Class

NSScrubber

A customizable item picker control for the Touch Bar.

macOS 10.12.2+

```
@MainActor
class NSScrubber
```

Overview

On supported MacBook Pro models, you can use a scrubber (an instance of the `NSScrubber` class) to provide a horizontally-oriented, item-picker control in the Touch Bar. Use a scrubber to let the user pick an item from a related collection, such as a photo from a library or a date from a date range.

Refer to the following sample code projects which demonstrate how to use `NSTouchBar` and related classes, including the `NSScrubber` class:

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

Each item that appears in a scrubber is a specialized view that supports selection and scrubber-appropriate decorations. The scrubber keeps track of its items by their index positions.

> **Note**
>
> Take care to understand the Touch Bar term *items*. An item for a scrubber *is* a view — an `NSScrubberItemView` instance — at a specific index position in the scrubber. This is analogous to a row in a table. An item for a bar (an instance of the `NSTouchBar` class), by contrast, is an `NSTouchBarItem` instance, which *has* a view.

There are many classes in the scrubber API, as well as a delegate protocol, a data source protocol, and a callback-based layout API. The design pattern is reminiscent of that used for a collection view (an instance of the `NSCollectionView` class). You might find it helpful to refer to the `NSCollectionView` overview for background. Be aware, though of the differences. For example, while scrubbers and collection views both employ a `makeItem(withIdentifier:owner:)` method, and both employ a reuse queue, a scrubber is subclassed from the `NSView` class while a collection view is subclassed from the `NSViewController` class.

A scrubber employs:

- The *scrubber object* itself (an instance of the `NSScrubber` class), which serves as a container view that shows a subview for each scrubber item, and which employs a reuse-queue pattern for efficiency and performance.
- A *data source* (conforming to the `NSScrubberDataSource` protocol), which provides scrubber items to the scrubber, on demand, from an associated data collection in your app. Specify the data source in the scrubber's `dataSource` property
- A *delegate* (conforming to the `NSScrubberDelegate` protocol), which responds to user interaction — such as with its `didBeginInteracting(with:)` and `didCancelInteracting(with:)` methods. Specify the delegate in the scrubber's `delegate` property. You can also use the delegate to respond to the highlighting and selection of scrubber items, and to respond to changes in which items are visible in the scrubber.
- A *layout* (an instance of a subclass of the `NSScrubberLayout` abstract class, typically the `NSScrubberFlowLayout` concrete subclass). You implement a layout to respond to calls, from the system, to return view specifications for the items to be displayed in the scrubber. The layout, in this way, assists in arranging and decorating the scrubber's contained items, and in providing appearance changes in response to user interaction. Specify the layout in the scrubber's `scrubberLayout` property.

Before learning how to use a scrubber in the Touch Bar, be sure you read the overview for the `NSTouchBar` class.

Scrubber data source and delegate

A scrubber employs a data source and a delegate, using a pattern similar to that used for collection views, as follows:

Data source. To supply items for a scrubber, implement an object that conforms to the `NSScrubberDataSource` protocol and specify that object in the scrubber's `dataSource` property. There are two built-in item types, provided by the `NSScrubberTextItemView` and `NSScrubberImageItemView` concrete classes. For more on scrubber items, see Scrubber items.

The following code shows an example implementation of the `numberOfItems` datasource method, returning the count of items displayed by the scrubber.

Swift Objective-C

```
func numberOfItems(for scrubber: NSScrubber) -> Int {
   return self.scrubberItems.count;
}
```

In addition to the count of scrubber items, you use the datasource method to provide individual items with the `scrubber(_:viewForItemAt:)` method. An example implementation is shown in the following code.

Swift Objective-C

```
func scrubber(_ scrubber: NSScrubber, viewForItemAt index: Int) -> NSScrubberItemVie
    itemView.title = String(index)
    return itemView
}
```

To optimize resource usage and performance, a scrubber employs a reuse queue that's similar to the reuse queue for an `NSCollectionView` object.

Delegate. To respond to user interactions and to visibility, highlighting, and selection changes, implement a delegate object that conforms to the `NSScrubberDelegate` protocol and specify that object in the scrubber's `delegate` property.

The following code shows a minimal implementation of the `scrubber(_:didSelectItemAt:)` delegate method for a scrubber.

Swift Objective-C

```
func scrubber(_ scrubber: NSScrubber, didSelectItemAt index: Int) {
    // Log the index value for the item the user selected
    print("\(#function) at index \(index)")
}
```

Choose a scrubber touch-interaction model

A scrubber offers many built-in permutations for touch interaction. By subclassing a scrubber, you can customize touch interaction.

