

Accessibility: testing Adobe's PDF to HTML conversion tools

Tagged PDF version

Table of contents

Accessibility: testing Adobe's PDF to HTML conversion tools	1
Table of contents	2
About this document.....	3
No tags version.....	3
Accessible forms	5
The importance of understanding screen reader "forms mode"	5
Structure.....	5
Explicit labelling	5
Non-form elements	6
Understanding forms mode	6
Option 1: explanatory text first, form last	7
Option 2: start a new page	8
Option 3: use pop-up	8
Option 4: provide an invisible link	8
Fieldsets and legends	9
Keep legend text short	9
Limitations of fieldset and legend.....	10
Legend workaround	10
Accessible PDF forms	10
Label positioning	11
Other common problems	12
Validation.....	13

About this document

This document is designed to test the accessibility benefits of the PDF to HTML conversion tools available from the Adobe website at:

http://www.adobe.com/products/acrobat/access_onlinetools.html¹

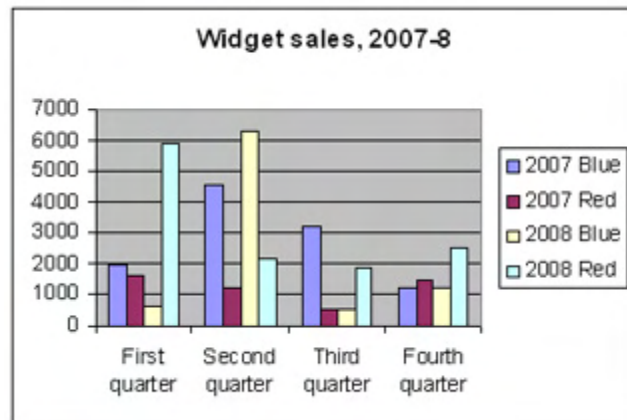
No tags version

This version of the document was tagged on conversion from Word to PDF and so contains structural elements such as headings, links and alt text.

It has one image - a chart of sales figures - and one data table with

two levels of headings. It also contains one footnote relating to the first paragraph

of text. The document therefore contains many of the structural elements that regularly appear in PDFs and which, if not handled



correctly, are likely to cause accessibility problems.

Table 1: data for the chart above

	2007		2008	
	Blue	Red	Blue	Red
First quarter	2001	1625	622	5874
Second quarter	4527	1255	6311	2145
Third quarter	3205	552	559	1878
Fourth quarter	1254	1445	1245	2545

¹ This is a footnote that relates nominally to the first paragraph of text on this page, specifically the link to Adobe's website.

Lastly, the document also contains the text from a previous PWS article (on accessible forms). This is included in order to test the outcomes in different formats of a table of contents being inserted into the document. A table of contents with active links to the various sections of a document is probably the single most important accessibility feature that can be added to any PDF of more than a page or two in length.

Accessible forms

The importance of understanding screen reader "forms mode"

There are many excellent tutorials on how to create accessible forms. Highly recommended are Stanford University's Creating Accessible Forms and Cameron Adams' Fancy Form Design. This article takes a different approach. Its principal focus is on the pitfalls of accessible form design and how to work around them (both in HTML and PDF) – a topic that is not as well covered in the literature as it might be. A crucial part of this is understanding screen reader "forms mode".

Structure

Initially the focus of this article will be on three key structural aspects of creating accessible forms: explicit labelling of input fields; the appropriate use of non-form elements such as paragraphs and headings; and the correct use of fieldsets and legends to label and group form elements. Get these right and, with the possible exception of validation, the rest is relatively easy.

Explicit labelling

Any accessible forms tutorial will tell you that it is important to provide explicitly associated labels for all input fields (text, checkbox, radio button, password, textarea and select). Implicit labelling should not be used as many assistive technologies do not handle it correctly. The following is an example of an input field that is explicitly associated with its label.

```
<label for="name">Name</label>
```

```
<input id="name" name="name" title="Name" type="text" />
```

The explicit relationship is defined by the input field's id="name" and the label's for="name" attribute/value pairs. As long as both the "id" and "for" attributes have the same value (in this case "name") there should be few problems. However, for the small number of assistive technologies that do struggle with this (ZoomText 9.0 reader, for example), it is advisable to add title="Name" to the input field.

Right, that's the easy bit out of the way.

