## WCAG 2.0 Accessibility Testing checklist

<table>
<thead>
<tr>
<th>Accessibility requirement</th>
<th>How to test</th>
<th>Pass / Fail / n/a</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-text Content WCAG 2.0 1.1.1 Level A</td>
<td>Using the WAT Toolbar, select the show images option and check for missing alt text on images. For those images with an alt text check they are relevant</td>
<td></td>
</tr>
<tr>
<td>Multimedia content Captions WCAG 2.0 1.2.2 Level A</td>
<td>To check if captions are present look for a CC option on the video player; if they are present turn on the option and check that they are synchronized with what is happening on screen.</td>
<td></td>
</tr>
<tr>
<td>Text transcripts WCAG 2.0 1.2.3 Level A</td>
<td>Check that a transcript is provided via a link either in the video player or adjacent to it. Check the transcript is clear and indicates who is speaking and includes all the speech content available in the audio of the video.</td>
<td></td>
</tr>
<tr>
<td>Adaptable (Info and Relationships) Markup WCAG 2.0 1.3.1 Level A</td>
<td>On the WAT accessibility toolbar use the Depreciated Elements options to find out if tags such as font and u have been used In addition you will need to check the code for instances of \texttt{&lt;b&gt;} tag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>On the WAT Toolbar from the ‘Structure’ tab select ‘Headings Structure’ and if used review the list of headings to check they follow a logical sequence H1\textgreater h2\textgreater h3</td>
<td></td>
</tr>
<tr>
<td>Data Tables</td>
<td>On the WAT Toolbar from the ‘Tables’ tab select the ‘Show Data Table’ option. All the table cells will be highlighted on the page. Check if column headings and rows (if present) are labelled correctly with the <code>&lt;th&gt;</code> tag</td>
<td></td>
</tr>
<tr>
<td>Forms</td>
<td>On the WAT Toolbar click on the ‘Structure’ tab and select ‘Fieldset/Labels’. Fieldset and label elements will be highlighted on the page. Make sure a ‘for’ attribute and an ‘id’ attribute have both been used for each form field</td>
<td></td>
</tr>
<tr>
<td>Reading and Navigation order (Meaningful sequence)</td>
<td>On the WAT Toolbar click on the ‘Tables’ tab and select ‘Linearize’ to make the page into one column then review the content to check it makes sense.</td>
<td></td>
</tr>
<tr>
<td>Sensory specific language</td>
<td>Read the web page and check if any language uses any sensory specific language</td>
<td></td>
</tr>
<tr>
<td>Distinguishable</td>
<td>Scan the page and check all images and text don’t just use colour to be understandable</td>
<td></td>
</tr>
<tr>
<td>Use of color (Conveying information by colour alone)</td>
<td>If a page has audio that automatically starts when it has loaded and the audio lasts longer than three seconds check to see if there is an option to turn off the audio – this should also be keyboard accessible.</td>
<td></td>
</tr>
<tr>
<td>Accessibility requirement</td>
<td>How to test</td>
<td>Pass / Fail / n/a</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Colour contrast</td>
<td>Use the Contrast Analyser tool and check colours for text and images are &gt;=4.5:1 (3:1 for text over the equivalent of 14pt bold or 18pt</td>
<td></td>
</tr>
<tr>
<td>WCAG 2.0 1.4.3 Level AA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Relative Layout</td>
<td>Use the WAT toolbar and select Text Size &gt; Larger /Largest and look for cropping or overlapping of text</td>
<td></td>
</tr>
<tr>
<td>WCAG 2.0 1.4.4 Level AA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graphic text</td>
<td>Check contrast (as above) and that alt text is accurate, background is uncluttered, and the text is a minimum of 12pt</td>
<td></td>
</tr>
<tr>
<td>WCAG 2.0 1.4.5 Level AA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.4.8 Visual Presentation</td>
<td>Check there is no fully justified text (aligned to both the left and right margins)</td>
<td></td>
</tr>
<tr>
<td>WCAG 2.0 1.4.8 Level AAA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Adaptable

#### (Info and Relationships)

<table>
<thead>
<tr>
<th>Markup</th>
<th>On the WAT accessibility toolbar use the Deprecated Elements options to find out if tags such as font and u have been used. In addition you will need to check the code for instances of &lt;b&gt; tag</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>WCAG 2.0 1.3.1 Level A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Headings</td>
<td>On the WAT Toolbar from the ‘Structure’ tab select ‘Headings Structure’ and if used review the list of headings to check they follow a logical sequence H1&gt;h2&gt;h3</td>
<td></td>
</tr>
<tr>
<td>WCAG 2.0 1.3.1 Level A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Tables</td>
<td>On the WAT Toolbar from the ‘Tables’ tab select the ‘Show Data Table’ option. All the table cells will be highlighted on the page. Check if column headings and rows (if present) are labelled correctly with the &lt;th&gt; tag</td>
<td></td>
</tr>
<tr>
<td>WCAG 2.0 1.3.1 Level A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accessibility requirement</td>
<td>How to test</td>
<td>Pass / Fail / n/a</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td><strong>Forms</strong></td>
<td>On the WAT Toolbar click on the ‘Structure’ tab and select ‘Fieldset/Labels’. Fieldset and label elements will be highlighted on the page. Make sure a ‘for’ attribute and an ‘id’ attribute have both been used for each form field</td>
<td></td>
</tr>
<tr>
<td>WCAG 2.0 1.3.1 Level A</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Reading and Navigation order (Meaningful sequence)</strong></td>
<td>On the WAT Toolbar click on the ‘Tables’ tab and select ‘Linearize’ to make the page into one column then review the content to check it makes sense.</td>
<td></td>
</tr>
<tr>
<td>WCAG 2.0 1.3.2 Level A</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Resizable text</strong></td>
<td>Select the IE menu in the WAT toolbar and select Text Size &gt; Larger and look for cropping or overlapping of text – also look at any dropdown menus and combo boxes that contain options. Repeat for Text size &gt; Largest</td>
<td></td>
</tr>
<tr>
<td>WCAG 2.0 1.4.4 Level AA</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Keyboard access</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Keyboard</strong></td>
<td>Using the keyboard tab through a webpage and test you can select and interact with all links, form elements and all other interface controls</td>
<td></td>
</tr>
<tr>
<td>WCAG 2.0 2.1.1 Level A</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>No Keyboard Trap</strong></td>
<td>On the page you want to check hold down the tab key and watch the focus run through the page. It should keep on cycling back around while the tab key is held down. If, however, it stops at a particular element such as a Flash movie, this indicates a keyboard trap, which means you should not be able tab back or forwards anymore.</td>
<td></td>
</tr>
<tr>
<td>WCAG 2.0 2.1.2 Level A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accessibility requirement</td>
<td>How to test</td>
<td>Pass / Fail / n/a</td>
</tr>
<tr>
<td>----------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Timing</td>
<td>Login to the site and check if any information is provided about timeouts - at a minimum there should be information about how long the timeout is and ideally a mechanism to either extend it once logged in or an option to extend when a warning is given that you are about to be logged out from the site.</td>
<td></td>
</tr>
<tr>
<td>Pause, Stop, Hide</td>
<td>For any element that is automatically updating or that moves on screen for more than 5 seconds check there is a way to turn off or pause the movement</td>
<td></td>
</tr>
<tr>
<td>Seizures</td>
<td>Use the PEAT tool see Section F.2 Other software tools</td>
<td></td>
</tr>
<tr>
<td>Accessibility requirement</td>
<td>How to test</td>
<td>Pass / Fail / n/a</td>
</tr>
<tr>
<td>---------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td><strong>Navigation</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Skip links (bypass blocks)</td>
<td>If you cannot visually see a skip to content link - tab through the page and see if a skip link appears. If not, view the html code and look for a skip link option. Also test that the skip link works on all pages.</td>
<td></td>
</tr>
<tr>
<td>WCAG 2.0 2.4.1 Level A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Page titles</td>
<td>Review the page title and check it clearly relates to the page content and is not left blank or just duplicates the homepage page title.</td>
<td></td>
</tr>
<tr>
<td>WCAG 2.0 2.4.2 Level A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Focus Order</td>
<td>Tab through the page and check the sequence makes sense</td>
<td></td>
</tr>
<tr>
<td>WCAG 2.0 2.4.3 Level A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hypertext links</td>
<td>Review the page and check all links make sense in the context of the page, especially make sure that any links with the same text that go to different pages such as ‘More information’ are clearly differentiated by their surrounding text.</td>
<td></td>
</tr>
<tr>
<td>WCAG 2.0 2.4.4 Level A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sitemaps and search</td>
<td>Review the page and ensure that search feature and sitemap are available</td>
<td></td>
</tr>
<tr>
<td>WCAG 2.0 2.4.5 Level A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Headings and labels</td>
<td>Scan the headings and labels on the page and ensure they are descriptive and ideally unique</td>
<td></td>
</tr>
<tr>
<td>WCAG 2.0 2.4.6 Level A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accessibility requirement</td>
<td>How to test</td>
<td>Pass / Fail / n/a</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Focus Visible</td>
<td>Tab through the page and check the sequence makes sense</td>
<td></td>
</tr>
<tr>
<td>WCAG 2.0 2.4.7 Level AA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Location</td>
<td>If in a sequence of pages such as an order form is it clear what page you are on either through the breadcrumb or by a step reference such as step 2 of 5?</td>
<td></td>
</tr>
<tr>
<td>WCAG 2.0 2.4.8 Level AAA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Language reference</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Language of Page</td>
<td>View the page code and check the language has been specified, for example &lt;html lang=&quot;en&quot;&gt; for English in html and xml:lang=&quot;en&quot; for xhtml.</td>
<td></td>
</tr>
<tr>
<td>WCAG 2.0 3.1.1 Level A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Using more than one language</td>
<td>Review the page for any non English words and then review the code to check the right language attribute has been used for the text.</td>
<td></td>
</tr>
<tr>
<td>WCAG 2.0 3.1.2 Level AA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Behaviour</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Focus and content change</td>
<td>Using a keyboard tab through all the selectable elements on screen and check no unexpected changes of context occur when any component receives focus</td>
<td></td>
</tr>
<tr>
<td>WCAG 2.0 3.2.1 Level A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input and content change</td>
<td>Check putting data / making selections for all controls on the page such as form fields does not result in an unexpected change of focus unless there is a clear warning on the page it is going to happen</td>
<td></td>
</tr>
<tr>
<td>WCAG 2.0 3.2.2 Level A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Change of request</td>
<td>Do web links that open in a new window indicate they will do so either in the alt text (if an image) or title of link?</td>
<td></td>
</tr>
<tr>
<td>--------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>WCAG 2.0 3.2.5 Level AAA</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Input Assistance</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Error identification</td>
<td>Systematically go through a form and check all error messages by trying to submit it by missing out or putting in the wrong data into fields.</td>
<td></td>
</tr>
<tr>
<td>WCAG 2.0 reference 3.3.1 Level A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Labels and instructions</td>
<td>Check each field has a descriptive label and where appropriate information has been provided on field formats to minimize input errors and for more complex form a clear overview of what filling in the form will involve.</td>
<td></td>
</tr>
<tr>
<td>WCAG 2.0 3.3.2 Level A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Error prevention</td>
<td>Go through the steps of purchasing a product or service and before submission check there is a stage where you can change or delete all aspects of the order.</td>
<td></td>
</tr>
<tr>
<td>WCAG 2.0 3.3.4 Level AA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accessibility requirement</td>
<td>How to test</td>
<td>Pass / Fail / n/a</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Parsing / Validation</td>
<td>Use the X/HTML validator at: <a href="http://validator.w3.org/">http://validator.w3.org/</a> or a similar tool to check your page code for key errors such as tags not being closed or nested correctly and duplication of id’s on the page.</td>
<td></td>
</tr>
<tr>
<td>WCAG 2.0 4.1.1 Level A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name, role and value</td>
<td>use a screen reader to check all interface controls are accessible for html and all other technologies such as Flash - see Appendix B for a basic Jaws tutorial on testing Flash.</td>
<td></td>
</tr>
<tr>
<td>WCAG 2.0 4.1.2 Level A</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix A - Using Jaws to Evaluate Web Accessibility

