Class

# NSTouchBarItem

A UI control shown in the Touch Bar on supported models of MacBook Pro.

iOS 13.0+ | iPadOS 13.0+ | Mac Catalyst 13.1+ | macOS 10.12.2+

```
@MainActor
class NSTouchBarItem
```

## Overview

An instance of the `NSTouchBarItem` class is called an *item*. It appears to the user on the Touch Bar, typically along with other items, within the (invisible) bounds of the view for an `NSTouchBar` object, called a *bar*.

You use an item by adding it or its identifier to one or another of a bar's arrays, depending on your app's architecture and on the user customization you want to support. Because of the close interaction between bars and items, be sure you have read the overview for the `NSTouchBar` class before continuing here to learn about items.

AppKit provides a rich set of subclasses of `NSTouchBarItem`, each of which is described in the corresponding class reference document:

- An `NSCandidateListTouchBarItem` object (a *candidate-list item*), along with its delegate, provides a list of textual suggestions for the current text view
- An `NSColorPickerTouchBarItem` object (a *color picker item*) provides a system-defined color picker
- An `NSCustomTouchBarItem` object (a *custom item*) contains a responder of your choice, such as a view, a button, or a scrubber (an instance of the `NSScrubber` class)
- An `NSGroupTouchBarItem` object (a *group item*) provides a bar to contain other items

- An `NSPopoverTouchBarItem` object (a *popover item*) provides a two-state control that, when touched or pressed, expands into its second state, showing the contents of a bar it owns
- An `NSSharingServicePickerTouchBarItem` object (a *sharing service picker item*), along with its delegate, provides a list of objects eligible for sharing
- An `NSSliderTouchBarItem` object (a *slider item*) provides a slider control for choosing a value in a range

The two most commonly-used item classes are `NSCustomTouchBarItem` and `NSPopoverTouchBarItem`.

Refer to the following sample code projects which demonstrate how to use `NSTouchBarItem` and related classes:

- Creating and Customizing the Touch Bar
- Integrating a Toolbar and Touch Bar into Your App

## Custom items

You typically use a *custom item* (an instance of the `NSCustomTouchBarItem` class) to hold a view. For example, to place a button in the Touch Bar, proceed as follows:

1. Use an `NSButton` convenience initializer such as `init(title:image:target:action:)` to create and configure the button.
2. Set the `view` property for a custom item to point to the new button.

> **Note**
>
> When you create custom items, it's important to use convenience initializers, available starting in macOS 10.12, for the `NSButton`, `NSSegmentedControl`, and `NSSlider` classes. These initializers take care of sizing their controls correctly for the Touch Bar, and they configure appearance appropriately for the Touch Bar. If you don't use the convenience initializers, it's your app's responsibility to ensure correct sizing and appearance.

## Popover items

A *popover item* (an instance of the `NSPopoverTouchBarItem` class) — the second commonly-used type — lets you provide a new bar (an `NSTouchBar` object) when a user taps, or presses-and-holds, on the collapsed representation of the popover item.

In its expanded state, a popover appears as an overlay above other items in the Touch Bar.

To show a bar when a user taps a popover item, specify a bar in the item's `popoverTouchBar` property. Enable press-and-hold by specifying a bar in the `pressAndHoldTouchBar` property. The press-and-hold feature is suitable only for a simple popover, such as one that contains a single segmented control (an instance of the `NSSegmentedControl` class) or slider (an instance of the `NSSliderTouchBarItem` class).

> Note
>
> If your popover bar requires significant user interaction and contains many items or many scroll views, don't enable press-and-hold; doing so can result in an awkward user experience.

The system automatically shows a chevron in the popover item under the following conditions: You specify the same `NSTouchBar` object for both `pressAndHoldTouchBar` and `popoverTouchBar` properties, *and* you use the default view for the popover item's `collapsedRepresentation` property.

If you provide a popover item that contains a scrubber (an `NSScrubber` instance), you'll likely want to dismiss both the scrubber and the popover after the user makes their selection in the scrubber. A good approach to achieve this user interaction is to subclass `NSPopoverTouchBarItem`, employing your instance of the subclass as the scrubber's delegate. You can then configure the delegate object, within its `didFinishInteracting(with:)` method, to call the popover's `dismissPopover(_:)` method.

