WECO FALL WEBINAR SERIES: ACCESSIBILITY CODE BASICS

Thursday, October 17, 2019

12 - 12:30 PM (CDT)

CAPTIONING PROVIDED BY CAPTION ACCESS LLC

support@captionaccess.com

http://www.captionaccess.com

Dane: This is Accessibility Code Basics. I'm Dane Dunham, your instructor and accessibility specialist here at WeCo.

This is what we'll touch on today. We'll talk about some devices that people with disabilities use to interact with computers. Then we'll go over some of the commonly encountered code errors and how to remedy them. We'll wrap up with some takeaway resources and information about WeCo.

This is the device type. Most commonly, people with disabilities will interact with their computer using one of these. There's refreshable braille displays. There's screen reader software, which can only read one element on a webpage at a time. And then I'll quickly mention these others: specialized keyboard, pointer, or mouse, speech recognition software, magnification, and eye tracking software.

This is the first code example. We have a label element provided for a form field, but the label was not properly associated with the text field because there was not an ID associated with the text field, which prevents assistive technology like screen readers from recognizing the label. So when screen readers encounter this kind of text field, they might just say something like, "Text Field" or "Edit Blank" to identify it as an editable text field, but not necessarily identifying what needs to be put in the text field.

It's a very common error we see on a lot of sites. It's very commonplace. It's why it's first on our list today. That violates WCAG 3.3.2, 1.3.1, and 4.1.2.

This shows the fix for that. In this case, it's really simple. Just add an ID to the form field, because all the other required pieces of code were already in place. Now, the assistive technology can convey the form field to users. When you have properly associated form field labels, it also provides a larger click target -- a larger area to click the mouse for people with mobility impairments. A lot of web browsers will let you click on the label of a text field to put focus inside that form field.

Alternatively, you could encapsulate the input element inside the label element itself, and nest

it that way. That would also work.

Next slide is slide 8. This one is custom controls that can present accessibility issues. This shows a code sample which was taken from a custom checkbox written using angular JS. This checkbox used appropriate tab index and aria attributes to allow assistive technology to detect it as a checkbox and read it correctly and work with it correctly, to use the keyboard and such to toggle it. But we can't always guarantee that these types of implementations will be successful, because the support for aria and that kind of custom markup varies from one browser to another, and from assistive technology to assistive technology, and even within variations of those.

So it's very difficult to ensure that all of this type of custom controls will be accessible across the board. We definitely don't recommend using those unless there's no standard/native HTML element for your use case. For this checkbox in particular, screen readers might have difficulty detecting that the element is interactive, or might not report state changes, like checked or unchecked, to the users. This violates WCAG 2.1.1. and 4.1.2.

And here's the fix. For this one, to resolve any potential issues I talked about earlier, we replace the custom checkbox with a standard input element and the associated label. In this case, the HTML controls have broad support both across browsers and assistive technologies, since HTML has been around since the dawn of the World Wide Web. In this case no additional effort needs to be made to make them accessible, beyond providing a properly associated label as we discussed in the first issue. That means that assistive technologies like screen readers can properly convey things like state and role, that it's a checkbox, and that it's labeled to the users.

This one is a misuse of HTML heading elements. This is also very common. Sometimes we see heading elements that are used to format text in specific ways, or make it bold, or italicized, or however they have the headings formatted in the content management system. We'll see headings formatted sometimes that way on purpose. But headings are used to provide structure to screen readers to create a logical and hierarchal structure of the content.

In this case, a heading element was used to provide the visual styling for the disclaimer text further in the webpage, which is very confusing for screen reader users and other users who rely on the headers for semantic outlines of the page, because if the text inside a heading element is not the start of a new section of content -- like in this case -- it would be really confusing for users who are navigating by heading. Most screen reader users do that, by the way. They familiarize themselves with unfamiliar pages by quickly jumping from heading to heading, so if they don't represent logical sections of the page, these users can quickly become confused or disoriented or unsure if they've missed sections of content. This violates WCAG 1.3.1.

In this next slide, we replaced the for element with a div element. In HTML div elements do not have a default. Any additional styling can be applied to that element, if you want the text to appear a certain way visually. Since the disclaimer was not the start of a new section of

content, having it inside a plain div element like that clarifies the purpose because it will be just read as plaintext by assistive technology, which it is in this case. And that will also resolve any confusion that heading may have caused.

In general, when creating a new site, it's good practice to try to determine what the logical structure of the headings of the page will be first, and then apply the visual styling to the headings and any other portions of text afterward.

This is modals not coded properly. Modals are very common these days, in things like video players or confirmation dialogues, email signup forms, etc. We have all seen them. In this case, a div element is used to provide both the structure and to contain the content of the modal on a page. Without any additional markup, which we'll get to in the next slide, screen readers and assistive technologies will not say anything when that's shown on screen. As I mentioned previously, the div elements do not have any semantic meaning, so the assistive technologies will not necessarily detect them. So that violates WCAG 4.1.2.

This shows the correction, which adds these features, which I'll discuss individually. First we gave the div element the role of dialogue with the dialog attribute. This is referred to as a dialogue in screen reader speak. Second, we used the object labeled by attribute to associate the contained element with the first element inside the dialogue, which will be visible in the heading at the top of the dialogue, which provides the title that the assistive technology or screen readers can announce automatically when the dialogue shows up. You could also use the aria label attribute to provide a title directly, if the dialogue that you're working on doesn't have a visible title already.

Finally, the aria modal attribute is added to inform assistive technology of the content behind the modal, that it's not active and shouldn't be attempted to be interacted with by assistive technology or have it receive focus, which is important because screen reader users can usually navigate around with the tab key or a virtual cursor that their screen reader uses to read content on the page. And if they try to interact with something outside the modal, that can be very confusing.

