**React Custom Hooks**

**React Custom Hooks** are JavaScript functions starting with ‘**use’** containing reusable logic.

**Table of Content**

* [What is React Custom Hook ?](https://www.geeksforgeeks.org/reactjs-custom-hooks/#what-is-react-custom-hook-)
* [Need for Custom Hooks](https://www.geeksforgeeks.org/reactjs-custom-hooks/#need-for-custom-hooks)
* [Building a custom hook](https://www.geeksforgeeks.org/reactjs-custom-hooks/#building-a-custom-hook)
* [Rules to Build React Custom Hooks](https://www.geeksforgeeks.org/reactjs-custom-hooks/#rules-to-build-react-custom-hooks)

**What is React Custom Hook ?**

We know that hooks like useState, and useEffect are reusable components. Sometimes we make components that we have to reuse again and again in the application. In this case, we can convert the component to hooks by extracting logic from it.

**Need for Custom Hooks**

The main reason why you should be using Custom hooks is to maintain the concept of [DRY](https://www.geeksforgeeks.org/7-common-programming-principles-that-every-developer-must-follow/)(Don’t Repeat Yourself) in your React apps.

For example, suppose you have some logic that makes use of some built-in hooks and you need to use the logic in multiple functional components. So, the easier way to do it is to create a separate function that wraps the logic inside it and then call it from those components. Here, the separate function you created is the custom hook.

**Building a custom hook**

Creating a custom hook is the same as creating a JavaScript function whose name starts with “use”. It can use other hooks inside it, return anything you want it to return, take anything as parameters.

**Note:** It is important to name your custom hooks starting with “use”, because without it React can’t realize that it is a custom hook and therefore can’t apply the rules of hooks to it. So, you should name it starting with “use”.

**Step 1: Create a React Component**

Let us first create a component where we are not creating custom hook

* Javascript

|  |
| --- |
| // Filename - src/components/FirstComponent.js    import React , {useState ,useEffect} from "react";    **function** FirstComponent(props){        const [counter , setCounter] = useState(initializer);        // Increases the value of counter by 1  **function** resetCounter(){          setCounter(counter + 1):      }        useEffect(() => {          // Some logic          console.log(counter);      } , [counter]);        const clickedButton = resetCounter;    **return** (          <div>              <h1> This is the First Component</h1>              <button onClick={clickedButton}>                 Click here!              </button>          </div>      );  }    export **default** FirstComponent; |

Suppose we have to use this counter in multiple components then we would require a custom hook that can perfrom the same function multiple times.

**Step 2: Creating a custom hook**

Create the custom hook as shown in the example below

* Javascript

|  |
| --- |
| // Filename- src/components/useCustomHook.js    import {useState , useEffect} from "react";    // Remember to start the name of your custom hook with "use"  **function** useCustomHook(initializer , componentName){      const [counter , setCounter] = useState(initializer);        // Increases the value of counter by 1  **function** resetCounter(){          setCounter(counter + 1);      }          useEffect(() => {          // Some logic that will be used in multiple components          console.log("The button of the "          + componentName + " is clicked "          + counter + " times.");      } , [counter , componentName]);        // Calls the useEffect hook if the counter updates  **return** resetCounter;  }    export **default** useCustomHook; |

**Step 3: Using the custom hook in components**

To use the custom hook in your components just import the “useCustomHook” function from “useCustomHook.js” file in the “src” folder.

* **src/components/FirstComponent.js:**This component will import customHook
* **src/components/SecondComponent.js:**This component will also import customHook
* **src/App.js:**We will render the components in this file
* Javascript
* Javascript
* Javascript

|  |
| --- |
| // Filename - First Component    import React from "react";    // importing the custom hook  import useCustomHook from "./useCustomHook";    **function** FirstComponent(props){        // ClickedButton = resetCounter;      const clickedButton = useCustomHook(0 , "FirstComponent");    **return** (          <div>              <h1> This is the First Component</h1>              <button onClick={clickedButton}>                    Click here!              </button>          </div>      );  }    export **default** FirstComponent; |

**Output:**



**Explanation:**

Note that the useEfffect is called after the first render even though we declare the dependency array. That means the callback function passed to the useEffect hook is executed after the first render and when the variables in the dependency array get updated. There is no straight way to prevent this first execution.

Note that both the counters (defined in “usecustomHook”) of the two components are different. The two components use two different “counter” state variable(see “useCustomhook.js”), they don’t share the states. Therefore, in this React app, each component has its own “counter” state variable.

Similarly, each component has its own useEffect and they are executed independently of each other. If the counter of the FirstComponent is updated, the useEffect of the FirstComponent is called, similarly if the counter of the SecondComponent is updated, the useEffect of the SecondComponent is called independently of the FirstComponent (See the above output).

**Rules to Build React Custom Hooks**

* Custom hooks must start with “use” to follow the naming convention and signal their potential use of React hooks.
* Typically, custom hooks encapsulate stateful logic, allowing its reuse across multiple components seamlessly.
* Custom hooks are not components; they are functions that contain reusable logic, making them versatile and easy to integrate.
* When using other hooks, ensure that custom hooks adhere to the rules of React hooks, maintaining order and dependency arrays.
* Custom hooks should encapsulate a specific concern, promoting modularity and improving the organization of code in React applications.