

[Swift](#) / ThrowingTaskGroup

Structure

ThrowingTaskGroup

A group that contains throwing, dynamically created child tasks.

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

```
@frozen
struct ThrowingTaskGroup<ChildTaskResult, Failure> where ChildTask
Result : Sendable, Failure : Error
```

Overview

To create a throwing task group, call the `withThrowingTaskGroup(of:returning:body:)` method.

Don't use a task group from outside the task where you created it. In most cases, the Swift type system prevents a task group from escaping like that because adding a child task to a task group is a mutating operation, and mutation operations can't be performed from concurrent execution contexts like a child task.

Refer to [TaskGroup](#) documentation for detailed discussion of semantics shared between all task groups.

Cancellation behavior

A task group becomes canceled in one of the following ways:

- when `cancelAll()` is invoked on it,
- when an error is thrown out of the `withThrowingTaskGroup(...){ } closure`,
- when the [Task](#) running this task group is canceled.

Since a `ThrowingTaskGroup` is a structured concurrency primitive, cancellation is automatically propagated through all of its child-tasks (and their child tasks).

A canceled task group can still keep adding tasks, however they will start being immediately canceled, and may act accordingly to this. To avoid adding new tasks to an already canceled task group, use `addTaskUnlessCancelled(priority:body:)` rather than the plain `addTask(priority:body:)` which adds tasks unconditionally.

For information about the language-level concurrency model that `ThrowingTaskGroup` is part of, see [Concurrency in The Swift Programming Language](#).

See Also

[TaskGroup](#)

See Also

[DiscardingTaskGroup](#)

See Also

[ThrowingDiscardingTaskGroup](#)

Topics

Adding Tasks to a Throwing Task Group

```
func addTask(priority: TaskPriority?, operation: sending () async throws -> ChildTaskResult)
```

Adds a child task to the group.

```
func addTask(executorPreference: (any TaskExecutor)?, priority: TaskPriority?, operation: sending () async throws -> ChildTaskResult)
```

Adds a child task to the group.

```
func addTask(name: String?, priority: TaskPriority?, operation: sending () async throws -> ChildTaskResult)
```

Adds a child task to the group.

```
func addTask(name: String?, executorPreference: (any TaskExecutor)?, priority: TaskPriority?, operation: sending () async throws -> ChildTaskResult)
```

Adds a child task to the group.

```
func addTaskUnlessCancelled(priority: TaskPriority?, operation: sending () async throws -> ChildTaskResult) -> Bool
```

Adds a child task to the group, unless the group has been canceled. Returns a boolean value indicating if the task was successfully added to the group or not.

```
func addTaskUnlessCancelled(name: String?, executorPreference: (any TaskExecutor)?, priority: TaskPriority?, operation: sending () async throws -> ChildTaskResult) -> Bool
```

Adds a child task to the group, unless the group has been canceled. Returns a boolean value indicating if the task was successfully added to the group or not.

```
func addTaskUnlessCancelled(executorPreference: (any TaskExecutor)?, priority: TaskPriority?, operation: sending () async throws -> ChildTaskResult) -> Bool
```

Adds a child task to the group, unless the group has been canceled. Returns a boolean value indicating if the task was successfully added to the group or not.

```
func addTaskUnlessCancelled(name: String?, priority: TaskPriority?, operation: sending () async throws -> ChildTaskResult) -> Bool
```

Adds a child task to the group, unless the group has been canceled. Returns a boolean value indicating if the task was successfully added to the group or not.

```
func addImmediateTask(name: String?, priority: TaskPriority?, executorPreference: consuming (any TaskExecutor)?, operation: sending () async throws -> ChildTaskResult)
```

Add a child task to the group and immediately start running it in the context of the calling thread/task.

```
func addImmediateTaskUnlessCancelled(name: String?, priority: TaskPriority?, executorPreference: consuming (any TaskExecutor)?, operation: sending () async throws -> ChildTaskResult) -> Bool
```

Add a child task to the group and immediately start running it in the context of the calling thread/task.

