Sample Code

Creating and Customizing the Touch Bar

Adopt Touch Bar support by displaying interactive content and controls for your macOS apps.

Download

macOS 11.0+ | Xcode 13.0+

Overview

You can use the Touch Bar to display interactive content and controls for supported models of MacBook Pro.

This sample uses a split-view controller architecture to navigate various ways to implement the `NSTouchBar` class. The `PrimaryViewController` class shows the available use cases. When you select one of the use cases, the secondary view shows what you can do with the Touch Bar controls.

The description in each secondary view controller reflects its target item. For example, the `ButtonViewController` class shows how to use an `NSButton` object. The sample hosts all view controllers in separate storyboards.

Refer to this sample if you're looking for specific ways to use and customize the `NSTouchBar`. It shows how to create an `NSTouchBarItem` for use with buttons, views, pickers, sliders, popovers, scrubbers, text fields, and text views.

The sample also displays a secondary window that shows a typical use case with the Touch Bar that uses an `NSScrubber` to pick an image that changes the window background.

Configure the Sample Code Project

In Xcode, select your development team on the macOS target's Signing and Capabilities tab.

Simulate the Touch Bar in Xcode

When adopting Touch Bar support for your app, but running Xcode on a Mac without a Touch Bar, enable the Xcode Touch Bar simulator by choosing Window > Touch Bar -> Show Touch Bar.

Create the Touch Bar

Create a Touch Bar by adding one to a view controller in the storyboard, or by creating one programmatically by overriding the `NSResponder` `makeTouchBar()` function. The following example demonstrates overriding `makeTouchBar()` for an `NSViewController` that displays a color-picker button or `NSColorPickerTouchBarItem`:

```
override func makeTouchBar() -> NSTouchBar? {
    let touchBar = NSTouchBar()
    touchBar.delegate = self

    touchBar.customizationIdentifier = colorPickerBar
    touchBar.defaultItemIdentifiers = [selectedItemIdentifier]
    touchBar.customizationAllowedItemIdentifiers = [selectedItemIdentifier]
    touchBar.principalItemIdentifier = selectedItemIdentifier

    return touchBar
}
```

Add Content to the Touch Bar

Add content to the Touch Bar by adopting the `NSTouchBarDelegate` protocol and implementing `touchBar(_:makeItemForIdentifier:)`. The following example adds a Sharing Service button:

```
func touchBar(_ touchBar: NSTouchBar, makeItemForIdentifier identifier: NSTouchBarIt
    guard identifier == services else { return nil }

    let services = NSSharingServicePickerTouchBarItem(identifier: identifier)
    services.delegate = self

    return services
}
```

Add a Scrubber to the Touch Bar

The `NSScrubber` class provides a way to add a flexible, horizontally oriented picker. Add a scrubber view to an `NSCustomTouchBarItem`, and that view adds a scrubber to the Touch Bar.

```
func touchBar(_ touchBar: NSTouchBar, makeItemForIdentifier identifier: NSTouchBarIt
    let scrubberItem: NSCustomTouchBarItem

    scrubberItem = NSCustomTouchBarItem(identifier: identifier)
    scrubberItem.customizationLabel = NSLocalizedString("Choose Photo", comment: "")

    let scrubber = NSScrubber()
    scrubber.register(NSScrubberImageItemView.self, forItemIdentifier: itemViewIdent
    scrubber.mode = .free
    scrubber.selectionBackgroundStyle = .roundedBackground
    scrubber.delegate = self
    scrubber.dataSource = self
    scrubber.showsAdditionalContentIndicators = true
    scrubber.scrubberLayout = NSScrubberFlowLayout()

    scrubberItem.view = scrubber

    // Set the scrubber's width to be 400.
    let viewBindings: [String: NSView] = ["scrubber": scrubber]
    let hconstraints =
        NSLayoutConstraint.constraints(withVisualFormat: "H:[scrubber(400)]",
                                       options: [],
                                       metrics: nil,
                                       views: viewBindings)
    NSLayoutConstraint.activate(hconstraints)

    return scrubberItem
}
```

Add a Popover to the Touch Bar

An `NSCustomTouchBarItem` class shows its own Touch Bar when the user taps and holds on it. This class is a two-state control that can expand into its second state, which shows the contents of a bar it owns. The second Touch Bar displays when this item *pops*. The following example shows how to create an `NSPopoverTouchBarItem` with an `NSScrubber` view with a set of numbered text items:

```swift
func touchBar(_ touchBar: NSTouchBar, makeItemForIdentifier identifier: NSTouchBarIt
    guard identifier == scrubberPopover else { return nil }

    let popoverItem = NSPopoverTouchBarItem(identifier: identifier)
    popoverItem.collapsedRepresentationLabel =
        NSLocalizedString("Scrubber Popover", comment: "")
    popoverItem.customizationLabel =
        NSLocalizedString("Scrubber Popover", comment: "")

    let scrubber = PopoverScrubber()
    scrubber.register(NSScrubberTextItemView.self, forItemIdentifier: textScrubber)

    scrubber.mode = .free
    scrubber.selectionBackgroundStyle = .roundedBackground
    scrubber.delegate = self
    scrubber.dataSource = self
    scrubber.presentingItem = popoverItem

    popoverItem.collapsedRepresentation = scrubber

    popoverItem.popoverTouchBar =
        PopoverTouchBarSample(presentingItem: popoverItem)

    return popoverItem
}
```

See Also

Essentials

{} Integrating a Toolbar and Touch Bar into Your App

Provide users quick access to your app’s features from a toolbar and corresponding Touch Bar.

`class NSTouchBar`

An object that provides dynamic contextual controls in the Touch Bar of supported models of MacBook Pro.

`protocol NSTouchBarDelegate`

A protocol that allows you to provide the items for a bar dynamically.

```
protocol NSTouchBarProvider
```

A protocol that an object adopts to create a bar object in your app.