

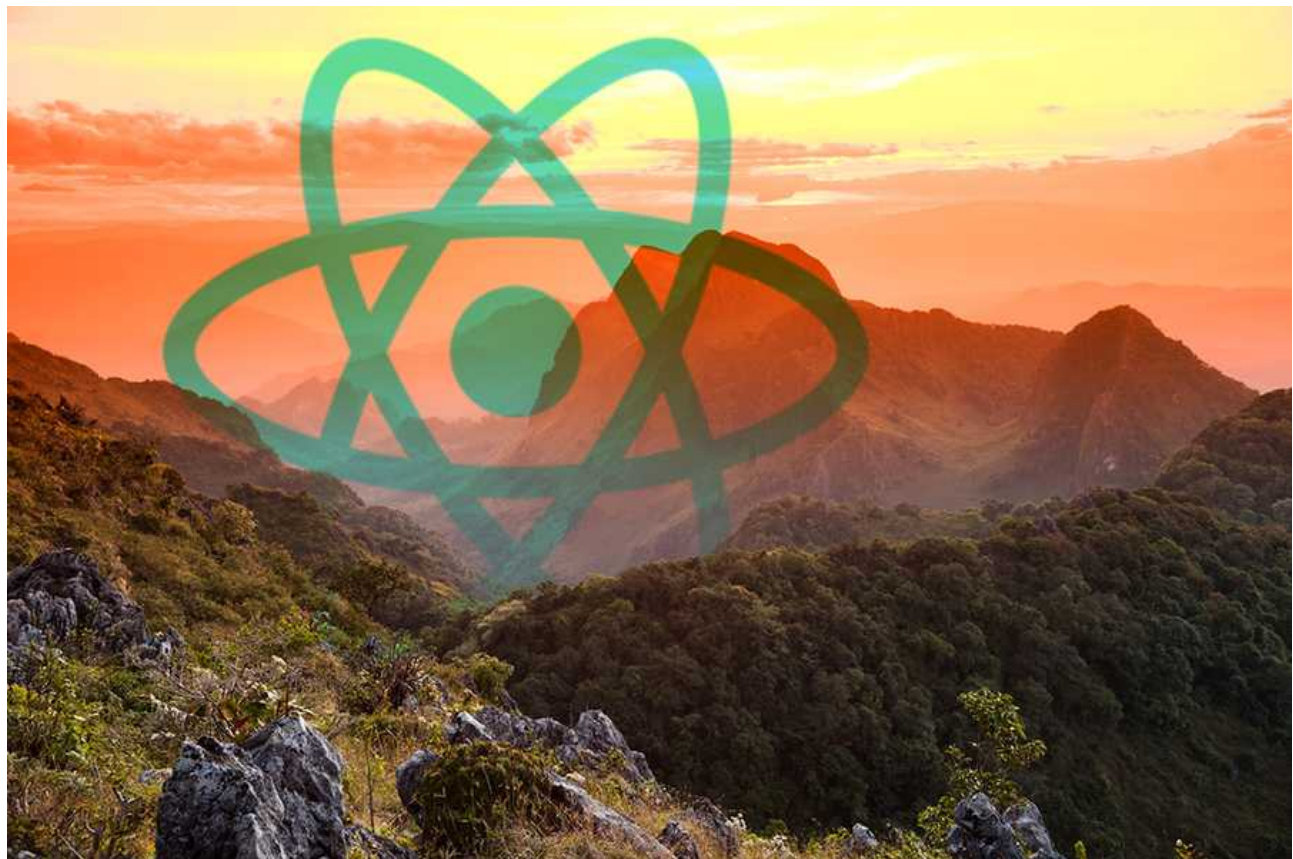
React useState with Callback

AUGUST 30, 2020 BY ROBIN WIERUCH - [EDIT THIS POST](#)

 Follow on Twitter

17k

 Follow on Facebook



f

t

in

If you have started to use [React's useState hook](#) for your application, you may be missing a [callback function](#), because only the initial state can be passed to the hook. In React class components, the `setState` method offers an optional second argument to pass a callback function. However, this second argument isn't available for React's `useState` hook. If you are moving from [React class components to function components](#), this may be a concern for you. In this tutorial, I want to explain you how to implement it.

