- Main actor-isolated property 'persistentContainer' can not be referenced from a nonisolated context

📎 Add '@MainActor' to make instance method 'getAuditScreenByData()' part of global actor 'MainActor' Fix

```
func getAuditScreenByData() -> ([[String:String]], [String : [Date]]){  
    var nearByKitScreenDict = [[String:String]]()  
    var auditKitSterilizedDates = [String:[Date]]()
```

```
let context = appDelegateObj.persistentContainer.viewContext  
// Fetch Kit records  
let fetchRequest : NSFetchedResultsController<NearByKitCollection>  
fetchRequest.predicate = NSPredicate(format: "ki
```

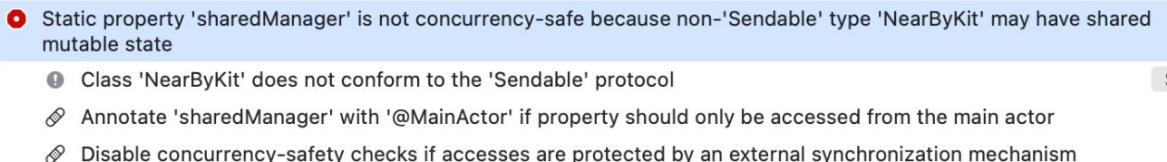
● Main actor-isolated property 'persistentContainer' can not be referenced from a nonisolated context
📎 Add '@MainActor' to make instance method 'getAuditScreenByData()' part of global actor 'MainActor' Fix

```
@MainActor func getAuditScreenByData() -> ([[String:String]], [String : [Date]]){  
    var nearByKitScreenDict = [[String:String]]()  
    var auditKitSterilizedDates = [String:[Date]]()  
  
    let context = appDelegateObj.persistentContainer.viewContext  
    // Fetch Kit records
```



Common Error - Static Property because Non-Sendable

```
class NearByKit{  
  
    static let sharedManager = NearByKit()  
  
    // Packets Data  
    private var upTime: String?  
    private var firstMPAAutoClavePowerReverseCycl  
    private var sumMPCycles: String?  
}
```



```
class NearByKit{
```

```
    @MainActor static let sharedManager = NearByKit()
```



```
class NearByKit{
```

```
    nonisolated(unsafe) static let sharedManager = NearByKit()
```

Common Error '@unchecked Sendable'

```
class Services: NSObject, URLSessionDelegate {  
    var baseUrl = "http://52.63.35.48:8080/test/"  
}
```

✖ Non-final class 'Services' cannot conform to 'Sendable'; use '@unchecked Sendable'
✖ Stored property 'baseUrl' of 'Sendable'-conforming class 'Services' is mutable

```
class Services: NSObject, URLSessionDelegate, @unchecked Sendable {  
    var baseUrl = "http://52.63.35.48:8080/test/"
```



```
@available(iOS 7.0, *)  
public protocol URLSessionDelegate : NSObjectProtocol, Sendable {
```

```
    optional func urlSession(_ session: URLSession, didBecomeInvalidWithError error: (any Error)?)
```

```
    optional func urlSession(_ session: URLSession, didReceive challenge: URLAuthenticationChallenge, completionHandler: @escaping @Sendable  
        (URLSession.AuthChallengeDisposition, URLCredential?) -> Void)
```

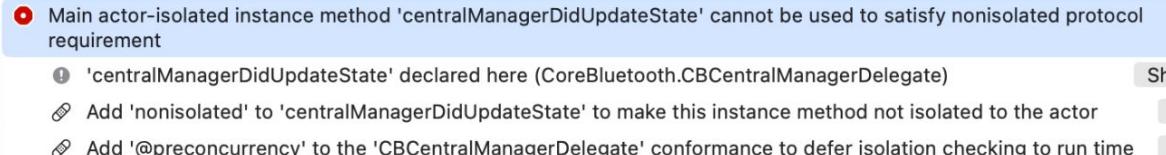
```
    optional func urlSession(_ session: URLSession, didReceive challenge: URLAuthenticationChallenge) async -> (URLSession.AuthChallengeDisposition,  
        URLCredential?)
```

```
@available(iOS 7.0, *)  
optional func urlSessionDidFinishEvents(forBackgroundURLSession session: URLSession)
```

