Class

NSWritingToolsCoordinator

An object that manages interactions between Writing Tools and your custom text view.

macOS 15.2+

```
@MainActor
class NSWritingToolsCoordinator
```

Mentioned in

Adding Writing Tools support to a custom AppKit view

Overview

Add a `NSWritingToolsCoordinator` object to a custom view when you want to add Writing Tools support to that view. The coordinator manages interactions between your view and the Writing Tools UI and back-end capabilities. When creating a coordinator, you supply a delegate object to respond to requests from the system and provide needed information. Your delegate delivers your view's text to Writing Tools, incorporates suggested changes back into your text storage, and supports the animations that Writing Tools creates to show the state of an operation.

Create the `NSWritingToolsCoordinator` object when setting up your UI, and initialize it with a custom object that adopts the `NSWritingToolsCoordinator.Delegate` protocol. Add the coordinator to the `writingToolsCoordinator` property of your view. When a coordinator is present on a view, the system adds UI elements to initiate Writing Tools operations.

When defining the delegate, choose an object from your app that has access to your view and its text storage. You can adopt the `NSWritingToolsCoordinator.Delegate` protocol in the view itself, or in another type that your view uses to manage content. During the interactions with

Writing Tools, the delegate gets and sets the contents of the view's text storage and supports Writing Tools behaviors.

> **Note**
>
> You don't need to create an `NSWritingToolsCoordinator` object if you display text using a `UITextView`, `NSTextField`, `NSTextView`, `TextField`, or `TextEditor` view. Those views already include the required support to handle Writing Tools interactions.

Topics

Creating a coordinator object

`init(delegate: (any NSWritingToolsCoordinator.Delegate)?)`

Creates a writing tools coordinator and assigns the specified delegate object to it.

Checking the availability of Writing Tools

`class var isWritingToolsAvailable: Bool`

A Boolean value that indicates whether Writing Tools features are currently available.

Managing Writing Tools interactions

`var delegate: (any NSWritingToolsCoordinator.Delegate)?`

The object that handles Writing Tools interactions for your view.

`protocol Delegate`

An interface that you use to manage interactions between Writing Tools and your custom text view.

`var view: NSView?`

The view that currently uses the writing tools coordinator.

Getting the host views for effects

`var effectContainerView: NSView?`

The view that Writing Tools uses to display visual effects during the text-rewriting process.

`var decorationContainerView: NSView?`

The view that Writing Tools uses to display background decorations such as proofreading marks.

Configuring the experience

`var preferredBehavior: NSWritingToolsBehavior`

The level of Writing Tools support you want the system to provide for your view.

`var behavior: NSWritingToolsBehavior`

The actual level of Writing Tools support the system provides for your view.

`var preferredResultOptions: NSWritingToolsResultOptions`

The type of content you allow Writing Tools to generate for your custom text view.

`var resultOptions: NSWritingToolsResultOptions`

The type of content the system generates for your custom text view.

Reporting changes to Writing Tools

`func updateRange(NSRange, with: NSAttributedString, reason: NSWritingToolsCoordinator.TextUpdateReason, forContextWithIdentifier: UUID)`

Informs the coordinator about changes your app made to the text in the specified context object.

`func updateForReflowedTextInContextWithIdentifier(UUID)`

Informs the coordinator that a change occurred to the view or its text that requires a layout update.

`enum TextUpdateReason`

Constants that specify the reason you updated your view’s content outside of the Writing Tools workflow.

Managing the current state

`func stopWritingTools()`

Stops the current Writing Tools operation and dismisses the system UI.

`var state: NSWritingToolsCoordinator.State`

The current level of Writing Tools activity in your view.

`enum State`

The states that indicate the current activity, if any, Writing Tools is performing in your view.

Getting the supporting types

`enum ContextScope`

Options that indicate how much of your content Writing Tools requested.

`enum TextReplacementReason`

Options that indicate whether Writing Tools is animating changes to your view’s text.

`enum TextAnimation`

The types of animations that Writing Tools performs during an interactive update of your view.

Instance Properties

`var includesTextListMarkers: Bool`

Relationships

Inherits From

`NSObject`

Conforms To

`CVarArg`
`CustomDebugStringConvertible`
`CustomStringConvertible`
`Equatable`
`Hashable`
`NSObjectProtocol`
`Sendable`

See Also

Writing Tools for custom views

Supporting Writing Tools via the pasteboard

Adopt a simplified version of the Writing Tools experience in a custom view using the pasteboard and macOS services.

Adding Writing Tools support to a custom AppKit view

Integrate Writing Tools support, including support for inline replacement animations, to your custom text views on macOS.

`protocol Delegate`

An interface that you use to manage interactions between Writing Tools and your custom text view.

`class Context`

A data object that you use to share your custom view's text with Writing Tools.

`class AnimationParameters`

An object you use to configure additional tasks or animations to run alongside the Writing Tools animations.

{} Enhancing your custom text engine with Writing Tools

Add Writing Tools support to your custom text engine to enhance the text editing experience.