# VirtualizedList

Base implementation for the more convenient <FlatList> and <SectionList> components, which are also better documented. In general, this should only really be used if you need more flexibility than \_FlatList \_ provides, e.g. for use with immutable data instead of plain arrays.

Virtualization massively improves memory consumption and performance of large lists by maintaining a finite render window of active items and replacing all items outside of the render window with appropriately sized blank space. The window adapts to scrolling behavior, and items are rendered incrementally with low-pri (after any running interactions) if they are far from the visible area, or with hi-pri otherwise to minimize the potential of seeing blank space.

# **Example**

TypeScript

**JavaScript** 

```
VirtualizedListExample
                                                                                                 ∧ Expo
import React from 'react';
import {
  SafeAreaView,
  View,
  VirtualizedList,
  StyleSheet,
  Text,
  StatusBar,
} from 'react-native';
type ItemData = {
  id: string;
  title: string;
};
const getItem = ( data: unknown, index: number): ItemData
=> ({
  id: Math.random().toString(12).substring(0),
  title: `Item ${index + 1}`,
});
const getItemCount = ( data: unknown) => 50;
type ItemProps = {
  title: string;
};
                                                             Preview
                                                                              My Device
                                                                                        iOS
                                                                                            Android
                                                                                                      Web
```

#### Some 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 are 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**

## **ScrollView Props**

Inherits ScrollView Props.

#### data

Opaque data type passed to getItem and getItemCount to retrieve items.

TYPE any

Required getItem

(data: any, index: number) => any;

A generic accessor for extracting an item from any sort of data blob.

TYPE function

Required

getItemCount

```
(data: any) => number;
```

Determines how many items are in the data blob.

TYPE function

Required

renderItem

```
(info: any) => ?React.Element<any>
```

Takes an item from data and renders it into the list

TYPE function

### CellRendererComponent

CellRendererComponent allows customizing how cells rendered by renderItem/ListItemComponent are wrapped when placed into the underlying ScrollView. This component must accept event handlers which notify VirtualizedList of changes within the cell.

TYPE

React.ComponentType<CellRendererProps>

### **ItemSeparatorComponent**

Rendered in between each item, but not at the top or bottom. By default, highlighted and leadingItem 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

### 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

### ListItemComponent

Each data item is rendered using this element. Can be a React Component Class, or a render function.

TYPE

component, function

### ListFooterComponent

Rendered at the bottom of all the items. Can be a React Component (e.g. SomeComponent), or a React element (e.g. <SomeComponent />).

ТҮРЕ	
component, element	

### **ListFooterComponentStyle**

Styling for internal View for ListFooterComponent.

TYPE	REQUIRED
ViewStyleProp	No

### ListHeaderComponent

Rendered at the top of all the items. Can be a React Component (e.g. SomeComponent), or a React element (e.g. <SomeComponent />).

TYPE	
component, element	

### **ListHeaderComponentStyle**

Styling for internal View for ListHeaderComponent.

TYPE	
View Style	

### debug

debug will turn on extra logging and visual overlays to aid with debugging both usage and implementation, but with a significant perf hit.

TYPE boolean

#### disableVirtualization

**Deprecated.** Virtualization provides significant performance and memory optimizations, but fully unmounts react instances that are outside of the render window. You should only need to disable this for debugging purposes.

```
TYPE boolean
```

#### 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
```

### getItemLayout

```
(
  data: any,
  index: number,
) => {length: number, offset: number, index: number}
```

TYPE			
function			

#### horizontal

If true, renders items next to each other horizontally instead of stacked vertically.

TYPE	
boolean	

#### 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.

ТҮРЕ	DEFAULT
number	10

#### initialScrollIndex

Instead of starting at the top with the first item, start at initialScrollIndex. This disables the "scroll to top" optimization that keeps the first initialNumToRender items always rendered and immediately renders the items starting at this initial index. Requires getItemLayout to be implemented.

TYPE	
number	

#### inverted

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

ТҮРЕ	
boolean	

### listKey

A unique identifier for this list. If there are multiple VirtualizedLists at the same level of nesting within another VirtualizedList, this key is necessary for virtualization to work properly.

ТҮРЕ	REQUIRED
string	True

### keyExtractor

```
(item: any, index: number) => string;
```

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 <code>item.key</code>, then <code>item.id</code>, and then falls back to using the index, like React does.

TYPE	
function	

#### maxToRenderPerBatch

The maximum number of items to render in each incremental render batch. The more rendered at once, the better the fill rate, but responsiveness may suffer because rendering content may interfere with responding to button taps or other interactions.

ТҮРЕ	
number	

#### onEndReached

Called once when the scroll position gets within within onEndReachedThreshold from the logical end of the list.

```
TYPE

(info: {distanceFromEnd: number}) => void
```

#### onEndReachedThreshold

How far from the end (in units of visible length of the list) the trailing edge of the list must be from the end of the content to trigger the onEndReached callback. Thus, a value of 0.5 will trigger onEndReached when the end of the content is within half the visible length of the list.

ТҮРЕ	DEFAULT
number	2

#### onRefresh

() => void;

If provided, a standard RefreshControl will be added for "Pull to Refresh" functionality. Make sure to also set the refreshing prop correctly.

