Contents

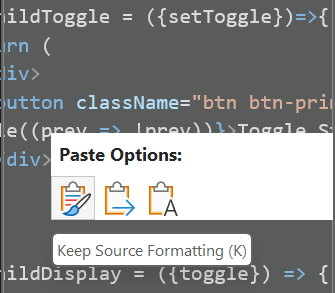
[What’s a Hook? 2](#_Toc192232942)

[useState 2](#_Toc192232943)

[useEffect 3](#_Toc192232944)

[useContext 4](#_Toc192232945)

Pasting in word with formatted code from vs code (right click in work and click on paste brush icon)



# What’s a Hook?

**What is a Hook?** A Hook is a special function that lets you “hook into” React features. For example, useState is a Hook that lets you add React state to function components.

## useState

The React useState Hook allows us to track state in a function component.

State generally refers to data or properties that need to be tracking in an application.

import React, { useState } from "react";

const UseStateExample = () => {

  //using vanila javascript, the dom value of cnt wont update on UI

  let cnt = 0;

  const incrementCnt = () => {

    cnt = cnt + 1;

    console.log(`cnt ${cnt}`);

  };

  // using useState,  the dom value of count  update on UI

  const [count, setCount] = useState(0);

  const increment = () => {

    setCount((previous) => previous + 1);

    console.log(`count ${count}`);

  };

  return (

    <div>

      <h1>{cnt}</h1>

      <button className="btn btn-primary" onClick={incrementCnt}>

        Increment Using vanila Javascript

      </button>

      <h1>{count}</h1>

      <button className="btn btn-primary" onClick={increment}>

        Increment Using React State

      </button>

    </div>

  );

};

export default UseStateExample;

## useEffect

If we want to run a function, every time a value changes, we can use useEffect hook.

It lets you run the code after the component renders, on a set of circumstances.

If I want to run a function every time state changes the value, we can use useEffect

Example of usage: We can use this for fetching data from an API, immediately when page renders

console.log("you will see useEffect running twice on page load, this is only for development  build, but for production build you see only once");

  useEffect(() => {

    console.log("runs every time  state changes");

  });

  useEffect(() => {

    console.log("runs only once during initial page render");

  }, []);

  let [value, setValue] = useState("N");

  useEffect(() => {

    console.log("runs on initial render and also if value changes");

  }, [value]);

## useContext

<https://medium.com/zestgeek/mastering-reacts-usecontext-hook-simplifying-state-management-65894e6dc431>

In the world of React development, efficient state management is key to building robust and scalable applications. While there are various state management solutions available, React provides its own built-in mechanism called the useContext hook, which offers a straightforward and elegant way to manage state across components

**Understanding the useContext Hook**

The useContext hook is a part of React’s hooks API introduced in React 16.8. It allows components to consume state or context without the need for prop drilling, which can lead to cleaner and more maintainable code. Context provides a way to share values like themes, user authentication status, or preferred language across the component tree without having to pass props down manually at every level.

Problem: If we want to pass parent values to child, then we need to pass them as part of props. Imagine we have 4 levels of parent -> child, then we need to pass 3 levels down

import UseContextExample from "./hooks/UseContextExample";

function App() {

  return (

    <div className="container-fluid mt-5 mx-5">

      <UseContextExample />

    </div>

  );

}

export default App;

import React, { useState } from 'react'

const UseContextExample = () => {

  const[toggle,setToggle]=useState(false);

    return (

    <div>

        <h1>Parent Component </h1>

        <ChildToggle setToggle={setToggle}/>

        <ChildDisplay toggle={toggle}/>

    </div>

  )

}

const ChildToggle = ({setToggle})=>{

    return (

      <div>

      <button className="btn btn-primary" type="button"   onClick={() => setToggle((prev => !prev))}>Toggle State</button>

      </div>

    )

}

const ChildDisplay = ({toggle}) => {

    return (

      <div>Current State is {toggle ?"ON":"OFF"}</div>

    )

}

export default UseContextExample;

Using Context

import React, { createContext, useState,useContext } from "react";

//Technically, all this components will be there in seperate files, but here we are storing in one file.

// We should import GlobalStateContext in each component

export const GlobalStateContext = createContext();

const UseContextExample = () => {

  const [toggle, setToggle] = useState(false);

  return (

    <GlobalStateContext.Provider value={{toggle,setToggle}}>

      <div>

        <h1>Parent Component</h1>

        <ChildToggle/>

        <ChildDisplay/>

      </div>

    </GlobalStateContext.Provider>

  );

};

const ChildToggle = () => {

    const {setToggle}=useContext(GlobalStateContext)

    return (

    <div>

      <button

        className="btn btn-primary"

        type="button"

        onClick={() => setToggle((prev) => !prev)}

      >

        Toggle State

      </button>

    </div>

  );

};

const ChildDisplay = () => {

    const {toggle}=useContext(GlobalStateContext)

  return <div>Current State is {toggle ? "ON" : "OFF"}</div>;

};

export default UseContextExample;

**Conclusion**

The useContext hook in React is a versatile tool for managing global state and context in your applications. Whether you’re handling user authentication, language preferences, themes, or any other shared data, useContext simplifies your code by eliminating prop drilling and providing a cleaner way to access context values. Incorporate useContext into your React projects to enhance maintainability, readability, and scalability, making your development process more efficient and enjoyable.