Controlled Components in React

In React, a controlled component is a component where form elements derive their value from a React state.

When a component is controlled, the value of form elements is stored in a state, and any changes made to the value are immediately reflected in the state.

To create a controlled component, you need to use the value prop to set the value of form elements and the onChange event to handle changes made to the value.

The value prop sets the initial value of a form element, while the onChange event is triggered whenever the value of a form element changes. Inside the onChange event, you need to update the state with the new value using a state update function.

Changing the state Object:

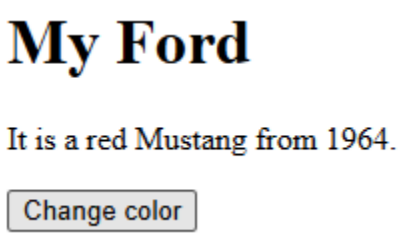
To change a value in the state object, use the **this.setState()** method.

When a value in the state object changes, the component will re-render, meaning that the output will change according to the new value(s).

```
class Car extends React.Component {
 constructor(props) {
  super(props);
  this.state = {
   brand: "Ford",
   model: "Mustang",
   color: "red",
   year: 1964
  };
 }
 changeColor = () => {
  this.setState({color: "blue"});
 }
 render() {
  return (
   <div>
    <h1>My {this.state.brand}</h1>
    <p>
     It is a {this.state.color}
     {this.state.model}
```

```
      from {this.state.year}.
     </p>
     <button
      type="button"
      onClick={this.changeColor}
     >Change color</button>
    </div>
   );
  }
}
```

this.state.*propertyname*



Always use the setState() method to change the state object, it will ensure that the component knows its been updated and calls the render().

Another example:

```
import { useState, useEffect } from 'react';


function ExampleComponent() {
  const [count, setCount] = useState(0);
```

```
 useEffect(() => {
  document.title = `Count: ${count}`;
 }, [count]);

 return (
  <div>
   <p>Count: {count}</p>
   <button onClick={() => setCount(count + 1)}>Increment</button>
  </div>
 );
}
export default ExampleComponent;
```

Mounting:

componentDidMount

The componentDidMount() method is called after the component is rendered.

This is where you run statements that requires that the component is already placed in the DOM.

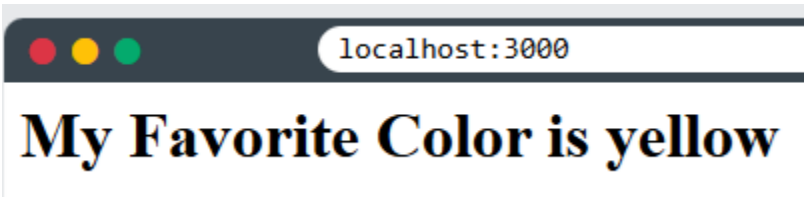
```
import React from 'react';
import ReactDOM from 'react-dom/client';

class Header extends React.Component {
 constructor(props) {
  super(props);
  this.state = {favoritecolor: "red"};
 }
```

```
  componentDidMount() {
    setTimeout(() => {
      this.setState({favoritecolor: "yellow"})
    }, 1000)
  }
  render() {
    return (
      <h1>My Favorite Color is {this.state.favoritecolor}</h1>
    );
  }
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Header />);
```

Output: After some seconds, color changed



Updating:

The next phase in the lifecycle is when a component is *updated*.

A component is updated whenever there is a change in the component's state or props.

componentDidUpdate

The componentDidUpdate method is called after the component is updated in the DOM.

The example below might seem complicated, but all it does is this:

When the component is *mounting* it is rendered with the favorite color "red".

When the component *has been mounted,* a timer changes the state, and the color becomes "yellow".

This action triggers the *update* phase, and since this component has a componentDidUpdate method, this method is executed and writes a message in the empty DIV element:

```
import React from 'react';
import ReactDOM from 'react-dom/client';

class Header extends React.Component {
 constructor(props) {
  super(props);
  this.state = {favoritecolor: "red"};
 }
 componentDidMount() {
  setTimeout(() => {
   this.setState({favoritecolor: "yellow"})
  }, 1000)
 }
 componentDidUpdate() {
  document.getElementById("mydiv").innerHTML =
  "The updated favorite is " + this.state.favoritecolor;
 }
 render() {
  return (
   <div>
   <h1>My Favorite Color is {this.state.favoritecolor}</h1>
```

```
    <div id="mydiv"></div>
    </div>
  );
 }
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Header />);
```

These are optional methods, the only required method is render().