Introduction

It is important to evaluate the accessibility of web content with a screen reader, but screen readers can be very complicated programs for the occasional user, so many people avoid them. This doesn't need to be the case. While screen readers are complicated, it is possible to test web content for accessibility without being a "power user."


It works for 40 minutes and you will then to restart your machine to use it again.

While working in JAWS, keep the following guidelines in mind:

- While JAWS can be used for accessing Windows and most Windows applications, here we will be focusing on accessing web content only.
- Make sure that NumLock is off.
- You will probably want to test JAWS in Internet Explorer, even if it is not your primary browser.
- Maximize the browser window.
- Remember that screen reader users typically do not use a mouse. As you become more comfortable with JAWS, try using only the keyboard.
- Keep in mind that most IE shortcut keys will work when using JAWS.
- The page *may not* scroll while you read, so you may hear content being read by JAWS that isn't visible on the screen.
Reading

The most basic function of a screen reader is to read the content on the page. There are dozens of keyboard shortcuts that allow you to read content by line, sentence, word, character, etc. The following is a list of essential reading shortcuts. With these shortcuts, you should be able to navigate through most content.

- **Insert + ↓**: Say All
- **Page Up/Page Down**: Increase/Decrease voice rate while using Say All
- **Ctrl**: Stop Reading
- **Insert + ↑**: Current line
- **Insert + ←/→**: Previous/next word
- **↑**: Prior line
- **↓**: Next line
- **←/→**: Previous/Next character
  - **Rewind/Fast Forward during Say All**
- **F5 / Shift + F5**: Page refresh / Hard page refresh. If you get lost, this is how you can start over.

October 2011  AbilityNet Charity No. England and Wales 1067673 – Scotland SC039866
Telephone: 0800 269545 - email: enquiries@abilitynet.org.uk - web: www.abilitynet.org.uk
Navigation

Sighted users visually navigate through web content in a number of ways. They skim for headings, lists, tables, etc. Most of these methods are available to screen reader users if the site is correctly structured and well organized. To navigate forward and backward through links and form elements on the page, use **Tab** and **Shift + Tab**. Other shortcuts are listed below.

Quick Keys

The following Quick Keys will help you navigate common page elements.

- **H**: Headings
- **F**: Forms
- **T**: Tables
- **N**: Non-link text (i.e., skip to the first thing that isn't a link - useful for skipping table of contents, etc.)
- **B**: Buttons
  - The Search button is often the first button on a page. You can often navigate to the search form but selecting **B** to jump to the button and then **Shift + Tab** to navigate to the previous form element – the search text box.
- **L**: Lists
- **I**: Items in a list
- **1-6**: Headings level 1-6
- **Shift + Quick Key**: Navigate through element in reverse order (works with most Quick Keys)
- **Ctrl + Insert + Quick Key**: For most Quick Keys, this will display a list of that type of element (e.g., **Ctrl + Insert + T** for a list of tables)

Other navigation shortcuts

- **Insert + F1**: Help with current element
- **Insert + F5**: List of Form Elements
- **Insert + F6**: List of Headings
- **Insert + F7**: List of Links
- **Ctrl + Home**: Top of the page
- **Ctrl + End**: Bottom of the Page (**Shift + Tab** when you are at the top of the page will go to the last link or form element on a page)
- **Alt + D** or **F6**: Browser address bar
Data Tables

- There are two main uses for tables on the web: for layout and to organize data. For simple data tables, the use of the `<th>` element will help make information more understandable. Even though most modern screen readers can usually determine appropriate headers for simple data tables, that is by no means a guarantee of accessibility; you should still use table headers appropriately. Screen readers still rely on appropriate headers to navigate through more complex tables.

- To skip to a table, press the T key. To navigate between cells, hold down Ctrl + Alt and use ↑ / ↓ / ← / → to move from cell to cell. Ctrl + Alt + 5 (on the numeric keypad) will read the row and/or column headers for the current cell.
Forms

Forms are used to interact on the web. They allow users to search for content, select and purchase merchandise, to fill out surveys and questionnaires, to register for courses, and a long list of other actions. There are three main things you can do to make forms more accessible for screen reader users:

1. Use the `<label>` element to explicitly associate form controls and their descriptions. If a form control does not have an associated label, a screen reader may attempt to determine the appropriate label based on the proximity of adjacent text.
2. Use the `<fieldset>` element to group related form elements. This usually includes, but is not limited to, checkboxes and radio buttons.
3. Avoid JavaScript jump menus.

Use the following shortcuts to navigate through and interact with forms:
When you access a form element, press Enter to go into forms mode.

- Once you are in forms mode, use Tab and Shift + Tab to navigate through the form controls.
- Simply begin typing in text, password, or text area elements. Quick keys are disabled when you are inserting text in forms mode.
- Use Spacebar to select and deselect checkboxes.
- Use ↑ / ↓ to select from a group of radio buttons.
- Use ↑ / ↓ or the First letter to select an element in combo boxes.
- If you would like to leave forms mode, hit the + key on the numeric keypad.
- JAWS automatically leaves forms mode when you press Enter to submit the form, navigate to a new page, or select a button.
Navigating a web page example

- When Internet Explorer is loaded press Ctrl+O to open the Open dialog box. Type “www.abilitynet.org.uk”, then press Enter
- You are now in the home page of the AbilityNet website
- Press Ctrl to stop Jaws reading the whole page

You will find that, even on a “screen reader friendly” site such as ours there is a lot of text to plough through. The first items encountered on a web page are invariably menus and the feeling of having to plough through text to get to what you want to hear is a very common one. There are several ways you can alleviate this and jump more quickly through a page’s contents – i.e. Ctrl + Up/Down Arrows to jump up and down through blocks of text, or B to jump to a button, X for a checkbox etc. Refer to your ‘Jaws and Internet Explorer Hot-Keys’ crib for the full list.

