FlatList

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

- Fully cross-platform.
- Optional horizontal mode.
- Configurable viewability callbacks.
- Header support.
- Footer support.
- Separator support.
- Pull to Refresh.
- Scroll loading.
- ScrollToIndex support.
- Multiple column support.

If you need section support, use `<SectionList>`.

Example

TypeScript JavaScript

flatlist-simple Expo

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

const DATA = [
  {
    id: 'bd7acbea-c1b1-46c2-aed5-3ad53abb28ba',
    title: 'First Item',
  },
  {
    id: '3ac68afc-c605-48d3-a4f8-fbd91aa97f63',
    title: 'Second Item',
  },
  {
    id: '58694a0f-3da1-471f-bd96-145571e29d72',
    title: 'Third Item',
  },
];

type ItemProps = {title: string};
```

Preview My Device iOS Android Web

To render multiple columns, use the `numColumns` prop. Using this approach instead of a `flexWrap` layout can prevent conflicts with the item height logic.

More complex, selectable example below.

- By passing `extraData={selectedId}` to `FlatList` we make sure `FlatList` itself will re-render when the state changes. Without setting this prop, `FlatList` would not know it needs to re-render any items because it is a `PureComponent` and the prop comparison will not show any changes.
- `keyExtractor` tells the list to use the `id`s for the react keys instead of the default `key` property.

TypeScript JavaScript

flatlist-selectable Expo

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

type ItemData = {
  id: string;
  title: string;
};

const DATA: ItemData[] = [
  {
    id: 'bd7acbea-c1b1-46c2-aed5-3ad53abb28ba',
    title: 'First Item',
  },
  {
    id: '3ac68afc-c605-48d3-a4f8-fbd91aa97f63',
    title: 'Second Item',
  },
  {
    id: '58694a0f-3da1-471f-bd96-145571e29d72',
```

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

ScrollView Props

Inherits ScrollView Props, unless it is nested in another FlatList of same orientation.

Required **`renderItem`**

```
renderItem({
  item: ItemT,
  index: number,
  separators: {
    highlight: () => void;
    unhighlight: () => void;
    updateProps: (select: 'leading' | 'trailing', newProps: any) => void;
  }
}): JSX.Element;
```

Takes an item from `data` and renders it into the list.

Provides additional metadata like `index` if you need it, as well as a more generic `separators.updateProps` function which let you set whatever props you want to change the rendering of either the leading separator or trailing separator in case the more common `highlight` and `unhighlight` (which set the `highlighted: boolean` prop) are insufficient for your use case.

TYPE
function

- `item` (Object): The item from `data` being rendered.
- `index` (number): The index corresponding to this item in the `data` array.
- `separators` (Object)
  - `highlight` (Function)
  - `unhighlight` (Function)
  - `updateProps` (Function)
    - `select` (enum('leading', 'trailing'))
    - `newProps` (Object)

Example usage:

```
<FlatList
  ItemSeparatorComponent={
    Platform.OS !== 'android' &&
    (({highlighted}) => (
      <View
        style={[style.separator, highlighted && {marginLeft: 0}]}
      />
    ))
  }
  data={[{title: 'Title Text', key: 'item1'}]}
  renderItem={({item, index, separators}) => (
    <TouchableHighlight
      key={item.key}
      onPress={() => this._onPress(item)}
      onShowUnderlay={separators.highlight}
      onHideUnderlay={separators.unhighlight}>
      <View style={{backgroundColor: 'white'}}>
        <Text>{item.title}</Text>
      </View>
    </TouchableHighlight>
  )}
/>
```

Required **`data`**

An array (or array-like list) of items to render. Other data types can be used by targetting `VirtualizedList` directly.

TYPE
ArrayLike

`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

`ListFooterComponent`

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

TYPE
component, element

`ListFooterComponentStyle`

Styling for internal View for `ListFooterComponent`.

TYPE
View Style

`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

`columnWrapperStyle`

Optional custom style for multi-item rows generated when `numColumns > 1`.

TYPE
View Style

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

`getItemLayout` is an optional optimization that allows skipping the measurement of dynamic content if you know the size (height or width) of items ahead of time. `getItemLayout` is efficient if you have fixed size items, for example:

```
  getItemLayout={(data, index) => (
    {length: ITEM_HEIGHT, offset: ITEM_HEIGHT * index, index}
  )}
```

Adding `getItemLayout` can be a great performance boost for lists of several hundred items. Remember to include separator length (height or width) in your offset calculation if you specify `ItemSeparatorComponent`.

