

SectionList

A performant interface for rendering sectioned lists, supporting the most handy features:

- Fully cross-platform.
- Configurable viewability callbacks.
- List header support.
- List footer support.
- Item separator support.
- Section header support.
- Section separator support.
- Heterogeneous data and item rendering support.
- Pull to Refresh.
- Scroll loading.

If you don't need section support and want a simpler interface, use [`<FlatList>`](#) .

Example

SectionList Example



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

const DATA = [
  {
    title: 'Main dishes',
    data: ['Pizza', 'Burger', 'Risotto'],
  },
  {
    title: 'Sides',
    data: ['French Fries', 'Onion Rings', 'Fried Shrimps'],
  },
  {
    title: 'Drinks',
    data: ['Water', 'Coke', 'Beer'],
  },
  {
    title: 'Desserts',
    data: ['Cheese Cake', 'Ice Cream'],
  },
];
```

Preview



My Device

iOS

Android

Web

This is a convenience wrapper around `<VirtualizedList>`, and thus inherits its props (as well as those of `<ScrollView>`) that aren't explicitly listed here, along with the following caveats:

- Internal state is not preserved when content scrolls out of the render window. Make sure all your data is captured in the item data or external stores like Flux, Redux, or Relay.
- This is a `PureComponent` which means that it will not re-render if props remain shallow-equal. Make sure that everything your `renderItem` function depends on is passed as a prop (e.g. `extraData`) that is not `===` after updates, otherwise your UI may not update on changes. This includes the `data` prop and parent component state.
- In order to constrain memory and enable smooth scrolling, content is rendered asynchronously offscreen. This means it's possible to scroll faster than the fill rate

and momentarily see blank content. This is a tradeoff that can be adjusted to suit the needs of each application, and we are working on improving it behind the scenes.

- By default, the list looks for a `key` prop on each item and uses that for the React key. Alternatively, you can provide a custom `keyExtractor` prop.

Reference

Props

VirtualizedList Props

Inherits VirtualizedList Props.

Required

renderItem

Default renderer for every item in every section. Can be over-ridden on a per-section basis. Should return a React element.

TYPE
function

The render function will be passed an object with the following keys:

- 'item' (object) - the item object as specified in this section's `data` key
- 'index' (number) - Item's index within the section.
- 'section' (object) - The full section object as specified in `sections`.
- 'separators' (object) - An object with the following keys:
 - 'highlight' (function) - `() => void`
 - 'unhighlight' (function) - `() => void`
 - 'updateProps' (function) - `(select, newProps) => void`
 - 'select' (enum) - possible values are 'leading', 'trailing'

- 'newProps' (object)

Required

sections

The actual data to render, akin to the `data` prop in `FlatList`.

TYPE
array of <code>Sections</code>

extraData

A marker property for telling the list to re-render (since it implements `PureComponent`). If any of your `renderItem`, `Header`, `Footer`, etc. functions depend on anything outside of the `data` prop, stick it here and treat it immutably.

TYPE
any

initialNumToRender

How many items to render in the initial batch. This should be enough to fill the screen but not much more. Note these items will never be unmounted as part of the windowed rendering in order to improve perceived performance of scroll-to-top actions.

TYPE	DEFAULT
number	10

inverted

Reverses the direction of scroll. Uses scale transforms of -1.

TYPE	DEFAULT
boolean	false

ItemSeparatorComponent

Rendered in between each item, but not at the top or bottom. By default, `highlighted`, `section`, and `[leading/trailing][Item/Section]` props are provided. `renderItem` provides `separators.highlight/unhighlight` which will update the `highlighted` prop, but you can also add custom props with `separators.updateProps`. Can be a React Component (e.g. `SomeComponent`), or a React element (e.g. `<SomeComponent />`).

TYPE
component, function, element

keyExtractor

Used to extract a unique key for a given item at the specified index. Key is used for caching and as the React key to track item re-ordering. The default extractor checks `item.key`, then falls back to using the index, like React does. Note that this sets keys for each item, but each overall section still needs its own key.

TYPE
(item: object, index: number) => string

ListEmptyComponent

Rendered when the list is empty. Can be a React Component (e.g. `SomeComponent`), or a React element (e.g. `<SomeComponent />`).

TYPE
component, element

ListFooterComponent

Rendered at the very end of the list. Can be a React Component (e.g. `SomeComponent`), or a React element (e.g. `<SomeComponent />`).

TYPE
component, element

ListHeaderComponent

Rendered at the very beginning of the list. Can be a React Component (e.g. `SomeComponent`), or a React element (e.g. `<SomeComponent />`).

TYPE
component, element

onRefresh

If provided, a standard RefreshControl will be added for "Pull to Refresh" functionality. Make sure to also set the `refreshing` prop correctly. To offset the RefreshControl from the top (e.g. by 100 pts), use `progressViewOffset={100}` .

TYPE
function

onViewableItemsChanged

Called when the viewability of rows changes, as defined by the `viewabilityConfig` prop.

TYPE
<code>(callback: {changed: ViewToken[], viewableItems: ViewToken[]}) => void</code>

refreshing

Set this true while waiting for new data from a refresh.

TYPE	DEFAULT
boolean	false

removeClippedSubviews

Note: may have bugs (missing content) in some circumstances - use at your own risk.

This may improve scroll performance for large lists.

TYPE	DEFAULT
boolean	false

renderSectionFooter

Rendered at the bottom of each section.

TYPE
<code>(info: {section: Section}) => element null</code>

renderSectionHeader

Rendered at the top of each section. These stick to the top of the `ScrollView` by default on iOS. See `stickySectionHeadersEnabled`.

TYPE
<code>(info: {section: Section}) => element null</code>



SectionSeparatorComponent

Rendered at the top and bottom of each section (note this is different from `ItemSeparatorComponent` which is only rendered between items). These are intended to separate sections from the headers above and below and typically have the same highlight response as `ItemSeparatorComponent`. Also receives `highlighted`, `[leading/trailing][Item/Section]`, and any custom props from `separators.updateProps`.

TYPE
<code>component, element</code>

stickySectionHeadersEnabled

Makes section headers stick to the top of the screen until the next one pushes it off. Only enabled by default on iOS because that is the platform standard there.

TYPE	DEFAULT
boolean	false  Android
	true  iOS

Methods

flashScrollIndicators() ◀ iOS