To specify a scrubber's touch-interaction model, set values for the following, cooperating scrubber properties: `mode`, `isContinuous`, and `itemAlignment`. Here's how to choose the right permutation of values for these properties:

Scrolling or fixed. Decide whether you want the scrubber to *scroll* to track horizontal finger movement across the scrubber, or to remain *fixed* in place as the finger moves.

- For scrolling, specify the `NSScrubber.Mode.free` value for the scrubber's `mode` property.
- For a fixed scrubber, specify the `NSScrubber.Mode.fixed` value for the `mode` property (this is the default value). In this case, if the user's finger reaches the left or right edge of the scrubber view and there are items beyond the edge, the scrubber automatically scrolls to bring those items into view.

Selection style. Decide whether you want item selection to take place only upon a deliberate selection gesture, or continuously during horizontal finger movement on the scrubber.

- For deliberate selection, specify a value of `false` for the scrubber's `isContinuous` property (this is the default value). In *free* (scrolling) mode, the user must then tap an item to highlight and select it. In *fixed* (non-scrolling) mode, ending interaction with the scrubber, by lifting the finger, selects the most-recently highlighted item. However, if there is already a highlighted item before interaction starts, and the user resumes interacting with the (fixed mode) scrubber on that item, selection changes continuously, tracking the user's finger — even though the `isContinuous` property value is `false`.
- For continuous selection, specify a value of `true` for the `isContinuous` property. Item selection behavior then depends on the `mode` and `itemAlignment` property values, as described in Position-based scrubber item selection.

Item alignment. The setting in the scrubber's `itemAlignment` property affects two things: 1) item highlighting and selection, and 2) the resting position of scrubber items after manual or automatic scrolling. Available values for this property are `NSScrubber.Alignment.leading`, `NSScrubber.Alignment.center`, `NSScrubber.Alignment.trailing`, and `NSScrubber.Alignment.none`. See the `NSScrubber.Alignment` enumeration for details on how these constants work.

Your choices for scrolling, selection, and alignment jointly impact highlighting and selection behavior. For details on highlighting and selection, see Position-based scrubber item selection. Your choice for alignment also impacts scrubber-item resting-position behavior following a scroll interaction. For details on resting position, see Scrubber item resting position.

Position-based scrubber item selection

In free mode with continuous selection style (the `mode` property value is `NSScrubber.Mode.free` and the `isContinuous` property value is YES for this configuration), the scrubber item on the alignment axis is automatically highlighted and selected. The *alignment axis* is the left edge, right edge, or center of the scrubber, as you specify by setting the value of the `itemAlignment` property using constants from the `NSScrubber.Alignment` enumeration. Specifying an alignment axis of `NSScrubber.Alignment.none` is equivalent to a value of `NSScrubber.Alignment.center` for position-based item selection.

In free mode with deliberate selection style (the `mode` property value is `NSScrubber.Mode.free` and the `isContinuous` property value is NO for this configuration), the system ignores the `itemAlignment` property value in terms of item selection.

In fixed mode (the `mode` property value is `NSScrubber.Mode.fixed` for this configuration), the system ignores the `itemAlignment` property value in terms of item selection — no matter which value you specify for the `isContinuous` property.

Scrubber item resting position

The value you provide in the `itemAlignment` property specifies the automatic scrubber item resting position that follows manual or automatic scrolling. (This value also affects item highlighting and selection, as described in Choose a scrubber touch-interaction model.) The system respects your setting for resting position irrespective of the values of the `mode` and `isContinuous` properties.

Specifically:

- `NSScrubber.Alignment.leading` — In a left-to-right language, the scrubber comes to rest, following manual or automatic scrolling, so that the left edge of the leftmost scrubber item is coincident with the left edge of the scrubber.
- `NSScrubber.Alignment.center` — The scrubber comes to rest, following manual or automatic scrolling, so that a scrubber item is perfectly centered in the scrubber.
- `NSScrubber.Alignment.trailing` — In a left-to-right language, the scrubber comes to rest, following manual or automatic scrolling, so that the right edge of the rightmost scrubber item is coincident with the right edge of the scrubber.
- `NSScrubber.Alignment.none` — Following manual or automatic scrolling, the scrubber comes to rest without attempting to align any scrubber item.

Scrubber layout

A scrubber configures the views for its items with the help of two classes, `NSScrubberLayout` and `NSScrubberLayoutAttributes`, as described in this section.

