# UIScrollView

A view that allows the scrolling and zooming of its contained views.

## Declaration

```
class UIScrollView : UIView
```

## Overview

`UIScrollView` is the superclass of several UIKit classes including `UITableView` and `UITextView`.

The central notion of a `UIScrollView` object (or, simply, a scroll view) is that it is a view whose origin is adjustable over the content view. It clips the content to its frame, which generally (but not necessarily) coincides with that of the application's main window. A scroll view tracks the movements of fingers and adjusts the origin accordingly. The view that is showing its content "through" the scroll view draws that portion of itself based on the new origin, which is pinned to an offset in the content view. The scroll view itself does no drawing except for displaying vertical and horizontal scroll indicators. The scroll view must know the size of the content view so it knows when to stop scrolling; by default, it "bounces" back when scrolling exceeds the bounds of the content.

The object that manages the drawing of content displayed in a scroll view should tile the content's subviews so that no view exceeds the size of the screen. As users scroll in the scroll view, this object should add and remove subviews as necessary.

Because a scroll view has no scroll bars, it must know whether a touch signals an intent to scroll versus an intent to track a subview in the content. To make this determination, it temporarily intercepts a touch-down event by starting a timer and, before the timer fires, seeing if the touching finger makes any movement. If the timer fires without a significant change in position, the scroll view sends tracking events to the touched subview of the content view. If the user then drags their finger far enough before the timer elapses, the scroll view cancels any tracking in the subview and performs the scrolling itself. Subclasses can override the `touchesShouldBegin(_:with:in:)`, `isPagingEnabled`, and `touchesShouldCancel(in:)` methods (which are called by the scroll view) to affect how the scroll view handles scrolling gestures.

A scroll view also handles zooming and panning of content. As the user makes a pinch-in or pinch-out gesture, the scroll view adjusts the offset and the scale of the content. When the gesture ends, the object managing the content view should update subviews of the content as necessary. (Note that the gesture can end and a finger could still be down.) While the gesture is in progress, the scroll view does not send any tracking calls to the subview.

The `UIScrollView` class can have a delegate that must adopt the `UIScrollViewDelegate`protocol. For zooming and panning to work, the delegate must implement both `viewForZooming(in:)` and `scrollViewDidEndZooming(_:with:atScale:)`; in addition, the maximum (`maximumZoomScale`) and minimum (`minimumZoomScale`) zoom scale must be different.

## State Preservation

If you assign a value to this view's `restorationIdentifier` property, it attempts to preserve its

scrolling-related information between app launches. Specifically, the values of the `zoomScale`, `contentInset`, and `contentOffset` properties are preserved. During restoration, the scroll view restores these values so that the content appears scrolled to the same position as before. For more information about how state preservation and restoration works, see App Programming Guide for iOS.

# Topics

## Responding to Scroll View Interactions

`var delegate: UIScrollViewDelegate?`
The delegate of the scroll-view object.

`protocol UIScrollViewDelegate`
The methods declared by the `UIScrollViewDelegate` protocol allow the adopting delegate to respond to messages from the `UIScrollView` class and thus respond to, and in some affect, operations such as scrolling, zooming, deceleration of scrolled content, and scrolling animations.

## Managing the Content Size and Offset

`var contentSize: CGSize`
The size of the content view.

`var contentOffset: CGPoint`
The point at which the origin of the content view is offset from the origin of the scroll view.

`func setContentOffset(CGPoint, animated: Bool)`
Sets the offset from the content view's origin that corresponds to the receiver's origin.

## Managing the Content Inset Behavior

`var adjustedContentInset: UIEdgeInsets`
The insets derived from the content insets and the safe area of the scroll view.

`var contentInset: UIEdgeInsets`
The custom distance that the content view is inset from the safe area or scroll view edges.

`var contentInsetAdjustmentBehavior: UIScrollView.ContentInsetAdjustmentBehavior`
The behavior for determining the adjusted content offsets.

`enum UIScrollView.ContentInsetAdjustmentBehavior`
Constants indicating how safe area insets are added to the adjusted content inset.

`func adjustedContentInsetDidChange()`
Called when the adjusted content insets of the scroll view change.

## Getting the Layout Guides

`var frameLayoutGuide: UILayoutGuide`
The layout guide based on the untransformed frame rectangle of the scroll view.

`var contentLayoutGuide: UILayoutGuide`
The layout guide based on the untranslated content rectangle of the scroll view.

## Configuring the Scroll View

`var isScrollEnabled: Bool`
A Boolean value that determines whether scrolling is enabled.

`var isDirectionalLockEnabled: Bool`
A Boolean value that determines whether scrolling is disabled in a particular direction.

