Class

NSResponder

An abstract class that forms the basis of event and command processing in AppKit.

macOS

```
@MainActor
class NSResponder
```

Overview

The core classes—`NSApplication`, `NSWindow`, and `NSView`—inherit from `NSResponder`, as must any class that handles events. The responder model uses three components: event messages, action messages, and the responder chain.

`NSResponder` also plays an important role in the presentation of error information. The default implementations of the `presentError(_:)` and `presentError(_:modalFor:delegate:didPresent:contextInfo:)` methods send `willPresentError(_:)` to `self`, thereby giving subclasses the opportunity to customize the localized information presented in error alerts. `NSResponder` then forwards the message to the next responder, passing it the customized `NSError` object. The exact path up the modified responder chain depends on the type of application window:

- Window that the document owns: view > superviews > window > window controller > document object > document controller > the application object
- Window with window controller but no documents: view > superviews > window > window controller > the application object
- Window with no window controllers: view > superviews > window > the application object

`NSApplication` displays a document-modal error alert and, if the error object has a recovery attempter, gives it a chance to recover from the error. A recovery attempter is an object that conforms to the NSErrorRecoveryAttempting informal protocol.

> **Note**
>
> In macOS 10.15 and later, `NSResponder` and its descendants call the `dealloc` method on the main thread. This method helps to avoid situations where an asynchronous block unexpectedly deallocates an object on a background queue.

Topics

Changing the First Responder

`var acceptsFirstResponder: Bool`

A Boolean value that indicates whether the responder accepts first responder status.

`func becomeFirstResponder() -> Bool`

Notifies the receiver that it's about to become first responder in its `NSWindow`.

`func resignFirstResponder() -> Bool`

Notifies the receiver that it's been asked to relinquish its status as first responder in its window.

`func validateProposedFirstResponder(NSResponder, for: NSEvent?) -> Bool`

Allows controls to determine when they should become first responder.

Managing the Next Responder

`var nextResponder: NSResponder?`

The next responder after this one, or `nil` if it has none.

Responding to Mouse Events

`func mouseDown(with: NSEvent)`

Informs the receiver that the user has pressed the left mouse button.

`func mouseDragged(with: NSEvent)`

Informs the receiver that the user has moved the mouse with the left button pressed.

`func mouseUp(with: NSEvent)`

Informs the receiver that the user has released the left mouse button.

`func mouseMoved(with: NSEvent)`

Informs the receiver that the mouse has moved.

`func mouseEntered(with: NSEvent)`

Informs the receiver that the cursor has entered a tracking rectangle.

`func mouseExited(with: NSEvent)`

Informs the receiver that the cursor has exited a tracking rectangle.

`func rightMouseDown(with: NSEvent)`

Informs the receiver that the user has pressed the right mouse button.

`func rightMouseDragged(with: NSEvent)`

Informs the receiver that the user has moved the mouse with the right button pressed.

`func rightMouseUp(with: NSEvent)`

Informs the receiver that the user has released the right mouse button.

`func otherMouseDown(with: NSEvent)`

Informs the receiver that the user has pressed a mouse button other than the left or right one.

`func otherMouseDragged(with: NSEvent)`

Informs the receiver that the user has moved the mouse with a button other than the left or right button pressed.

`func otherMouseUp(with: NSEvent)`

Informs the receiver that the user has released a mouse button other than the left or right button.

Responding to Key Events

`func keyDown(with: NSEvent)`

Informs the receiver that the user has pressed a key.

`func keyUp(with: NSEvent)`

Informs the receiver that the user has released a key.

`func interpretKeyEvents([NSEvent])`

Handles a series of key events.

`func performKeyEquivalent(with: NSEvent) -> Bool`

Handle a key equivalent.

`func flushBufferedKeyEvents()`

Clears any unprocessed key events when overridden by subclasses.

Responding to Pressure Changes

`func pressureChange(with: NSEvent)`

Indicates a pressure change as the result of a user input event on a system that supports pressure sensitivity.

Responding to Other Kinds of Events

`func cursorUpdate(with: NSEvent)`

Informs the receiver that the mouse cursor has moved into a cursor rectangle.

`func flagsChanged(with: NSEvent)`

Informs the receiver that the user has pressed or released a modifier key (Shift, Control, and so on).

`func tabletPoint(with: NSEvent)`

Informs the receiver that a tablet-point event has occurred.

`func tabletProximity(with: NSEvent)`

Informs the receiver that a tablet-proximity event has occurred.

`func helpRequested(NSEvent)`

Displays context-sensitive help for the receiver if help has been registered.

`func scrollWheel(with: NSEvent)`

Informs the receiver that the mouse's scroll wheel has moved.

`func quickLook(with: NSEvent)`

Performs a Quick Look on the content at the location specified by the supplied event.

`func changeMode(with: NSEvent)`

Informs the responder that performed a double-tap on the side of an Apple Pencil.

Responding to Action Messages

`func supplementalTarget(forAction: Selector, sender: Any?) -> Any?`

Finds a target for an action method.

`protocol NSStandardKeyBindingResponding`

Methods that responder subclasses implement to support key binding commands, such as inserting tabs and newlines, or moving the insertion point.