Reference - URLSessionDelegate

@preconcurrency

```
extension HomeViewController: CBCentralManagerDelegate {  
  
    // Checking bluetooth connectivity  
    func centralManagerDidUpdateState(_ central: CBCentralManager) {  
        switch central.state {  
            case .unknown:  
                print("central.state is .unknown")  
            case .resetting:  
                print("central.state is .resetting")  
            case .unsupported:  
                print("central.state is .unsupported")  
        }  
    }  
}
```



```
extension HomeViewController: @preconcurrency CBCentralManagerDelegate {  
  
    // Checking bluetooth connectivity  
    func centralManagerDidUpdateState(_ central: CBCentralManager) {  
        switch central.state {  
            case .unknown:  
                print("central.state is .unknown")  
            case .resetting:  
                print("central.state is .resetting")  
        }  
    }  
}
```



Static property

Static property 'rate' is not concurrency-safe because it is nonisolated global shared mutable state

Convert 'rate' to a 'let' constant to make 'Sendable' shared state immutable

Fix

Annotate 'rate' with '@MainActor' if property should only be accessed from the main actor

Fix

Disable concurrency-safety checks if accesses are protected by an external synchronization mechanism

Fix

```
private struct Initialize {  
    static var rate = "rate"
```

```
private struct Initialize {  
    static let rate = "rate"
```

Other Common Errors

- **● Main actor-isolated property 'model' can not be mutated from a nonisolated context**
Mutation of this property is only permitted within the actor
Add '@MainActor' to make instance method 'present(context:model:)' part of global actor 'MainActor'
- > **● Main actor-isolated property 'model' can not be mutated from a nonisolated context**
- **● Call to main actor-isolated initializer 'init(arrangedSubviews:)' in a synchronous nonisolated context**
Calls to initializer 'init(arrangedSubviews:)' from outside of its actor context are implicitly asynchronous
Add '@MainActor' to make instance method 'addViews(context:views:)' part of global actor 'MainActor'
 - > **● Main actor-isolated property 'spacing' can not be mutated from a nonisolated context**
 - > **● Main actor-isolated property 'distribution' can not be mutated from a nonisolated context**
 - > **● Main actor-isolated property 'alignment' can not be mutated from a nonisolated context**
 - > **● Main actor-isolated property 'axis' can not be mutated from a nonisolated context**
 - > **● Main actor-isolated property 'translatesAutoresizingMaskIntoConstraints' can not be mutated from a nonisolated context**
 - > **● Main actor-isolated property 'isUserInteractionEnabled' can not be mutated from a nonisolated context**
 - > **● Call to main actor-isolated instance method 'fill(with:edges:)' in a synchronous nonisolated context**
 - > **● Main actor-isolated property 'isUserInteractionEnabled' can not be mutated from a nonisolated context**
 - > **● Call to main actor-isolated instance method 'constraint(equalTo:constant:)' in a synchronous nonisolated context**
 - > **● Main actor-isolated property 'topAnchor' can not be referenced from a nonisolated context**
 - > **● Main actor-isolated property 'topAnchor' can not be referenced from a nonisolated context**
 - > **● Call to main actor-isolated instance method 'constraint(equalTo:constant:)' in a synchronous nonisolated context**
 - > **● Main actor-isolated property 'trailingAnchor' can not be referenced from a nonisolated context**
 - > **● Main actor-isolated property 'trailingAnchor' can not be referenced from a nonisolated context**
 - > **● Call to main actor-isolated instance method 'constraint(equalTo:constant:)' in a synchronous nonisolated context**
 - > **● Main actor-isolated property 'bottomAnchor' can not be referenced from a nonisolated context**
 - > **● Main actor-isolated property 'bottomAnchor' can not be referenced from a nonisolated context**
 - > **● Call to main actor-isolated instance method 'constraint(equalTo:constant:)' in a synchronous nonisolated context**
 - > **● Main actor-isolated property 'leadingAnchor' can not be referenced from a nonisolated context**
 - > **● Main actor-isolated property 'leadingAnchor' can not be referenced from a nonisolated context**
 - > **● Call to main actor-isolated class method 'activate' in a synchronous nonisolated context**

Other Common Errors

- ✖ Capture of 'observer' with non-sendable type 'AnyObject?' in a `@Sendable` closure
- ✖ Reference to captured var 'observer' in concurrently-executing code
- ✖ Mutation of captured var 'observer' in concurrently-executing code
- > ✖ Call to main actor-isolated instance method 'pushViewController(_:animated:)' in a synchronous nonisolated context
- > ✖ Main actor-isolated property 'navigationController' can not be referenced from a nonisolated context

Common Errors

Migrating to Swift 6 Documentation / Migrating to Swift 6 / Common Compiler Errors

Common Compiler Errors

Identify, understand, and address common problems you can encounter while working with Swift concurrency.



