Structure

CheckedContinuation

A mechanism to interface between synchronous and asynchronous code, logging correctness violations.

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

```
struct CheckedContinuation<T, E> where E : Error
```

Overview

A *continuation* is an opaque representation of program state. To create a continuation in asynchronous code, call the `withUnsafeContinuation(function:_:)` or `withUnsafeThrowingContinuation(function:_:)` function. To resume the asynchronous task, call the `resume(returning:)`, `resume(throwing:)`, `resume(with:)`, or `resume()` method.

> **Important**
>
> You must call a resume method exactly once on every execution path throughout the program.

Resuming from a continuation more than once is undefined behavior. Never resuming leaves the task in a suspended state indefinitely, and leaks any associated resources. `CheckedContinuation` logs a message if either of these invariants is violated.

`CheckedContinuation` performs runtime checks for missing or multiple resume operations. `UnsafeContinuation` avoids enforcing these invariants at runtime because it aims to be a low-overhead mechanism for interfacing Swift tasks with event loops, delegate methods, callbacks, and other non-`async` scheduling mechanisms. However, during development, the ability to verify that the invariants are being upheld in testing is important. Because both types have the same

interface, you can replace one with the other in most circumstances, without making other changes.

Topics

Initializers

`init(continuation: UnsafeContinuation<T, E>, function: String)`

Creates a checked continuation from an unsafe continuation.

Instance Methods

`func resume()`

Resume the task awaiting the continuation by having it return normally from its suspension point.

`func resume(returning: sending T)`

Resume the task awaiting the continuation by having it return normally from its suspension point.

`func resume(throwing: E)`

Resume the task awaiting the continuation by having it throw an error from its suspension point.

`func resume(with: sending Result<T, E>)`

Resume the task awaiting the continuation by having it either return normally or throw an error based on the state of the given `Result` value.

`func resume<Er>(with: sending Result<T, Er>)`

Resume the task awaiting the continuation by having it either return normally or throw an error based on the state of the given `Result` value.

Relationships

Conforms To

`Sendable`, `SendableMetatype`

See Also

Continuations

`func withCheckedContinuation<T>(isolation: isolated (any Actor)?, function: String, (CheckedContinuation<T, Never>) -> Void) async -> sending T`

Invokes the passed in closure with a checked continuation for the current task.

`func withCheckedThrowingContinuation<T>(isolation: isolated (any Actor)?, function: String, (CheckedContinuation<T, any Error>) -> Void) async throws -> sending T`

Invokes the passed in closure with a checked continuation for the current task.

`struct UnsafeContinuation`

A mechanism to interface between synchronous and asynchronous code, without correctness checking.

`func withUnsafeContinuation<T>(isolation: isolated (any Actor)?, (UnsafeContinuation<T, Never>) -> Void) async -> sending T`

Invokes the passed in closure with a unsafe continuation for the current task.

~~`typealias UnsafeThrowingContinuation`~~ Deprecated

`func withUnsafeThrowingContinuation<T>(isolation: isolated (any Actor)?, (UnsafeContinuation<T, any Error>) -> Void) async throws -> sending T`

Invokes the passed in closure with a unsafe continuation for the current task.