If you place a segmented control in a bar for a popover item, take care *not* to use `NSSegmentedControl.SwitchTracking.momentary` option of the `NSSegmentedControl.SwitchTracking` enumeration because doing so interferes with the user's operation of the control.

## Other common item types

To provide a *slider item*, always use the `NSSliderTouchBarItem` class, which employs a standard slider but is optimized for user interaction with the Touch Bar. (That is, don't instead add an `NSSlider` object directly to a custom item.)

A *group item* (an instance of the `NSGroupTouchBarItem` class) is a container that provides a bar, in its `groupTouchBar` property, with its own array of items. You can enable customization for the items in a group's contained bar, in the same way you would for items directly within a top-level bar. Using a group item lets you provide different user customization rules for different parts of the Touch Bar. Using a group item also lets you enable centering of the group within the Touch Bar.

A *spacing item* lets you add custom spacing between items in a bar. Specify a spacing item for a bar by assigning the `fixedSpaceSmall`, `fixedSpaceLarge`, or `flexibleSpace` identifier to an item, and adding that item to the bar’s items array. The system automatically instantiates and configures spacing items based on the identifiers you specify.

## Configuration

You must configure each item with a unique identifier, and can optionally assign a visibility priority or tag it as a principal item.

**NSTouchBarItem identification.** You must provide a unique identifier for each item in the bar, apart from spacing items. Specify an identifier, of type `NSTouchBarItem.Identifier` (called an *item identifier*), for each item when you initialize it. The item identifier serves as a persistable weak reference to the item. The system uses item identifiers to populate bars and to track and record changes for user customization.

**NSTouchBarItem priority for visibility.** If the system is showing a bar in the Touch Bar, but horizontal space is constrained and the bar defines more items than will fit, the system hides some of the items. You influence this hide/show behavior by setting a value for the `visibilityPriority` property of each item.

Lower-visibility-priority items get hidden by the system, as needed, before higher-visibility-priority items do.

To set visibility priority, use the constants in the `NSTouchBarItem.Priority` enumeration, or assign an integer value. The value `0` indicates `normal` visibility priority. Visibility priority increases with increasing numerical value. The `low` constant provides a value of `−1000`; the `high` constant, a value `+1000`. You can use integers outside of this range if you need to.

The system hides or shows groups of identical-priority items (defined within a single bar) together. The one exception to this rule is for items whose visibility priority is `normal`; these items get hidden one-by-one, with the normal-priority item farthest to the right getting hidden first. If horizontal space later increases in the Touch Bar, and hidden, normal-priority items become eligible for display, the system first shows the most recently-hidden of those items.

**Principal Items.** Within a bar, you can optionally specify an item as having special significance by employing the `principalItemIdentifier` property. The system attempts to center a principal item within the Touch Bar. If you want a group of items to appear centered in the Touch Bar, designate the group item (of type `NSTouchBarItem`) as the principal item.

If more than one bar in the responder chain is eligible to be visible in the Touch Bar, and more than one of those has a principal item, the system determines which one to center in the Touch Bar.

## Fonts, images, and colors

When using a button in a custom item, don’t attempt to set the button title’s font. In the Touch Bar, the system specifies fonts for standard controls.

If you need to specify a font, such as for custom drawing, use the `systemFont(ofSize:)` class method (or related methods) of the `NSFont` class. Use a font size of `0` to automatically obtain appropriate sizing for the Touch Bar.

If you use an image in a button or other control in the Touch Bar, take care to employ a template image. Template images in the Touch Bar respond automatically to system white-point changes, and automatically react to user interactions. The overview in this document lists the built-in Touch Bar template images.

To use your own image assets, use Retina-resolution images, designated as `@2x` in your asset catalog and with a maximum height of 30 points (corresponding to 60 pixels).