The data isolation guarantees made by the compiler affect all Swift code. This means complete concurrency checking can surface latent issues, even in Swift 5 code that doesn't use any concurrency language features directly. With the Swift 6 language mode enabled, some of these potential issues can also become errors.

After enabling complete checking, many projects can contain a large number of warnings and errors. *Don't get overwhelmed!* Most of these can be tracked down to a much smaller set of root causes. And these causes, frequently, are a result of common patterns which aren't just easy to fix, but can also be very instructive while learning about Swift's concurrency system.

Unsafe Global and Static Variables

Global state, including static variables, are accessible from anywhere in a program. This visibility makes them particularly susceptible to concurrent access. Before data-race safety, global variable patterns relied on programmers carefully accessing global state in ways that avoided data-races without any help from the compiler.

Experiment

These code examples are available in package form. Try them out yourself in [Swift Playgrounds](#).

[Filter](#)

<https://www.swift.org/migration/documentation/swift-6-concurrency-migration-guide/incrementaladoption>

- [Common Compiler Errors](#)
- [Unsafe Global and Static Variables](#)
- [Sendable Types](#)
- [Non-Sendable Types](#)
- [Protocol Conformance Isolation Mismatch](#)
- [Under-Specified Protocol](#)
- [Isolated Conforming Type](#)
- [Crossing Isolation Boundaries](#)
- [Implicitly-Sendable Types](#)
- [Preconcurrency Import](#)
- [Latent Isolation](#)
- [Computed Value](#)
- [Sending Argument](#)
- [Sendable Conformance](#)
- [Non-Isolated Initialization](#)
- [Non-Isolated Deinitialization](#)

StrictConcurrency

```
1 // swift-tools-version: 5.8
2 // The swift-tools-version declares the minimum version of Swift required to build this package.
3
4 import PackageDescription
5
6 let package = Package(
7     name: "NetworkKit",
8     platforms: [
9         .iOS(.v15),
10        .macOS(.v12),
11        .watchOS(.v8)
12    ],
13    products: [
14        // Products define the executables and libraries a package produces, making them visible to other packages.
15        .library(
16            name: "NetworkKit",
17            targets: ["NetworkKit"])
18    ],
19    targets: [
20        // Targets are the basic building blocks of a package, defining a module or a test suite.
21        // Targets can depend on other targets in this package and products from dependencies.
22        .target(
23            name: "NetworkKit",
24            swiftSettings: [
25                .enableUpcomingFeature("StrictConcurrency")
26            ],
27            .testTarget(
28                name: "NetworkKitTests",
29                dependencies: ["NetworkKit"])
30        ]
31    )
32 )
```

StrictConcurrency

```
public func sendRequest<T: Decodable>(endpoint: EndPoint,
                                         resultHandler: @escaping (Result<T, NetworkError>) -> Void) {

    guard let urlRequest = createRequest(endPoint: endpoint) else {
        return
    }
    let urlTask = URLSession.shared.dataTask(with: urlRequest) { data, response, error in
        guard error == nil else {
            resultHandler(.failure(.invalidURL))
            return
        }
        guard let response = response as? HTTPURLResponse, 200...299 ~= response.statusCode else {
            resultHandler(.failure(.unexpectedStatusCode))
            return
        }
    }
}
```

⚠ Capture of 'resultHandler' with non-sendable type '(Result<T, NetworkError>) -> Void' in a '@Sendable' closure; this is an error in the Swift 6 language mode

ⓘ A function type must be marked '@Sendable' to conform to 'Sendable'

Show

```
public func sendRequest<T: Decodable>(endpoint: EndPoint,
                                         resultHandler: @Sendable @escaping (Result<T, NetworkError>) -> Void) {

    guard let urlRequest = createRequest(endPoint: endpoint) else {
        return
    }
    let urlTask = URLSession.shared.dataTask(with: urlRequest) { data, response, error in
        guard error == nil else {
            resultHandler(.failure(.invalidURL))
            return
        }
        guard let response = response as? HTTPURLResponse, 200...299 ~= response.statusCode else {
            resultHandler(.failure(.unexpectedStatusCode))
            return
        }
    }
}
```

Zero data Races safety errors

 Swift Package Index  Supporters Add a Package Blog FAQ Search

Track the adoption of Swift 6 strict concurrency checks for data race safety. How many packages are Ready for Swift 6?