Use some of these keystrokes to review the web page. You should hear a different voice spoken to represent links, headings and images. Press Alt + F1 to hear what any items you are unsure of are (don’t forget to close all messages with Esc).

- Press Ins+F7 to bring up the links list box for this page
- Use the up and down arrows to find the link “Sign up for our eNewsletter” (you can also press the first letter to jump to items in a list) and then tab around the dialog box to familiarise yourself with its contents.
- Tab to Move to Link (this will take you to this link on the page, as opposed to activating it)
- Arrow down to read the text underneath
Appendix B - Testing Flash with Jaws

Overview
This tutorial explains the basics of how the Jaws screen reader works and how it interacts with Flash.

Jaws is the most popular screen reader used by visually impaired people in the UK. Other common screen readers include Window-Eyes and Hal.

Audio feedback
A screen reader user relies on audio feedback to understand what is happening on screen. To successfully use a screen reader you need to learn keyboard commands for interacting with the computer and also be a proficient touch typist.

Because of the reliance on audio for information it is very important that all non text visual elements on a screen have an equivalent textual description. Failure to provide this is one of the most common accessibility issues.

Issues with Flash
This issue is particularly important with Flash because by default most movie clips and buttons don't have descriptive labels - they are often unlabelled or have an inadequate description. There is also an issue with text in Flash, for accessibility you should always use dynamic rather than static text for words on screen as a screen reader does not read static text at all. Label text for input fields is also accessible.

Accessibility panel
To make an element on screen accessible you need to ensure that in its accessibility panel the checkbox 'make object accessible is checked'. This is typically all you need to do for dynamic and label text but for buttons and movies you usually need to add a description in the name field to give extra information about the object.

Changing elements on screen
Another issue is indicating to a screen reader user when the state of a movie or button has changed for example when an option in a multi choice question is selected. The only way to do this is to change the name description dynamically using ActionScript using the accprops function.
Walk through using Jaws
Because it is easy to make a mistake adding descriptions to elements in a Flash movie it is important to test with the Jaws screen reader. Learning the most commonly used keyboard commands will help you get started using Jaws.

Starting Jaws
When you first load Jaws it will start speaking out information about where it has focus this is typically the desktop. To run Jaws through a Flash movie start the file in Internet Explorer. Note at this point in time Firefox only has very limited screen reader support.

Initial automatic read through
When you bring up a Flash page Jaws will normally say 'Flash movie start' and proceed to read at a rapid pace through the movie - often a user will read through a page like this to orientate themselves and then go back to the beginning.

When the screen reader hits the end of the movie it typically announces this by saying 'Flash movie end'. To go back to the beginning of the Flash movie press Ctrl + Home at which point it will normally read out the page title and then continue with 'Flash movie start'.

Also to go back to the start at any point during the initial rapid reading of the page hit the Ctrl key this will normally take you back to the start of the Flash movie.

Moving through the movie at your own pace
To start moving through the movie at your own pace press the down arrow to go to the first object on the screen which is typically the next button, press the down arrow again to go to the next element on screen which should be the previous button - this sequence is known as the reading order. To go back press the up arrow which will read the previous element, in this case the next button.

You can speed up the process by pressing Ctrl + down arrow to jump blocks on the screen but you should only do this if you are familiar with what is on the screen as it can miss out important information navigating in this way.

When you come across a selectable movie or button such as a multiple choice question you can select it by hitting the Enter key - on screens 1 and 2 you should get audio feedback to say the button has been selected.

Different reading modes
The default reading mode when going through a page is called virtual cursor mode however when Jaws encounters a form field it typically changes to forms mode so you can input data into the field for example your answer to the pie chart question on the visual interaction screen. Once you have input your data you can press Tab to go to the answer or submit button and then press Enter. You should at this point revert
to virtual cursor mode but sometimes is does not as in screen 5 and you have to manually reset to virtual cursor mode by pressing the plus key on the number pad.

You can then continue through the movie using the down arrow

**Links list**
Also there is a shortcut way of listing all the links and form fields on the page by pressing Insert + F5 which will bring up a list of options use the up and down arrow keys to make your choice and then hit Enter to activate the link or go to the input field.
Table of Contents

Ajax Best Practice checklist..................................................................................................................3

Introduction........................................................................................................................................4

Key accessibility issues with Ajax......................................................................................................7

User does not know updates will occur............................................................................................8
Users does not notice an update..........................................................................................................8
Unexpected changes in focus................................................................................................................8
Browser behaviours changed..............................................................................................................8
Users may not have the technology available or enabled.................................................................8

Ajax Best Practice ................................................................................................................................9

Be clear Ajax is the best solution........................................................................................................9
Inform users early in the page that dynamic updates will occur........................................................9
Highlight the areas that have been updated.......................................................................................10
Don’t change the focus ........................................................................................................................12
Replace element instead of creating new one ....................................................................................13
Offer the option to disable automatic updates..................................................................................13
Ensure the site works if JavaScript isn't enabled ...............................................................................13
Use progressive enhancement / Hijax for backward proofing...........................................................14
Ensure keyboard accessibility............................................................................................................15
Use ARIA for future proofing.............................................................................................................16

Presenting Document Structure ......................................................................................................17

Dynamic Content Updates.................................................................................................................18

Enhancing Keyboard Accessibility..................................................................................................19

Widget Accessibility............................................................................................................................20

Examples............................................................................................................................................21

Scripting libraries...............................................................................................................................22

ARIA tutorials and examples ............................................................................................................22
Ajax Best Practice checklist

Below are the key issues to consider when developing websites with Ajax, they are provided for easy reference only; you should refer to the full document for detailed guidance.

- Always consider if Ajax is the best solution
- Inform users early in the page that dynamic updates will occur
- Highlight the areas that have been updated
- Don't change the focus
- Replace element instead of creating new one
- Offer the option to disable automatic updates
- Ensure the site works if JavaScript isn't enabled
- Use progressive enhancement / Hijax for backward proofing
- Ensure keyboard accessibility
- Implement ARIA for future proofing
Introduction

Ajax is an innovative way of using existing technologies to create highly interactive web applications. Ajax allows portions of the page to be updated without having to refresh and reload the entire page. It can increase site performance and site usability significantly. Unfortunately it can cause accessibility issues if not implemented considering the various ways users will interact with the site – for example using a screen reader or relying on just the keyboard to navigate.

The classic example of Ajax is googlemaps – where the ability to scroll smoothly is a result of Ajax.

Fig 1 Googlemaps is a classic example of Ajax enhancing functionality.

Examples of Ajax implementation are mapping applications, real-time searches, chats, real-time form validation and processing, live shopping cart applications, e-mail applications, and accordion menus.

Ajax stands for Asynchronous JavaScript And XML. Ajax itself is not a technology — it is a combination of technologies used in a certain way. Ajax applications work 'behind the scenes' with the web server to dynamically update the content of a web page.

Ajax uses XHTML (or HTML) and CSS to create an interface and visually style information. The XHTML and CSS elements can be dynamically modified to display new information or styling. These changes are typically done using the Document Object Model (DOM).

JavaScript is used to manipulate the interface elements and is also used to establish a communication with a web server through the XMLHttpRequest DOM object. This allows data (typically in XML format) to be transmitted between the client and the web server.

**Good use of Ajax**

Ajax can add significant benefits to how a site works and can often enhance accessibility if implemented well. For example Auto-suggest drop downs can really help users with Dyslexia and other reading difficulties, and also means people with motor impairments have to type less.

For example the National Rail enquires bring up station suggestions as you type.

![National Rail Enquiries](image)

*Fig 2 Ajax used to enhance usability and accessibility in a dropdown menu*

In addition Ajax can be used to stop the user from having to take redundant steps - for example when signing up for an online account rather than having to repeatedly check for available logins Ajax can be used to check the availability of usernames without having to submit the signup form.

Other benefits include:
- To keep a consistent interface – making the web more like an application than a web page
- Data entry fields that are specific to the current page (for example titles and descriptions in Flickr)
- Automatic storage of data in a certain interval to prevent data loss
- Upload of files while the user is entering more data

**Bad use of Ajax**

Ajax is a tool that can enhance websites but it is also quite easy to use it badly.
For example in a form submission the purpose is to take data from a form and submit it to the server. Using Ajax introduces several potential problems, such as it breaks the expected behaviour for your users who expect a page refresh – a particular issue for screen reader users, also the risk of multiple submissions increases as users think the form does not work properly. And you will still need to provide a non-JavaScript solution for those using browsers or other tools which cannot handle JavaScript, or who have it turned off.