Layout implementation

A *layout* is a concrete implementation of the NSScrubberLayout abstract class. AppKit provides two concrete, preconfigured layout subclasses: NSScrubberFlowLayout and NSScrubberProportionalLayout. If you use one of these built-in layout types, there's no additional layout code to write, apart from adding your choice of built-in layout to the scrubber's scrubberLayout property. This Swift example shows this simple step for the flow layout:

```
myInformationScrubber.scrubberLayout = NSScrubberFlowLayout()
```

To create a custom layout, subclass the NSScrubberLayout class and implement its callback methods. Unlike a view delegate (such as used for a table view), which provides *views* on demand, scrubber layout callbacks provide *view specifications* on demand. Using these callbacks, you specify:

- Scrubber item geometry
- Scrubber item appearance
- Layout life cycle for state management

Specify the overall visual dimensions of a custom scrubber when you create it, using the init(frame:) or init(coder:) initializer, or by using Interface Builder.

Return the total width and height for the elements in a custom scrubber, including those not currently visible, using the scrubberContentSize property in your layout. Specify height and width in points. To use the standard height, specify a value of 30.

Specify the geometry and appearance for items in your custom scrubber, using the two required callback methods that each return instances of the NSScrubberLayoutAttributes class. The system calls one or another of these methods, as it needs to, as a user interacts with a layout's owning scrubber:

Callback method	How to use
layoutAttributesForItem(at:)	Return *one* NSScrubberLayoutAttributes instance that specifies the view attribute values for the one scrubber item at the index position requested by the system in the method call.
layoutAttributesForItems(in:)	Return the *set of* NSScrubberLayoutAttributes instances that, together, specify the per-item view attributes for the items within the visible rectangle requested by the system in the method call. The set you return must contain one layout attributes object for each item in the rectangle.

You can explicitly invalidate a layout by calling the `invalidateLayout()` method. Do this whenever your app changes a scrubber's information in a way that requires a layout update. For example, if you change the text shown in one or more items, invalidate the layout.

You can specify layout life cycle in terms of the conditions under which a layout should be automatically invalidated, such as when the user selects something different in the layout's owning scrubber. The API for automatic invalidation consists of the following two properties and one method:

- `shouldInvalidateLayoutForSelectionChange`
- `shouldInvalidateLayoutForHighlightChange`
- `shouldInvalidateLayoutForChange(fromVisibleRect:toVisibleRect:)`

For example, if you design a scrubber's layout characteristics to depend on which of its items is selected by the user, return a value of `true` from the scrubber's `shouldInvalidateLayoutForSelectionChange` method.A *layout attributes* object is an instance of the `NSScrubberLayoutAttributes` class, which you configure to describe the view for a single item. The class offers the following built-in attributes for you to work with:

- `itemIndex` — The item's index position within the scrubber
- `frame` — The item's frame rectangle
- `alpha` — The item's transparency

You can specify additional item attributes by subclassing the `NSScrubberLayoutAttributes` class. For example, you could specify a geometric transform attribute.

If you're using a custom `NSScrubberLayout` subclass, provide an implementation for the `invalidateLayout()` method to clear any custom layout state, such as by discarding cached data.

Prepare for redrawing

The flip side of layout invalidation (as described in Layout implementation) is preparation for redrawing, which you perform in a layout's `prepare()` method. The goal of layout preparation is to optimize performance. A scrubber calls the `prepare()` method exactly once between invalidation and redrawing. Complete as much one-time, up-front layout work as you can, in advance of redrawing, in this step. For example, your `prepare()` implementation should perform initial layout calculations and should fill caches needed during drawing.

After the `prepare()` method returns, the system updates the scrubber view hierarchy with repeated calls to three `NSScrubberLayout` methods: `layoutAttributesForItem(at:)`, `layoutAttributesForItems(in:)`, and `scrubberContentSize`. Implement these methods to provide return values as quickly as possible, taking advantage of the work you did during layout preparation.

Scrubber items

The view that represents a scrubber item is provided by your data source object, using the `scrubber(_:viewForItemAt:)` protocol method. AppKit provides two purpose-built view classes you can use, both of which are concrete subclasses of the abstract `NSScrubberItemView` class:

- `NSScrubberImageItemView` has `image`, `imageView`, and `imageAlignment` properties
- `NSScrubberTextItemView` has `textField` and `title` properties

To create a custom item, subclass these or their abstract superclass, `NSScrubberItemView`.

Scrubbers and the responder chain

To show a scrubber, associate it with an `NSTouchBar` object (adding it, as the view for a custom item or popover item, to the bar) and then associate the bar with the appropriate responder object in your app. The system then shows the scrubber in the Touch Bar only at appropriate times. For more information on bars and the responder chain, read the overview for the `NSTouchBar` class.