So these techniques really help ensure that assistive technology can accurately convey the modal to the users.

This one is content read out of sequence. In this example, which we got from the WCAG failure example #1, there's a table of products and locations that's read out of order, because in this case, CSS had been used to visually position the elements on screen to make them look like a table. However, since assistive technology relies on the underlying HTML markup, this will not be read properly since the order of the HTML elements is not correct. It doesn't match the actual order that they would be on screen.

Specifically, a screen reader user looking at something like this might hear products, locations, computers, portable MP3 players, Wisconsin, Idaho, etc. because that's the order they appear

in the CSS source. And that violates WCAG 1.3.2.

To correct that, we rearranged the items in the code and placed them in a proper HTML table structure, to let screen reader users read the items not only in the correct order, but also to use table navigation commands and navigate within the rows and columns to get a sense of the overall layout. We have also allowed the headers to be repeated because they're coded as header elements in the table. So when they go between the cells in each row, the screen readers will repeat the column headers as well, so the users can better understand the information, since screen readers only read one element at a time. A lot of times screen readers are configured to repeat row and column headers if they're present in the tables.

This is off-screen content that is not hidden from the assistive technology. This is showing both HTML and CSS for a form that was hidden off-screen, and a button that we're not showing here for simplicity's sake, used to toggle the visibility of the form. For regular users this works well because the CSS is causing the form and anything else contained inside that element to be placed outside the visual viewport, or just off-screen in common terms, so it's not visible.

But because it's still present in the HTML and just simply moved out of the visible screen area, screen readers will still pick it up because it's not specifically removed from assistive technology as well. They might try to read it with a screen reader, which can lead to confusion about the button to toggle the visibility. It may not work for form fields positioned off-screen, which can really lead to a lot of confusion for those users. And that violates WCAG 1.3.2.

The correction for this is to replace the CSS for floating the content off the visible screen area with "display: none" or "visibility: hidden" in this case would also work. They both remove it visually as well as from assistive technology. In a case like this, it's important to ensure that the hiding of the form also hides it from assistive technology. That will really help with any confusion that might arise from the assistive technology still being able to interact with it while it's off-screen.

Then you would use Javascript to toggle between this CSS class and any other that the developer chooses for the div element on the button click, to allow the button to act as a toggle for displaying and hiding the search form. In addition to that, certain aria attributes would also be needed on the toggle button to convey the state the button to assistive technology. But that falls outside the scope of this particular slide.

So that's all I have for the common issues. Next slide?

I'll talk about some resources we have at WeCo. Something that's available to everyone is our Free Accessibility Library, on our website weco.com. It contains a lot of information about things like accessibility laws and guidance in the United States and worldwide, best practice tips from WeCo's certified test team and the accessibility specialists, as well as free tools and resources to help you not only learn about accessibility, but keep your products and services accessible as well, and more. We recommend you bookmark it for easy reference, and feel free

to refer to it as often as you like.

WeCo also provides you with some free and easy ways to stay on top of accessibility. We have our blog, as well as our email quick tips. A few times each month, you'll be notified about our latest blog entry related to accessibility and how people with disabilities interact with IT. In our blog entries we cover everything from accessibility experiences to experiences that people who live with particular disabilities have, as well as legislative updates. And our quick tip emails provide you with technical accessibility tips in shorter email messages, complete with informative video shorts to demonstrate easy ways you can keep your websites and software accessible. It makes it easier for you to learn to keep accessibility top of mind and to remember to implement what you learned today.

We have some upcoming events. Please feel free to go to our events page to learn about and get the latest information on our upcoming webinars and workshops. We have Meetups and other events as well. If you need something special that's not covered there, we also have customized training. Please feel free to contact us about that if that's something you're interested in.

So feel free to keep in touch, if you'd like. We're always here to help. And our website is displayed on the slide, but it's weco.com. On Facebook we can be found at WeCo or The Wehrman Collaborative. On Twitter we are @WeCo5. We also have an email address and phone number listed here.

And that's all I have for you today. Does anyone have any questions? I'll open up the floor if anyone has any questions.

Laura: Hey Dane.

Dane: Let me come into the chat or email.

[Background noises.]

Dane: I'm sorry, say that again?

Laura: Dane, it's Laura. As you know, you can ask again if anyone has any questions.

Dane: Sure. I'll do that. Does anyone have any questions? Sorry about that.

If no one has any questions, I'll go ahead and end this webinar. Thank you, everyone, for participating today. Again, apologies for the technical difficulties at the beginning.

I'm sorry, did someone say something? Well, thank you all for coming.

Laura: Actually, Dane, it looks like Colette -- I think that's how you pronounce your name -- is

asking if we'll get a copy of the presentation.

Dane: That's a very good question. I honestly do not know at this time. We can look into that. I do not believe so, but I'm not certain at this time. But that is a good question.

Any other questions?

Speaker: Yeah, thank you, Dane.

Dane: You're welcome. All right. It looks like we're good. Laura, do we have anything else coming into the chat?

Laura: We do not. Everyone is saying "thank you very much" to you, Dane. And the handouts are very helpful, that everyone got.

Dane: Great. So I think I will go ahead and conclude the webinar. And thank you, everyone, for joining us today. We look forward to serving you in the future, hopefully.

Laura: Thank you, everyone, for attending this webinar. Have a great rest of your day. Thank you again for being patient with our technical difficulties. But we have it under control.

Dane: All right. I think I'll go ahead and conclude the webinar.

Laura: Everyone says, thank you Dane and team.

Dane: Certainly. I think I'll go ahead and conclude this.

Laura: All right. Thank you again, everyone, for coming. We are gonna end this now.

[End of transcript.]