

[Swift / withThrowingTaskGroup\(of:returning:isolation:body:\)](#)

Function

withThrowingTaskGroup(of:returning:isolation:body:)

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

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

```
@backDeployed(before: macOS 15.0, iOS 18.0, watchOS 11.0, tvOS 18.0, visionOS 2.0)
func withThrowingTaskGroup<ChildTaskResult, GroupResult>(
    of childTaskResultType: ChildTaskResult.Type = ChildTaskResult.self,
    returning returnType: GroupResult.Type = GroupResult.self,
    isolation: isolated (any Actor)? = #isolation,
    body: (inout ThrowingTaskGroup<ChildTask
Result, any Error>) async throws -> GroupResult
) async rethrows -> GroupResult where ChildTaskResult : Sendable
```

Discussion

A group *always* waits for all of its child tasks to complete before it returns. Even canceled tasks must run until completion before this function returns. Canceled child tasks cooperatively react to cancellation and attempt to return as early as possible. After this function returns, the task group is always empty.

To collect the results of the group's child tasks, you can use a `for-await-in` loop:

```
var sum = 0
for try await result in group {
    sum += result
}
```

If you need more control or only a few results, you can call `next()` directly:

```
guard let first = try await group.next() else {
    group.cancelAll()
    return 0
}
let second = await group.next() ?? 0
group.cancelAll()
return first + second
```

Error Handling

Throwing an error in one of the child tasks of a task group doesn't immediately cancel the other tasks in that group. However, throwing out of the body of the `withThrowingTaskGroup` method does cancel the group, and all of its child tasks. For example, if you call `next()` in the task group and propagate its error, all other tasks are canceled. For example, in the code below, nothing is canceled and the group doesn't throw an error:

```
try await withThrowingTaskGroup(of: Void.self) { group in
    group.addTask { throw SomeError() }
}
```

In contrast, this example throws `SomeError` and cancels all of the tasks in the group:

```
try await withThrowingTaskGroup(of: Void.self) { group in
    group.addTask { throw SomeError() }
    try await group.next()
}
```

An individual task throws its error in the corresponding call to `Group.next()`, which gives you a chance to handle the individual error or to let the group rethrow the error.

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

See Also

[TaskGroup](#)

See Also

[ThrowingTaskGroup](#)

See Also

[ThrowingDiscardingTaskGroup](#)

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.

`struct ThrowingTaskGroup`

A group that contains throwing, dynamically created 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.