To set colors on objects within an `NSTouchBarItem` object, use AppKit named colors and use a bezel color property (available starting in macOS 10.12.1). Named colors appear correctly in the Touch Bar, support appearance vibrancy, and respond to system white-point changes. In a button or a segmented control, employ the bezel color property to ensure appropriate appearance in the Touch Bar.

To set the background color on a button within a custom item, use code like this:

```
myButton.bezelColor = NSColor.controlColor
```

To set color on text and glyphs in the Touch Bar, use the following colors from the `NSColor` class:

- `labelColor`
- `secondaryLabelColor`
- `tertiaryLabelColor`
- `quaternaryLabelColor`

The system automatically changes the relative brightness and the white-point of these colors, depending on the ambient light, and depending on other factors such as keyboard backlight level. Always use these colors, or colors that dynamically derive from these colors, for control backgrounds, text, icons, and glyphs in the Touch Bar.

## Handling touch events

The easiest way to handle touch events in an item is to use AppKit controls, such as by adding a button, a segmented control, or a scrubber to the item. Standard AppKit controls convey touch events to your specified targets automatically, so use standard controls whenever possible in your app.

If standard controls are insufficient, you can create composite views with a combination of standard controls, custom views, and gesture recognizers that you manually add to those custom views.

If you require the lowest-level of control for touch event processing, you can use the `NSTouch` class directly. You might go this route, for example, to provide good user feedback in the case of a control placed within a scroll view.Direct use of touch methods allows fine-grained control over interaction. You can, for example, highlight a control immediately upon a user touching it, and then remove the highlight if the user then, without lifting the finger, performs a scroll gesture.

If using the `NSTouch` class directly, be sure to implement the `touchesCancelled(with:)` responder method, because users can perform touch interactions that result in canceled touches.

## Topics

### Creating a bar item

`init(identifier: NSTouchBarItem.Identifier)`

Creates a new item with the specified identifier.

`struct Identifier`

An identifier for an item in the Touch Bar.

`init?(coder: NSCoder)`

Initializes and returns a new item from a storyboard or nib file.

### Identifying a bar item

`var identifier: NSTouchBarItem.Identifier`

The identifier for this item.

`struct Identifier`

An identifier for an item in the Touch Bar.

### Managing item visibility

`var visibilityPriority: NSTouchBarItem.Priority`

Determines which items are shown in a bar when space is limited.

`struct Priority`

Priorities for the visibility of a Touch Bar item.

`var isVisible: Bool`

A Boolean value that reflects whether or not the item is visible.

## Configuring bar customization

`var customizationLabel: String`

The user-visible string identifying this item during bar customization.

## Subclassing bar items

`var viewController: NSViewController?`

The view controller associated with this item.

`var view: NSView?`

The view associated with this item.

## Using template images

`class let touchBarAddDetailTemplateName: String`

A template image for showing additional detail for an item.

`class let touchBarAddTemplateName: String`

A template image for creating a new item.

`class let touchBarAlarmTemplateName: String`

A template image for setting or showing an alarm.

`class let touchBarAudioInputMuteTemplateName: String`

A template image for muting audio input or denoting that audio input is muted.

`class let touchBarAudioInputTemplateName: String`

A template image for denoting audio input.

`class let touchBarAudioOutputMuteTemplateName: String`

A template image for muting audio output or for denoting that audio output is muted.

`class let touchBarAudioOutputVolumeHighTemplateName: String`

A template image for setting the audio output volume to a high level, or for denoting that the audio is at the peak volume.

`class let touchBarAudioOutputVolumeLowTemplateName: String`

A template image for setting the audio output volume to a low level, or for denoting that it is set to a low level.

`class let touchBarAudioOutputVolumeMediumTemplateName: String`

A template image for setting the audio output volume to a medium level, or for denoting that it is set to a medium level.

`class let touchBarAudioOutputVolumeOffTemplateName: String`

A template image for setting the audio output volume to silent, or for denoting that it is set to silent.

`class let touchBarBookmarksTemplateName: String`

A template image for showing app-specific bookmarks.

`class let touchBarColorPickerFillName: String`

A template image for showing a color picker so the user can select a fill color.

