Class

NSWindowController

A controller that manages a window, usually a window stored in a nib file.

macOS

```
@MainActor
class NSWindowController
```

Overview

Managing a window entails:

- Loading and displaying the window
- Closing the window when appropriate
- Customizing the window's title
- Storing the window's frame (size and location) in the defaults database
- Cascading the window in relation to other document windows of the app

A window controller can manage a window by itself or as a role player in AppKit's document-based architecture, which also includes `NSDocument` and `NSDocumentController` objects. In this architecture, a window controller is created and managed by a "document" (an instance of an `NSDocument` subclass) and, in turn, keeps a reference to the document.

The relationship between a window controller and a nib file is important. Although a window controller can manage a programmatically created window, it usually manages a window in a nib file. The nib file can contain other top-level objects, including other windows, but the window controller's responsibility is this primary window. The window controller is usually the owner of the nib file, even when it is part of a document-based app. Regardless of who is the file's owner, the window controller is responsible for freeing all top-level objects in the nib file it loads.

For simple documents—that is, documents with only one nib file containing a window—you need to do little directly with `NSWindowController`; AppKit creates one for you. However, if the default window controller is not sufficient, you can create a custom subclass of `NSWindowController`. For documents with multiple windows or panels, your document must create separate instances of `NSWindowController` (or of custom subclasses of `NSWindowController`), one for each window or panel. An example is a CAD app that has different windows for side, top, and front views of drawn objects. What you do in your `NSDocument` subclass determines whether the default `NSWindowController` or separately created and configured `NSWindowController` objects are used.

Subclassing NSWindowController

You should create a subclass of `NSWindowController` when you want to augment the default behavior, such as to give the window a custom title or to perform some setup tasks before the window is loaded. In your class's initialization method, be sure to invoke on `super` either one of the `initWithWindowNibName:...` initializers or the `init(window:)` initializer. The initializer you choose depends on whether the window object originates in a nib file or is programmatically created.

You can also implement an `NSWindowController` subclass to avoid requiring client code to get the corresponding nib's filename and pass it to `init(windowNibName:)` or `init(windowNibName:owner:)` when instantiating the window controller. The best way to do this is to override `windowNibName` to return the nib's filename and instantiate the window controller by passing `nil` to `init(window:)`. Using the `init(window:)` designated initializer simplifies compliance with Swift initializer requirements.

Typically, you override one of the methods listed below.

Method Name	Description
`windowWillLoad()`	Override to perform tasks before the window nib file is loaded.
`windowDidLoad()`	Override to perform tasks after the window nib file is loaded.
`windowTitle(forDocumentDisplayName:)`	Override to customize the window title.

You can also override `loadWindow()` to get different nib-searching or nib-loading behavior, although there is usually no need to do this.

Topics

Initializing Window Controllers

`init(window: NSWindow?)`

Returns a window controller initialized with a given window.

`convenience init(windowNibName: NSNib.Name)`

Returns a window controller initialized with a nib file.

`convenience init(windowNibName: NSNib.Name, owner: Any)`

Returns a window controller initialized with a nib file and a specified owner for that nib file.

`convenience init(windowNibPath: String, owner: Any)`

Returns a window controller initialized with a nib file at an absolute path and a specified owner.

Loading and Displaying the Window

`func loadWindow()`

Loads the receiver's window from the nib file.

`func showWindow(Any?)`

Displays the window associated with the receiver.

`var isWindowLoaded: Bool`

A Boolean value that indicates whether the nib file containing the receiver's window has been loaded.

`var window: NSWindow?`

The window owned by the receiver.

`func windowDidLoad()`

Sent after the window owned by the receiver has been loaded.

`func windowWillLoad()`

Sent before the window owned by the receiver is loaded.

Accessing the Document

`var document: AnyObject?`

The document associated with the window controller.

`func setDocumentEdited(Bool)`

Sets the document edited flag for the window controller.

Closing the Window

`func close()`

Closes the window if it was loaded.

`var shouldCloseDocument: Bool`

A Boolean value that indicates whether the receiver necessarily closes the associated document when the window it manages is closed.

Getting Nib and Storyboard Information

`var owner: AnyObject?`

The owner of the nib file containing the window managed by the receiver.

`var storyboard: NSStoryboard?`

The storyboard file from which the window controller was loaded.

`var windowNibName: NSNib.Name?`

The name of the nib file that stores the window associated with the receiver.

`var windowNibPath: String?`

The full path of the nib file that stores the window associated with the receiver.

Accessing Window Attributes and Content

`var shouldCascadeWindows: Bool`

A Boolean value that indicates whether the window will cascade in relation to other document windows when it is displayed.

`var windowFrameAutosaveName: NSWindow.FrameAutosaveName`

The name under which the frame rectangle of the window owned by the receiver is stored in the defaults database.

`func synchronizeWindowTitleWithDocumentName()`

Synchronizes the displayed window title and the represented filename with the information in the associated document.

`func windowTitle(forDocumentDisplayName: String) -> String`

Returns the window title to be used for a given document display name.

`var contentViewController: NSViewController?`

The view controller for the window’s content view.

`func dismissController(Any?)`

Dismisses the window controller.

Window Property Wrappers

`struct WindowLoading`

A property wrapper that loads the receiver’s window.

Initializers

`init?(coder: NSCoder)`

Instance Properties

`var previewRepresentableActivityItems: [any NSPreviewRepresentableActivityItem]?`

Relationships

Inherits From

`NSResponder`

Conforms To

`CVarArg`
`CustomDebugStringConvertible`
`CustomStringConvertible`
`Equatable`

`Hashable`
`NSCoding`
`NSObjectProtocol`
`NSSeguePerforming`
`NSStandardKeyBindingResponding`
`NSTouchBarProvider`
`NSUserActivityRestoring`
`Sendable`
`SendableMetatype`

See Also

Content Controllers

`class NSViewController`

A controller that manages a view, typically loaded from a nib file.

`class NSTitlebarAccessoryViewController`

An object that manages a custom view—known as an accessory view—in the title bar–toolbar area of a window.