# Additional User Props

In this lesson, we'll learn how users can pass additional props to prop getters!

WE'LL COVER THE FOLLOWING

## Passing Additional Props to Prop Getters

With the prop getter, `getTogglerProps`, we can cater to other props the user may be interested in passing down to the toggle element.

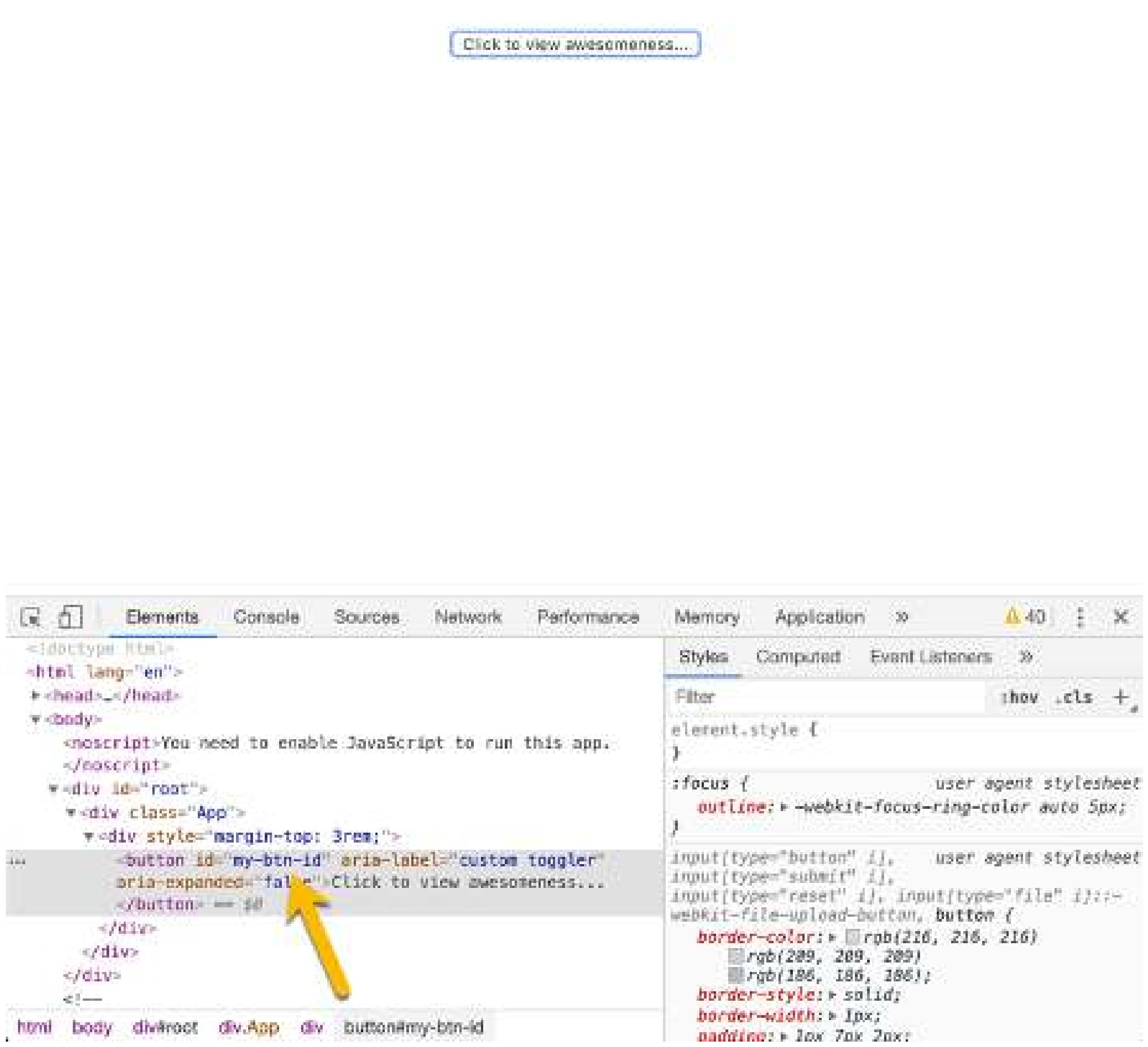
For example, we currently have this usage by the user:

```
<button {...getTogglerProps()}>Click to view awesomeness...</button>
```

If the user needs some more props that we haven't catered for in the prop collection, they could do this:

```
<button id='my-btn-id' aria-label='custom toggler' {...getTogglerProps()}>
        Click to view awesomeness...
      </button>
```

Passing `id` and `aria-label` works just fine.



```
<!doctype html>
<html lang="en">
 ▸<head>…</head>
 ▾<body>
    <noscript>You need to enable JavaScript to run this app.
    </noscript>
  ▾<div id="root">
    ▾<div class="App">
      ▾<div style="margin-top: 3rem;">
          <button id="my-btn-id" aria-label="custom toggler"
          aria-expanded="false">Click to view awesomeness...
          </button> == $0
        </div>
      </div>
    </div>
    <!--
```

```
element.style {
}
:focus {                                  user agent stylesheet
  outline: ▸ -webkit-focus-ring-color auto 5px;
}
input[type="button" i],                   user agent stylesheet
input[type="submit" i],
input[type="reset" i], input[type="file" i]::-
webkit-file-upload-button, button {
  border-color: ▸ rgb(216, 216, 216)
      rgb(209, 209, 209)
      rgb(186, 186, 186);
  border-style: ▸ solid;
  border-width: ▸ 1px;
  padding: ▸ 1px 7px 2px;
```

However, since these are still part of the “toggler props”, though not common enough to make them a part of the default props collection, we could allow for the following usage:

```
<button
        {...getTogglerProps({
          id: 'my-btn-id',
          'aria-label': 'custom toggler'
        })}
      >
      Click to view awesomeness...
</button>
```

It’s arguably more expressive. So, let’s extend the `getTogglerProps` function to handle this use case.

```
...
const getTogglerProps = useCallback(

    ({ ...customProps }) => ({
      onClick: toggle,
      'aria-expanded': expanded,
      ...customProps
    }),
    [toggle, expanded]
  )
```

Since the user will invoke `getTogglerProps` with an object of custom props, we could use the `rest` parameter to hold a reference to all the custom props.

```
({ ...customProps }) => ({

})
```

Then we could `spread` over all custom props into the returned props collection.

```
({ ...customProps }) => ({
      onClick: toggle,
      'aria-expanded': expanded,
      // look here 👇
      ...customProps
})
```

With this extension to `getTogglerProps` we've handled the case for custom props not catered to in the default props collection we expose.

## The Final Output

Here it is all put together.

```
.Expandable-panel {
    margin: 0;
    padding: 1em 1.5em;
    border: 1px solid hsl(216, 94%, 94%);;
    min-height: 150px;
  }
```

How interesting!

## Quick Quiz

Let's take a quick quiz before moving forward

Q

What would happen if we passed custom props like so?

```
({ customProps }) => ({
 ...
```

COMPLETED 0%

1 of 1

---

In the next section, we'll consider an even more interesting case.