`class let touchBarColorPickerFontName: String`

A template image for showing a color picker so the user can select a text color.

`class let touchBarColorPickerStrokeName: String`

A template image for showing a color picker so the user can select a stroke color.

`class let touchBarCommunicationAudioTemplateName: String`

A template image for initiating or denoting audio communication.

`class let touchBarCommunicationVideoTemplateName: String`

A template image for initiating or denoting video communication.

`class let touchBarComposeTemplateName: String`

A template image for opening a new document or view in edit mode.

`class let touchBarDeleteTemplateName: String`

A template image for deleting the current or selected item.

`class let touchBarDownloadTemplateName: String`

A template image for downloading an item.

`class let touchBarEnterFullScreenTemplateName: String`

A template image for entering full screen mode.

`class let touchBarExitFullScreenTemplateName: String`

A template image for exiting full screen mode.

`class let touchBarFastForwardTemplateName: String`

A template image for moving forward through media playback or slides.

`class let touchBarFolderTemplateName: String`

A template image for opening or representing a folder.

`class let touchBarFolderCopyToTemplateName: String`

A template image for copying an item to a destination.

`class let touchBarFolderMoveToTemplateName: String`

A template image for moving an item to a destination.

`class let touchBarGetInfoTemplateName: String`

A template image for showing information about an item.

`class let touchBarGoBackTemplateName: String`

A template image for returning to the previous screen or location.

`class let touchBarGoDownTemplateName: String`

A template image for moving to the next item in a list.

`class let touchBarGoForwardTemplateName: String`

A template image for moving to the next screen or location.

`class let touchBarGoUpTemplateName: String`

A template image for moving to the previous item in a list.

`class let touchBarHistoryTemplateName: String`

A template image for showing history information, such as recent downloads.

`class let touchBarIconViewTemplateName: String`

A template image for showing items in an icon view.

`class let touchBarListViewTemplateName: String`

A template image for showing items in a list view.

`class let touchBarMailTemplateName: String`

A template image for creating an email message.

`class let touchBarNewFolderTemplateName: String`

A template image for creating a new folder.

`class let touchBarNewMessageTemplateName: String`

A template image for creating a new message, or for denoting the use of messaging.

`class let touchBarOpenInBrowserTemplateName: String`

A template image for opening an item in the user’s browser.

`class let touchBarPauseTemplateName: String`

A template image for pausing media playback or slides.

`class let touchBarPlayTemplateName: String`

A template image for starting or resuming playback of media or slides.

`class let touchBarPlayPauseTemplateName: String`

A template image for toggling between playing and pausing media or slides.

`class let touchBarPlayheadTemplateName: String`

A template image for denoting the current playback position within a timeline track.

`class let touchBarQuickLookTemplateName: String`

A template image for opening an item in Quick Look.

`class let touchBarRecordStartTemplateName: String`

A template image for starting recording.

`class let touchBarRecordStopTemplateName: String`

A template image for stopping recording or stopping playback of media or slides.

`class let touchBarRefreshTemplateName: String`

A template image for refreshing displayed data.

`class let touchBarRewindTemplateName: String`

A template image for moving backwards through media or slides.

`class let touchBarRotateLeftTemplateName: String`

A template image for rotating an item counterclockwise.

`class let touchBarRotateRightTemplateName: String`

A template image for rotating an item clockwise.

