React supports two types of components: controlled components and uncontrolled components.

*In most cases, we recommend using*[*controlled components*](https://reactjs.org/docs/forms.html#controlled-components)*to implement forms. In a controlled component, form data is handled by a React component. The alternative is uncontrolled components, where form data is handled by the DOM itself.*

**Controlled Components**

In a controlled component, the form data is handled by the state *within the component*. The state within the component serves as *“the single source of truth”* for the input elements that are rendered by the component.

Let’s look at a code example.

Here we have a simple component that renders one textbox on the page and echos back whatever the user types in the textbox. Here it is in use:

The controlled component in action!

If we look at the code for this component, on line 4 we create the state object. It holds a single property called *message*. This is where the value that is entered into the textbox is stored.

In order to store the value, we need an event to be fired when the user types in the textbox. How do we do that?

If you look at lines 19 and 20 of the code, you can see that:

1. The textbox has a value attribute bound to the message property in the state.
2. We have an onChange event handler declared.

***These 2 points tell you that this is a controlled component.***

***We don’t need a form element on the page for the component to be a controlled component.***

When changes are made to any of the input elements that have an *event handler*, the handler is fired.

The handler calls setState() as you can see in line 9 above. This updates the state within the component.

You should never set the state directly like this:

this.state.message = 'dont update state like this';

Updating the state in this way will not cause a re-render of the component and the changes made by the user will not be displayed in the UI.

When a state update occurs via setState(), it causes the component to re-render and the newly entered value is displayed in the element.

The data flow is uni-directional from the component state to the input element.

Working with controlled components can be a bit cumbersome. If there are a large number of input elements on the page each element requires setup with a *value* attribute and an *event handler*.

import React, { Component } from 'react';

class App extends Component {

state = {

message: ''

}

updateMessage = (newText) => {

console.log(newText);

this.setState(() => ({

message: newText

}));

}

render() {

return (

<div className="App">

<div className="container">

<input type="text"

placeholder="Your message here.."

value={this.state.message}

onChange={(event) => this.updateMessage(event.target.value)}

/>

<p>the message is: {this.state.message}</p>

</div>

</div>

);

}

}

export default App;

Example 2:

class NameForm extends React.Component {  
  constructor(props) {  
    super(props);  
   **this.state = {value: ''};**  
  
    this.handleChange = this.handleChange.bind(this);  
    this.handleSubmit = this.handleSubmit.bind(this);  
  }  
  
 **handleChange(event) {**  
 **this.setState({value: event.target.value});**  
 **}**  
  
  handleSubmit(event) {  
    alert('A name was submitted: ' + this.state.value);  
    event.preventDefault();  
  }  
  
  render() {  
    return (  
      <form onSubmit={this.handleSubmit}>  
        <label>  
          Name:  
 **<input type="text" value={this.state.value} onChange={this.handleChange} />**  
        </label>  
        <input type="submit" value="Submit" />  
      </form>  
    );  
  }  
}

**Uncontrolled Components**

Uncontrolled components act more like traditional HTML form elements. The data for each input element is stored in the DOM, not in the component. Instead of writing an event handler for all of your state updates, you use a ***ref*** to retrieve values from the DOM.

:

*Refs provide a way to access DOM nodes or React elements created in the render method.*

Let’s look at a code sample.

If we modify the controlled component that we were working with above to be an uncontrolled component, here’s what we get:

You can see that we have added a constructor on line 4 and in it we are setting up the event handler and creating a *ref* to *this.input.*

We reference *this.input* again on line 19 where we set it as the *ref* attribute on the input element. With those pieces in place, each time we type into the textbox, the value is echoed in the console.

import React, { Component } from 'react';

class App2 extends Component {

constructor(props){

super(props);

this.handleChange = this.handleChange.bind(this);

this.input = React.createRef();

}

handleChange = (newText) => {

console.log(newText);

}

render() {

return (

<div className="App2">

<div className="container">

<input type="text"

placeholder="Your message here.."

ref={this.input}

onChange={(event) => this.handleChange(event.target.value)}

/>

</div>

</div>

);

}

}

export default App2;

Ex1:

|  |
| --- |
| Import React, { useRef } from "react"; |
|  |  |
|  | export default function App() { |
|  | const inputRef = useRef(null); |
|  | const handleSubmitButton = () => { |
|  | alert(inputRef.current.value); |
|  | }; |
|  | return ( |
|  | <div className="App"> |
|  | <input type="text" ref={inputRef} /> |
|  | <input type="submit" value="submit" onClick={handleSubmitButton} /> |
|  | </div> |
|  | ); |
|  | } |

Ex2:

class NameForm extends React.Component {  
  constructor(props) {  
    super(props);  
    this.handleSubmit = this.handleSubmit.bind(this);  
  }  
  
 **handleSubmit(event) {**  
 **alert('A name was submitted: ' + this.input.value);**  
 **event.preventDefault();**  
 **}**  
  
  render() {  
    return (  
      **<form onSubmit={this.handleSubmit}>**  
        <label>  
          Name:  
          **<input type="text" ref={(input) => this.input = input} />**  
        </label>  
        <input type="submit" value="Submit" />  
      </form>  
    );  
  }  
}

A few key points regarding *refs:*

1. Refs are created using React.createRef().
2. Refs are attached to input elements using the *ref* attribute on the element in question.
3. Refs are often used as instance properties on a component. The ref is set in the constructor (as shown above) and the value is available throughout the component.
4. You cannot use the *ref* attribute ***on*** functional components because an instance is not created. Here is an example:

5. ***BUT***, You can use a *ref*attribute ***inside*** a functional component, like so:

[More from the React Docs:](https://reactjs.org/docs/refs-and-the-dom.html)

*There are a few good use cases for refs:*

*Managing focus, text selection, or media playback.*

*Triggering imperative animations.*

*Integrating with third-party DOM libraries.*

*Avoid using refs for anything that can be done declaratively.*

**Wrap Up and Key Takeaways**

1. Use controlled components whenever possible.
2. Controlled components do not require a form element in order to be considered a controlled component.
3. If a component has an input element that has a value attribute bound to state and an event handler to update said state, it is a controlled component.
4. For pages that have a large number of input elements, it can be cumbersome to work with controlled components.
5. Data flow is uni-directional in controlled components with the state within the component acting as the single source of truth.
6. All state changes within a controlled component should be made via the setState function.
7. Uncontrolled components store their data in the DOM like a traditional HTML input element.
8. React.createRef() is used to create instance variables within uncontrolled component constructors. These variables are then associated with input elements via the *ref* attribute.
9. Refs cannot be used ***on*** functional components as there is no instance.
10. Refs can be used ***inside*** functional components.