AppKit / NSView

Class

NSView

The infrastructure for drawing, printing, and handling events in an app.

macOS

```
@MainActor
class NSView
```

Mentioned in

- Supporting Writing Tools via the pasteboard
- Adding Writing Tools support to a custom AppKit view

Overview

You typically don't use `NSView` objects directly. Instead, you use objects that descend from `NSView` or you subclass `NSView` yourself and override its methods to implement the behavior you need. An instance of the `NSView` class (or one of its subclasses) is commonly known as a view object, or simply as a view.

Views handle the presentation and interaction with your app's visible content. You arrange one or more views inside an `NSWindow` object, which acts as a wrapper for your content. A view object defines a rectangular region for drawing and receiving mouse events. Views handle other chores as well, including the dragging of icons and working with the `NSScrollView` class to support efficient scrolling.

AppKit handles most of your app's `NSView` management. Unless you're implementing a concrete subclass of `NSView` or working intimately with the content of the view hierarchy at runtime, you don't need to know much about this class's interface. For any view, there are many methods that you can use as-is. The following methods are commonly used.

- `frame` returns the location and size of the `NSView` object.
- `bounds` returns the internal origin and size of the `NSView` object.
- `needsDisplay` determines whether the `NSView` object needs to be redrawn.
- `window` returns the `NSWindow` object that contains the `NSView` object.
- `draw(_:)` draws the `NSView` object. (All subclasses must implement this method, but it's rarely invoked explicitly.) An alternative to drawing is to update the layer directly using the `updateLayer()` method.

For more information on how `NSView` instances handle event and action messages, see Cocoa Event Handling Guide. For more information on displaying tooltips and contextual menus, see Displaying Contextual Menus and Managing Tooltips.

Subclassing notes

`NSView` is perhaps the most important class in AppKit when it comes to subclassing and inheritance. Most user-interface objects you see in a Cocoa application are objects that inherit from `NSView`. If you want to create an object that draws itself in a special way, or that responds to mouse clicks in a special way, you would create a custom subclass of `NSView` (or of a class that inherits from `NSView`). Subclassing `NSView` is such a common and important procedure that several technical documents describe how to both draw in custom subclasses and respond to events in custom subclasses. See Cocoa Drawing Guide and Cocoa Event Handling Guide (especially "Handling Mouse Events" and "Mouse Events").

Handling events in your subclass

If you subclass `NSView` directly and handle specific types of events, don't call `super` in the implementations of your event-related methods. Views inherit their event-handling capabilities from their `NSResponder` parent class. The default behavior for responders is to pass events up the responder chain, which isn't the behavior you typically want for a custom view. Therefore, don't call `super` if your view implements any of the following methods and handles the event:

- `mouseDown(with:)`
- `mouseDragged(with:)`
- `mouseUp(with:)`
- `mouseMoved(with:)`
- `mouseEntered(with:)`
- `mouseExited(with:)`
- `rightMouseDragged(with:)`

- `rightMouseUp(with:)`
- `otherMouseDown(with:)`
- `otherMouseDragged(with:)`
- `otherMouseUp(with:)`
- `scrollWheel(with:)`
- `keyDown(with:)`
- `keyUp(with:)`
- `flagsChanged(with:)`
- `tabletPoint(with:)`
- `tabletProximity(with:)`

Note

`NSView` changes the default behavior of `rightMouseDown(with:)` so that it calls `menu(for:)` and, if non `nil`, presents the contextual menu. In macOS 10.7 and later, if the event is not handled, `NSView` passes the event up the responder chain. Because of these behaviorial changes, call `super` when implementing `rightMouseDown(with:)` in your custom `NSView` subclasses.

If your view descends from a class other than `NSView`, call `super` to let the parent view handle any events that you don't.

Topics

Creating a view object

`init(frame: NSRect)`

Initializes and returns a newly allocated `NSView` object with a specified frame rectangle.

`init?(coder: NSCoder)`

Initializes a view using from data in the specified coder object.

`func prepareForReuse()`

Restores the view to an initial state so that it can be reused.

Configuring the view

- View Hierarchy

  Manage the subviews, superview, and window of a view and respond to notifications when the view hierarchy changes.

- View Coordinates

  Manage the frame and bounds rectangles that determine the size and position of the view in the view hierarchy.

- Appearance

  Change the view’s visibility, vibrancy, and focus ring and respond to appearance-related changes.

- Core Animation Support

  Manage the layer object that provides the view’s visual representation and accelerates drawing operations.

- Related UI

  Manage contextual menus, cursors, tool tips, and other system-provided windows and content.

Managing the view’s content

- Layout

  Specify the size and position your view relative to other nearby views using rules that update your view hierarchy automatically.

- Drawing

  Draw the content of custom views and update that content when the view’s size or appearance changes.

- Printing

  Create a printable version of your view’s content and handle pagination and printer-related behaviors.

```
protocol NSViewContentSelectionInfo
```

Managing interactions

Event Handling

Respond to mouse, keyboard, touch, and tablet events and gestures that originate inside your view.

Deprecated

Deprecated Symbols

Review unsupported symbols and their replacements.

Structures

```
struct LayoutRegion
```

Instance Properties

```
var prefersCompactControlSizeMetrics: Bool
```

When this property is true, any NSControls in the view or its descendants will be sized with compact metrics compatible with macOS 15 and earlier. Defaults to false

```
var writingToolsCoordinator: NSWritingToolsCoordinator?
```

Instance Methods

```
func edgeInsets(for: NSView.LayoutRegion) -> NSEdgeInsets
```

```
func layoutGuide(for: NSView.LayoutRegion) -> NSLayoutGuide
```

```
func rect(for: NSView.LayoutRegion) -> NSRect
```

Enumerations

```
enum Invalidations
```

Changes that cause aspects of a view to be invalid and require an update.

Relationships

Inherits From

`NSResponder`

Inherited By

`NSBackgroundExtensionView`
`NSBox`
`NSClipView`
`NSCollectionView`
`NSControl`
`NSGlassEffectContainerView`
`NSGlassEffectView`
`NSGridView`
`NSOpenGLView`
`NSProgressIndicator`
`NSRulerView`
`NSScrollView`
`NSScrubber`
`NSScrubberArrangedView`
`NSSplitView`
`NSStackView`
`NSTabView`
`NSTableCellView`
`NSTableHeaderView`
`NSTableRowView`
`NSText`
`NSTextInsertionIndicator`
`NSVisualEffectView`

Conforms To

`CVarArg`
`Copyable`
`CustomDebugStringConvertible`
`CustomStringConvertible`
`Equatable`
`Hashable`
`NSAccessibilityElementProtocol`
`NSAccessibilityProtocol`
`NSAnimatablePropertyContainer`
`NSAppearanceCustomization`
`NSCoding`
`NSDraggingDestination`

`NSObjectProtocol`
`NSStandardKeyBindingResponding`
`NSTouchBarProvider`
`NSUserActivityRestoring`
`NSUserInterfaceItemIdentification`
`PlaygroundLiveViewable`

See Also

View fundamentals

`class NSControl`

A specialized view, such as a button or text field, that notifies your app of relevant events using the target-action design pattern.

`class NSCell`

A mechanism for displaying text or images in a view object without the overhead of a full `NSView` subclass.

`class NSActionCell`

An active area inside a control.