`class let touchBarSearchTemplateName: String`

A template image for showing a search field or for initiating a search.

```
class let touchBarShareTemplateName: String
```
A template image for sharing content with others directly or via social media.

```
class let touchBarSidebarTemplateName: String
```
A template image for showing a sidebar in the current view.

```
class let touchBarSkipBackTemplateName: String
```
A template image for skipping to the previous chapter or location during media playback.

```
class let touchBarSkipToStartTemplateName: String
```
A template image for skipping to the start of media playback.

```
class let touchBarSkipBack30SecondsTemplateName: String
```
A template image for skipping back 30 seconds during media playback.

```
class let touchBarSkipBack15SecondsTemplateName: String
```
A template image for skipping back 15 seconds during media playback.

```
class let touchBarSkipAhead15SecondsTemplateName: String
```
A template image for skipping ahead 15 seconds during media playback.

```
class let touchBarSkipAhead30SecondsTemplateName: String
```
A template image for skipping ahead 30 seconds during media playback.

```
class let touchBarSkipToEndTemplateName: String
```
A template image for skipping to the end of media playback.

```
class let touchBarSkipAheadTemplateName: String
```
A template image for skipping to the next chapter or location during media playback.

```
class let touchBarSlideshowTemplateName: String
```
A template image for starting a slideshow.

```
class let touchBarTagIconTemplateName: String
```
A template image for applying a tag to an item.

```
class let touchBarTextBoxTemplateName: String
```
A template image for inserting a text box.

```
class let touchBarTextListTemplateName: String
```
A template image for inserting a list or converting text to list form.

```
class let touchBarTextBoldTemplateName: String
```

A template image for making selected text bold.

```
class let touchBarTextItalicTemplateName: String
```

A template image for italicizing the selected text.

```
class let touchBarTextUnderlineTemplateName: String
```

A template image for underlining text.

```
class let touchBarTextStrikethroughTemplateName: String
```

A template image for striking through text.

```
class let touchBarTextJustifiedAlignTemplateName: String
```

A template image for fully justifying text.

```
class let touchBarTextLeftAlignTemplateName: String
```

A template image for aligning text to the left.

```
class let touchBarTextCenterAlignTemplateName: String
```

A template image for centering text.

```
class let touchBarTextRightAlignTemplateName: String
```

A template image for aligning text to the right.

```
class let touchBarUserTemplateName: String
```

A template image for showing or representing user information.

```
class let touchBarUserAddTemplateName: String
```

A template image for creating a new user account.

```
class let touchBarUserGroupTemplateName: String
```

A template image for showing or representing a group of users.

```
class let touchBarVolumeUpTemplateName: String
```

A template image for increasing the audio output volume.

```
class let touchBarVolumeDownTemplateName: String
```

A template image for reducing the audio output volume.

## Relationships

### Inherits From

`NSObject`

### Inherited By

`NSButtonTouchBarItem`
`NSCandidateListTouchBarItem`
`NSColorPickerTouchBarItem`
`NSCustomTouchBarItem`
`NSGroupTouchBarItem`
`NSPickerTouchBarItem`
`NSPopoverTouchBarItem`
`NSSharingServicePickerTouchBarItem`
`NSSliderTouchBarItem`
`NSStepperTouchBarItem`

### Conforms To

`CVarArg`
`Copyable`
`CustomDebugStringConvertible`
`CustomStringConvertible`
`Equatable`
`Hashable`
`NSCoding`
`NSObjectProtocol`
`Sendable`

---

## See Also

### Touch Bar items

`class NSCandidateListTouchBarItem`

A bar item that, along with its delegate, provides a list of textual suggestions for the current text view.

`class NSColorPickerTouchBarItem`

A bar item that provides a system-defined color picker.

`class NSCustomTouchBarItem`

A bar item that contains a responder of your choice, such as a view, a button, or a scrubber.

`class NSGroupTouchBarItem`

A bar item that provides a bar to contain other items.

`class NSPopoverTouchBarItem`

A bar item that provides a two-state control that can expand into its second state, showing the contents of a bar that it owns.

`class NSSharingServicePickerTouchBarItem`

A bar item that, along with its delegate, provides a list of objects eligible for sharing.

`class NSSliderTouchBarItem`

A bar item that provides a slider control for choosing a value in a range.

`class NSStepperTouchBarItem`

A bar item that provides a stepper control for incrementing or decrementing a value.

`class NSUserInterfaceCompressionOptions`

An object that specifies how user interface elements resize themselves when space is constrained.

`class NSButtonTouchBarItem`

A bar item that provides a button.

`class NSPickerTouchBarItem`

A bar item that provides a picker control with multiple options.

`enum ControlRepresentation`

Constants that specify display styles for picker bar items.

`enum SelectionMode`

Constants that specify selection modes for picker bar items.