Creating custom buttons (2): Toggle Buttons

In a previous post we discussed what it takes to create action buttons that execute a specific action.

In this post we'll discuss toggle buttons, a special type of button that changes based on the status of the button, like if the audio track is playing or not.

Toggle button

Like we did with the action button, we need to look at the roles and attributes we need to have in a custom button.

The biggest difference is the `aria-pressed` attribute that we will add programmatically. This attribute tells assistive technology tools whether the button is pressed or not.

Role	Attribute	Element	Usage
button		div	Identifies the element as a button widget. Accessible name for the button is defined by the text content of the element.
	tabindex="0"	div, a	Includes the element in the tab sequence. Needed on the a element because it does not have a href attribute
	aria-pressed="false"	a	Identifies the button as a toggle button Indicates the toggle button is not pressed

Role	Attribute	Element	Usage
	aria-pressed="true"	a	Identifies the button as a toggle button. Indicates the toggle button is pressed.

With that in mind the structure of the HTML we will use for this example is almost identical to the action button from the previous post.

```
<div tabindex="0" role="button" id="toggle">
  Play
</div>
```

All the work for the toggle is done in Javascript. I've broken the code into sections to make it easier to explain:

The first action is to capture a reference to the element we use as toggle.

```
const toggleButton = document.getElementById("toggle");
```

The `toggleButtonClickHandler` function checks if we're working with a real button or elements with a role of button then we execute the `toggleButtonState` function passing the event as the function parameter.

```
function toggleButtonClickHandler(event) {
  if (
    event.currentTarget.tagName === "button" ||
    event.currentTarget.getAttribute("role") === "button"
  ) {
    toggleButtonState(event.currentTarget);
  }
}
```

The `toggleButtonKeydownHandler` function handles the keyboard navigation. It tests if the key pressed are either the `Space` or `Enter` keys and runs the `toggleButtonState` function.

We use the event.key read-only property to identify the key that was pressed and only execute the `toggleButtonState` function if the user clicke the `space bar` or `enter` keys.

Older examples use the event.keyCode property to read the key that was pressed but the property has been deprecated and it is system dependant.

For the spacebar key detection we use two possible values:

- `Spacebar` will work with older browsers
- The space string (`" "`) works in modern browsers.

```
function toggleButtonKeydownHandler(event) {
  if (
    event.key === "Spacebar" ||
    event.key === " " ||
    event.key === "Enter") {
    toggleButtonState(event.currentTarget);
  }
}
```

`toggleButtonState` is where we do the bulk of the work.

We first create a constant to set the `aria-pressed` attribute to true by default.

We then use a ternary operator to set the `aria-pressed` attribute. It toggles between true and false based on the existing value of the attribute.

We then use the value of the `aria-pressed` attribute to change the text of the button. If the value of `aria-pressed` is true, we set the text to the string `Play`, otherwise we set the text of the button to the string `False`.

```
function toggleButtonState(button) {
  const isAriaPressed = button.getAttribute("aria-pressed") === "true";

  button.setAttribute("aria-pressed", isAriaPressed ? "false" : "true");

  if (isAriaPressed) {
```

```
    button.textContent = "Play";
  } else {
    button.textContent = "Pause";
  }
}
```

Finally we add event listeners for the `pointerdown` and `keydown` events.

```
toggleButton.addEventListener("pointerdown", toggleButtonClickHandler);
toggleButton.addEventListener("keydown", toggleButtonKeydownHandler);
```

The full example is here: