What are some common ARIA states used on custom control elements?

Semantic form control elements like input, select, textarea, button, and fieldset have built-in states that are conveyed to assistive technologies.

For example, you could use the disabled attribute to disable a button or the checked attribute to indicate that a checkbox is checked.

But if you are creating a custom control element, you need to use ARIA attributes to convey the state of the control to assistive technologies.

In this lecture video, we will discuss a few common ARIA states that you can use on custom control elements.

The first ARIA state we will discuss is aria-selected. This state is used to indicate that an element is selected. You can use this state on custom controls like a tabbed interface, a listbox, or a grid.

Here is an example of how you can use aria-selected on a custom tab control:

<div role="tablist">

<button role="tab" aria-selected="true">Tab 1</button>

<button role="tab" aria-selected="false">Tab 2</button>

<button role="tab" aria-selected="false">Tab 3</button>

</div>

Tabs are used to display multiple panels of content in a limited space. The aria-selected state is used to indicate which tab is currently selected.

When the user selects a tab, the aria-selected state of the selected tab is set to true, and the aria-selected state of the other tabs is set to false.

Another common ARIA state is aria-disabled. This state is used to indicate that an element is disabled only to people using assistive technologies, such as screen readers. It is important to note that aria-disabled does not actually disable the element. It is up to you, the developer, to make it look and act like a disabled element. This attribute is also commonly used on native HTML elements in place of the disabled attribute. Which one you choose will depend on the context the button is being used.

Here is an example of how you can use aria-disabled on a custom edit button:

<div role="button" tabindex="-1" aria-disabled="true">Edit</div>

The aria-disabled attribute is used to tell screen reader users that the edit button is disabled and cannot be interacted with. Again, it does not actually disable the button. When using aria-disabled,you will need to apply styling and JavaScript to make the control look and behave like a disabled button.

In most cases, you will probably use the native button element, but there are cases where you might need to use a custom control. So, it is essential to know how to convey the state of the control to assistive technologies.

The next ARIA state we will discuss is aria-haspopup. This state is used to indicate that an interactive element will trigger a popup element when activated. You can only use the aria-haspopup attribute when the popup has one of the following roles: menu, listbox, tree, grid, or dialog. The value of aria-haspopup must be either one of these roles or true, which defaults to the menu role.

Here is an example of a file editor menu that uses aria-haspopup:

<button id="menubutton" aria-haspopup="menu" aria-controls="filemenu" aria-expanded="false">File</button>

<ul id="filemenu" role="menu" aria-labelledby="menubutton" hidden>

<li role="menuitem" tabindex="-1">Open</li>

<li role="menuitem" tabindex="-1">New</li>

<li role="menuitem" tabindex="-1">Save</li>

<li role="menuitem" tabindex="-1">Delete</li>

</ul>

The aria-haspopup state is used to indicate that the File menu button will open a popup menu when activated. Screen reader users may hear this additional information when they navigate to the button.

You will need to use JavaScript to show and hide the popup menu, and to implement proper keyboard support for interacting with the menu. Also, please note that the ARIA menu role refers to a very specific type of menu. It generally refers to a list of actions that the user can invoke, similar to a menu on a desktop application. It does not include more common uses of what we typically refer to as "menus", such as navigation menus. Realistically, most "menus" you create on the web will not be ARIA menus and you will not use aria-haspopup with them.

The next ARIA state we will discuss is aria-required. The aria-required attribute is used to indicate that a field needs to be filled out before the form is submitted. Here is an example of working with the aria-required attribute for a custom form control.

<div id="name-label">Full Name\*</div>

<div role="textbox" contenteditable aria-labelledby="name-label" aria-required="true" id="name"></div>

We need to use the contenteditable attribute so users can type in their input. We are also using the aria-required attribute set to true to indicate that this custom form control is required.

To make the form control look like a normal form control, you would need to add CSS. You would also need to add JavaScript to prevent the form from being submitted without content.

If the label already has the word required, then you should omit the aria-required attribute. This ensures that screen readers only announce the word required once.

In most cases, you will probably use the native label and form elements with the required attribute. But if you need to create a custom form control, then it is important to add the aria-required attribute when necessary.

Additionally, the aria-required attribute can also be used on native form inputs, such as the input, textarea, and select elements. This is often preferred to the native required attribute, since the required attribute may have potential usability and accessibility concerns, particularly with the default error handling provided by the browser. Ultimately, you will need to test in order to determine which attribute is best for your situation.

The last ARIA state we will discuss is aria-checked. This attribute is used to indicate whether an element is in the checked state. It is most commonly used when creating custom checkboxes, radio buttons, switches, and listboxes.

Here is an example of how you can use aria-checked on a custom checkbox control:

<div role="checkbox" aria-checked="true" tabindex="0">Checkbox</div>

Native checkbox elements have a built-in checked state that is conveyed to assistive technologies. But if you are creating a custom checkbox control, you will need to use the aria-checked attribute to indicate its state.

When the user interacts with the custom checkbox control, you will need to use the aria-checked state to reflect the new state of the checkbox. When the checkbox is checked, the aria-checked attribute is set to true. When it is unchecked, it is set to false.

Native elements typically have better support and built-in accessibility features.

However, if you must create custom controls, using ARIA attributes is essential to convey the state of these controls to assistive technologies effectively.

As always, test your work to ensure that ARIA attributes are applied correctly and that the custom control functions in a way that is both accessible and user-friendly.