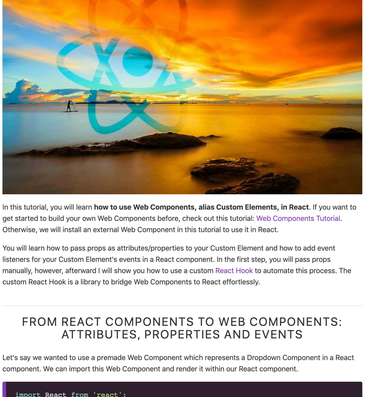
How to use Web Components in React

JUNE 12, 2019 BY ROBIN WIERUCH - EDIT THIS POST



In this tutorial, you will learn **how to use Web Components, alias Custom Elements, in React**. If you want to get started to build your own Web Components before, check out this tutorial: Web Components Tutorial. Otherwise, we will install an external Web Component in this tutorial to use it in React.

You will learn how to pass props as attributes/properties to your Custom Element and how to add event listeners for your Custom Element's events in a React component. In the first step, you will pass props manually, however, afterward I will show you how to use a custom React Hook to automate this process. The custom React Hook is a library to bridge Web Components to React effortlessly.

FROM REACT COMPONENTS TO WEB COMPONENTS: ATTRIBUTES, PROPERTIES AND EVENTS

Let's say we wanted to use a premade Web Component which represents a Dropdown Component in a React component. We can import this Web Component and render it within our React component.

```
import React from 'react';

import 'road-dropdown';

const Dropdown = props => (
  return <road-dropdown />;
);
```

You can install the Web Component via `npm install road-dropdown`. So far, the React Component is only rendering the Custom Element, but no props are passed to it. It isn't as simple as passing the props as attributes the following way, because you need to pass objects, arrays, and functions in a different way to Custom Elements.

```
import React from 'react';

import 'road-dropdown';

const Dropdown = props => {
  // doesn't work for objects/arrays/functions
  return <road-dropdown {...props} />;
};
```

Let's see how our React component would be used in our React application to get to know the props that we need to pass to our Web Component:

```
const props = {
  label: 'Label',
  option: 'option1',
  options: {
    option1: { label: 'Option 1' },
    option2: { label: 'Option 2' },
  },
  onChange: value => console.log(value),
};

return <Dropdown {...props} />;
```

Passing the `label` and `option` property unchanged as attributes to our Web Components is fine:

```
import React from 'react';

import 'road-dropdown';

const Dropdown = ({ label, option, options, onChange }) => {
  return (
    <road-dropdown
      label={label}
      option={option}
    />
  );
};
```

However, we need to do something about the `options` object and the `onChange` function, because they need to be adjusted and cannot be passed simply as attributes. Let's start with the object: Similar to arrays, the object needs to be passed as JSON formatted string to the Web Component instead of a JavaScript object:

```
import React from 'react';

import 'road-dropdown';

const Dropdown = ({ label, option, options, onChange }) => {
  return (
    <road-dropdown
      label={label}
      option={option}
      options={JSON.stringify(options)}
    />
  );
};
```

That's it for the object. Next, we need to take care about the function. Rather than passing it as attribute, we need to register an event listener for it. That's where we can use React's useLayoutEffect when the component is rendered for the first time:

```
import React from 'react';

import 'road-dropdown';

const Dropdown = ({ label, option, options, onChange }) => {
  const ref = React.createRef();

  React.useLayoutEffect(() => {
    const { current } = ref;

    current.addEventListener('onChange', customEvent =>
      onChange(customEvent.detail)
    );
  }, [ref]);

  return (
    <road-dropdown
      ref={ref}
      label={label}
      option={option}
      options={JSON.stringify(options)}
    />
  );
});
```

We are creating a reference for our Custom Element -- which is passed as ref attribute to the Custom Element -- to add an event listener in our React hook. Since we are dispatching a custom event from the custom dropdown element, we can register on this `onChange` event and propagate the information up with our own `onChange` handler from the props. A custom event comes with a detail property to send an optional payload with it.

Note: If you would have a built-in DOM event (e.g. `click` or `change` event) in your Web Component, you could also register to this event. However, this Web Component already dispatches a custom event which matches the naming convention of React components.

An improvement would be to extract the event listener callback function in order to remove the listener when the component unmounts.

```
import React from 'react';

import 'road-dropdown';

const Dropdown = ({ label, option, options, onChange }) => {
  const ref = React.createRef();

  React.useLayoutEffect(() => {
    const handleChange = customEvent => onChange(customEvent.detail);

    const { current } = ref;

    current.addEventListener('onChange', handleChange);

    return () => current.removeEventListener('onChange', handleChange);
  }, [ref]);

  return (
    <road-dropdown
      ref={ref}
      label={label}
      option={option}
      options={JSON.stringify(options)}
    />
  );
};
```

That's it for adding an event listener for our callback function that is passed as prop to our Dropdown Component. Therefore, we used a reference attached to the Custom Element to register this event listener. All other properties are passed as attributes to the Custom Element. The `option` and `label` properties are passed without modification. In addition, we passed the `options` object as stringified JSON format. In the end, you should be able to use this Web Component in React now.

REACT TO WEB COMPONENTS LIBRARY

The previous section has shown you how to wire Web Components into React Components yourself. However, this process could be automated with a wrapper that takes care about formatting objects and arrays to JSON and registering functions as event listeners. Let's see how this works with the `useCustomElement` React Hook which can be installed via `npm install use-custom-element`:

```
import React from 'react';

import 'road-dropdown';

import useCustomElement from 'use-custom-element';

const Dropdown = props => {
  const [customElementProps, ref] = useCustomElement(props);

  return <road-dropdown {...customElementProps} ref={ref} />;
};
```

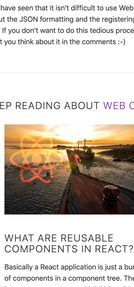
The custom hook gives us all the properties in a custom format by formatting all arrays and objects to JSON, keeping the strings, integers, and booleans intact, and removing the functions from the custom props. Instead, the functions will be registered as event listeners within the hook. Don't forget to pass the ref attribute to your Web Component as well, because as you have seen before, it is needed to register all callback functions to the Web Component.

If you want to know more about this custom hook to integrate Web Components in React, check out its documentation. There you can also see how to create a custom mapping for props to custom props, because you may want to map an `onClick` callback function from the props to a built-in `click` event in the Web Component. Also, if you have any feedback regarding this hook, let me know about it. In the end, if you are using this Web Components hook for your projects, support it by giving it a star.

. . .

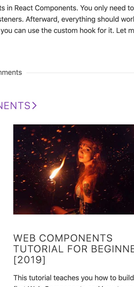
You have seen that it isn't difficult to use Web Components in React Components. You only need to take care about the JSON formatting and the registering of event listeners. Afterward, everything should work out of the box. If you don't want to do this tedious process yourself, you can use the custom hook for it. Let me know what you think about it in the comments :-)

KEEP READING ABOUT WEB COMPONENTS>



WHAT ARE REUSABLE COMPONENTS IN REACT?

Basically a React application is just a bunch of components in a component tree. There is one root component which kicks of the rendering for all the other components below. Commonly these components...



WEB COMPONENTS TUTORIAL FOR BEGINNERS [2019]

This tutorial teaches you how to build your first Web Components and how to use them in your applications. Before we get started, let's take a moment to learn more about Web Components in general: In...