Home > Kanagasabapathy Rajkumar > NetworkKit

NetworkKit

sabapathyk7/NetworkKit

Use this Package

Powerful network layer seamlessly integrating Combine Framework, Async/Await, and Closures.

Written by Kanagasabapathy, Kanagasabapathy Rajkumar, and 1 other contributor.

In development for 7 months, with 37 commits and 9 releases.

There are no open issues and no open pull requests. The last pull request was merged/closed 11 days ago.

This package has no package dependencies.

MIT licensed 13 stars

1 library No executables

No plugins No macros

Zero data race safety errors
What is data race safety and how is it tested?

apple 85 | async-await 46 | closure 7 | combine 93 | ios 1.4K | network 28
package 43 | swift 3.8K | swift-package-manager 497

View on GitHub Try in a Playground

1.0.8 Latest Release
Released 1 day ago

main Default Branch
Modified 1 day ago

Do you maintain this package?
Get shields.io compatibility badges and learn how to control our build system. [Learn more](#).

Compatibility

6.0 (beta) 5.10 5.9 5.8

1.0.8 (beta) and main (beta)

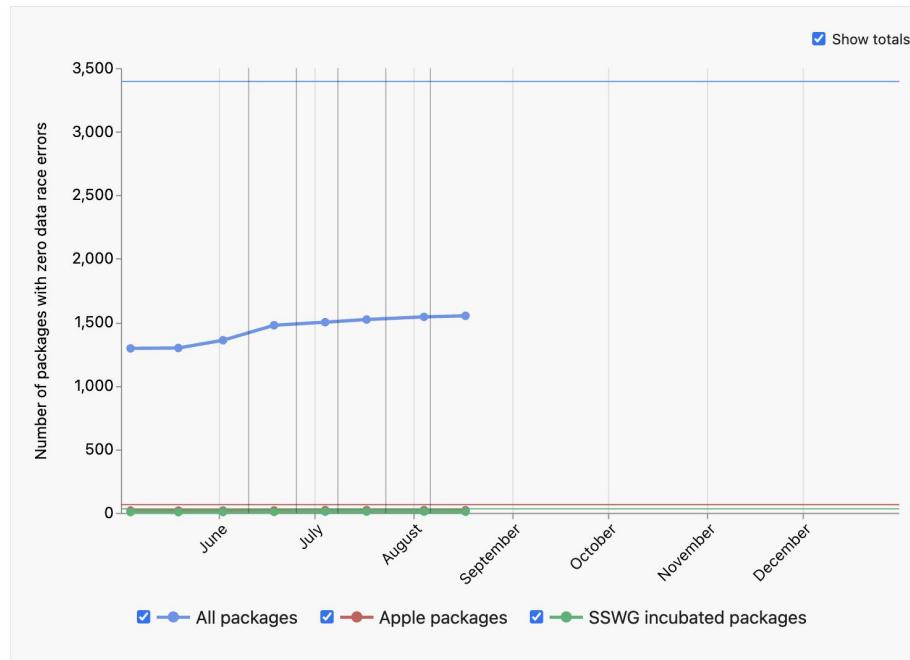
Full Build Results



Zero data Races safety errors

Total packages with Swift 6 zero data race safety errors

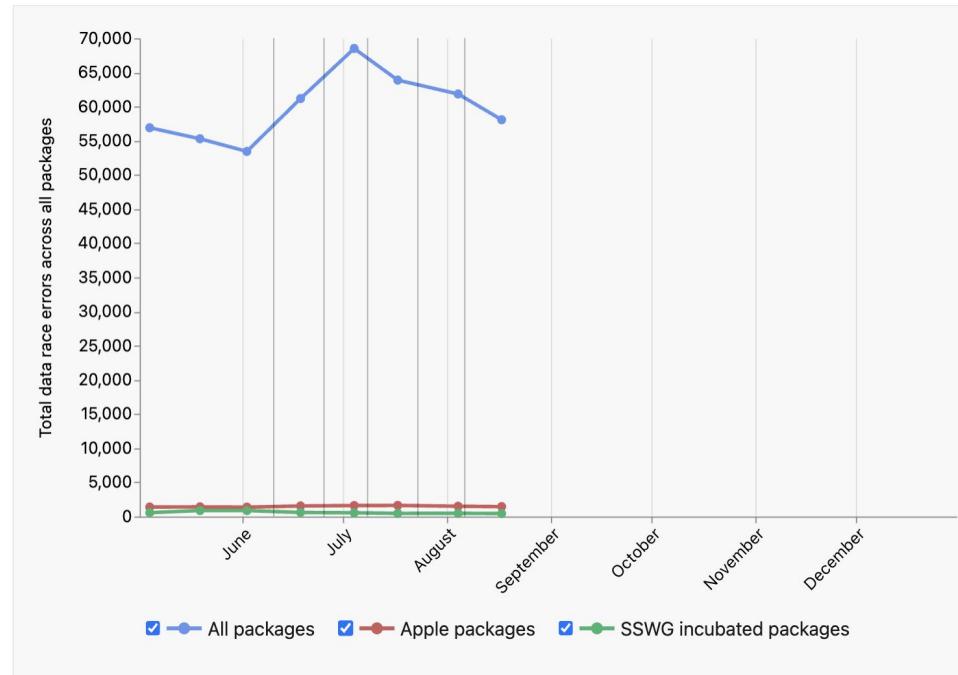
This chart shows packages with zero data race safety compiler diagnostics during a successful build on at least one tested platform.



