Sample Code

Organize Your User Interface with a Stack View

Group individual views in your app's user interface into a scrollable stack view.

Download

macOS 10.13+ | Xcode 13.0+

Overview

NSStackView simplifies the process of stacking arbitrary views together in one place. You embed a stack view inside an NSScrollView object, making the stack's content scrollable when the user resizes the window. This sample shows how to use NSStackView with each individual view either disclosed (expanded) or hidden (collapsed).

The sample's architecture uses the following protocols:

- StackItemHeader. Represents the header's view and its ability to expand and collapse its corresponding stack item body.
- StackItemBody. Represents the stack item's primary interface.
- StackItemContainer. Represents each stack item, combining both the header view and the body view.
- StackItemHost. Discloses each stack item container.

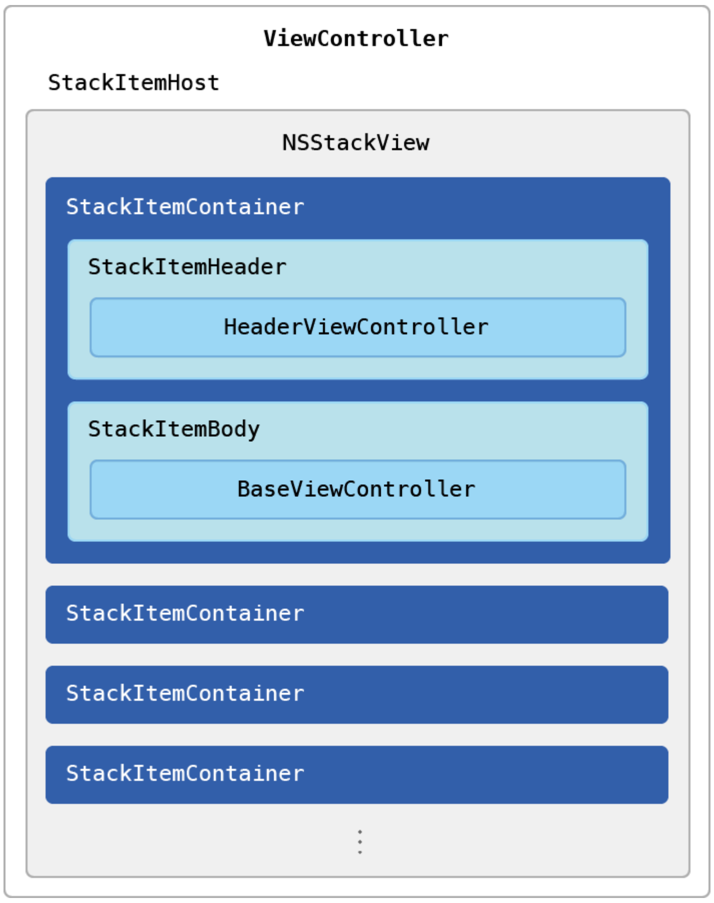
The following code shows the StackItemContainer class and its associated header and body components:

```
class StackItemContainer {
    // Disclosure state of this container.
    var state: NSControl.StateValue
    let header: StackItemHeader
    let body: StackItemBody

    init(header: StackItemHeader, body: StackItemBody) {
        self.header = header
        self.body = body
        self.state = .on
    }
```

This design simplifies adding stack items as view controllers. When you adopt these protocols, all stack items behave the same way and follow the same management rules.

The following illustration shows an example stack view:



Configure the Sample Code Project

In Xcode, choose your development team on the macOS target's General tab.

Configure the System

This sample uses window state restoration so that when the app is relaunched, the user interface returns to the state the user left it in. For state restoration to work, you must deselect the "Close windows when quitting an app" checkbox on the General Pane in System Preferences. To reset state restoration, use Command-Option-Quit when quitting the sample.

Configure the Stack View

Each stack item's header and body view controllers are loaded from a storyboard and added to the stack view in the `addViewController()` setup function.

Restore the User Interface

To restore the stack view's window on relaunch, use the `NSWindowRestoration` protocol. The disclosure states of all stack view containers are also restored. Each body view controller saves and restores various parts of its user interface.

To restore the states of stack view items, add the `NSRestorableState` to each item's view controller:

- `FormViewController`. Restores the form field text content.
- `TableViewController`. Restores the table view's selection.
- `CollectionViewController`. Restores the collection view's selection.

For `FormViewController`, restoration looks like this:

```
/// Encode state. Helps save the restorable state of this view controller.
override func encodeRestorableState(with coder: NSCoder) {

    coder.encode(textField.stringValue, forKey: FormViewController.formTextKey)
    super.encodeRestorableState(with: coder)
}

/// Decode state. Helps restore any previously stored state.
override func restoreState(with coder: NSCoder) {

    super.restoreState(with: coder)
    if let restoredText = coder.decodeObject(forKey: FormViewController.formTextKey
        textField.stringValue = restoredText
    }
}
```

Choose a Disclosure Appearance

This sample provides two disclosure options:

- A triangle
- A rounded button

The sample project conditionally decides which type of header disclosure control to use by referencing the `DisclosureTriangleAppearance` compilation flag. This flag is predefined in the "Active Compilation Conditions" build settings for passing conditional compilation flags to the Swift compiler. For more about configuring Xcode build settings, see Xcode Help.

If a disclosure appearance is defined, each section in the stack view gets the `NSButton` disclosure style (a triangle). If you prefer the `roundedDisclosure` style (a rounded button with up and down arrows), remove the compilation flag from the Xcode target's build settings.

See Also

Container views

{} Localization-friendly layouts in macOS

This project demonstrates localization-friendly auto layout constraints.

Grid View

Arrange views in a flexible grid, and handle the layout associated with those views.

`class NSSplitView`

A view that arranges two or more views in a linear stack running horizontally or vertically.

`class NSStackView`

A view that arranges an array of views horizontally or vertically and updates their placement and sizing when the window size changes.

`class NSTabView`

A multipage interface that displays one page at a time.

Scroll View

Provide an interface for navigating content that is too large to fit in the available space.