Controlled Components

What a controlled component is, binding an input to state with value and onChange, using the same pattern for checkboxes/selects -- with simple examples.

Intermediate 11 min

Last updated: August 18, 2026
Read online: https://www.learnforgex.com/en/topics/controlled-components

---

Controlled Components

In Events we saw `onChange`, and in State we saw `useState`, separately. This lesson brings the two together into the **controlled component** -- the fundamental pattern for writing forms in React.

What Is a Controlled Component?

Normally, an `<input>` keeps its own value inside itself (in the browser's DOM) -- React might not even be aware of it. In a **controlled component**, the input's value is DETERMINED by React's state instead: the input doesn't hold a value on its own, it always gets its value from state.

Binding an Input to State with value

The first step in making an input controlled is binding the `value` attribute to a state variable:

```
import { useState } from "react";

function ControlledInputExample() {
  const [text, setText] = useState("");

  function handleChange(event) {
    setText(event.target.value);
  }

  return (
    <div>
      {/* value={text} says that React's state should DECIDE the value the
          input displays on screen -- the input no longer holds its own
          value, it gets its value entirely from the `text` state. */}
      <input type="text" value={text} onChange={handleChange} />
      <p>You typed: {text}</p>
    </div>
  );
}
```

Since we wrote `value={text}`, the value shown in the input is ALWAYS whatever `text` currently holds in state. But that alone isn't enough -- something also needs to happen when the user types, which we'll see in the next section.

Updating State with onChange

`value` alone makes the input read-only -- the user can't type anything. To keep state up to date, you also need `onChange`:

```
import { useState } from "react";

function WhyControlledMattersExample() {
  const [text, setText] = useState("");

  // Because the value lives in state, we can instantly use it in ANOTHER
  // part of the screen as the user types each letter -- for example to
  // show a character count or an uppercase preview.
  return (
    <div>
      <input
        type="text"
        value={text}
        onChange={(event) => setText(event.target.value)}
        placeholder="Type something..."
      />
      <p>Character count: {text.length}</p>
      <p>Uppercase preview: {text.toUpperCase()}</p>
    </div>
  );
}
```

Every keystroke runs `onChange`, `setText` updates state, React re-renders, and the input's `value` reflects that new state. This loop (type → update state → re-render → show in input) is the heart of a controlled component. Since the value lives in state, we can also use it instantly somewhere else on screen at the same time (a character count, a preview, etc.).

Controlled Checkboxes and Selects

The same pattern isn't limited to text inputs -- checkboxes and selects can be controlled too:

```
import { useState } from "react";

function ControlledCheckboxExample() {
  const [isSubscribed, setIsSubscribed] = useState(false);

  // Checkboxes use checked instead of value -- the logic is the same:
  // React's state decides whether the checkbox is checked or not.
  function handleChange(event) {
    setIsSubscribed(event.target.checked);
  }

  return (
    <div>
      <label>
        <input type="checkbox" checked={isSubscribed} onChange={handleChange} />
        Subscribe to newsletter
      </label>
      <p>{isSubscribed ? "You are subscribed." : "You are not subscribed."}</p>
    </div>
  );
}
```

Checkboxes use `checked` instead of `value`, but the logic is the same: React's state decides whether it's checked.

```
import { useState } from "react";

function ControlledSelectExample() {
  const [language, setLanguage] = useState("tr");

  return (
    <div>
      {/* Selects are controlled with the same pattern: value + onChange. */}
      <select value={language} onChange={(event) => setLanguage(event.target.value)}>
        <option value="tr">Turkish</option>
        <option value="en">English</option>
      </select>
      <p>Selected language: {language}</p>
    </div>
  );
}
```

A `<select>` is controlled with `value` and `onChange` too, just like a text input.

Why Controlled Components?

One of the most concrete benefits of keeping a value in React's state is that you can easily do things with it -- like resetting it programmatically:

```
import { useState } from "react";

function ResettingControlledInputExample() {
  const [text, setText] = useState("");

  function handleClear() {
    // Because the value lives in state, clearing the input is as simple
    // as resetting the state -- there's no need to touch the actual DOM
    // element (e.g. via useRef or event.target.reset()).
    setText("");
  }

  return (
    <div>
      <input
        type="text"
        value={text}
        onChange={(event) => setText(event.target.value)}
      />
      <button onClick={handleClear}>Clear</button>
    </div>
  );
}
```

Since the value lives in state, clearing the input is as simple as `setText("")`. If we kept the value outside React (in the browser's own DOM), we'd need a DOM reference (`useRef`) or the form's own `reset()` method to do the same thing.

Summary and Glossary

A controlled component is a form element bound to state with `value` (or `checked` for checkboxes) and kept up to date with `onChange`. Since the value always lives in React's state, reading, validating, or resetting it becomes easier -- you don't have to rely on a value the input holds on its own.

Glossary

Controlled Component — A form element whose value is determined by React's state and updated with `onChange`.

Uncontrolled Component — A form element that holds its own value (in the browser's DOM), not tied to React state.