

# ScrollView

Component that wraps platform ScrollView while providing integration with touch locking "responder" system.

Keep in mind that ScrollViews must have a bounded height in order to work, since they contain unbounded-height children into a bounded container (via a scroll interaction). In order to bound the height of a ScrollView, either set the height of the view directly (discouraged) or make sure all parent views have bounded height. Forgetting to transfer `{flex: 1}` down the view stack can lead to errors here, which the element inspector makes quick to debug.

Doesn't yet support other contained responders from blocking this scroll view from becoming the responder.

`<ScrollView>` vs `<FlatList>` - which one to use?

`ScrollView` renders all its react child components at once, but this has a performance downside.

Imagine you have a very long list of items you want to display, maybe several screens worth of content. Creating JS components and native views for everything all at once, much of which may not even be shown, will contribute to slow rendering and increased memory usage.

This is where `FlatList` comes into play. `FlatList` renders items lazily, when they are about to appear, and removes items that scroll way off screen to save memory and processing time.

`FlatList` is also handy if you want to render separators between your items, multiple columns, infinite scroll loading, or any number of other features it supports out of the box.

## Example

## ScrollView

```
import React from 'react';
import {
  StyleSheet,
  Text,
  SafeAreaView,
  ScrollView,
  StatusBar,
} from 'react-native';

const App = () => {
  return (
    <SafeAreaView style={styles.container}>
      <ScrollView style={styles.scrollView}>
        <Text style={styles.text}>
          Lorem ipsum dolor sit amet, consectetur
adipiscing elit, sed do
          eiusmod tempor incididunt ut labore et dolore
magna aliqua. Ut enim ad
          minim veniam, quis nostrud exercitation ullamco
laboris nisi ut
          aliquip ex ea commodo consequat. Duis aute irure
dolor in
          reprehenderit in voluptate velit esse cillum
dolore eu fugiat nulla
          pariatur. Excepteur sint occaecat cupidatat non
proident, sunt in
```

Preview



My Device

iOS

Android

Web

# Reference

## Props

### View Props

Inherits View Props.

### StickyHeaderComponent

A React Component that will be used to render sticky headers, should be used together with `stickyHeaderIndices`. You may need to set this component if your sticky header uses custom transforms, for example, when you want your list to have an animated and hidable header. If a component has not been provided, the default `ScrollViewStickyHeader` component will be used.

TYPE
component, element

## **alwaysBounceHorizontal** ◀ iOS

When true, the scroll view bounces horizontally when it reaches the end even if the content is smaller than the scroll view itself.

TYPE	DEFAULT
bool	true when <code>horizontal={true}</code> false otherwise

## **alwaysBounceVertical** ◀ iOS

When true, the scroll view bounces vertically when it reaches the end even if the content is smaller than the scroll view itself.

TYPE	DEFAULT
bool	false when <code>vertical={true}</code> true otherwise

## **automaticallyAdjustContentInsets** ◀ iOS

Controls whether iOS should automatically adjust the content inset for scroll views that are placed behind a navigation bar or tab bar/toolbar.

TYPE	DEFAULT
bool	true

## **automaticallyAdjustKeyboardInsets** ◀ iOS

Controls whether the ScrollView should automatically adjust its `contentInset` and `scrollViewInsets` when the Keyboard changes its size.

TYPE	DEFAULT
bool	false

## **automaticallyAdjustsScrollIndicatorInsets** ◀ iOS

Controls whether iOS should automatically adjust the scroll indicator insets. See Apple's [documentation on the property](#).

TYPE	DEFAULT
bool	true

## **bounces** ◀ iOS

When true, the scroll view bounces when it reaches the end of the content if the content is larger than the scroll view along the axis of the scroll direction. When `false`, it disables all bouncing even if the `alwaysBounce*` props are true.

TYPE	DEFAULT
bool	true

## bouncesZoom ◀ iOS

When `true`, gestures can drive zoom past min/max and the zoom will animate to the min/max value at gesture end, otherwise the zoom will not exceed the limits.

TYPE	DEFAULT
bool	true

## canCancelContentTouches ◀ iOS

When `false`, once tracking starts, won't try to drag if the touch moves.

TYPE	DEFAULT
bool	true

## centerContent ◀ iOS

When `true`, the scroll view automatically centers the content when the content is smaller than the scroll view bounds; when the content is larger than the scroll view, this property has no effect.

TYPE	DEFAULT
bool	false

## contentContainerStyle

These styles will be applied to the scroll view content container which wraps all of the child views. Example:

```
return (  
  <ScrollView contentContainerStyle={styles.contentContainer}>  
    </ScrollView>  
  );  
  ...  
  const styles = StyleSheet.create({  
    contentContainer: {  
      paddingVertical: 20  
    }  
  });
```

TYPE
<a href="#">View Style</a>

**contentInset** ◀ iOS

The amount by which the scroll view content is inset from the edges of the scroll view.

