## What are refs in React again?

Before we start, let's do a quick recap of what refs (or references) in React are.

We will usually use references in React to store the reference to a DOM element, such as a button or an input field.

Having the element available to us as a variable in our code allows us to use browser APIs only available through the DOM.

const inputElement = document.getElementById('input-element-id');

inputElement.focus();

React makes these operations a bit easier by allowing us to store references to DOM elements like this:

const App = () => {

const inputRef = useRef(null);

return (

<>

<input

ref={inputRef}

value={name}

onChange={handleChange}

/>

<button

onClick={() => inputRef.current.focus()}

>

Focus

</button>

</>

);

};

Make sure that you pass down a variable that has been initialized with useRef (or createRef in class components).

If we now want to focus input element, we just need to call inputRef.current.focus().

**ReactJS Forwarding Refs**

When a child component needs to reference its parent component’s current node, the parent component needs a way to send down its ref to the child. The technique is called ref forwarding.

The forwardRef method in React allows parent components to move down (or “forward”) refs to their children.

ForwardRef gives a child component a reference to a DOM entity created by its parent component in React.

This helps the child to read and modify the element from any location where it is used.

**How does forwardRef work in React?**

In React, parent components typically use props to transfer data down to their children.

Consider you make a child component with a new set of props to change its behavior. We need a way to change the behavior of a child component without having to look for the state or re-rendering the component. We can do this by using refs.

We can access a DOM node that is represented by an element using refs.

As a result, we will make changes to it without affecting its state or having to re-render it.

**Syntax:**

React.forwardRef((props, ref) => {})

**Parameters:** It takes a function with props and ref arguments.

**Return Value:** This function returns a JSX Element.

## What is forwardRef in React?

[React forwardRef](https://reactjs.org/docs/forwarding-refs.html) is a method that allows parent components pass down (i.e., “forward”) refs to their children. Using forwardRef in React gives the child component a reference to a DOM element created by its parent component. This then allows the child to read and modify that element anywhere it is being used.

Typically in React, parent components pass data down to their children via props.

To change the behavior of a child component you render it with a new set of props. To modify a child component such that it exhibits a slightly different behavior, we need a way to make this change without reaching for the state or re-rendering the component.

We can achieve this by using refs. With refs, we have access to a DOM node that is represented by an element. As a result, we can modify it without touching its state or re-rendering it.

Because refs hold a reference to the DOM element itself, we can manipulate it with native JavaScript functions that are unavailable in the React library. For instance, we can initiate focus on input field when a button is clicked:

import \* as React from "react";

import ReactDOM from "react-dom";

export default function App() {

const ref = React.useRef();

function focus() {

ref.current.focus();

}

return (

<div className="App">

<input ref={ref} placeholder="my input" />

<button onClick={focus}>Focus</button>

</div>

);

}

Similarly, we could use JavaScript to achieve a similar effect, though it is not recommended, and even marked as a bad practice to access DOM directly when using React. The pure JavaScript equivalent to getting a reference would be:

document.getElementById('myInput).focus()

This information is important because forwardRef allows you to define internally what element the ref will point at.

### When to use refs in React

#### Managing focus, text selection, or media playback

## Working with refs in function components

In function components, we can’t simply make use of the function createRef as the same because it will create a new reference each time it gets called. We could opt to use effects and state to handle references but React provides a much easier way with useRef. useRef takes care of returning the same ref each time as on the initial rendering.

## Forwarding refs in React using forwardRef

When a child component needs to reference its parent component’s current node, the parent component needs a way to send down its ref to the child. The technique is called ref forwarding.

[Ref forwarding](https://reactjs.org/docs/forwarding-refs.html) is a technique for automatically passing a ref through a component to one of its children. It’s very useful when building reusable component libraries. forwardRef is a function used to pass the ref to a child component.

Let’s take an example of a new library with an InputText component that will provide a lot of functionality, though, for now, we’ll keep it simple:

const InputText = (props) => (

<input {...props} />

));

The InputText() component will tend to be used throughout the application in a similar manner as a regular DOM input, therefore accessing its DOM node may be unavoidable for managing focus, selection, or animations related to it.

In the example below, other components in the application have no access to the DOM input element generated by the InputText() component and is, thus, restricting some of the operations we have already foreseen we would need to meet our application requirements, such as controlling the focus of the input programmatically.

Here is when React.forwardRef enters to obtain a ref passed as props, and then forward it to the DOM input that it renders:

const InputText = React.forwardRef((props, ref) => (

 <input ref={ref} {...props} />

));

Now that our component supports forwardRef, let’s use it in the context of our application to build a button that will automatically focus the input when it’s clicked. The code looks as follows:

import \* as React from "react";

import ReactDOM from "react-dom";

const InputText = React.forwardRef((props, ref) => (

 <input ref={ref} {...props} />

));

export default function App() {

 const ref = React.useRef();

 function focus() {

   ref.current.focus();

 }

 return (

   <div className="App">

     <InputText ref={ref} placeholder="my input" />

     <button onClick={focus}>Focus</button>

   </div>

 );

}

const rootElement = document.getElementById("root");

ReactDOM.render(<App />, rootElement);

Here’s a clarification for the code above:

* We define a ref in the component that needs the ref and pass it to the button component
* React will pass the ref through and forward it down to <input ref={ref}> by specifying it as a JSX attribute
* When the ref is attached, ref.current will point to the <input> DOM node
* The second ref argument in the InputRef component only exists when you define a component with React.forwardRef call
* Regular function or class components don’t receive the ref argument, and ref is not available in props either
* Ref forwarding is not limited to DOM components. You can forward refs to class component instances, too

## Conclusion

Refs in React are a powerful tool that enable direct access to DOM nodes and thus open a whole new spectrum of methods and options to build more performant, feature-rich and clean components.

However, accessing DOM directly is often seen as a bad practice in React, and for a reason, when used improperly, it can turn all its benefits into real problems. As a general rule, it should be avoided and used only under very specific circumstances and with thorough examination.