

---

[UIKit](#) / [NSStackView](#)

## Class

# NSStackView

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

macOS 10.9+

```
@MainActor
class NSStackView
```

---

## Overview

A stack view employs Auto Layout (the system's constraint-based layout feature) to arrange and align an array of views according to your specification. To use a stack view effectively, you need to understand the basics of Auto Layout constraints as described in [Auto Layout Guide](#).

## Basic Features of Stack Views

A stack view supports vertical and horizontal layouts and interacts dynamically with window resizing and Cocoa animations. You can easily reconfigure the contents of a stack view at runtime. That is, after you create and configure a stack view in Interface Builder, you can add or remove views dynamically without explicitly working with layout constraints. For example, if you configure a stack view with three checkboxes and dynamically add a fourth, the stack view automatically adds constraints as needed, according to the stack view's configuration. The new checkbox gains dynamic layout configuration from the stack view.

Stack views are nestable: a stack view is a valid element in the [views](#) array of another stack view.

### Important

Do not add views or constraints to a stack view's private views. A stack view's private views might change in future versions of macOS and are not guaranteed to be encoded or decoded with the `NSCoder` class.

For more information on `NSStackView`, see [Organize Your User Interface with a Stack View](#).

## Layout Direction and Gravity Areas

A stack view has three so-called *gravity areas* that each identify a section of the stack view's layout. A horizontal stack view, which is the default type, has a leading, a center, and a trailing gravity area. The ordering of these areas depends on the value of the stack view's `userInterfaceLayoutDirection` property (inherited from the `NSView` class). In a left to right language, the leading gravity area in a horizontal stack view is on the left. To enforce a left to right layout independently of language, explicitly set the layout direction by calling the inherited `userInterfaceLayoutDirection` method on your stack view instance.

To specify vertical layout, use the `orientation` property and the `NSUserInterfaceLayoutOrientation.vertical` constant from the `NSUserInterfaceLayoutOrientation` enumeration. In a vertical stack view, the gravity areas always are top, center, and bottom.

## View Detachment and Hiding

A stack view can automatically detach and reattach its views in response to layout changes, such as window resizing performed by the user, or resizing/repositioning of another view in the same view hierarchy. A view in a detached state is not present in the stack view's view hierarchy, but it still consumes memory. A view that is hidden, but not detached, remains part of the view hierarchy and continues to participate in Auto Layout, but it is not visible and doesn't receive input events.

To allow views to detach, set the so-called *clipping resistance* for a stack view to a value lower than its default of `required`. See the `setClippingResistancePriority(_:for:)` method.

You can influence which views detach first (and reattach last). Do this by setting the so-called *visibility priority* for each view whose detachment order you want to specify. A view with a lower visibility priority detaches before one with a higher priority, and reattaches after it. See the `NSStackView.VisibilityPriority` enumeration and the `setVisibilityPriority(_:for:)` method.

To explicitly detach a view from a stack view, call the `setVisibilityPriority(_:for:)` method with a value of `notVisible`. To explicitly reattach a view to a stack view, call the same method with a value of `mustHold`. If you hide a view that belongs to a stack view (by setting the

view's `isHidden` property to `true`), the view detaches from the stack view by default. Use the `detachesHiddenViews` property to change the default behavior.

The system calls a stack view delegate method when a view is about to be detached and when a view has been reattached, giving you the opportunity to run code at those times. See [NSStackViewDelegate](#).

---

## Topics

### Creating a Stack View

```
convenience init(views: [NSView])
```

Creates and returns a stack view with a specified array of views.

### Responding to Stack-Related Changes

```
var delegate: (any NSStackViewDelegate)?
```

The delegate object for the stack view.

```
protocol NSStackViewDelegate
```

A set of methods you use to respond to a stack view detaching and reattaching views.

### Managing Views in Gravity Areas

```
func addView(NSView, in: NSStackView.Gravity)
```

Adds a view to the end of the stack view gravity area.

```
func insertView(NSView, at: Int, in: NSStackView.Gravity)
```

Adds a view to a stack view gravity area at a specified index position.

```
func setViews([NSView], in: NSStackView.Gravity)
```

Specifies an array of views for a specified gravity area in the stack view, replacing any previous views in that area.

```
func removeView(NSView)
```

Removes a specified view from the stack view.

```
enum Gravity
```

The gravity areas available in a stack view.

