

Version: 3.x

useAnimatedStyle

`useAnimatedStyle` lets you create a styles object, similar to `StyleSheet` styles, which can be animated using [shared values](#).

Styles defined using `useAnimatedStyle` have to be passed to `style` property of an [Animated component](#). Styles are automatically updated whenever an associated shared value or React state changes.

For animating properties use [useAnimatedProps](#) instead.

Reference

```
import { useAnimatedStyle } from 'react-native-reanimated';

function App() {
  const animatedStyles = useAnimatedStyle(() => {
    return {
      opacity: sv.value ? 1 : 0,
    };
  });

  return <Animated.View style={[styles.box, animatedStyles]} />;
}
```

▼ Type definitions

Arguments

updater

A function returning an object with style properties you want to animate. You can animate any style property available in React Native.

dependencies Optional

An optional array of dependencies.

Only relevant when using Reanimated [without the Babel plugin on the Web](#).

Returns

`useAnimatedStyle` returns an animated style object which has to be passed to the `style` property of an [Animated component](#) that you want to animate.

Example

Preview Code



```
import React from 'react';
import { Button, View, StyleSheet } from 'react-native';
import Animated, {
  useSharedValue,
  withSpring,
  useAnimatedStyle,
} from 'react-native-reanimated';

export default function App() {
  const translateX = useSharedValue(0);

  const handlePress = () => {
    translateX.value += 50;
  };
}
```

Remarks

- The callback passed to the `useAnimatedStyle` is first run on the [JavaScript thread](#) and immediately after on the [UI thread](#). Some functions like [measure](#) and [scrollTo](#) have

implementations only on the UI thread. To safely use them inside of `useAnimatedStyle` callback add a check to your code:

```
function App() {
  const animatedStyles = useAnimatedStyle(() => {
    if (!_WORKLET) {
      return { offset: 0 };
    }

    // safely use measure or scrollTo functions
  });
}
```

- Make sure not to mutate shared values in `useAnimatedStyle` 's callback. In many cases this can cause an infinite loop.
- Only define the dynamic part of your styles with `useAnimatedStyle` and keep the static ones separately using `StyleSheet` API or (if you really have to) with inline styles. That way you avoid lots of unnecessary style recalculations. Static and dynamic styles can be easily merged using the `[]` syntax:




```
function App() {
  const animatedStyles = useAnimatedStyle(() => ({
    offset: sv.value,
  }));

  return <Animated.View style={[styles.box, animatedStyles]} />;
}

const styles = StyleSheet.create({
  box: {
    height: 120,
    width: 120,
    backgroundColor: '#b58df1',
  },
});
```

- You can share animated props between components to avoid code duplication.

Platform compatibility

Android	iOS	Web
		

 [Edit this page](#)