```
TYPE function
```

#### onScrollToIndexFailed

```
(info: {
  index: number,
  highestMeasuredFrameIndex: number,
  averageItemLength: number,
}) => void;
```

Used to handle failures when scrolling to an index that has not been measured yet.

Recommended action is to either compute your own offset and scrollto it, or scroll as far as possible and then try again after more items have been rendered.

```
TYPE function
```

#### onStartReached

Called once when the scroll position gets within within onStartReachedThreshold from the logical start of the list.

```
TYPE

(info: {distanceFromStart: number}) => void
```

#### onStartReachedThreshold

How far from the start (in units of visible length of the list) the leading edge of the list must be from the start of the content to trigger the onStartReached callback. Thus, a value of 0.5 will trigger onStartReached when the start of the content is within half the visible length of the list.

ТҮРЕ	DEFAULT
number	2

### onViewableItemsChanged

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

```
TYPE

(callback: {changed: ViewToken[], viewableItems: ViewToken[]}) => void
```

### persistentScrollbar

ТҮРЕ	
bool	

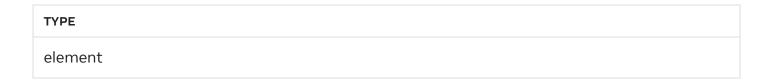
### progressViewOffset

Set this when offset is needed for the loading indicator to show correctly.

TYPE	
number	

### refreshControl

A custom refresh control element. When set, it overrides the default <RefreshControl> component built internally. The onRefresh and refreshing props are also ignored. Only works for vertical VirtualizedList.



### refreshing

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

TYPE boolean

### removeClippedSubviews

This may improve scroll performance for large lists.

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

TYPE boolean

### renderScrollComponent

```
(props: object) => element;
```

Render a custom scroll component, e.g. with a differently styled RefreshControl.

TYPE			
function			

### viewabilityConfig

See ViewabilityHelper.js for flow type and further documentation.

TYPE		
ViewabilityConfig		

### viewabilityConfigCallbackPairs

List of ViewabilityConfig/onViewableItemsChanged pairs. A specific onViewableItemsChanged will be called when its corresponding ViewabilityConfig's conditions are met. See ViewabilityHelper.js for flow type and further documentation.

array of ViewabilityConfigCallbackPair

### updateCellsBatchingPeriod

Amount of time between low-pri item render batches, e.g. for rendering items quite a ways off screen. Similar fill rate/responsiveness tradeoff as maxToRenderPerBatch.

TYPE	
number	

#### windowSize

Determines the maximum number of items rendered outside of the visible area, in units of visible lengths. So if your list fills the screen, then windowSize={21} (the default) will render the visible screen area plus up to 10 screens above and 10 below the viewport. Reducing this number will reduce memory consumption and may improve performance, but will increase the chance that fast scrolling may reveal momentary blank areas of unrendered content.

```
number
```

### **Methods**

### flashScrollIndicators()

```
flashScrollIndicators();
```

### getScrollableNode()

```
getScrollableNode(): any;
```

### getScrollRef()

```
getScrollRef():
    | React.ElementRef<typeof ScrollView>
    | React.ElementRef<typeof View>
    | null;
```

### getScrollResponder()

```
getScrollResponder () => ScrollResponderMixin | null;
```

Provides a handle to the underlying scroll responder. Note that this.\_scrollRef might not be a ScrollView, so we need to check that it responds to getScrollResponder before calling it.

### scrollToEnd()

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

Scrolls to the end of the content. May be janky without getItemLayout prop.

#### **Parameters:**

NAME	TYPE
params	object

Valid params keys are:

• 'animated' (boolean) - Whether the list should do an animation while scrolling.

Defaults to true.

### scrollToIndex()

```
scrollToIndex(params: {
  index: number;
  animated?: boolean;
  viewOffset?: number;
  viewPosition?: number;
});
```

Valid params consist of:

- 'index' (number). Required.
- 'animated' (boolean). Optional.
- 'viewOffset' (number). Optional.
- 'viewPosition' (number). Optional.

### scrollToItem()

```
scrollToItem(params: {
  item: ItemT;
  animated?: boolean;
  viewOffset?: number;
  viewPosition?: number;
);
```

Valid params consist of:

- 'item' (Item). Required.
- 'animated' (boolean). Optional.
- 'viewOffset' (number). Optional.
- 'viewPosition' (number). Optional.

## scrollToOffset()

```
scrollToOffset(params: {
  offset: number;
  animated?: boolean;
});
```

Scroll to a specific content pixel offset in the list.

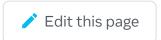
Param offset expects the offset to scroll to. In case of horizontal is true, the offset is the x-value, in any other case the offset is the y-value.

Param animated (true by default) defines whether the list should do an animation while scrolling.

Is this page useful?







Last updated on Aug 17, 2023