Other key things to avoid when using Ajax

- Adding any functionality that makes using the page dependant on the browser having JavaScript enabled
- Changing so much in the page that the user feels inclined to bookmark it (which they can’t)
- Changing so much that the user wants to go Back (which they can’t)
**Key accessibility issues with Ajax**

By its nature, Ajax tends to update and manipulate interface elements 'on the fly'. Ajax also can submit information to the server without user interaction or may do so in methods that are not obvious to the user. For example, most users expect forms to be submitted, validated, and processed when a submit button is selected, but with Ajax this submission and processing can occur at any time.

The reality of Adaptive technology (AT) such as Screen readers and Voice Recognition software is that it does have JavaScript support so you need to think how this software will work with Ajax on your website.

Screen readers used by people with visual impairments struggle the most with inaccessible Ajax because they rely on a webpage giving them information about what they can do and where they can go – Ajax breaks this model and means Screen reader users can be very confused as to what is happening on a page that uses Ajax to update content.

This section explains what the key accessibility issues are with Ajax, in summary;

- User does not know updates will occur
- Users does not notice an update
- Unexpected changes in focus
- Browser behaviours altered
- Users may not have the technology available or enabled
User does not know updates will occur
When a screen reader hits a website it takes a snapshot of the web page, and places this content in a virtual buffer – and then uses this to interact with the page. The problem with Ajax is that the way it works means this virtual buffer might not be updated when Ajax makes a change to the page so the screen reader user may not be aware anything has happened. This will be very disorientating especially if they have been given no warning that parts of the page will update.

Users does not notice an update
For users with moderate vision impairment who use Screen magnification software who zoom the page on average by x4 or x5 magnification they might not notice changes that have occurred outside the areas they're interacting with. They can therefore miss out on important information especially if the changed content takes place above their current location on the page. Also sighted users may not notice the updates because they are distracted or have cognitive, memory or attention impairments.

Unexpected changes in focus
If your Ajax automatically shifts focus on the page when an update occurs it can be very distracting for some users, especially if there is no easy way to return to the previous position. This is particularly true for screen reader users as they will have no visual indicator that anything has happened.

For example, on a site that dynamically updates a series of share prices located at the top of the page, a sudden shift in focus each time a price change occurs could make the page unusable.

Browser behaviours changed
While Bookmarks and the back button work for traditional, multi-page web applications; as users surf websites, their browsers' location bars update with new web addresses (Urls) that can be pasted into emails or bookmarked for later use. The back and forward buttons also function correctly and move users between the pages they have visited.

Ajax applications are different; they are sophisticated programs that live within a single web page. Browsers by default are not able to cope with the dynamic changes that Ajax introduces so pages cannot be bookmarked reliably and the back button not longer works.

Users may not have the technology available or enabled
Ajax requires JavaScript to be enabled. Although assistive technologies can now handle many uses of JavaScript they don't all provide complete support. In addition many organisations do not allow staff to use JavaScript enabled browsers for security reasons and mobile support for JavaScript is not uniform.
Ajax Best Practice

Below is guidance on the best approach to using Ajax accessibly on your website.

Be clear Ajax is the best solution

While Ajax can enhance a webpage it is not always the simplest or best solution. Always be clear what functionality the web application needs, and review if Ajax is the best solution. Use Ajax for things that really need and benefit from it — where it's the only way to make something work.

Also be aware grafting Ajax onto existing applications adds another layer of complexity to your application. And in a lot of situations it will duplicate functionality that already exists. Ajax inherently does things differently than a postback application. And with the various .NET Ajax frameworks you'll end up either working that logic into the existing page (script callbacks, My Ajax.NET) or in external handlers (Ajax.NET).

Inform users early in the page that dynamic updates will occur

Not all users are familiar with Ajax interfaces and screen reader users have no reliable way of knowing Ajax is being used unless you inform them, also magnifier users are often unaware that changes are taking place on a page unless you tell them to look out for the changes.

So a critical accessibility requirement is to let visitors to your site know that changes may take place so they can expect and look for these changes. Place this information near the top of the page and ensure it standouts.
Below is an example of an Ajax application, a calculator that gives information that the form requires JavaScript and will update the page dynamically.

You can see the form at: http://www.standards-schmandards.com/exhibits/ajax/v2.php

Figure 3 an Example of Ajax where information is provided that the page will be updated

It is strongly recommend that you ensure your pages function without JavaScript; but at the very least you should use the HTML <noscript> tag to provide an explanation of what functionality is available on a webpage if a user comes to the page with a browser that does not support JavaScript or has it turned off

Highlight the areas that have been updated

To address the issue of users not noticing updates have happened you need to provide clear indicators that updates have occurred. The most reliable visual approach is to use colour and animation.

Using colour alone is not clear enough as some people cannot distinguish between certain colours or cannot see colour at all. So any change needs a high level of contrast; the best approach is to use colours that sit opposite each other on the colour wheel – known as complementary colours.

Figure 4 Example of high contrast with complementary colours
When used with equal brightness, these colours have a tendency to compete with each other for attention (a phenomenon referred to as simultaneous contrast). As a result, they can be used effectively to bring attention to areas of change.

Animation is also effective because of the visual systems natural attraction to motion. Incorporating a simple animation in a page lets users know something is happening in response to an action they make.

A good example is known as the yellow fade technique which was pioneered by 37Signals for use in their online project management application Basecamp, a web-based project management and collaboration tool. See the original article - Web Interface Design Tip: The Yellow Fade Technique at http://37signals.com/svn/archives/000558.php

The technique uses a combination of both colour change and animation to highlight updated interface elements.

If you look at Figure 5 below you can see an example of how the technique is implemented

![Communicating Change](http://www.lukew.com/resources/articles/ajax_design.asp)

**Fig 5 Example of an update change using Ajax**

Source: [http://www.lukew.com/resources/articles/ajax_design.asp](http://www.lukew.com/resources/articles/ajax_design.asp)

The animation must not be too quick or last too long so you have the balance between a user noticing it and not being annoyed by its intrusiveness.

See the article Ajax; The "Yellow Fade" Technique [http://www.dmxzone.com/go?11436](http://www.dmxzone.com/go?11436) or more detailed information about how it can be applied to JavaScript.

**Screen reader users**

Unfortunately this approach does not help screen reader users who will not see the colour change or animations on screen.

When a screen reader hits a webpage it takes a snapshot of the web page, and places this content in a virtual buffer – and then uses this to interact with the page. In older versions of screen readers such as Jaws before version 7.1 the user had to manually update the virtual buffer so they could access a version of the page that had been modified by Ajax but in newer versions (9+) the virtual buffer is updated automatically. This however does not solve the problem of them being notified there has been a change. One approach is to use
the JavaScript focus() to move focus to changed content but this is not reliable and can cause disorientation see next point – Don’t change the focus.

The longer term solution is for screen reader and other Assistive technologies such as Magnifiers to provide support for WAI ARIA live regions which will enable feedback to be given on changes to a page. For more details see the section ‘Implement ARIA for future proofing’.

Don’t change the focus

Do not automatically shift focus on the page when an update occurs. Changing focus without warning can be very distracting for some users, especially if there is no easy way to return to the previous position. For example, on a site that dynamically updates a series of stock prices located at the bottom of the page, a sudden shift in focus each time a price change occurs could make the page unusable. Instead highlight the areas that have been updated as discussed in the previous.

Also if you add a large amount of content to a webpage through an Ajax update the screen may “jump” to make room for the new elements. This may confuse users – especially Screen magnification users looking at a zoomed in version of the webpage (typically x4 or x5) as they may suddenly find themselves looking at another section of the page than they were before the update. So take care to minimize the amount of screen “jumps” so the focus is not changed too much on the page.

Some changes in focus, however, may be appropriate. If the user has activated a button to check for new mail, it may be correct to move focus to the list of mail messages when the update has completed. Make certain that the user is aware that an action is going to occur and expects to focus to change to the results.

In addition one technique to help screen reader users is to provide internal page links like skip links that are shortcuts to parts of the page which are dynamically updated via Ajax regularly to give them a quick way to navigate to the new content.
Replace element instead of creating new one

When updating the page with Ajax replace data where possible rather than creating and adding new elements to the page. This is because not all Adaptive technology (AT) can handle dynamic additions to a page via the Document Object Model (DOM). When adding elements, append them at the end of a page or parent them to an existing element rather than the document itself. Be aware that adding navigation elements such as additional menu sections within the page may affect the page tab order for AT users so you need to ensure you test your pages with AT’s ensure this is not going to cause a problem.