Non-form elements

One of the most serious and least well understood form accessibility problems is caused by the inclusion of non-form elements between form fields. Common examples include headings and paragraphs of explanatory text. To see why this is a problem it is necessary to understand screen reader "forms mode" (in JAWS terminology – in Window-Eyes the equivalent is known as "browse mode off" and in Hal "interactive mode").

Understanding forms mode

All screen readers have two distinct modes, one for operating forms and one for everything else. The reason for this is that in normal web browsing mode many keys have special functions: for example, in JAWS, pressing H takes you to the next heading, L takes you to the next list, P takes you to the next paragraph, and so on. However, when completing a form these keys are needed for input, hence the need to switch to a different mode with a different set of commands.

In forms mode, screen reader users typically move from one field to the next by pressing the Tab key (or the up/down arrow keys to navigate within sets of grouped controls such as radio buttons or dropdowns). Provided that labels are explicitly associated with their

respective input fields this works well. Labels are voiced correctly and focus is assigned to each input field in turn ready for inputting data.

However, most screen reader users tabbing from one form field to the next in forms mode won't hear any content contained in non-form elements such as paragraphs or headings. This is because in forms mode screen readers can only read the very limited number of HTML elements that can receive focus. These are: form elements, links, and nothing else (not `<p>`, not `<h1>`, nor anything else).

(An exception to this rule is Hal which turns off interactive mode when the user tabs to the next form control.)

Of course, the problem would disappear altogether if paragraphs and headings could receive focus (as is proposed in the current HTML 5 working draft). In theory they can now – by adding a `tabindex="0"` attribute and value to a non-form element. But there are two problems with this. One, the HTML doesn't validate against a strict doctype and two, JAWS reads the text but then announces the existence of an input field even though none exists (Window-Eyes assigns focus but remains silent). Even though HTML 5 may come to the rescue one day, for the time being we need to work around the problem.

Option 1: explanatory text first, form last

The tidiest solution is to place all explanatory text at the top of the page before the form starts. The text will be read in normal web browsing mode until the first form element is encountered at which point the screen reader will prompt the user to enter forms mode in order to input data. This works well as long as there is no content placed after the form or explanatory text within it.

Option 2: start a new page

If it is necessary to include explanatory text within the form a good solution is to create a multi-page form. Each time some explanatory text is needed a new page is started, with the text (as in option 1 above) positioned at the top. This works because clicking a "next page" button forces the screen reader to exit forms mode and re-enter normal web browsing mode as well as opening the new page. The text can then be read in the normal way before re-entering forms mode to continue.

Option 3: use pop-up

Sometimes a simple pop-up box might be the best solution. As links can receive focus in forms mode, you can use a link to open a pop-up containing explanatory text. However, be sure to use an accessible pop-up such as Ian Lloyd's [The Perfect Pop-up](#). You also need to be careful how you "pop back down again". On closing the pop-up, the focus should be returned to the form element after the link that opened it. The likely alternative is that screen reader users will be returned to the top of the page, which at best will be highly confusing.

Option 4: provide an invisible link

If explanatory text must be included between form elements, and for some reason the above options are not available, a somewhat inelegant workaround exists. This is simply to provide a hidden (skip) link to some explanatory text below. Activating the link will force the screen reader to exit forms mode so that the text can be read. The down arrow or tab key will then take the user to the next form element. However, whilst this might benefit screen reader users, there

is potential for confusion for users of non-CSS enabled browsers who will see the link.

In options 2, 3 and 4 above the solution stems from clicking a button or a link to force the screen reader to exit forms mode in order to read any text contained within non-form elements.

Fieldsets and legends

The third key structural aspect concerns the appropriate use of fieldsets and legends. Fieldsets and legends are particularly important for enabling assistive technology users to make sense of groups of checkboxes or radio buttons. A typical set of radio buttons might be coded as follows:

```
<fieldset>
<legend>Age range</legend>
<input type="radio" id="under18" name="age" title="Age range
under 18" />
<label for="under 18">Under 18</label>
<input type="radio" id="18-64" name="age" title="Age range 18 to
64" />
<label for="18-64">18 to 64</label>
<input type="radio" id="over64" name="age" title="Age range 65 or
over" />
<label for="over64">65 or over</label>
</fieldset>
```

Keep legend text short

Coded in this way, JAWS will announce the legend text before each label in the group. For this reason it is advisable to keep legend text as short as possible. In this case the user will hear: "age range: under

18; age range: 18 to 24; age range: 65 or over", which is more user-friendly than, say, "please tell us your age range: under 18; please tell us your age range: 18 to 24; please tell us your age range: 65 or over".