Accessing Individual Results

```
func next() async throws -> ChildTaskResult?
```

```
func nextResult(isolation: isolated (any Actor)?) async -> Result<ChildTaskResult, Failure>?
```

Wait for the next child task to complete, and return a result containing either the value that the child task returned or the error that it threw.

```
func next(isolation: isolated (any Actor)?) async throws -> ChildTaskResult?
```

Wait for the next child task to complete, and return the value it returned or rethrow the error it threw.

```
var isEmpty: Bool
```

A Boolean value that indicates whether the group has any remaining tasks.

```
func waitForAll(isolation: isolated (any Actor)?) async throws
```

Wait for all of the group's remaining tasks to complete.

Accessing an Asynchronous Sequence of Results

```
func makeAsyncIterator() -> ThrowingTaskGroup<ChildTaskResult, Failure>.Iterator
```

Creates the asynchronous iterator that produces elements of this asynchronous sequence.

```
func allSatisfy((Self.Element) async throws -> Bool) async rethrows -> Bool
```

Returns a Boolean value that indicates whether all elements produced by the asynchronous sequence satisfy the given predicate.

```
func compactMap<ElementOfResult>((Self.Element) async throws -> ElementOfResult?) -> AsyncThrowingCompactMapSequence<Self, ElementOfResult>
```

Creates an asynchronous sequence that maps an error-throwing closure over the base sequence's elements, omitting results that don't return a value.

```
func compactMap<ElementOfResult>((Self.Element) async -> ElementOfResult?) -> AsyncCompactMapSequence<Self, ElementOfResult>
```

Creates an asynchronous sequence that maps the given closure over the asynchronous sequence's elements, omitting results that don't return a value.

```
func contains(Self.Element) async rethrows -> Bool
```

Returns a Boolean value that indicates whether the asynchronous sequence contains the given element.

```
func contains(where: (Self.Element) async throws -> Bool) async  
rethrows -> Bool
```

Returns a Boolean value that indicates whether the asynchronous sequence contains an element that satisfies the given predicate.

```
func drop(while: (Self.Element) async -> Bool) -> AsyncDropWhile  
Sequence<Self>
```

Omits elements from the base asynchronous sequence until a given closure returns false, after which it passes through all remaining elements.

```
func dropFirst(Int) -> AsyncDropFirstSequence<Self>
```

Omits a specified number of elements from the base asynchronous sequence, then passes through all remaining elements.

```
func filter((Self.Element) async -> Bool) -> AsyncFilterSequence<Self>
```

Creates an asynchronous sequence that contains, in order, the elements of the base sequence that satisfy the given predicate.

```
func first(where: (Self.Element) async throws -> Bool) async rethrows ->  
Self.Element?
```

Returns the first element of the sequence that satisfies the given predicate.

```
func map<Transformed>((Self.Element) async throws -> Transformed) ->  
AsyncThrowingMapSequence<Self, Transformed>
```

Creates an asynchronous sequence that maps the given error-throwing closure over the asynchronous sequence's elements.

```
func map<Transformed>((Self.Element) async -> Transformed) -> AsyncMap  
Sequence<Self, Transformed>
```

Creates an asynchronous sequence that maps the given closure over the asynchronous sequence's elements.

```
func max() async rethrows -> Self.Element?
```

Returns the maximum element in an asynchronous sequence of comparable elements.

```
func max(by: (Self.Element, Self.Element) async throws -> Bool) async  
rethrows -> Self.Element?
```

Returns the maximum element in the asynchronous sequence, using the given predicate as the comparison between elements.

```
func min() async rethrows -> Self.Element?
```

Returns the minimum element in an asynchronous sequence of comparable elements.

```
func min(by: (Self.Element, Self.Element) async throws -> Bool) async  
rethrows -> Self.Element?
```

Returns the minimum element in the asynchronous sequence, using the given predicate as the comparison between elements.

```
func prefix(Int) -> AsyncPrefixSequence<Self>
```

Returns an asynchronous sequence, up to the specified maximum length, containing the initial elements of the base asynchronous sequence.

```
func prefix(while: (Self.Element) async -> Bool) rethrows -> Async  
PrefixWhileSequence<Self>
```

Returns an asynchronous sequence, containing the initial, consecutive elements of the base sequence that satisfy the given predicate.

```
func reduce<Result>(Result, (Result, Self.Element) async throws ->  
Result) async rethrows -> Result
```

Returns the result of combining the elements of the asynchronous sequence using the given closure.

```
func reduce<Result>(into: Result, (inout Result, Self.Element) async  
throws -> Void) async rethrows -> Result
```

Returns the result of combining the elements of the asynchronous sequence using the given closure, given a mutable initial value.

