**useReducer**

`useReducer` is a hook in React that is used for managing complex state logic in functional components. It is an alternative to the more common `useState` hook and is particularly useful when you have state transitions that depend on the previous state.

The `useReducer` hook takes two arguments: a reducer function and an initial state. The reducer function takes two parameters: the current state and an action, and returns the new state based on the action. Here's the basic syntax:

```jsx

import React, { useReducer } from 'react';

// Reducer function

const reducer = (state, action) => {

switch (action.type) {

case 'INCREMENT':

return { count: state.count + 1 };

case 'DECREMENT':

return { count: state.count - 1 };

default:

return state;

}

};

// Initial state

const initialState = { count: 0 };

// Component using useReducer

const Counter = () => {

const [state, dispatch] = useReducer(reducer, initialState);

const handleIncrement = () => {

dispatch({ type: 'INCREMENT' });

};

const handleDecrement = () => {

dispatch({ type: 'DECREMENT' });

};

return (

<div>

<p>Count: {state.count}</p>

<button onClick={handleIncrement}>Increment</button>

<button onClick={handleDecrement}>Decrement</button>

</div>

);

};

```

In the example above, we have a simple counter component using `useReducer`. When the "Increment" button is clicked, the `handleIncrement` function is called, which dispatches an action of type 'INCREMENT' to the reducer. The reducer then updates the state by incrementing the count. Similarly, when the "Decrement" button is clicked, the `handleDecrement` function dispatches an action of type 'DECREMENT' to decrement the count.

The benefit of using `useReducer` over `useState` becomes more apparent when you have more complex state transitions and logic in your application. It allows you to centralize state management in a single reducer function, making it easier to reason about and test your code.