**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.

To write an uncontrolled component, instead of writing an event handler for every state update, you can [use a ref](https://reactjs.org/docs/refs-and-the-dom.html) to get form values from the DOM.

For example, this code accepts a single name in an uncontrolled component:

class NameForm extends React.Component {

constructor(props) {

super(props);

this.handleSubmit = this.handleSubmit.bind(this);

this.input = React.createRef(); }

handleSubmit(event) {

alert('A name was submitted: ' + this.input.current.value); event.preventDefault();

}

render() {

return (

<form onSubmit={this.handleSubmit}>

<label>

Name:

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

<input type="submit" value="Submit" />

</form>

);

}

}

[**Try it on CodePen**](https://codepen.io/gaearon/pen/WooRWa?editors=0010)

Since an uncontrolled component keeps the source of truth in the DOM, it is sometimes easier to integrate React and non-React code when using uncontrolled components. It can also be slightly less code if you want to be quick and dirty. Otherwise, you should usually use controlled components.

If it’s still not clear which type of component you should use for a particular situation, you might find [this article on controlled versus uncontrolled inputs](https://goshakkk.name/controlled-vs-uncontrolled-inputs-react/) to be helpful.

**Default Values**

In the React rendering lifecycle, the value attribute on form elements will override the value in the DOM. With an uncontrolled component, you often want React to specify the initial value, but leave subsequent updates uncontrolled. To handle this case, you can specify a defaultValue attribute instead of value.

render() {

return (

<form onSubmit={this.handleSubmit}>

<label>

Name:

<input

defaultValue="Bob" type="text"

ref={this.input} />

</label>

<input type="submit" value="Submit" />

</form>

);

}

Likewise, <input type="checkbox"> and <input type="radio"> support defaultChecked, and <select> and <textarea> supports defaultValue.

**The file input Tag**

In HTML, an <input type="file"> lets the user choose one or more files from their device storage to be uploaded to a server or manipulated by JavaScript via the [File API](https://developer.mozilla.org/en-US/docs/Web/API/File/Using_files_from_web_applications).

<input type="file" />

In React, an <input type="file" /> is always an uncontrolled component because its value can only be set by a user, and not programmatically.

You should use the File API to interact with the files. The following example shows how to create a [ref to the DOM node](https://reactjs.org/docs/refs-and-the-dom.html) to access file(s) in a submit handler:

class FileInput extends React.Component {

constructor(props) {

super(props);

this.handleSubmit = this.handleSubmit.bind(this);

this.fileInput = React.createRef(); }

handleSubmit(event) {

event.preventDefault();

alert(

`Selected file - ${this.fileInput.current.files[0].name}` );

}

render() {

return (

<form onSubmit={this.handleSubmit}>

<label>

Upload file:

<input type="file" ref={this.fileInput} /> </label>

<br />

<button type="submit">Submit</button>

</form>

);

}

}

ReactDOM.render(

<FileInput />,

document.getElementById('root')

);