Class

NSSegmentedControl

Display one or more buttons in a single horizontal group.

macOS

```
@MainActor
class NSSegmentedControl
```

Overview

The `NSSegmentedControl` class uses an `NSSegmentedCell` class to implement much of the control's functionality. Most methods in `NSSegmentedControl` are simply cover methods that call the corresponding method in `NSSegmentedCell`. The methods of `NSSegmentedCell` that do not have covers relate to accessing and setting values for tags and tooltips, programatically setting the key segment, and establishing the mode of the control.

The features of a segmented control include the following:

- A segment can have an image, text (label), menu, tooltip, and tag.
- A segmented control can contain images or text, but not both.
- Either the control or individual segments can be enabled or disabled.
- Segmented controls have four tracking modes, described in `NSSegmentedControl.SwitchTracking`. You use these modes with the `trackingMode` property.
- Each segment can be either a fixed width or autosized to fit the contents.
- If a segment has text and is marked as autosizing, then the text may be truncated so that the control completely fits.
- If an image is too large to fit in a segment, it is clipped.

- If Full Keyboard Access is enabled in System Preferences > Keyboard, the keyboard may be used to move between and select segments.

Topics

Creating a segmented control

`convenience init(images: [NSImage], trackingMode: NSSegmentedControl.SwitchTracking, target: Any?, action: Selector?)`

`convenience init(labels: [String], trackingMode: NSSegmentedControl.SwitchTracking, target: Any?, action: Selector?)`

Configuring the cell

`class NSSegmentedCell`

An `NSSegmentedCell` object implements the appearance and behavior of a horizontal button divided into multiple segments. This class is used in conjunction with the `NSSegmentedControl` class to implement a segmented control.

Specifying the segment behavior

`var trackingMode: NSSegmentedControl.SwitchTracking`

The type of tracking behavior the control exhibits.

`enum SwitchTracking`

The following constants specify the type of tracking behavior a segmented control exhibits. They are used by `trackingMode`.

`var segmentStyle: NSSegmentedControl.Style`

The visual style used to display the control.

`enum Style`

The following constants specify the visual style used to display the segmented control. They are used by `segmentStyle`.

Specifying number of segments

`var segmentCount: Int`

The number of segments in the control.

Configuring the segment text

```
func label(forSegment: Int) -> String?
```

Returns the label of the specified segment.

```
func setLabel(String, forSegment: Int)
```

Sets the label for the specified segment.

```
func setAlignment(NSTextAlignment, forSegment: Int)
```

```
func alignment(forSegment: Int) -> NSTextAlignment
```

Configuring a segment image

```
func setImage(NSImage?, forSegment: Int)
```

Sets the image for the specified segment.

```
func image(forSegment: Int) -> NSImage?
```

Returns the image associated with the specified segment.

```
func setImageScaling(NSImageScaling, forSegment: Int)
```

Sets the scaling mode used to display the specified segment’s image.

```
func imageScaling(forSegment: Int) -> NSImageScaling
```

Returns the scaling mode used to display the specified segment’s image.

Configuring a segment menu

```
func setMenu(NSMenu?, forSegment: Int)
```

Sets the menu for the specified segment.

```
func menu(forSegment: Int) -> NSMenu?
```

Returns the menu for the specified segment.

```
func setShowsMenuIndicator(Bool, forSegment: Int)
```

```
func showsMenuIndicator(forSegment: Int) -> Bool
```

```
var isSpringLoaded: Bool
```

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

Managing the selected segment

`var selectedSegment: Int`

The index of the selected segment of the control, or −1 if no segment is selected.

`var indexOfSelectedItem: Int`

`func selectSegment(withTag: Int) -> Bool`

Selects the segment with the specified tag.

`func setSelected(Bool, forSegment: Int)`

Sets the selection state of the specified segment.

`func isSelected(forSegment: Int) -> Bool`

Returns a Boolean value indicating whether the specified segment is selected.

`var selectedSegmentBezelColor: NSColor?`

The color of the selected segment’s bezel, in appearances that support it.

`var doubleValueForSelectedSegment: Double`

When the tracking mode for the control is set to use a momentary accelerator, returns a value for the selected segment.

Adjusting the segment spacing

`func setWidth(CGFloat, forSegment: Int)`

Sets the width of the specified segment.

`func width(forSegment: Int) -> CGFloat`

Returns the width of the specified segment.

`var segmentDistribution: NSSegmentedControl.Distribution`

`enum Distribution`

`var activeCompressionOptions: NSUserInterfaceCompressionOptions`

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

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

Specifying the border shape

```
var borderShape: NSControl.BorderShape
```

```
enum BorderShape
```

Enabling and disabling segments

```
func setEnabled(Bool, forSegment: Int)
```

Sets the enabled state of the specified segment

```
func isEnabled(forSegment: Int) -> Bool
```

Returns a Boolean value indicating whether the specified segment is enabled.

Managing tags and tooltips

```
func tag(forSegment: Int) -> Int
```

```
func setTag(Int, forSegment: Int)
```

```
func setToolTip(String?, forSegment: Int)
```

```
func toolTip(forSegment: Int) -> String?
```

Relationships

Inherits From

`NSControl`

Conforms To

```
CVarArg
CustomDebugStringConvertible
CustomStringConvertible
Equatable
Hashable
NSAccessibilityElementProtocol
NSAccessibilityProtocol
```

`NSAnimatablePropertyContainer`
`NSAppearanceCustomization`
`NSCoding`
`NSDraggingDestination`
`NSObjectProtocol`
`NSStandardKeyBindingResponding`
`NSTouchBarProvider`
`NSUserActivityRestoring`
`NSUserInterfaceCompression`
`NSUserInterfaceItemIdentification`
`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 NSButton`

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

`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.

Slider

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