Action Messages

Implement action messages in your first responders to handle common tasks.

Handling Window Restoration

`class func allowedClasses(forRestorableStateKeyPath: String) -> [AnyClass]`

Returns the classes that support secure coding.

`func encodeRestorableState(with: NSCoder)`

Saves the interface-related state of the responder.

`func encodeRestorableState(with: NSCoder, backgroundQueue: OperationQueue)`

Saves the interface-related state of the responder to a keyed archiver either synchronously or asynchronously on the given operation queue.

`func restoreState(with: NSCoder)`

Restores the interface-related state of the responder.

`class var restorableStateKeyPaths: [String]`

Returns an array of key paths representing the restorable attributes of the responder.

`func invalidateRestorableState()`

Marks the responder’s interface-related state as dirty.

Supporting User Activities

`var userActivity: NSUserActivity?`

An object encapsulating a user activity supported by this responder.

`func updateUserActivityState(NSUserActivity)`

Updates the state of the given user activity.

Presenting and Customizing Error Information

```
func presentError(any Error) -> Bool
```

Presents an error alert to the user as an application-modal dialog.

```
func presentError(any Error, modalFor: NSWindow, delegate: Any?, did
Present: Selector?, contextInfo: UnsafeMutableRawPointer?)
```

Presents an error alert to the user as a document-modal sheet attached to document window.

```
func willPresentError(any Error) -> any Error
```

Returns a custom version of the supplied error object that’s more suitable for presentation in alert sheets and dialogs.

Dispatching Messages

```
func tryToPerform(Selector, with: Any?) -> Bool
```

Attempts to perform the method indicated by an action with a specified argument.

Managing a Responder’s Menu

```
var menu: NSMenu?
```

Returns the responder’s menu.

Updating the Services Menu

```
func validRequestor(forSendType: NSPasteboard.PasteboardType?, return
Type: NSPasteboard.PasteboardType?) -> Any?
```

Overridden by subclasses to determine what services are available.

Getting the Undo Manager

```
var undoManager: UndoManager?
```

The undo manager for this responder.

Testing Events

```
func shouldBeTreatedAsInkEvent(NSEvent) -> Bool
```

Indicates whether a pen-down event should be treated as an ink event.

Terminating the Responder Chain

`func noResponder(for: Selector)`

Handles the case where an event or action message falls off the end of the responder chain.

Touch and Gesture Events

`func beginGesture(with: NSEvent)`

Informs the receiver that the user has begun a touch gesture.

`func endGesture(with: NSEvent)`

Informs the receiver that the user has ended a touch gesture.

`func magnify(with: NSEvent)`

Informs the receiver that the user has begun a pinch gesture.

`func rotate(with: NSEvent)`

Informs the receiver that the user has begun a rotation gesture.

`func swipe(with: NSEvent)`

Informs the receiver that the user has begun a swipe gesture.

`func touchesBegan(with: NSEvent)`

Informs the receiver that new set of touches has been recognized.

`func touchesMoved(with: NSEvent)`

Informs the receiver that one or more touches has moved.

`func touchesCancelled(with: NSEvent)`

Informs the receiver that tracking of touches has been cancelled for any reason.

`func touchesEnded(with: NSEvent)`

Returns that a set of touches have been removed.

`func wantsForwardedScrollEvents(for: NSEvent.GestureAxis) -> Bool`

Returns whether to forward elastic scrolling gesture events up the responder.

`func smartMagnify(with: NSEvent)`

Informs the receiver that the user performed a smart zoom gesture.

`func wantsScrollEventsForSwipeTracking(on: NSEvent.GestureAxis) -> Bool`

Implement this method to track gesture scroll events such as a swipe.

`enum GestureAxis`

Constants that specify the direction of travel for a gesture.

Supporting the Touch Bar

`var touchBar: NSTouchBar?`

The `NSTouchBar` object associated with the responder.

`func makeTouchBar() -> NSTouchBar?`

Your custom subclass of the `NSResponder` class should override this method to create and configure your subclass's default `NSTouchBar` object.

Performing Text Find Actions

`func performTextFinderAction(Any?)`

Performs all find oriented actions.

Supporting Tabbed Windows

`func newWindowForTab(Any?)`

Creates a new window to show as a tab in a tabbed window.

Creating Responders

`init()`

Creates a new responder object.

`init?(coder: NSCoder)`

Creates a new responder object with data in an unarchiver.

Instance Methods

`func contextMenuKeyDown(NSEvent)`

`func mouseCancelled(with: NSEvent)`

```
func showWritingTools(Any?)
```

Relationships

Inherits From

`NSObject`

Inherited By

`NSApplication`
`NSDrawer`
`NSPopover`
`NSView`
`NSViewController`
`NSWindow`
`NSWindowController`

Conforms To

`CVarArg`
`Copyable`
`CustomDebugStringConvertible`
`CustomStringConvertible`
`Equatable`
`Hashable`
`NSCoding`
`NSObjectProtocol`
`NSStandardKeyBindingResponding`
`NSTouchBarProvider`
`NSUserActivityRestoring`
`Sendable`