## Managing the Arranged Subviews

```
func addArrangedSubview(NSView)
```

Adds the specified view to the end of the arranged subviews list.

```
func insertArrangedSubview(NSView, at: Int)
```

Adds the provided view to the array of arranged subviews at the specified index.

```
func removeArrangedSubview(NSView)
```

Removes the provided view from the stack's array of arranged subviews.

```
var arrangedSubviews: [NSView]
```

The array of views arranged by the stack view.

## Inspecting a Stack View

```
var views: [NSView]
```

The array of views owned by the stack view.

```
func views(in: NSStackView.Gravity) -> [NSView]
```

Returns the array of views in the specified gravity area in the stack view.

```
var detachedViews: [NSView]
```

An array that contains the detached views from all the stack view's gravity areas.

```
func clippingResistancePriority(for: NSLayoutConstraint.Orientation) ->  
NSLayoutConstraint.Priority
```

Returns the Auto Layout priority for resisting clipping of views in the stack view when Auto Layout attempts to reduce the stack view's size.

```
func huggingPriority(for: NSLayoutConstraint.Orientation) -> NSLayout  
Constraint.Priority
```

Returns the Auto Layout priority for the stack view to minimize its size to fit its contained views as closely as possible, for a specified user interface axis.

## Configuring the Stack View Layout

```
var orientation: NSUserInterfaceLayoutOrientation
```

The horizontal or vertical layout direction of the stack view.

`enum NSUserInterfaceLayoutOrientation`

The stack view layout directions, and user interface axes for hugging priority and clipping resistance.

`var alignment: NSLayoutConstraint.Attribute`

The view alignment within the stack view.

`var spacing: CGFloat`

The minimum spacing, in points, between adjacent views in the stack view.

`class let useDefaultSpacing: CGFloat`

`var edgeInsets: NSEdgeInsets`

The geometric padding, in points, inside the stack view, surrounding its views.

~~`var hasEqualSpacing: Bool`~~

A Boolean value that indicates whether the spacing between adjacent views should be equal to each other.

Deprecated

`var distribution: NSStackView.Distribution`

`enum Distribution`

## Configuring Views in a Stack View

`func customSpacing(after: NSView) -> CGFloat`

Returns the custom spacing, in points, between a specified view in the stack view and the view that follows it.

`func setCustomSpacing(CGFloat, after: NSView)`

Specifies the custom spacing, in points, between a specified view and the view that follows it in the stack view.

`func visibilityPriority(for: NSView) -> NSStackView.VisibilityPriority`

Returns the visibility priority for a specified view in the stack view.

`func setVisibilityPriority(NSStackView.VisibilityPriority, for: NSView)`

Sets the Auto Layout priority for a view to remain attached to the stack view when Auto Layout reduces the stack view's size.

`struct VisibilityPriority`

The various Auto Layout priorities for a view in the stack view to remain attached.

```
class let useDefaultSpacing: CGFloat
```

## Configuring Dynamic Behavior for a Stack View

```
var detachesHiddenViews: Bool
```

A Boolean value that indicates whether the stack view removes hidden views from its view hierarchy.

```
func setClippingResistancePriority(NSLayoutConstraint.Priority, for: NSLayoutConstraint.Orientation)
```

Sets the Auto Layout priority for resisting clipping of views in the stack view when Auto Layout attempts to reduce the stack view's size.

```
func setHuggingPriority(NSLayoutConstraint.Priority, for: NSLayoutConstraint.Orientation)
```

Sets the Auto Layout priority for the stack view to minimize its size, for a specified user interface axis.

---

## Relationships

### Inherits From

NSView

### Conforms To

CVarArg  
CustomDebugStringConvertible  
CustomStringConvertible  
Equatable  
Hashable  
NSAccessibilityElementProtocol  
NSAccessibilityProtocol  
NSAnimatablePropertyContainer  
NSAppearanceCustomization  
NSCoding  
NSDraggingDestination  
NSObjectProtocol  
NSStandardKeyBindingResponding

NSTouchBarProvider

NSUserActivityRestoring

NSUserInterfaceItemIdentification

Sendable

SendableMetatype