**React Documents**

**How the hell are you supposed to make forms?**

After all, forms are central to many web apps out there. And yet, form handling in React seems to be a bit of a… corner stone?

Fear no more. Let me show you the differences between the approaches, as well as when you should use each.

## The Uncontrolled

Uncontrolled inputs are like traditional HTML form inputs:

class Form extends Component {

render() {

return (

<div>

<input type="text" />

</div>

);

}

}

They remember what you typed. You can then get their value using [a ref](https://facebook.github.io/react/docs/refs-and-the-dom.html). For example, in onClick handler of a button:

class Form extends Component {

handleSubmitClick = () => {

const name = this.\_name.value;

// do something with `name`

}

render() {

return (

<div>

<input type="text" ref={input => this.\_name = input} />

<button onClick={this.handleSubmitClick}>Sign up</button>

</div>

);

}

}

In other words, you have to ‘pull’ the value from the field when you need it. This can happen when the form is submitted.

That is the simplest way to implement the form inputs. There certainly are valid cases for using it: in simple forms in the real world; and when learning React.

It’s not as powerful, though, so let’s see those controlled inputs next.

For what it’s worth, I’m giving away two chapters of my book about forms. Get it below if you want; and continue reading the article.

## **The Controlled**

A controlled input accepts its current value as a prop, as well as a callback to change that value. You could say it’s a more “React way” of approaching this (which doesn’t mean you should always use it).

<input value={someValue} onChange={handleChange} />

Which is fine and all… but the value of this input has to live in the state somewhere. Typically, the component that renders the input (aka the form component) saves that in its state:

class Form extends Component {

constructor() {

super();

this.state = {

name: '',

};

}

handleNameChange = (event) => {

this.setState({ name: event.target.value });

};

render() {

return (

<div>

<input

type="text"

value={this.state.name}

onChange={this.handleNameChange}

/>

</div>

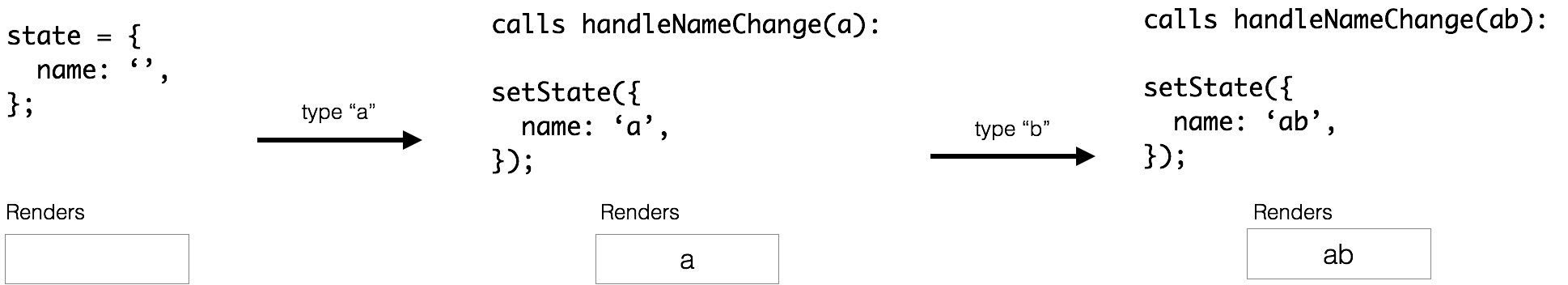
);

}

}

(Of course, it can be in the state of another component, or even in the separate state store, [like Redux](https://goshakkk.name/should-i-put-form-state-into-redux/).)

Every time you type a new character, handleNameChange is called. It takes in the new value of the input and sets it in the state.



* It starts out as an empty string — ''.
* You type a and handleNameChange gets an a and calls setState. The input is then re-rendered to have the value of a.
* You type b. handleNameChange gets the value of ab and sets that to the state. The input is re-rendered once more, now with value="ab".

This flow kind of ‘pushes’ the value changes to the form component**, so the Form component always has the current value of the input, without needing to ask for it explicitly.**

This means your data (state) and UI (inputs) are always in sync. The state gives the value to the input, and the input asks the Form to change the current value.

This also means that the form component can respond to input changes immediately; for example, by:

* in-place feedback, like validations
* disabling the button unless all fields have valid data
* enforcing a specific input format, like credit card numbers

But if you don’t need any of that and consider uncontrolled to be simpler, go for it.

**Resources:--https://goshakkk.name/controlled-vs-uncontrolled-inputs-react/**