TYPE	DEFAULT
object: {top: number, left: number, bottom: number, right: number}	{top: 0, left: 0, bottom: 0, right: 0}

**contentInsetAdjustmentBehavior** ◀ iOS

This property specifies how the safe area insets are used to modify the content area of the scroll view. Available on iOS 11 and later.

TYPE	DEFAULT
enum('automatic', 'scrollableAxes', 'never', 'always')	'never'

**contentOffset**

Used to manually set the starting scroll offset.

TYPE	DEFAULT
Point	{x: 0, y: 0}

## **decelerationRate**

A floating-point number that determines how quickly the scroll view decelerates after the user lifts their finger. You may also use string shortcuts "normal" and "fast" which match the underlying iOS settings for `UIScrollViewDecelerationRateNormal` and `UIScrollViewDecelerationRateFast` respectively.

- 'normal' 0.998 on iOS, 0.985 on Android.
- 'fast', 0.99 on iOS, 0.9 on Android.

TYPE	DEFAULT
enum( 'fast', 'normal' ), number	'normal'

## **directionalLockEnabled** ◀ iOS

When true, the ScrollView will try to lock to only vertical or horizontal scrolling while dragging.

TYPE	DEFAULT
bool	false

## **disableIntervalMomentum**

When true, the scroll view stops on the next index (in relation to scroll position at release) regardless of how fast the gesture is. This can be used for pagination when the page is less than the width of the horizontal ScrollView or the height of the vertical ScrollView.

TYPE	DEFAULT
bool	false

## disableScrollViewPanResponder

When true, the default JS pan responder on the ScrollView is disabled, and full control over touches inside the ScrollView is left to its child components. This is particularly useful if `snapToInterval` is enabled, since it does not follow typical touch patterns. Do not use this on regular ScrollView use cases without `snapToInterval` as it may cause unexpected touches to occur while scrolling.

TYPE	DEFAULT
bool	false

## endFillColor Android

Sometimes a scrollview takes up more space than its content fills. When this is the case, this prop will fill the rest of the scrollview with a color to avoid setting a background and creating unnecessary overdraw. This is an advanced optimization that is not needed in the general case.

TYPE
<u>color</u>

## fadingEdgeLength Android

Fades out the edges of the the scroll content.

If the value is greater than 0, the fading edges will be set accordingly to the current scroll direction and position, indicating if there is more content to show.



TYPE	DEFAULT
number	0

## horizontal

When `true`, the scroll view's children are arranged horizontally in a row instead of vertically in a column.

TYPE	DEFAULT
bool	false

## indicatorStyle ◀ iOS

The style of the scroll indicators.

- 'default' same as `black`.
- 'black', scroll indicator is `black`. This style is good against a light background.
- 'white', scroll indicator is `white`. This style is good against a dark background.

TYPE	DEFAULT
enum('default', 'black', 'white')	'default'

## invertStickyHeaders

If sticky headers should stick at the bottom instead of the top of the ScrollView. This is usually used with inverted ScrollViews.

TYPE	DEFAULT
bool	false



## keyboardDismissMode

Determines whether the keyboard gets dismissed in response to a drag.

- 'none', drags do not dismiss the keyboard.
- 'on-drag', the keyboard is dismissed when a drag begins.

### iOS Only

- 'interactive', the keyboard is dismissed interactively with the drag and moves in synchrony with the touch, dragging upwards cancels the dismissal. On Android this is not supported and it will have the same behavior as 'none'.

TYPE	DEFAULT
enum( 'none', 'on-drag' )  Android	'none'
enum( 'none', 'on-drag', 'interactive' )  iOS	

## keyboardShouldPersistTaps

Determines when the keyboard should stay visible after a tap.

- 'never' tapping outside of the focused text input when the keyboard is up dismisses the keyboard. When this happens, children won't receive the tap.
- 'always', the keyboard will not dismiss automatically, and the scroll view will not catch taps, but children of the scroll view can catch taps.
- 'handled', the keyboard will not dismiss automatically when the tap was handled by children of the scroll view (or captured by an ancestor).
- false, **deprecated**, use 'never' instead
- true, **deprecated**, use 'always' instead

TYPE	DEFAULT
enum( 'always', 'never', 'handled', false, true )	'never'

## maintainVisibleContentPosition

When set, the scroll view will adjust the scroll position so that the first child that is currently visible and at or beyond `minIndexForVisible` will not change position. This is useful for lists that are loading content in both directions, e.g. a chat thread, where new messages coming in might otherwise cause the scroll position to jump. A value of 0 is common, but other values such as 1 can be used to skip loading spinners or other content that should not maintain position.