Choose between a scrubber and a scroll view

When choosing between a scrubber and a scroll view, use a scrubber unless the amount of content, or the nature of your content, doesn't work well in a scrubber. Scrubber interaction is optimized for the Touch Bar, typically making a scrubber the better option for letting the user pick from among several choices, such as dates in a calendar.

Topics

Initializing a scrubber

`init(frame: NSRect)`

Initializes and returns a newly allocated scrubber object with the specified frame rectangle.

`init(coder: NSCoder)`

Initializes and returns a newly allocated scrubber object from a storyboard or nib file.

Configuring the scrubber

`var dataSource: (any NSScrubberDataSource)?`

The object that provides the data for the scrubber.

`var delegate: (any NSScrubberDelegate)?`

The object that acts as the delegate of the scrubber.

Creating scrubber items

`func register(AnyClass?, forItemIdentifier: NSUserInterfaceItemIdentifier)`

Registers a class for the scrubber to use when it creates new items.

`func register(NSNib?, forItemIdentifier: NSUserInterfaceItemIdentifier)`

Registers a nib file for the scrubber to use when it creates new items in the scrubber.

`func makeItem(withIdentifier: NSUserInterfaceItemIdentifier, owner: Any?) -> NSScrubberItemView?`

Creates or returns a reusable item object with the specified identifier.

Changing the layout

`var scrubberLayout: NSScrubberLayout`

An object used to describe the layout of items within the scrubber.

`var mode: NSScrubber.Mode`

A setting that determines whether interaction with the scrubber is fixed or free.

`enum Mode`

The scrolling behavior for a scrubber.

`var itemAlignment: NSScrubber.Alignment`

A setting that specifies the snapping behavior of items in the scrubber.

`enum Alignment`

The specified preferred alignment of items within the scrubber, when they come to rest following a user's scrolling or paging interaction.

`var isContinuous: Bool`

A Boolean value that, together with the `mode` property, determines scrubber interaction style.

Configuring the scrubber's appearance

`var backgroundColor: NSColor?`

The color displayed behind the scrubber content.

`var backgroundView: NSView?`

A view that is displayed behind the scrubber content.

`var showsAdditionalContentIndicators: Bool`

A Boolean value that specifies whether the scrubber should display the existence of additional items beyond the leading and trailing edges.

`var showsArrowButtons: Bool`

A Boolean value that specifies whether arrow buttons should be displayed at the leading and trailing edges of the scrubber.

Configuring the selection appearance

`var floatsSelectionViews: Bool`

A Boolean value that determines the behavior of the item selection decorations as the scrubber's selection changes.

`var selectionOverlayStyle: NSScrubberSelectionStyle?`

The style overlaid on selected items.

`var selectionBackgroundStyle: NSScrubberSelectionStyle?`

The style applied to the background of selected items.

Reloading content

`func reloadData()`

Reloads the content of the entire scrubber, and deselects the currently selected item.

`func reloadItems(at: IndexSet)`

Reloads the items at the specified indexes.

Getting the state of the scrubber

`var numberOfItems: Int`

The number of items represented by the scrubber.

`var highlightedIndex: Int`

The index of the highlighted item in the scrubber.

`var selectedIndex: Int`

The index of the selected item in the scrubber.

Inserting, moving, and deleting items

`func insertItems(at: IndexSet)`

Inserts new items at the specified indexes into the scrubber.

`func moveItem(at: Int, to: Int)`

Moves an item from one index to another in the scrubber.

`func removeItems(at: IndexSet)`

Removes the items at the specified indexes from the scrubber.

Animating multiple changes to the scrubber

`func performSequentialBatchUpdates(() -> Void)`

Combines multiple scrubber content updates into a single action.

Scrolling items

`func scrollItem(at: Int, to: NSScrubber.Alignment)`

Scrolls an item to a specified alignment within the scrubber.

Locating items in the scrubber

`func itemViewForItem(at: Int) -> NSScrubberItemView?`

Returns the view for the item at the specified index.

Relationships

Inherits From

`NSView`

Conforms To

CVarArg
CustomDebugStringConvertible
CustomStringConvertible
Equatable
Hashable
NSAccessibilityElementProtocol
NSAccessibilityProtocol
NSAnimatablePropertyContainer
NSAppearanceCustomization
NSCoding
NSDraggingDestination
NSObjectProtocol
NSStandardKeyBindingResponding
NSTouchBarProvider
NSUserActivityRestoring
NSUserInterfaceItemIdentification
Sendable
SendableMetatype

See Also

Scrubbers

`protocol` NSScrubberDataSource

A set of methods that a scrubber data source object implements to provide items to the scrubber from an associated data collection in your app.

`protocol` NSScrubberDelegate

A set of methods that a scrubber delegate implements to respond to user interactions.