Instance Method

task(priority:_:)

Adds an asynchronous task to perform before this view appears.

iOS 15.0+ | iPadOS 15.0+ | Mac Catalyst 15.0+ | macOS 12.0+ | tvOS 15.0+ | visionOS 1.0+ | watchOS 8.0+

```
nonisolated
func task(
    priority: TaskPriority = .userInitiated,
    _ action: @escaping () async -> Void
) -> some View
```

Parameters

`priority`
The task priority to use when creating the asynchronous task. The default priority is `userInitiated`.

`action`
A closure that SwiftUI calls as an asynchronous task before the view appears. SwiftUI will automatically cancel the task at some point after the view disappears before the action completes.

Return Value

A view that runs the specified action asynchronously before the view appears.

Mentioned in

Understanding the navigation stack

Discussion

Use this modifier to perform an asynchronous task with a lifetime that matches that of the modified view. If the task doesn't finish before SwiftUI removes the view or the view changes identity, SwiftUI cancels the task.

Use the `await` keyword inside the task to wait for an asynchronous call to complete, or to wait on the values of an `AsyncSequence` instance. For example, you can modify a `Text` view to start a task that loads content from a remote resource:

```
let url = URL(string: "https://example.com")!
@State private var message = "Loading..."

var body: some View {
    Text(message)
        .task {
            do {
                var receivedLines = [String]()
                for try await line in url.lines {
                    receivedLines.append(line)
                    message = "Received \(receivedLines.count) lines"
                }
            } catch {
                message = "Failed to load"
            }
        }
}
```

This example uses the `lines` method to get the content stored at the specified URL as an asynchronous sequence of strings. When each new line arrives, the body of the `for-await-in` loop stores the line in an array of strings and updates the content of the text view to report the latest line count.

See Also

Responding to view life cycle updates

`func onAppear(perform: (() -> Void)?) -> some View`

Adds an action to perform before this view appears.

`func onDisappear(perform: (() -> Void)?) -> some View`

Adds an action to perform after this view disappears.

```
func task<T>(id: T, priority: TaskPriority, () async -> Void) -> some
View
```

Adds a task to perform before this view appears or when a specified value changes.