Offer the option to disable automatic updates

A solution to give Adaptive technology (AT) users more control in how they interact with the page is to give them the ability to manually request page updates, for example by providing links and/or form buttons to refresh the page on-demand. Screen reader and magnifier users may be unaware of on-the-page changes. It can also be difficult for users with reading difficulties to keep up with automatic updates. If possible, store users' preferences for requesting page updates for future visits to the site.

Ensure the site works if JavaScript isn't enabled

As its name suggests, Ajax requires JavaScript. This means that Ajax applications will not work in web browsers and devices that do not support JavaScript. For best practice if you use Ajax you should ensure that the features or functions on the page such as a site search will work when JavaScript is disabled or not supported.

A good practice example of this is the site http://www.fixmystreet.com/ which uses Ajax as a way of interacting and scrolling an area map –see Figure 6.

Figure 6 Fix my Street with JavaScript disabled
With JavaScript disabled the map is still presented you just lose the ability to dynamically scroll around the map but direction buttons are present to allow you to still move around the map.

In case of an advanced Ajax application you may need to consider providing an HTML alternative. If the Ajax application is impossible to use by group of users for example because of keyboard accessibility issues then a link to an HTML alternative is a must.

**Use progressive enhancement / Hijax for backward proofing**

Progressive enhancement (PE) is a strategy for web design that emphasizes accessibility, semantic markup, and external style sheets and scripting technologies.

Progressive enhancement uses web technologies in a layered fashion - starting at the most basic and increasing in complexity.

For example below is an example of how you would code a page with Ajax functionality:

- **Layer 1**: HTML code for the most basic browsers.
- **Layer 2**: enhancement for browsers that support CSS
- **Layer 3**: The next level of enhancement is for browsers that support JavaScript.
- **Layer 4**: A separate JavaScript function to ensure cross browser support for Document Object Model
- **Layer 5**: Ajax functionality

This approach allows everyone to access the basic content and functionality of a web page, using any browser or Internet connection, while also providing those with better bandwidth or more advanced browser software an enhanced version of the page.

For more detail and tutorials see:


**Hijax**

Hijax is a technique of applying progressive enhancement to Ajax which has been created by Jeremy Keith author of Bulletproof Ajax (ISBN-10: 0321472667)

In a nutshell the approach is:

- Plan for Ajax from the start.
- Implement Ajax at the end.

For more information see [http://www.uie.com/articles/hijax/](http://www.uie.com/articles/hijax/)

**Ensure keyboard accessibility**

Keyboard accessibility also needs to be considered when using Ajax for functionality, many Ajax applications have limited or no keyboard support – for example the standard Google maps interface has no keyboard support.

This has lead to some developers implementing more accessible versions for example in the screenshot below in **Figure 7** is an example of a keyboard accessible Google maps where you can tab to the directional arrow keys and use the ‘Enter’ key to navigate around the map.

![Figure 7 Keyboard accessible Google map by Patrick H. Lauke](image)

See the article Keyboard-accessible Google Maps by Patrick H. Lauke that explores this issue: [http://dev.opera.com/articles/view/keyboard-accessible-google-maps/](http://dev.opera.com/articles/view/keyboard-accessible-google-maps/)
Use ARIA for future proofing

WAI-ARIA is a W3C protocol for enhancing and supporting accessibility of scripted and dynamic content. ARIA provides accessible interactive controls (such as tree menus, drag and drop, sliders, sort controls, etc.), content roles for identifying page structure (navigation, search, main content, etc.), areas that can be dynamically updated (called "live regions" in ARIA), better support for keyboard accessibility and interactivity, and much more.

Most up-to-date browsers, script libraries and screen readers support ARIA. However only Internet Explorer 8 supports ARIA, previous versions have no or little support.

Although ARIA is not yet universally supported, when used with existing HTML and scripting accessibility techniques, it can provide additional accessibility support where it is supported while not causing compatibility issues where it is not yet supported.

Web sites are increasingly using more advanced and complex user interface controls, such as tree controls for Web site navigation. To provide an accessible user experience to people with disabilities, assistive technologies need to be able to interact with these controls. However, the information that the assistive technologies need is not available with most current Web technologies including Ajax.

Another example of an accessibility barrier is drag-and-drop functionality that is not available to users who use a keyboard only and cannot use a mouse. Even relatively simple Web sites can be difficult if they require an extensive amount of keystrokes to navigate with only a keyboard.

Many Web applications developed with Ajax (also known as Ajax), DHTML, and other technologies pose additional accessibility challenges. For example, if the content of a Web page changes in response to user actions or time- or event-based updates, that new content may not be available to some people, such as people who are blind or people with cognitive disabilities who use a screen reader.

WAI-ARIA addresses these accessibility challenges by defining how information about this functionality can be provided to assistive technology. With WAI-ARIA, an advanced Web application can be made accessible and usable to people with disabilities.

WAI-ARIA is discussed in more detail in the rest of this section.
Presenting Document Structure

With current versions of HTML and XHTML, there is no mechanism for identifying the function or purpose of page elements in a programmatically determinable way - in other words, you cannot, in your code, identify your main content, navigation, search, etc. as such. While you can and should use a proper heading structure in your document, this does not provide a standard way for users to access or determine the semantic role of page elements.

This problem is highlighted by the need for "Skip to main content" or "Skip navigation" links. Because a browser or screen reader has no way of knowing what portion of the page contains the navigation elements, it is necessary for site authors to create a specific link which allows keyboard users to bypass those links. Even directly accessing commonly used page functionality, such as search requires that the user browse through or listen to the page and find or discover it.

WAI-ARIA provides the ability for developers to specify roles for document areas (and many other things). The available Document Landmark Roles are:

- **Banner**: Site-orientated content, such as the name of the web site, title of the page, and/or the logo
- **Navigation**: The area that contains the navigation links for the document or web site
- **Main**: The main or central content of the document
- **Search**: This section contains the search functionality for the site
- **Article**: Stand-alone content that makes sense out of context from the rest of the document. Examples might be a blog posting, a comment on a blog, a forum post, etc. Specifically, a blog posting might be identified as an article and individual blog comments might also be marked up with a role of article within that blog posting.
- **Complementary**: Supporting content for the main content
- **Contentinfo**: Informational child content, such as footnotes, copyrights, links to privacy statement, links to preferences, and so on.
Most pages can apply at least a few landmark roles now - `<ul role="navigation">` or `<div role="main">` or `<form role="search">`. This function of ARIA is not limited to rich applications, but can be applied to nearly any current web page.

Assistantive technologies such as screen readers are beginning to provide support for document roles. They may provide shortcut keys to jump to specific structural elements (for instance, S for search) and/or provide a list of all structural roles in the document. Additionally, ARIA provides new roles for other types of elements, such as `role="presentation"` for tables used in layout. It also allows you to identify required form elements, provide better form labelling and descriptions, and allow you to provide instant feedback to screen reader users.

**Dynamic Content Updates**

When content changes dynamically within a web page, it may cause accessibility problems. What happens if a screen reader is currently reading an element that is updated? If the updated content is important, should you interrupt the user and set focus immediately to the new content, do you simply inform the user of the update, or do you do nothing? How do you set focus or allow the user to jump to the updated content?

With standard scripting, the developer must dictate what happens when a content update, such as an Ajax-driven feedback message, occurs. The developer can simply allow the update to occur and not inform the user of it, alert the user of the update through some sort of embedded audio sound, or can set focus directly to the updated content. The developer must script each of these situations, thus removing the control from the user.

With WAI-ARIA, the developer can identify regions that dynamically change as a live region changes. A live region allows content updates in a way that a screen reader understands. It also allows the developer to add additional functionality to alert the user, provide controls for the live region, determine the amount of new content that would be read, and much more.

To create a live region, the developer adds the `aria-live` property to the element with a value of off, polite, assertive, or rude. The value, or politeness level (or alternatively the intrusiveness level), specifies what a screen reader should do when the element is updated.

A value of off (`aria-live="off"`) tells the screen reader to not announce the update. If/when the screen reader user encounters the updated content; it will be read at that time. This would be used for non-important or irrelevant content updates.

A value of polite will notify the user of the content change as soon as he/she is done with the current task. This may take the form of a beep or an audio indication of the update. The user can then choose to directly jump to the updated content. This value would be the
most common for content updates, especially for things like status notification, weather or stock updates, chat messages, etc.

An aria-live value of assertive will result in the user being alerted to the content change immediately or as soon as possible. Assertive would be used for important updates, such as error messages

aria-live="rude" is reserved for the most critical of updates. It would immediately inform the user of the update and possibly set focus immediately to the updated content.

The high level of fidelity with ARIA live regions allows great flexibility both for developers and for end users.