The optional `autoscrollToTopThreshold` can be used to make the content automatically scroll to the top after making the adjustment if the user was within the threshold of the top before the adjustment was made. This is also useful for chat-like applications where you want to see new messages scroll into place, but not if the user has scrolled up a ways and it would be disruptive to scroll a bunch.

Caveat 1: Reordering elements in the scrollview with this enabled will probably cause jumpiness and jank. It can be fixed, but there are currently no plans to do so. For now, don't re-order the content of any ScrollViews or Lists that use this feature.

Caveat 2: This uses `contentOffset` and `frame.origin` in native code to compute visibility. Occlusion, transforms, and other complexity won't be taken into account as to whether content is "visible" or not.

TYPE
object: {minIndexForVisible: number, autoscrollToTopThreshold: number}

## maximumZoomScale ◀ iOS

The maximum allowed zoom scale.

TYPE	DEFAULT
number	1.0

## minimumZoomScale ◀ iOS

The minimum allowed zoom scale.

TYPE	DEFAULT
number	1.0

## nestedScrollEnabled ▶ Android

Enables nested scrolling for Android API level 21+.

TYPE	DEFAULT
bool	false

## onContentSizeChange

Called when scrollable content view of the ScrollView changes.

The handler function will receive two parameters: the content width and content height (contentWidth, contentHeight).

It's implemented using onLayout handler attached to the content container which this ScrollView renders.

TYPE
function

## onMomentumScrollBegin

Called when the momentum scroll starts (scroll which occurs as the ScrollView starts gliding).

TYPE
function

## onMomentumScrollEnd

Called when the momentum scroll ends (scroll which occurs as the ScrollView glides to a stop).

TYPE
function

## onScroll

Fires at most once per frame during scrolling. The frequency of the events can be controlled using the `scrollEventThrottle` prop. The event has the following shape (all values are numbers):

```
{
  nativeEvent: {
    contentInset: {bottom, left, right, top},
    contentOffset: {x, y},
    contentSize: {height, width},
    layoutMeasurement: {height, width},
    zoomScale
  }
}
```

TYPE
function

## onScrollBeginDrag

Called when the user begins to drag the scroll view.

TYPE
function

## onScrollEndDrag

Called when the user stops dragging the scroll view and it either stops or begins to glide.

TYPE
function

## onScrollToTop iOS

Fires when the scroll view scrolls to top after the status bar has been tapped.

TYPE
function

## overScrollMode Android

Used to override default value of overScroll mode.

Possible values:

- 'auto' - Allow a user to over-scroll this view only if the content is large enough to meaningfully scroll.
- 'always' - Always allow a user to over-scroll this view.
- 'never' - Never allow a user to over-scroll this view.

TYPE	DEFAULT
<code>enum( 'auto' , 'always' , 'never' )</code>	<code>'auto'</code>

## **pagingEnabled**

When true, the scroll view stops on multiples of the scroll view's size when scrolling. This can be used for horizontal pagination.

Note: Vertical pagination is not supported on Android.

TYPE	DEFAULT
<code>bool</code>	<code>false</code>

## **persistentScrollbar** Android

Causes the scrollbars not to turn transparent when they are not in use.

TYPE	DEFAULT
<code>bool</code>	<code>false</code>

## **pinchGestureEnabled** iOS

When true, ScrollView allows use of pinch gestures to zoom in and out.

TYPE	DEFAULT
<code>bool</code>	<code>true</code>

## **refreshControl**

A RefreshControl component, used to provide pull-to-refresh functionality for the ScrollView. Only works for vertical ScrollViews (`horizontal` prop must be `false`).

See [RefreshControl](#).

TYPE
element

## **removeClippedSubviews**

Experimental: When `true`, offscreen child views (whose `overflow` value is `hidden`) are removed from their native backing superview when offscreen. This can improve scrolling performance on long lists.

TYPE	DEFAULT
bool	false

## **scrollEnabled**

When `false`, the view cannot be scrolled via touch interaction.

Note that the view can always be scrolled by calling `scrollTo`.

TYPE	DEFAULT
bool	true

## **scrollEventThrottle** ◀ iOS

This controls how often the scroll event will be fired while scrolling (as a time interval in ms). A lower number yields better accuracy for code that is tracking the scroll position, but can lead to scroll performance problems due to the volume of information being sent



over the bridge. You will not notice a difference between values set between 1-16 as the JS run loop is synced to the screen refresh rate. If you do not need precise scroll position tracking, set this value higher to limit the information being sent across the bridge. The default value is 0, which results in the scroll event being sent only once each time the view is scrolled.