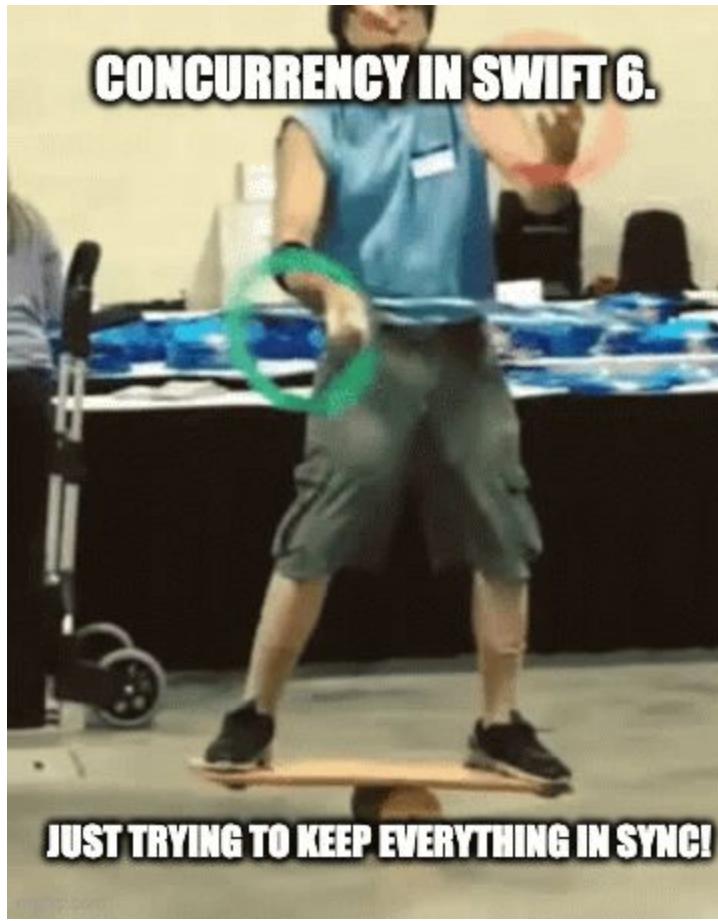
Data Races safety errors

Total Swift 6 data race safety errors

This chart shows the total number of all data race safety diagnostics across *all* packages.



CONCURRENCY IN SWIFT 6.



JUST TRYING TO KEEP EVERYTHING IN SYNC!

Key Takeaways from Swift Concurrency

- Use **async/await** for API Executions
 - ◆ Simplifies async code and more readability compared to callbacks/completion handlers
- Replace Callbacks/Completion Handlers with **Continuations**
 - ◆ Bridging existing callbacks
- Use **async-let** for Concurrent Operations
 - ◆ Run multiple async operations concurrently - Best fit for downloading images
- Use **Structured Concurrency** to Manage Task Lifecycles
 - ◆ Preventing potential memory leaks or child tasks
- Use **Sendable** for Safe Concurrency
 - ◆ Safe shared across isolation boundaries and reducing data races
- Use **Actors** and **Isolation** to Eliminate Data Races
 - ◆ Built-in mechanism
- Use **@preconcurrency** for Legacy Code
 - ◆ Silence concurrency warnings in legacy codebase
- Enable Concurrency Checking and Prepare for **Swift 6**
 - ◆ Concurrency checks and catch potential issues

Scan the QR Code to Download



Connect with me on LinkedIn - *sabapathy7*

Q & A

**Thank
You**