# Refactor useState() hooks to useReducer()

**useState()** : This is one of the basic hook which can be used to change the state by calling a

setState(newState) method.

const[state,setState]=useState(initialState);

When called first time this method returns the initialState and then updates the state to the value returned by the setState function. The setState function inputs the newState, which is the updated version of state as requested by the user.

**useReducer()** :This is an alternative method for the useState() method (discussed above). The syntax for above method is shown below:

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

The init parameter is not necessarily important. It takes reducer() method as its input with the initialArg for first ever call, and returns currentState along with dispatch method which can be used to assign action type in reducer method. Here the reducer method is used to change the state of the component.

Now since both these methods are used to change the state of the components some conflicts are there which results in us shifting to use either of the two.

**WHY SHOULD WE SHIFT TO useReducer() HOOK?**

Basically when dealing with small projects we are not finding it very difficult to handle state and calls to setState function but when we switch ourselves to more big projects then we find it difficult to traceback from where changes in states had started and also calling many setState method results in tracing callbacks which is difficult. So we are in need of such a method which can be passed easily to the next state without much callbacks. Also such a method should be a pure one, since while dealing with big projects, there are chances to have complex objects with many primitive values and there may be chances that at any time the type returned by method differs. This is not the case with pure methods.

All these features are provided by useReducer() hook which is an extra edge over useState() hook. So we can conclude that this method can give us more control over state management than useState().

**BENEFITS?**

1. useReducer() hook lets us optimize the performance for the components that triggers deep updates. Since it is easier for us to pass reducer function than to manage callbacks in setState method.
2. This can also help to make testing tasks easier since there are no side effects produced because the reducer method is a pure function.
3. It makes implementation of complex logic appear easy with if-else blocks or switch statements.
4. It helps in preventing state collapsing.

**DISADVANTAGES**

1. Since this hook is not any basic or fundamental one, so it would be difficult for beginners to make up with this hook but those who are familiar with redux can understand its concept easily

**WHEN TO IMPLEMENT THIS useReducer() HOOK?**

1. When in the component we have objects or arrays as state, then these hooks are beneficial.
2. When we have complex state transitions or complex logic which can suitably be managed by reducer function.
3. Different primitive values tied within one object. This case can be handled more easily by reducer function since it is a pure one.