Enhancing Keyboard Accessibility

In HTML, only links and form elements can receive keyboard focus. This means that as you 'tab' through a page, the browser stops or sets focus only on these types of elements. With scripting, however, you can add mouse interactivity to nearly any element. This means you can make normal page elements, such as a paragraph or span, interactive and responsive to the mouse (e.g., you can make plain text display and behave like a button). The functionality of these non-focusable elements cannot be made accessible to screen reader and keyboard-only users. Developers often have a need for page elements other than links and form elements to be interactive.

Additionally, it is often necessary to set focus to page elements. For instance, a form validation error message might be displayed as text (not a link or form element) within a page using scripting. Visual users can immediately see the error message. However, a screen reader user may not know that the new message is present. In order to set focus to the error message so it can be read by a screen reader, the message must be able to receive focus - something it cannot typically do unless it is a link or form element.

ARIA again provides mechanisms for non-focusable elements to receive focus through the tabindex property. ARIA extends the tabindex property so it can be applied to any element. By setting a tabindex value of 0 (tabindex="0"), an element will be placed in the tab order of the document. This means that the browser will stop and set focus to the element in the navigation order of the document (e.g., when the user tabs to the element). This allows additional functionality and interactivity to be applied to the element, such as triggering functionality when the element receives keyboard focus or when the user presses a key while the element has focus.

A tabindex value of -1 allows an element to receive focus, but only when that focus is set programmatically - meaning that the user activates a link to the element (<a href="#maincontent"> ...) or focus is set with scripting (e.g., document.getElementById('errormessage').focus();).
By expanding the focus capabilities in the web browser to elements that otherwise cannot receive focus, ARIA allows additional possibilities for providing accessibility.

**Widget Accessibility**

The term 'widget' is broadly used to describe interactive elements that are created and controlled through scripting. It refers to controls that are not native to HTML or to HTML controls that are greatly enhanced through scripting.

Examples of widgets would include sliders, drop-down and fly-out menus, tree systems, drag and drop controls, auto-completing text boxes, and tooltip windows, to name a few. Accessibility of widgets does not happen natively or naturally. While scripting can often provide accessibility for many of these elements, there is not an easy or standardized way of doing so. In short, making these widgets keyboard and screen reader accessible is often possible, but it is rather difficult and the end result is often only marginally accessible for practical purposes.

Similar issues arise with other widgets. How do you make a drag-and-drop element keyboard accessible? How do you present ordering details for sortable list items? How do you present visual tooltips or pop-up messages to blind users?

ARIA addresses many of these issues. By establishing a set of roles, properties, and values, ARIA allows developers to address these accessibility issues with relative ease.

The following provide some *very basic examples* of how ARIA might be implemented.
Examples
This example below demonstrates a solution to a common problem - the form element is required, but the word "required" is not contained within the label for the form element and so is unlikely be spoken by a screen reader. ARIA's aria-required="true" will inform a screen reader that the specified form element is required.

```
<label for="name">First Name</label>: <input name="name" id="name" aria-required="true"> <em>(required element)</em>
```

Item state
WAI-ARIA provides the ability for developers to specify states of document or form items.

The example below provides the state information for the button to assistive technology.

**Button not selected**

```
<input type="image" src="hioff.gif" alt="highlight" role="button" aria-pressed="false">
```

**Button selected**

```
<input type="image" src="hion.gif" alt="highlight" role="button" aria-pressed="true">
```

Styled text
In this example, the text is styled to visually appear like a button (used in many interfaces where it is difficult to style a normal form button). The role="button" tells the browser that the text should behave as a button. tabindex="0" puts the element into the keyboard navigation flow so keyboard users can activate the button. This example demonstrates the ability for ARIA to add accessible interactivity to any element.

```
<span style="background-color: #ddd; border: medium outset white;"
role="button" tabindex="0"
onkeydown="if(event.keyCode==13 || event.keyCode==32) alert('You activated me with the keyboard');"
onclick="alert('You clicked me with the mouse');">Push Me</span>
```
Scripting libraries
For complex widgets (interactive elements that are created and controlled through scripting) and interactions, it is highly likely that a developer, rather than building from scratch, will use existing scripting libraries to create such elements. Fortunately, ARIA is being implemented into many scripting libraries (such as jQuery, Dojo, YUI, and GWT). While developers can certainly implement ARIA into their advanced widgets and applications, using ARIA-supported libraries greatly simplifies the process of providing this level of accessibility.

ARIA cannot solve all accessibility issues and can be a bit difficult to implement, but it provides much greater levels of accessibility than is currently available through HTML and scripting alone. As support in browsers, assistive technologies, and scripting libraries increases, ARIA is becoming a great tool for ensuring accessibility of rich internet content and applications.

ARIA tutorials and examples
There are lots of good ARIA tutorials and examples to ARIA implementation at http://wiki.codetalks.org/wiki/index.php/Main_Page#List_of_ARIA_Resources

http://wiki.codetalks.org/wiki/index.php/Web_2.0_Accessibility_with_WAI-ARIA_FAQ

and

http://www.w3.org/WAI/intro/aria.php
# Table of Contents

Checklist .................................................................................................................................................. 3

Introduction ............................................................................................................................................... 4

Background ............................................................................................................................................... 5

Flash Accessibility from a users perspective ......................................................................................... 7
  Visual impairments .................................................................................................................................. 7
  Deaf and Hard of hearing ......................................................................................................................... 10
  Mobility impairments .............................................................................................................................. 10
  Dyslexia .................................................................................................................................................. 12
  Cognitive impairments ............................................................................................................................ 12
  Learning Disabilities ............................................................................................................................... 12

Adobe Flash accessibility guidelines in detail ......................................................................................... 13
  Assign text equivalents ............................................................................................................................ 14
  Animations ............................................................................................................................................. 16
  Accessible components ............................................................................................................................ 17
  Enable control over reading order ........................................................................................................ 18
  Keyboard accessibility: Facilitate keyboard access to all controls ..................................................... 19
  Expose structure .................................................................................................................................... 22
  Expose state of controls .......................................................................................................................... 23
  Use color wisely ...................................................................................................................................... 24
  Validate for accessibility ......................................................................................................................... 25
  Testing checklist ................................................................................................................................... 25

Additional issues ........................................................................................................................................ 26
  Embedding Flash in a webpage .............................................................................................................. 26

Appendix A - Testing Flash with Jaws ..................................................................................................... 27
Checklist

Below are the key issues to consider when building Flash, they are provided for easy reference only; you should refer to the full document for detailed guidance.

- Don’t disable the Flash Zoom In / Zoom out feature
- Don’t use a font size less than 12pt
- Use consistent page layout for your Flash - specifically for navigation
- Ensure links and buttons are a reasonable size to aid those with mobility impairments
- Do not disable the tab focus so keyboard users know where they are
- Use left justified text as much as possible
- Use a clear serif font
- Uses icons and graphics to supplement the text
- Ensure every non text element such as a movie clip or button on the page you want to make accessible has a text description.
- Only make the top level of animations accessible (make child movie inaccessible)
- Avoid continuous movement - if not possible provide a way for the use to stop or pause the animation
- Avoid flickering or flashing animation
- Use accessible components such as radio buttons and turn on their accessibility
- Create a logical reading order through the Flash movie
- Ensure all Flash controls are keyboard accessible
- Expose structure to screen readers by providing additional orientation information about the Flash
- Expose state of controls - make sure when elements change on screen the text equivalent is updated
- Use colour with care - do not convey information by colour alone
- Ensure good colour contrast between foreground and background
- Check for accessibility using the testing checklist and a screen reader
Introduction

This guide gives practical guidance on Flash accessibility.

Firstly it explains some of the background to Flash accessibility. It next goes on to explain about Flash Accessibility from a users’ perspective - exploring the issues for vision, hearing, mobility and cognitive impairments.

The main section of the guide then explains the key Flash accessibility issues based on the Adobe Flash accessibility design guidelines

This guide assumes you have a basic understanding of Flash and have some familiarity with ActionScript.
Background

Flash has made its powerful mark on the web and on standards of design in only a few short years. It is one of the most popular ways to present engaging dynamic content on the web – both because of its speed (vector-based animations, which can adapt to different display sizes and resolutions, play as they download) and for the smooth way it renders graphics.

Adobe has steadily improved how well Flash works with adaptive technology with each new release. It is not perfect but it has come a long way from the early days of 'total inaccessibility'. The next challenge with Flash is for developers to start using the accessibility features built into Flash to give disabled users a positive experience of the rich media.

Although Flash presented accessibility barriers for many people with physical disabilities, it is important to realise it is a medium that can enhance accessibility for people with cognitive and learning disabilities because of its visual approach.

