Class

NSButton

A control that defines an area on the screen that a user clicks to trigger an action.

macOS

```
@MainActor
class NSButton
```

Overview

Buttons are a standard control for initiating actions within your app. You can configure buttons with many different visual styles, but the behavior is the same. When a user clicks it, a button calls the action method of its associated target object. (If you configure a button as continuous, it calls its action method at timed intervals until the user releases the mouse button or the cursor leaves the button boundaries). You use the action method to perform your app-specific tasks.

There are multiple types of buttons, each with a different user interface and behavior. The `NSButtonCell` class defines the button types, and calling the `setButtonType(_:)` method configures them.

If you configure it as an accelerator button (type `NSAcceleratorButton` or `NSMultiLevelAcceleratorButton`), you can set a button to send action messages when changes in pressure occur when the user clicks the button.

Buttons can either have two states (on and off) or three states (on, off, and mixed). You enable a three-state button by calling the `allowsMixedState` method. On and off (also referred to as alternate and normal) states indicate that the user clicked or didn't click the button. Mixed is typically used for checkboxes or radio buttons, which allow for an additional intermediate state. For example, suppose the state of a checkbox denotes whether a text field contains bold text. If all text in the text field is bold, then the checkbox is on. If none of the text is bold, then the checkbox is off. If some of the text is bold, then the checkbox is mixed.

For most types of buttons, the value of the button matches its state—the value is 1 for on, 0 for off, or –1 for mixed. For pressure-sensitive buttons, the value of the button indicates pressure level instead.

NSButton and NSMatrix both provide a control view, which displays an NSButtonCell object. However, while a matrix requires you to access the button cell objects directly, most button class methods act as "covers" for identically declared button cell methods. In other words, the implementation of the button method invokes the corresponding button cell method for you, allowing you to be unconcerned with the existence of the button cell. The only button cell methods that don't have covers relate to the font you use to display the key equivalent and to specific methods for highlighting or showing the state of the button.

Topics

Creating standard buttons

`convenience init(checkboxWithTitle: String, target: Any?, action: Selector?)`

Creates a standard checkbox with the title you specify.

`convenience init(image: NSImage, target: Any?, action: Selector?)`

Creates a standard push button with the image you specify.

`convenience init(radioButtonWithTitle: String, target: Any?, action: Selector?)`

Creates a standard radio button with the title you specify.

`convenience init(title: String, image: NSImage, target: Any?, action: Selector?)`

Creates a standard push button with a title and image.

`convenience init(title: String, target: Any?, action: Selector?)`

Creates a standard push button with the title you specify.

Configuring the cell

`class NSButtonCell`

An object that defines the user interface of a button or other clickable region of a view.

Configuring buttons

`func setButtonType(NSButton.ButtonType)`

Sets the button's type, which affects its user interface and behavior when clicked.

`func getPeriodicDelay(UnsafeMutablePointer<Float>, interval: UnsafeMutablePointer<Float>)`

Returns by reference the delay and interval periods for a continuous button.

`func setPeriodicDelay(Float, interval: Float)`

Sets the message delay and interval periods for a continuous button.

`var contentTintColor: NSColor?`

A tint color to use for the template image and text content.

`var hasDestructiveAction: Bool`

A Boolean value that defines whether a button's action has a destructive effect.

`var alternateTitle: String`

The title that the button displays when the button is in an on state.

`var attributedTitle: NSAttributedString`

The title that the button displays in an off state, as an attributed string.

`var attributedAlternateTitle: NSAttributedString`

The title that the button displays as an attributed string when the button is in an on state.

`var title: String`

The title displayed on the button when it's in an off state.

`var symbolConfiguration: NSImage.SymbolConfiguration?`

The combination of point size, weight, and scale to use when sizing and displaying symbol images.

`var sound: NSSound?`

The sound that plays when the user clicks the button.

`var isSpringLoaded: Bool`

A Boolean value that indicates whether spring loading is enabled for the button.

`var maxAcceleratorLevel: Int`

An integer value indicating the maximum pressure level for a button of type `NSMultiLevelAcceleratorButton`.

`var tintProminence: NSTintProminence`

The tint prominence of the button. Use tint prominence to gently suggest a hierarchy when multiple buttons perform similar actions. A button with primary tint prominence suggests the most preferred option, while secondary prominence indicates a reasonable alternative. See `NSTintProminence` for a list of possible values.