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.

TYPE	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`.

TYPE
boolean

`keyExtractor`

```
(item: ItemT, 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 `item.key`, then `item.id`, and then falls back to using the index, like React does.

TYPE
function

`numColumns`

Multiple columns can only be rendered with `horizontal={false}` and will zig-zag like a `flexWrap` layout. Items should all be the same height - masonry layouts are not supported.

TYPE
number

`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

`onViewableItemsChanged`

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

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

`progressViewOffset`

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

TYPE
number

`refreshing`

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

TYPE
boolean

`removeClippedSubviews`

This may improve scroll performance for large lists. On Android the default value is `true`.

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

TYPE
boolean

`viewabilityConfig`

See `ViewabilityHelper.js` for flow type and further documentation.

TYPE
ViewabilityConfig

`viewabilityConfig` takes a type `ViewabilityConfig` an object with following properties

PROPERTY	TYPE
minimumViewTime	number
viewAreaCoveragePercentThreshold	number
itemVisiblePercentThreshold	number
waitForInteraction	boolean

At least one of the `viewAreaCoveragePercentThreshold` or `itemVisiblePercentThreshold` is required. This needs to be done in the `constructor` to avoid following error (ref):

```
  Error: Changing viewabilityConfig on the fly is not supported
```

```
constructor (props) {
  super(props)

  this.viewabilityConfig = {
      waitForInteraction: true,
      viewAreaCoveragePercentThreshold: 95
```

```
    }
  }



  <FlatList
      viewabilityConfig={this.viewabilityConfig}
    ...
```

minimumViewTime

Minimum amount of time (in milliseconds) that an item must be physically viewable before the viewability callback will be fired. A high number means that scrolling through content without stopping will not mark the content as viewable.

viewAreaCoveragePercentThreshold

Percent of viewport that must be covered for a partially occluded item to count as "viewable", 0-100. Fully visible items are always considered viewable. A value of 0 means that a single pixel in the viewport makes the item viewable, and a value of 100 means that an item must be either entirely visible or cover the entire viewport to count as viewable.

itemVisiblePercentThreshold

Similar to `viewAreaCoveragePercentThreshold`, but considers the percent of the item that is visible, rather than the fraction of the viewable area it covers.

waitForInteraction

Nothing is considered viewable until the user scrolls or `recordInteraction` is called after render.

`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.

TYPE
array of ViewabilityConfigCallbackPair

Methods

`flashScrollIndicators()`

```
flashScrollIndicators();
```

Displays the scroll indicators momentarily.

`getNativeScrollRef()`

```
getNativeScrollRef(): React.ElementRef<typeof ScrollViewComponent>;
```

Provides a reference to the underlying scroll component

`getScrollResponder()`

```
getScrollResponder(): ScrollResponderMixin;
```

Provides a handle to the underlying scroll responder.

`getScrollableNode()`

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

Provides a handle to the underlying scroll node.

`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;
});
```

Scrolls to the item at the specified index such that it is positioned in the viewable area such that `viewPosition` 0 places it at the top, 1 at the bottom, and 0.5 centered in the middle.

> Note: Cannot scroll to locations outside the render window without specifying the `getItemLayout` 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`.
- 'index' (number) - The index to scroll to. Required.
- 'viewOffset' (number) - A fixed number of pixels to offset the final target position.
- '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.

`scrollToItem()`

```
scrollToItem(params: {
  animated?: ?boolean,
  item: Item,
  viewPosition?: number,
});
```

Requires linear scan through data - use `scrollToIndex` instead if possible.

> Note: Cannot scroll to locations outside the render window without specifying the `getItemLayout` 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`.
- 'item' (object) - The item to scroll to. Required.
- 'viewPosition' (number)

`scrollToOffset()`

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

Scroll to a specific content pixel offset in the list.

Parameters:

NAME	TYPE
params Required	object

Valid `params` keys are:

- 'offset' (number) - The offset to scroll to. In case of `horizontal` being true, the offset is the x-value, in any other case the offset is the y-value. Required.
- 'animated' (boolean) - Whether the list should do an animation while scrolling. Defaults to `true`.

Is this page useful?

Edit this page

*Last updated on **Aug 17, 2023***