Limitations of fieldset and legend

Unfortunately not all screen readers handle fieldsets and legends as well as JAWS does. For example, in Window-Eyes, in normal web browsing mode, users can jump to the next fieldset by pressing E, but in forms mode ("browse mode off") this is no longer possible (the E key is needed for input). In addition, in order to hear the contents of legend tags, Window-Eyes users have to select the appropriate verbosity settings each time they switch on the computer.

Legend workaround

Because of the poor support for fieldset and legend in Window-Eyes and because Window-Eyes does read the contents of the title tag, it is a good idea to duplicate the legend text in each radio button's title tag as well as the text of its label, as in the above code sample.

Accessible PDF forms

With PDF forms created in LiveCycle Designer the equivalent of explicit labelling is taken care of as screen readers automatically read form element labels (or captions as they are called in LiveCycle Designer).

However, there is no direct equivalent of the HTML legend tag. To produce the same effect, in the Accessibility palette, the appropriate text should be added to the Custom Screen Reader Text field of the "exclusion group" containing the radio buttons. With check boxes a slightly different approach is required as there is no containing group.

In this case, Custom Screen Reader Text has to be added to each checkbox.

Text boxes can be added to a LiveCycle Designer forms but, as with paragraphs and headings in HTML forms, there is no way to make these available to screen readers (except Hal) in forms mode. All the same restrictions apply as for non-form elements in HTML.

Clearly there are still limitations in the capacity of various assistive technologies to make sense of forms, no matter how well they are coded. However, following the above guidelines will make forms highly accessible in the more advanced screen readers and as accessible as possible in the others.

Label positioning

Another common problem that warrants a mention here is the often large gap left between labels and their respective fields. This can be a serious problem for users of screen magnification software who may find it difficult to associate a label with the appropriate field. Happily, there are benefits to be had for most users from keeping the gap to a minimum.

Eye tracking tests have shown that form completion times are fastest when labels are positioned immediately above their respective input fields as users can view both at the same time. The next most efficient option identified in these studies was right-aligned label text positioned just to the left of the input field. The longest form completion times were measured for forms with labels aligned to the left margin (with necessarily a wider gap between label and input field). Completion times were almost double those for forms with labels with right-aligned text.

Other common problems

Briefly, some other points to note are:

- Always include an explicitly associated label with a search input box. If you don't want a visible label, hide it off screen with CSS. Contrary to the advice in some guides don't use `display:none` as this hides the label from screen readers too. Instead use off-screen positioning.
- Don't use a fieldset without an accompanying legend. It causes JAWS to misinterpret the legend text of nested fieldsets. It can also cause layout problems in some browsers (for example Safari on Windows).
- Don't use JavaScript dependent drop downs (selects) as they are not keyboard operable. Instead use a normal select with a button to activate the selection.
- Don't use tables to lay out forms, for all the usual reasons. They can also play havoc with your fieldset and legend plans and lead to unspeakable atrocities when, for example, radio buttons are placed one to a cell in a tabular grid. Just say no!
- Don't place any content after the submit button as it is unlikely that it will be read by screen reader users.
- Access keys often cause conflicts with application-specific keyboard commands. There is also no consistency from site to site. And, most screen reader users ignore them. They have been downgraded from a triple A accessibility requirement in

WCAG 1.0 to an "advisory" in WCAG 2.0. Might it not have been kinder to put them out of their misery altogether?

Validation

Lastly, a word on form validation. Although this is a topic that warrants an article of its own, this one wouldn't be complete without at least a nod in that direction. So here it is.

JavaScript alert boxes are a highly accessible way to provide form validation to screen reader users. But they are not ideal from an aesthetics point of view.

One solution is to use JavaScript to post error messages directly to the page for sighted users whilst offering screen reader users the option of validation via alerts. This can be done quite simply via a pair of radio buttons at the top of the form, hidden off screen with CSS. Depending on which option is selected the user will get error messages posted on-screen (typically the default) or via alerts.

Of course, to accommodate all users you also need to offer server-side validation for those who have JavaScript disabled or otherwise unavailable.