`enum NSTintProminence`

Controls how strongly the tint color applies in a view.

`var borderShape: NSControl.BorderShape`

`enum BorderShape`

Configuring button images

`var image: NSImage?`

The image that appears on the button when it's in an off state, or `nil` if there is no such image.

`var alternateImage: NSImage?`

An alternate image that appears on the button when the button is in an on state.

`var imagePosition: NSControl.ImagePosition`

The position of the button's image relative to its title.

`enum ImagePosition`

A constant for specifying the position of a button's image relative to its title.

`var isBordered: Bool`

A Boolean value that determines whether the button has a border.

`var isTransparent: Bool`

A Boolean value that indicates whether the button is transparent.

`var bezelStyle: NSButton.BezelStyle`

The appearance of the button's border.

`var bezelColor: NSColor?`

The color of the button's bezel, in appearances that support it.

`var showsBorderOnlyWhileMouseInside: Bool`

A Boolean value that determines whether the button displays its border only when the pointer is over it.

`var imageHugsTitle: Bool`

A Boolean value that determines how the button's image and title are positioned together within the button bezel.

`var imageScaling: NSImageScaling`

The scaling mode applied to make the cell's image fit the frame of the image view.

Managing button compression

`var activeCompressionOptions: NSUserInterfaceCompressionOptions`

The compression options active for this button.

`func compress(withPrioritizedCompressionOptions: [NSUserInterface CompressionOptions])`

Sets the priority compression options for this button.

`func minimumSize(withPrioritizedCompressionOptions: [NSUserInterface CompressionOptions]) -> NSSize`

Returns the minimum size of the button by using the compression options.

Managing button state

`var allowsMixedState: Bool`

A Boolean value that indicates whether the button allows a mixed state.

`var state: NSControl.StateValue`

The button's state.

`func setNextState()`

Sets the button to its next state.

`func highlight(Bool)`

Highlights (or unhighlights) the button.

Accessing key equivalents

`var keyEquivalent: String`

The key-equivalent character of the button.

`var keyEquivalentModifierMask: NSEvent.ModifierFlags`

The mask specifying the modifier keys for the button's key equivalent.

Handling keyboard events

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

Checks the button's key equivalent against the specified event and, if they match, simulates the button being clicked.

Relationships

Inherits From

NSControl

Inherited By

NSPopUpButton, NSStatusBarButton

Conforms To

CVarArg
CustomDebugStringConvertible
CustomStringConvertible
Equatable
Hashable
NSAccessibilityButton
NSAccessibilityElementProtocol
NSAccessibilityProtocol
NSAnimatablePropertyContainer
NSAppearanceCustomization
NSCoding
NSDraggingDestination
NSObjectProtocol
NSStandardKeyBindingResponding
NSTouchBarProvider
NSUserActivityRestoring
NSUserInterfaceCompression
NSUserInterfaceItemIdentification
NSUserInterfaceValidations

`Sendable`
`SendableMetatype`

See Also

Controls

Responding to control-based events using target-action

Handle user input by connecting buttons, sliders, and other controls to your app's code using the target-action design pattern.

`class` `NSColorWell`

A control that displays a color value and lets the user change that color value.

Combo Box

Display a list of values in a pop-up menu that lets the user select a value or type in a custom value.

`class` `NSComboButton`

A button with a pull-down menu and a default action.

Date Picker

Display a calendar date and provide controls for editing the date value.

`class` `NSImageView`

A display of image data in a frame.

`class` `NSLevelIndicator`

A visual representation of a level or quantity, using discrete values.

Path Control

A display of a file system path or virtual path information.

`class` `NSPopUpButton`

A control for selecting an item from a list.

`class` `NSProgressIndicator`

An interface that provides visual feedback to the user about the status of an ongoing task.

`class` `NSRuleEditor`

An interface for configuring a rule-based list of options.

`class` `NSPredicateEditor`

A defined set of rules that allows the editing of predicate objects.

Search Field

Provide a text field that is optimized for text-based search interfaces.

`class` `NSSegmentedControl`

Display one or more buttons in a single horizontal group.

Slider

Display a range of values from which the user selects a single value.