Version: 3.x

useAnimatedScrollHandler

> INFO
>
> This page was ported from an old version of the documentation.
>
> As we're rewriting the documentation some of the pages might be a little outdated.

This is a convenience hook that returns an event handler reference which can be used with React Native's scrollable components.

Arguments

`scrollHandlerOrHandlersObject` **[object with worklets]**

Object containing any of the following keys: `onScroll`, `onBeginDrag`, `onEndDrag`, `onMomentumBegin`, `onMomentumEnd`. The values in the object should be individual worklets. Each of the worklet will be triggered when the corresponding event is dispatched on the connected Scrollable component.

Each of the event worklets will receive the following parameters when called:

- `event` [object] - event object carrying the information about the scroll. The payload can differ depending on the type of the event (`onScroll`, `onBeginDrag`, etc.). Please consult **React Native's ScrollView documentation** to learn about scroll event structure.
- `context` [object] - plain JS object that can be used to store some state. This object will persist in between scroll event occurrences and you can read and write any data to it. When there are several event handlers provided in a form of an object of worklets, the `context` object will be shared in between the worklets allowing them to communicate with each other.

`dependencies` **[Array]**

Optional array of values which changes cause this hook to receive updated values during rerender of the wrapping component. This matters when, for instance, worklet uses values dependent on the component's state.

Example:

```
const App = () => {
  const [state, setState] = useState(0);
  const sv = useSharedValue(0);

  const handler = useAnimatedScrollHandler(
    {
      onEndDrag: (e) => {
        sv.value = state;
      },
    },
    dependencies
  );
  //...
  return <></>;
};
```

`dependencies` here may be:

- `undefined` (argument skipped) - worklet will be rebuilt if there is any change in any of the callbacks' bodies or any values from their closure(variables from outer scope used in worklet),
- empty array(`[]`) - worklet will be rebuilt only if any of the callbacks' bodies changes,
- array of values(`[val1, val2, ..., valN]`) - worklet will be rebuilt if there is any change in any of the callbacks bodies or in any values from the given array.

Returns

The hook returns a handler object that can be hooked into a scrollable container. Note that in order for the handler to be properly triggered, you should use containers that are wrapped with `Animated` (e.g. `Animated.ScrollView` and not just `ScrollView`). The handler should be passed under `onScroll` parameter regardless of whether it is configured to receive only scroll or also momentum or drag events.

Example

In the below example we define a scroll handler by passing a single worklet handler. The worklet handler is triggered for each of the scroll events dispatched to the `Animated.ScrollView` component to which we attach the handler.

```
import Animated, {
  useSharedValue,
  useAnimatedStyle,
  useAnimatedScrollHandler,
} from 'react-native-reanimated';

function ScrollExample() {
  const translationY = useSharedValue(0);

  const scrollHandler = useAnimatedScrollHandler((event) => {
    translationY.value = event.contentOffset.y;
  });

  const stylez = useAnimatedStyle(() => {
    return {
      transform: [
        {
          translateY: translationY.value,
        },
      ],
    };
  });

  return (
    <View style={styles.container}>
      <Animated.View style={[styles.box, stylez]} />
      <Animated.ScrollView style={styles.scroll} onScroll={scrollHandler}>
        <Content />
      </Animated.ScrollView>
    </View>
  );
}
```

If we are interested in receiving drag or momentum events instead of passing a single worklet object we can pass an object of worklets. Below for convenience, we only show how the `scrollHandler` should be defined in such a case. The place where we attach handler to a scrollable component remains unchanged regardless of the event types we want to receive:

```
const isScrolling = useSharedValue(false);

const scrollHandler = useAnimatedScrollHandler({
  onScroll: (event) => {
    translationY.value = event.contentOffset.y;
  },
  onBeginDrag: (e) => {
    isScrolling.value = true;
  },
  onEndDrag: (e) => {
    isScrolling.value = false;
  },
});
```

Edit this page