Platform specific accessibility - MSAA

It is important to note that because Flash accessibility relies on Microsoft Active Accessibility (MSAA) as the link between Flash and assistive technologies it will only work on the Windows platform and with only a number of the most popular ATs such as the Jaws and Window-Eyes screen readers. So at this moment in time you cannot make Flash fully accessible on the Mac or Linux platforms.

MSAA operates in the Windows operating system, much like a database; as content is played by the Flash player, it's pushed into this database. On the other side, a screen reader or other assistive technology picks up that information and renders it to the user on the other side so they can make sense of it.

Fig 1 MSAA sits in-between the Flash player and the Assistive technology such as a screen reader.

---

Flash accessibility guidelines

As a rich media technology Flash did not fit well within the First Web Content accessibility guidelines (WCAG 1.0) that were created in 1999, which focused very much on html and CSS. This meant that developers had very little guidance in the early days of Flash on how to address accessibility issues – beyond the guidelines requiring text alternatives, scripts to be accessible with adaptive technology and work with the keyboard.

The Web content Accessibility Guidelines 2.0 (WCAG 2.0) which was launched in December 2008 is more technology agnostic – being based on four principles - perceivable, operable, understandable, and robust.

WCAG 2.0 also introduces a new concept - ‘Accessibility Supported’ – using a technology in a way that is accessibility supported; means that it works with assistive technologies (AT) such as screen readers and the accessibility features of operating systems, browsers and other user agents.

This is helpful in setting a framework to make Flash accessible but it does not give detailed techniques for how to code accessible Flash.

Enter Adobe Best Practices for Accessible Flash Design…

To address the needs of developers Macromedia (before they merged with Adobe) in 2005 created Flash accessibility design guidelines which explain in detail how to code Flash accessibly.

The guidelines:

- Uses examples of disabled users to give some insight into Flash accessibility
- Does not prioritise accessibility guidelines – no levels as in W3C WCAG
- Is the most comprehensive guidance currently available

These guidelines are the basis of the Flash accessibility best practice covered in this guide.

You can download a copy here: Best Practices for Accessible Flash Design (PDF, 643k)

Adobe have also has produced a useful resource which covers the key principles of making Flash accessible, available on their website in a web friendly format.

Flash Accessibility from a users perspective

Historically Flash accessibility issues have heavily focused on screen reader users but it is important to be aware of the range of issues that affect other disability groups. This section outlines the most common accessibility issues for disabled people.

Around 11 million people in the UK have a disability under the Disability Discrimination Act DDA (note the Equalities Act will replace the DDA from October 2010). The groups that have specific issues with web and intranet accessibility are:

- Vision impairment
- Hearing impairment
- Motor difficulties
- Cognitive and learning impairments

Many have more than one disability

Visual impairments

Mild vision impairment

The main issue for people with mild vision impairment is the need to enlarge the text size – or at least a good clear font of at least the equivalent of 12pt and have good colour contrast.

This group also might potentially use the Zoom Flash feature – Zoom In / Zoom Out as shown below in Figure 2 which can be accessed by right clicking on the Flash movie. Unfortunately many users are not aware of this option and those who are aware do not always find it helpful, especially as there are no scroll bars - users have to use the mouse to navigate around the enlarged movie. Also this feature is frequently disabled by developers so might not be available at all.

![Figure 2: The zoom function available in Flash](image-url)
Moderate vision impairment - Magnification users

People with moderate vision impairments need to use Magnification technology to blow up the content on the screen to make it readable for them. Often magnifiers have a reader function built in to read out text as well. The most typically magnification range is x4 - x6 but it can increase to x32 which is in reality rarely used – an individual using this magnification would normally have to use a Screen reader instead. Flash historically has not supported magnifiers; however this has changed recently with the ZoomText Screen reader now having basic support.

Figure 3: screenshot of a magnified section of a website in the top half of the page

The most important issue for magnification users is having a consistent page layout as only a portion of the screen is seen at any one time, therefore it helps to know that, for example, the ‘back’, ‘forward’ and ‘help’ buttons are always in the same place. Information is easier to read in one column rather than across multiple columns; a problem commonly encountered in online forms is where one or two of the fields have been put on the right of the main set of fields; these are often missed when users are filling in forms. Additionally if a ‘search’ or ‘go’ button is not close to a form, for example because it has been right justified and the form has been left justified, users have to hunt around to find the button.
Screen reader users

The most common problems for screen reader users with limited or no vision are buttons and controls not being labelled properly or not being labelled at all. The most common problems are buttons and controls not being labelled properly or not being labelled at all – for example it is quite common for screen reader users to hear ‘Button Button Button’ when going through a list of Flash links. **Figure 4** gives you an example of a list of unlabeled buttons in a Flash movie that are unlabelled which makes them totally inaccessible.

![Figure 4: Inaccessible button list in Flash movie](image)

Other problems include orientation and comprehension issues. It is often hard to make sense of a movie if the reading order does not follow a logical sequence, especially if no summary information is provided about what content is on the screen.

Additionally a common complaint is around the use of looping Flash animations – which causes a screen reader to go to the top of the page every time the content in the movie refreshes which is a very frustrating experience.

Also there are interaction issues – often a screen reader user can navigate around a Flash application and read the content but when it comes to interacting with elements such as multiple choice and drag and drop options they become very frustrated because the controls don’t work properly and tend to be very visually focussed.
Deaf and Hard of hearing
Hard of hearing and Deaf users obviously need captions for any video and audio content in Flash that is not provided in another way on screen.

For British Sign Language (BSL) users there is a language issue because BSL as a language has a smaller vocabulary than standard English so a significant number of words will not be covered. For example, research by the BBC looked at the interaction between BSL users and their website and found that some words, for example ‘marinade’, were not understood by BSL users.

Their solution was to use a glossary for words whose meaning might be unclear – they had to term this ‘Jargon Buster’ as the word glossary itself does not exist in BSL!

Mobility impairments

Hand arm dexterity problems
People with hand and arm dexterity problems often use alternative devices to the standard mouse, or they will rely solely on the keyboard if it is too uncomfortable to use a pointing device.

For those that use an alternative pointing device it is important that links and buttons are of a reasonable size to allow for the reduced level of dexterity - if they are too small or too close together some people with find it difficult to navigate and interact with Flash screens.

Users relying on keyboard navigation need to be able to tab through Flash and selected options and interact in such a way that all the controls work. It is also important not to disable the tab focus the yellow rectangle you see when tabbing through Flash controls - if it is turned off then it is often hard to work out where you are on screen.
Voice Recognition users

Voice recognition users can face serious issues with Flash as there is currently no interface between Flash and the voice recognition software. Users therefore rely on a basic interaction system called mousegrid (see Figure 5). The user begins with a 3 x 3 numbered grid that fills the screen. By selecting a number the user focuses on a particular part of the screen. This step is repeated until the pointer is over the target link which is selected using the voice command ‘Click’.

Figure 5 Example of a mouse grid used in voice recognition software

This can make using Flash a long winded process but it is possible to navigate but drag and drop functionality that requires the use of a mouse may prove to be a problem – alternatives to drag and drop including Click and drop are covered later in this guide under the Adobe guidelines keyboard accessibility section.
Dyslexia
People with Dyslexia have a range of issues with seeing information on the screen – it is such a diverse group that it is difficult to cater for every need. At the very least the text should be left aligned as fully justified text can be difficult to read due to the uneven spacing between the words, which causes the gaps to stand out more than the words.

Also people with Dyslexia tend to prefer off white or pastel backgrounds – they find pure white to harsh to look at for extended periods of time.

Font styles are also an issue for people with Dyslexia – san-serif fonts such as Arial, Tahoma and Verdana are easier to read for most people; fonts such as Times, Tempas and old English text are very difficult to read. Typically slightly larger font size, for example 14pt, than standard (usually 12pt) is helpful, and wider line spacing can make the text easier to read.

Often people with Dyslexia will use third party software to read out text on the screen. Only quite recently has this software started to have any compatibility with Flash. For example TextHelp’s Read and Write has the ability, although quite limited to read Flash text with a Screen shot reader - still a long way from full functionality. A clear clean font with good contrast will help software like this to recognise the text accurately.

Cognitive impairments
Other impairments, for example those that affect memory and attention span, impact on people’s ability to process the information on the screen. The key requirements for this group are to have consistent layout that is easy to navigate and not to have too much distracting movement on the screen – animations that run once and them stop are fine, however continuous animation can be particularly distracting for this group.

Learning Disabilities
For those with learning disabilities it is not easy to make complex Flash applications fully accessible. The use of plenty of icons and graphics that are clear and intuitive, simple language and consistent navigation are elements that can be particularly helpful.
Adobe Flash accessibility guidelines in detail

In this section, the most important Flash accessibility guidelines are explained which are:

- Assign text equivalents
- Control Animation
- Use accessible components such as radio buttons
- Enable control over reading order
- Keyboard accessibility - Facilitate keyboard access to all controls
- Expose structure
- Expose state of controls
- Use color wisely
- Validate for accessibility

For more information about the Adobe guidelines see:
Assign text equivalents

Every non text element such as a movie clip or button on the page you want to make accessible must have a text description.