`var isPagingEnabled: Bool`
A Boolean value that determines whether paging is enabled for the scroll view.

`var scrollsToTop: Bool`
A Boolean value that controls whether the scroll-to-top gesture is enabled.

`var bounces: Bool`
A Boolean value that controls whether the scroll view bounces past the edge of content and back again.

`var alwaysBounceVertical: Bool`
A Boolean value that determines whether bouncing always occurs when vertical scrolling reaches the end of the content.

`var alwaysBounceHorizontal: Bool`
A Boolean value that determines whether bouncing always occurs when horizontal scrolling reaches the end of the content view.

## Getting the Scrolling State

`var isTracking: Bool`
Returns whether the user has touched the content to initiate scrolling.

`var isDragging: Bool`
A Boolean value that indicates whether the user has begun scrolling the content.

`var isDecelerating: Bool`
Returns whether the content is moving in the scroll view after the user lifted their finger.

`var decelerationRate: UIScrollView.DecelerationRate`
A floating-point value that determines the rate of deceleration after the user lifts their finger.

`struct UIScrollView.DecelerationRate`
Deceleration rates for the scroll view.

## Managing the Scroll Indicator and Refresh Control

`var indicatorStyle: UIScrollView.IndicatorStyle`
The style of the scroll indicators.

`enum UIScrollView.IndicatorStyle`
The style of the scroll indicators. You use these constants to set the value of the `indicatorStyle` style.

`var scrollIndicatorInsets: UIEdgeInsets`
The distance the scroll indicators are inset from the edge of the scroll view.

`var showsHorizontalScrollIndicator: Bool`
A Boolean value that controls whether the horizontal scroll indicator is visible.

`var showsVerticalScrollIndicator: Bool`
A Boolean value that controls whether the vertical scroll indicator is visible.

`func flashScrollIndicators()`
Displays the scroll indicators momentarily.

`var refreshControl: UIRefreshControl?`
The refresh control associated with the scroll view.

## Scrolling to a Specific Location

`func scrollRectToVisible(CGRect, animated: Bool)`
Scrolls a specific area of the content so that it is visible in the receiver.

## Managing Touches

```
func touchesShouldBegin(Set<UITouch>, with: UIEvent?, in: UIView) -> Bool
```

Overridden by subclasses to customize the default behavior when a finger touches down in displayed content.

```
func touchesShouldCancel(in: UIView) -> Bool
```

Returns whether to cancel touches related to the content subview and start dragging.

```
var canCancelContentTouches: Bool
```

A Boolean value that controls whether touches in the content view always lead to tracking.

```
var delaysContentTouches: Bool
```

A Boolean value that determines whether the scroll view delays the handling of touch-down gestures.

```
var directionalPressGestureRecognizer: UIGestureRecognizer
```

The underlying gesture recognizer for directional button presses.

## Zooming and Panning

`var panGestureRecognizer: UIPanGestureRecognizer`
The underlying gesture recognizer for pan gestures.

`var pinchGestureRecognizer: UIPinchGestureRecognizer?`
The underlying gesture recognizer for pinch gestures.

`func zoom(to: CGRect, animated: Bool)`
Zooms to a specific area of the content so that it is visible in the receiver.

`var zoomScale: CGFloat`
A floating-point value that specifies the current scale factor applied to the scroll view's content.

`func setZoomScale(CGFloat, animated: Bool)`
A floating-point value that specifies the current zoom scale.

`var maximumZoomScale: CGFloat`
A floating-point value that specifies the maximum scale factor that can be applied to the scroll view's content.

`var minimumZoomScale: CGFloat`
A floating-point value that specifies the minimum scale factor that can be applied to the scroll view's content.

`var isZoomBouncing: Bool`
A Boolean value that indicates that zooming has exceeded the scaling limits specified for the receiver.

`var isZooming: Bool`
A Boolean value that indicates whether the content view is currently zooming in or out.

`var bouncesZoom: Bool`
A Boolean value that determines whether the scroll view animates the content scaling when the scaling exceeds the maximum or minimum limits.

## Managing the Keyboard

`var keyboardDismissMode: UIScrollView.KeyboardDismissMode`
The manner in which the keyboard is dismissed when a drag begins in the scroll view.

`enum UIScrollView.KeyboardDismissMode`
The manner in which the keyboard is dismissed when a drag begins in the scroll view.

## Managing the Index

`var indexDisplayMode: UIScrollView.IndexDisplayMode`

The manner in which the index is shown while the user is scrolling.

`enum UIScrollView.IndexDisplayMode`

The manner in which the index is shown while the user is scrolling.