Note: If you are just looking for an out of the box solution, check out this [custom useState hook with callback function](#). That's what you are going to implement in this tutorial anyway. I will show how this works below as well.

REACT USESTATE CALLBACK

In [React Function Components with Hooks](#), you can implement a callback function for anything using the `useEffect` hook. For instance, if you want to have a callback function for a state change, you can make the `useEffect` hook dependent on this state:

```
import React from 'react';

const App = () => {
  const [count, setCount] = React.useState(0);

  React.useEffect(() => {
    if (count > 1) {
      console.log('Threshold of over 1 reached.');
```





```
    } else {
      console.log('No threshold reached.');
```

```
    }
  }, [count]);

  const handleClick = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <p>{count}</p>

      <button type="button" onClick={handleClick}>
        Increase
      </button>
    </div>
  );
};

export default App;
```

The function you pass to the `useEffect` hook is your callback function which runs after the provided state changes from the `useState` hook's second argument. If you perform changes in this callback function that should be reflected in your component's rendered output, you may want to use `useLayoutEffect` instead of `useEffect`.

If you are looking for an out of the box solution, check out [this custom hook](#) that works like `useState` but accepts as second parameter as callback function:

```
import React from 'react';
```

```
import useStateWithCallback from 'use-state-with-callback';

const App = () => {
  const [count, setCount] = useStateWithCallback(0, count => {
    if (count > 1) {
      console.log('Threshold of over 1 reached.');
```



```
    } else {
      console.log('No threshold reached.');
```



```
    }
  });

  const handleClick = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <p>{count}</p>

      <button type="button" onClick={handleClick}>
        Increase
      </button>
    </div>
  );
};

export default App;
```



The custom hook can be installed via `npm install use-state-with-callback`. At the end, the React team decided consciously against a second argument for `useState` for providing a callback function, because `useEffect` or `useLayoutEffect` can be used instead. However, if you are lazy, you can use the custom hook to get the same effect as `setState` from React class components.

REACT USESTATE WITH LAZY CALLBACK FUNCTION

If you want to have a lazy executable function instead, you can use the library as well:

```
import React from 'react';
import { useStateWithCallbackLazy } from 'use-state-with-callback';

const App = () => {
  const [count, setCount] = useStateWithCallbackLazy(0);

  const handleClick = () => {
```

```
    setCount(count + 1, (currentCount) => {
      if (currentCount > 1) {
        console.log('Threshold of over 1 reached.');
```

```
      } else {
        console.log('No threshold reached.');
```

```
      }
    });
  });
};

return (
  <div>
    <p>{count}</p>

    <button type="button" onClick={handleClick}>
      Increase
    </button>
  </div>
);
};

export default App;
```



This way, you can decide when to use the second argument for the callback function and you can decide for each case what the callback function should do specifically.



Show Comments



KEEP READING ABOUT [JAVASCRIPT](#) >

CALLBACK FUNCTIONS IN JAVASCRIPT

Functions are first-class citizens in JavaScript. That's why you will early on hear about callback functions in JavaScript, which are a super powerful asset when writing JavaScript code. Here I want...

REACT HOOKS MIGRATION

React Hooks were introduced to React to make state and side-effects available in React Function Components. Before it was only possible to have these in React Class Components; but since React's way...



THE ROAD TO REACT



Learn React by building real world applications. No setup configuration. No tooling. Plain React in 200+ pages of learning material. Learn React like



50.000+ readers.

GET THE BOOK >

Get it on Amazon.

TAKE PART

NEVER MISS AN ARTICLE ABOUT WEB DEVELOPMENT AND JAVASCRIPT.

- ✓ Join 50.000+ Developers
- ✓ Learn Web Development with JavaScript
- ✓ Tips and Tricks
- ✓ Access Tutorials, eBooks and Courses
- ✓ Personal Development as a Software Engineer

Your email address

SUBSCRIBE

[View our Privacy Policy.](#)



PORTFOLIO

[Online Courses](#)

[Open Source](#)

[Tutorials](#)

ABOUT

[About me](#)

[What I use](#)

[How to work with me](#)

[How to support me](#)

© Robin Wieruch



[Contact Me](#) [Privacy & Terms](#)