```
flashScrollIndicators();
```

Displays the scroll indicators momentarily.

recordInteraction()

```
recordInteraction();
```

Tells the list an interaction has occurred, which should trigger viewability calculations, e.g. if `waitForInteractions` is true and the user has not scrolled. This is typically called by taps on items or by navigation actions.

scrollToLocation()

```
scrollToLocation(params: SectionListScrollParams);
```

Scrolls to the item at the specified `sectionIndex` and `itemIndex` (within the section) positioned in the viewable area such that `viewPosition` 0 places it at the top (and may be covered by a sticky header), 1 at the bottom, and 0.5 centered in the middle.

Note: Cannot scroll to locations outside the render window without specifying the `getItemLayout` or `onScrollToIndexFailed` prop.

Parameters:

NAME	TYPE
params Required	object

Valid params keys are:

- 'animated' (boolean) - Whether the list should do an animation while scrolling. Defaults to `true`.
- 'itemIndex' (number) - Index within section for the item to scroll to. Required.
- 'sectionIndex' (number) - Index for section that contains the item to scroll to. Required.
- 'viewOffset' (number) - A fixed number of pixels to offset the final target position, e.g. to compensate for sticky headers.
- 'viewPosition' (number) - A value of `0` places the item specified by index at the top, `1` at the bottom, and `0.5` centered in the middle.

Type Definitions

Section

An object that identifies the data to be rendered for a given section.


TYPE
any

Properties:

NAME	TYPE	DESCRIPTION
data Required	array	The data for rendering items in this section. Array of objects, much like <code>FlatList</code> 's <code>data prop</code> .
key	string	Optional key to keep track of section re-ordering. If you don't plan on re-ordering sections, the array index will be used by default.
renderItem	function	Optionally define an arbitrary item renderer for this section, overriding the default <code>renderItem</code> for the list.
ItemSeparatorComponent	component, element	Optionally define an arbitrary item separator for this section, overriding the default

NAME	TYPE	<code>ItemSeparatorComponent</code> for the list.
<code>keyExtractor</code>	function	Optionally define an arbitrary key extractor for this section, overriding the default <code>keyExtractor</code> .

Is this page useful?  

 Edit this page

Last updated on **Aug 17, 2023**