**Creating a Form for Screen Reader Accessibility**

Web accessibility is becoming more and more important these days. If you are not used to implementing web accessibility, it sounds difficult. In reality, making website accessible takes only a little bit of effort and knowledge. HTML 5 supports a lot of accessibility functionality if you use the correct tags and follow the standard best practice.

Web accessibility is a large topic to cover. In this post, let’s focus on screen reader accessibility for forms.

**Before we start**

There are two Google Chrome plugins that I recommend to install.

* [axe](https://chrome.google.com/webstore/detail/axe-web-accessibility-tes/lhdoppojpmngadmnindnejefpokejbdd)
* [ChromeVox classic extension](https://chrome.google.com/webstore/detail/chromevox-classic-extensi/kgejglhpjiefppelpmljglcjbhoiplfn?hl=en)

Axe is a free chrome extension for web accessibility testing. It scans your page and report back with accessibility issues. Scanning any website and checking the issues will teach you a lot about accessibility.

ChromeVox classic extension is a free screen reader that you can use on Chrome. I am recommending this because it is free and probably just good enough to start with. However, ChromeVox does behave differently from the real screen readers out there. If the screen reader experience is critical for your website, I would insist on getting the real screen reader and test your site with it.

**Tips**

1. **Provide extra information with invisible paragraph**

You can provide extra information with invisible paragraph. This can be achieved by adding a class name with styling to hide the paragraph as below. In this way, screen reader will read out the extra information.

<script src="https://gist.github.com/mydatahack/77a25285d059c97d76e779da0f459ca2.js"></script>

1. **Use aria-required attribute for required input fields**

When your input field is required, setting the aria-required attribute to true will make screen reader to say the field is required. Having only the required attribute in the input field will not screen reader to indicate the field is required.

<script src="https://gist.github.com/mydatahack/a0620714373c4f5c25b1b16aea8ddd7c.js"></script>

1. **Carefully consider what you put in the place holder**

The screen reader will read the placeholder value as ‘hint: enter your first name’. For example, when you put first name as the placeholder value, the screen reader will read out as ‘first name hint: first name’. The extra information is redundant. Unless you have something more to add, placeholder value may not be needed.

1. **Do not use display none to customise radio or checkbox input**

If your input has display: none, the screen reader will ignore it. We often customise our radio or checkbox input with CSS by hiding the default browser input. Hide the default input by using sr-only class described in (1).

**Use tab index for radio input groups or checkbox – original (4) contains wrong information- updated above**

Tab index makes the section of form as tab focusable. For radio input group, the section of the radio input group as well as each radio button group (input & label) should have tabindex = 0. Otherwise hitting tab will skip the section and go to the next input.

<script src="https://gist.github.com/mydatahack/6fe71c950700af0001f54f7e922eb03d.js"></script>

The same applies for checkbox field.

<script src="https://gist.github.com/mydatahack/31efa97d2cedcac744476edb7ef67368.js"></script>

1. **Be aware of redundant ARIA attributes**

Role is not needed for input field with correct type (example [here](https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/Roles/checkbox_role)). When you use input with type=”checkbox” as in the example, role=”checkbox” is redundant.

The same goes for aria-checked attribute. For radio or checkbox input, we don’t need to explicitly assign value to aria-checked by JavaScript if the input field has the type specified. Screen reader will read if the input is checked or not.

<script src="https://gist.github.com/mydatahack/f428165e870c4cd97ff0d033f36a5b72.js"></script>

1. **Use HTML5 semantic elements instead of div**

This applies not only for a screen-reader accessible form. It applies to web accessibility as well as HTML best practice. Instead of using div, use HTML5 semantic element wherever you can. For example, using the nav tag instead of div with class=”nav”. Sticking to the best HTML5 practice will make website accessible.

1. **Test the site with a screen reader**

Accessibility testing tools like [axe](https://chrome.google.com/webstore/detail/axe-web-accessibility-tes/lhdoppojpmngadmnindnejefpokejbdd) is nice. But, it pays off to invest in a real screen reader for testing. Unless you use the real one, you will never know what the experience is like on your website.

**Example**

I made an example ([Accessibility Form Example](https://mydatahack.github.io/accessibility-form/)) where I implemented the tips presented in this post. Forms can be switched between screen reader friendly version and not so friendly version. It is not perfect, but is a good start for you to get started with screen reader accessibility. The source code is [here](https://github.com/mydatahack/web-accessibility-form-typescript-spa) if you are interested. It is build as a SAP with TypeScript.

**Resources**

Google’s web fundamentals is a good resource for anything related to front-end development. It offers a extensive section on [web accessibility](https://developers.google.com/web/fundamentals/accessibility). This is the best place to get started. If you want to be a web accessibility expert, [Deque University](https://dequeuniversity.com) offers online training. The website also has a heaps of resources about accessibility.

Here are the references I used for this post.

* [ARIA: checkbox role](https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/Roles/checkbox_role)
* [Using the aria-labelledby attribute](https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/ARIA_Techniques/Using_the_aria-labelledby_attribute)
* [Introducing to ARIA](https://developers.google.com/web/fundamentals/accessibility/semantics-aria/)
* [Radio input tip](https://dequeuniversity.com/rules/axe/3.3/radiogroup)
* [ARIA radiogroup and radio Example](https://www.w3.org/TR/2016/WD-wai-aria-practices-1.1-20160317/examples/radio/radio.html)
* [Bad ARIA practice](https://www.accessibility-developer-guide.com/knowledge/aria/bad-practices/)
* [Using the tabindex attribute](https://developer.paciellogroup.com/blog/2014/08/using-the-tabindex-attribute/)
* [Grouping Controls](https://www.w3.org/WAI/tutorials/forms/grouping/)