Instance Method

task(id:priority:_:)

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

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<T>(
    id value: T,
    priority: TaskPriority = .userInitiated,
    _ action: @escaping () async -> Void
) -> some View where T : Equatable
```

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 can automatically cancel the task after the view disappears before the action completes. If the `id` value changes, SwiftUI cancels and restarts the task.

Return Value

A view that runs the specified action asynchronously before the view appears, or restarts the task when the `id` value changes.

Discussion

This method behaves like `task(priority:_:)`, except that it also cancels and recreates the task when a specified value changes. To detect a change, the modifier tests whether a new value for the `id` parameter equals the previous value. For this to work, the value's type must conform to the `Equatable` protocol.

For example, if you define an equatable `Server` type that posts custom notifications whenever its state changes — for example, from *signed out* to *signed in* — you can use the task modifier to update the contents of a `Text` view to reflect the state of the currently selected server:

```
Text(status ?? "Signed Out")
    .task(id: server) {
        let sequence = NotificationCenter.default.notifications(
            named: .didUpdateStatus,
            object: server
        ).compactMap {
            $0.userInfo?["status"] as? String
        }
        for await value in sequence {
            status = value
        }
    }
```

This example uses the `notifications(named:object:)` method to create an asynchronous sequence of notifications, given by an `AsyncSequence` instance. The example then maps the notification sequence to a sequence of strings that correspond to values stored with each notification.

Elsewhere, the server defines a custom `didUpdateStatus` notification:

```
extension NSNotification.Name {
    static var didUpdateStatus: NSNotification.Name {
        NSNotification.Name("didUpdateStatus")
    }
}
```

Whenever the server status changes, like after the user signs in, the server posts a notification of this custom type:

```
let notification = Notification(
    name: .didUpdateStatus,
    object: self,
    userInfo: ["status": "Signed In"])
NotificationCenter.default.post(notification)
```

The task attached to the `Text` view gets and displays the status value from the notification’s user information dictionary. When the user chooses a different server, SwiftUI cancels the task and creates a new one, which then waits for notifications from the new server.

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(priority: TaskPriority, () async -> Void) -> some View`

Adds an asynchronous task to perform before this view appears.