Canceling Tasks

```
var isCancelled: Bool
```

A Boolean value that indicates whether the group was canceled.

```
func cancelAll()
```

Cancel all of the remaining tasks in the group.

Supporting Types

```
typealias Element
```

The type of element produced by this asynchronous sequence.

```
struct Iterator
```

A type that provides an iteration interface over the results of tasks added to the group.

```
typealias AsyncIterator
```

The type of asynchronous iterator that produces elements of this asynchronous sequence.

Deprecated

```
func add(priority: TaskPriority?, operation: () async throws -> ChildTaskResult) async -> Bool
```

Deprecated

```
func async(priority: TaskPriority?, operation: () async throws -> ChildTaskResult)
```

Deprecated

```
func asyncUnlessCancelled(priority: TaskPriority?, operation: () async throws -> ChildTaskResult) -> Bool
```

Deprecated

```
func spawn(priority: TaskPriority?, operation: () async throws -> ChildTaskResult)
```

Deprecated

```
func spawnUnlessCancelled(priority: TaskPriority?, operation: () async throws -> ChildTaskResult) -> Bool
```

Deprecated

Default Implementations

≡ AsyncSequence Implementations

Relationships

Conforms To

AsyncSequence

Conforms when `ChildTaskResult` conforms to `Copyable`, `ChildTaskResult` conforms to `Escapable`, `ChildTaskResult` conforms to `Sendable`, and `Failure` conforms to `Error`.

BitwiseCopyable

Conforms when `ChildTaskResult` conforms to `Copyable`, `ChildTaskResult` conforms to `Escapable`, `ChildTaskResult` conforms to `Sendable`, and `Failure` conforms to `Error`.

Copyable

Conforms when `ChildTaskResult` conforms to `Copyable`, `ChildTaskResult` conforms to `Escapable`, `ChildTaskResult` conforms to `Sendable`, and `Failure` conforms to `Error`.

See Also

Tasks

`struct Task`

A unit of asynchronous work.

`struct TaskGroup`

A group that contains dynamically created child tasks.

```
func withTaskGroup<ChildTaskResult, GroupResult>(of: ChildTaskResult.Type, returning: GroupResult.Type, isolation: isolated (any Actor)?, body: (inout TaskGroup<ChildTaskResult>) async -> GroupResult) async -> GroupResult
```

Starts a new scope that can contain a dynamic number of child tasks.

```
func withThrowingTaskGroup<ChildTaskResult, GroupResult>(of: ChildTaskResult.Type, returning: GroupResult.Type, isolation: isolated (any Actor)?, body: (inout ThrowingTaskGroup<ChildTaskResult, any Error>) async throws -> GroupResult) async rethrows -> GroupResult
```

Starts a new scope that can contain a dynamic number of throwing child tasks.

`struct TaskPriority`

The priority of a task.

`struct DiscardingTaskGroup`

A discarding group that contains dynamically created child tasks.

```
func withDiscardingTaskGroup<GroupResult>(returning: GroupResult.Type, isolation: isolated (any Actor)?, body: (inout DiscardingTaskGroup) async -> GroupResult) async -> GroupResult
```

Starts a new scope that can contain a dynamic number of child tasks.

`struct ThrowingDiscardingTaskGroup`

A throwing discarding group that contains dynamically created child tasks.

```
func withThrowingDiscardingTaskGroup<GroupResult>(returning: GroupResult.Type, isolation: isolated (any Actor)?, body: (inout ThrowingDiscardingTaskGroup<any Error>) async throws -> GroupResult) async throws -> GroupResult
```

Starts a new scope that can contain a dynamic number of child tasks.

```
struct UnsafeCurrentTask
```

An unsafe reference to the current task.