TYPE	DEFAULT
number	0

## scrollIndicatorInsets ◀ iOS

The amount by which the scroll view indicators are inset from the edges of the scroll view. This should normally be set to the same value as the `contentInset`.

TYPE	DEFAULT
object: {top: number, left: number, bottom: number, right: number}	{top: 0, left: 0, bottom: 0, right: 0}

## scrollPerfTag ▶ Android

Tag used to log scroll performance on this scroll view. Will force momentum events to be turned on (see `sendMomentumEvents`). This doesn't do anything out of the box and you need to implement a custom native `FpsListener` for it to be useful.

TYPE
string

## scrollToOverflowEnabled ◀ iOS

When `true`, the scroll view can be programmatically scrolled beyond its content size.

TYPE	DEFAULT
bool	false

## scrollsToTop ◀ iOS

When `true`, the scroll view scrolls to top when the status bar is tapped.

TYPE	DEFAULT
bool	true

## showsHorizontalScrollIndicator

When `true`, shows a horizontal scroll indicator.

TYPE	DEFAULT
bool	true

## showsVerticalScrollIndicator

When `true`, shows a vertical scroll indicator.

TYPE	DEFAULT
bool	true

## snapToAlignment ◀ iOS

When `snapToInterval` is set, `snapToAlignment` will define the relationship of the snapping to the scroll view.

Possible values:

- 'start' will align the snap at the left (horizontal) or top (vertical).
- 'center' will align the snap in the center.
- 'end' will align the snap at the right (horizontal) or bottom (vertical).

TYPE	DEFAULT
enum( 'start' , 'center' , 'end' )	'start'

## snapToEnd

Use in conjunction with `snapToOffsets`. By default, the end of the list counts as a snap offset. Set `snapToEnd` to false to disable this behavior and allow the list to scroll freely between its end and the last `snapToOffsets` offset.

TYPE	DEFAULT
bool	true

## snapToInterval

When set, causes the scroll view to stop at multiples of the value of `snapToInterval`. This can be used for paginating through children that have lengths smaller than the scroll view. Typically used in combination with `snapToAlignment` and `decelerationRate="fast"`. Overrides less configurable `pagingEnabled` prop.

TYPE
number

## snapToOffsets

When set, causes the scroll view to stop at the defined offsets. This can be used for paginating through variously sized children that have lengths smaller than the scroll view. Typically used in combination with `decelerationRate="fast"`. Overrides less configurable `pagingEnabled` and `snapToInterval` props.

TYPE
array of number

## **snapToStart**

Use in conjunction with `snapToOffsets`. By default, the beginning of the list counts as a snap offset. Set `snapToStart` to `false` to disable this behavior and allow the list to scroll freely between its start and the first `snapToOffsets` offset.

TYPE	DEFAULT
bool	true

## **stickyHeaderHiddenOnScroll**

When set to `true`, sticky header will be hidden when scrolling down the list, and it will dock at the top of the list when scrolling up.

TYPE	DEFAULT
bool	false

## **stickyHeaderIndices**

An array of child indices determining which children get docked to the top of the screen when scrolling. For example, passing `stickyHeaderIndices={[0]}` will cause the first child to be fixed to the top of the scroll view. You can also use like `[x,y,z]` to make multiple items

sticky when they are at the top. This property is not supported in conjunction with `horizontal={true}`.

TYPE
array of number

## zoomScale iOS

The current scale of the scroll view content.

TYPE	DEFAULT
number	1.0

## Methods

### flashScrollIndicators()

```
flashScrollIndicators();
```

Displays the scroll indicators momentarily.

### scrollTo()

```
scrollTo(  
  options?: {x?: number, y?: number, animated?: boolean} | number,  
  deprecatedX?: number,  
  deprecatedAnimated?: boolean,  
);
```

Scrolls to a given x, y offset, either immediately, with a smooth animation.

## Example:

```
scrollTo({x: 0, y: 0, animated: true})
```

Note: The weird function signature is due to the fact that, for historical reasons, the function also accepts separate arguments as an alternative to the options object. This is deprecated due to ambiguity (y before x), and SHOULD NOT BE USED.


## scrollToEnd()

```
scrollToEnd(options?: {animated?: boolean});
```

If this is a vertical ScrollView scrolls to the bottom. If this is a horizontal ScrollView scrolls to the right.

Use `scrollToEnd({animated: true})` for smooth animated scrolling, `scrollToEnd({animated: false})` for immediate scrolling. If no options are passed, `animated` defaults to `true`.

Is this page useful?  

 Edit this page

Last updated on **Sep 1, 2023**