You do this by adding a description to the Accessibility panel for each element on screen under the ‘Name’ field.

Typically the Accessibility panel can be found under Window > other panels > Accessibility or pressing Alt+F2 as shown below in Figure 6.

![Figure 6 Finding the accessibility panel](image)

To make an element such as a movie or button as well as adding a description you must ensure that the tick box ‘Make object accessible’ is selected. Figure 7 shows a screen shot below where you can see a button as a text description of “Continue to the next button” which will be read out by screen readers.
There are three different types of text you can use in Flash; static, dynamic and input. Static text should be avoided as this is not accessible – it is text that you cannot change once you have put it in the Flash movie.

Dynamic text is so called as you can change what it says by the Flash scripting language ActionScript depending on the choices the user makes when interacting with the movie.

You can make any dynamic or input text on the screen accessible just by ticking 'Make object accessible'. You don’t need to add an explanation as you do for buttons and movies. Dynamic and input text only have a 'Description' field available. You can also override this by putting your own text description in the 'Description' field.
Animations

A common complaint from screen reader users is around the use of looping Flash animations which can cause a screen reader to go to the top of the page every time the content in the movie refreshes which is a very frustrating experience – they cannot reach the end of the Flash movie.

The best solution is to make the animation inaccessible but add in a description of the animation in the top movie. So for example if you have an animation of a bicycle where you had the pedals and wheels moving independently you would turn off the accessibility of the child objects – the animations of the wheels and pedals by unticking the checkbox – ‘Make child objects accessible’ as shown in Figure 9 below. This makes the animation invisible to screen readers.

![Accessibility](image)

**Fig 9 making child objects inaccessible**

And then include a description in the top level movie such as “Animation of bicycle”

Other movement requirements

**Continuous movement**
An additional requirement under the WCAG 2.0 guidelines is to avoid continuous motion on a web page which lasts longer than 5 sections or provide a way to turn it off or pause it and avoid Flashing movement and animations

**WCAG 2.0 reference** [2.2.2 Pause, Stop, Hide - Level A](#)

**Flashing movement**
Also you should ensure no page content Flashes more than 3 times per second unless that Flashing content is sufficiently small and the Flashes are of low contrast and do not contain too much red.

**WCAG 2.0 reference** [2.3.1 Three Flashes or Below Threshold - Level A](#)

See the Web Accessibility Developer guidelines in this series of guides for more information about continuous and Flashing movement.
Accessible components
A component is a movie clip with ready made behaviours that would be time consuming to build from scratch, interface examples include radio buttons, checkboxes and combo boxes as show in Fig 10 below.

Fig 10 Example use of a ComboBox component
An important issue with components is that accessibility is turned off by default so it has to be turned on via adding an ActionScript call in the Flash movie. This only has to be done once for all instances of the component on the page. For Combo boxes the following snippet of ActionScript needs to be added to the movie.

```actionscript
import mx.accessibility.ComboBoxAccImpl;
ComboBoxAccImpl.enableAccessibility();
```

For radio buttons the code is:

```actionscript
import fl.accessibility.RadioButtonAccImpl;
RadioButtonAccImpl.enableAccessibility();
```

It is best to attach the components script to the first frame in the movie.

Failure to add in this code means the accessibility features will not be turned on and the components will be recognised only as 'buttons' rather than a combo box or checkboxes.

Radio buttons are the most accessible component, others work but need to be tested – for example in earlier versions of Flash, screen readers while they recognised that the combo box component is present it cannot read any of the entries in the drop down list. So if you are going to use components ensure they work with screen readers such as Jaws.
Enable control over reading order

For a screen reader user it is often hard to make sense of a movie if the reading order does not follow a logical sequence, especially if no summary information is provided about what content is on the screen.

In Flash a screen reader follows a reading order on the page; which in can either be left as the standard left to right top to bottom order or you can set a specific reading order via the TabIndex property. Note the order you place elements on the page can cause problems with the reading order – as far as possible place elements in the Flash move in a logical sequence.

In ActionScript The TabIndex on the page is set as the lowest number is read out first the highest last, so for example for the first two movie buttons in Fig 11 they are set as follows:

```
this.mc_exit.tabIndex =10;
this.mc_toolshed.tabIndex =20;
```

And so on...

![Fig 11 Screen reader reading order](image)

For a screen reader it is not just the selectable buttons that are part of the reading order - every accessible object on the screen is part of the reading order and has a specific TabIndex value.

As you can see from Fig 11, the order is set in a logical sequence that is understandable to screen readers but is not a logical visual sequence. The reason for this is that the page heading 'Opening a document - How much do you know?' is at the bottom of the page and not the top, so to avoid disorienting a screen reader it is set as fourth in the reading order sequence.
Keyboard accessibility: Facilitate keyboard access to all controls

It is important that all controls that can be used with the mouse can be accessed by using the keyboard only. This is critical for both screen reader users as well as users with mobility impairments.

By default mouse actions have keyboard support in Flash but you must ensure that you test all controls work with the keyboard.

Things to avoid

Inaccessible links
Avoid calling links using ActionScript directly on a button using code such as:

```actionscript
on (click) {
    getURL(index.html);
}
```

Instead you should use an ActionScript function in a frame – for example to add a link to go to the Adobe homepage

```actionscript
function gotoAbobehomepage (event:MouseEvent):void
{
    var adobeURL:URLRequest = new URLRequest("http://www.adobe.com/");
    navigateToURL(adobeURL);
}
```

```actionscript
home_mc.addEventListener(MouseEvent.CLICK, gotoAdobeSite);
```

Note that this script uses the ActionScript MouseEvent, but this is method is also works with the keyboard.

Empty hit areas
Hit areas are empty button clips with a shape defined in the hit state. These allow you to reuse a single library of objects repeatedly by placing them over text objects and just modifying the scripts used.

The problem with this technique is that screen readers assume that if the contents of the up state of a button clip is empty, then it is not a button and therefore not accessible. The solution is to place a transparent movie clip the up state so screen readers will recognize the button and allow the user to use it.
Drag and drop

Drag and drop screens are the most challenging interaction for keyboard and screen reader users. There is also the issue for many mouse users find the ‘drag’ element of drag and drop difficult to do because of the strain it puts on their hands. One solution is to use ‘click and drop’ instead which means they only need to click once to pick up a description and click again over the placement location to drop it as shown in Figure 12.

Fig 12: Placing an option with mouse click rather than dragging

The only way to make a Flash drag and drop screen accessible to a Voice Recognition user is to use a click and drop approach

Keyboard only users

Keyboard users should also be provided with an alternative way to interact with the screen – this can be done by pressing key combinations to place options.

The example in Figure 13 incorporates a solution that allows a keyboard only learner to use key combinations to replace the descriptions next to each term - using A - E for the descriptions and 1-5 for the placement options. If a keyboard user is detected by the use of the tab key then the number and letters are highlighted on screen for improved usability as shown in Figure 13 below.

Figure 13: Keyboard accessible drag and drop screen
The recommended approach for screen readers is to redirect via a script detect in the Flash movie to take the user to an alternative version of the task rather than trying to make the drag and drop fully accessible.

There is however a good tutorial for making click and drop accessible. - Creating self-voicing puzzles in Flash on the Adobe developer site - http://www.adobe.com/devnet/Flash/articles/self_voicing_puzzle.html.

Screen reader detection

Often a screen reader user can navigate around a Flash application and read the content but when it comes to interacting with elements such as multiple choice and drag and drop options they become very frustrated because the controls don’t work properly and tend to be very visually focussed – this typically stems from developers not testing with a screen reader through the whole design process.

It is possible to get around this issue by redirecting screen reader users. Flash can detect if Access technology like a screen reader or magnifier is running - which is activated by using the following ActionScript code:

```actionscript
if(Accessibility.isActive()){ gotoAndPlay("screenreader");}
```

The code checks to see if Access technology is present by checking if the Accessibility.isActive() function is true. If it is the movie redirects to an alternative version of the screen called ‘screen reader’ - if no screen reader is detected it ignores the code and continues loading the standard screen. Note that Flash cannot distinguish between different screen readers and other AT’s like magnifiers. The screen reader detect code should not be placed in the first frame of the movie as it might be missed – better to place it a few frames into the movie such as frame 5.
Expos structure
Movies created with Flash can be complex in terms of layout, structure, and navigation. Unlike in accessible html pages a screen reader cannot interrogate a Flash movie to find out about its structure.

You therefore need to include enough information within the movie for it to make sense to a non sighted screen reader user. For example in an html page screen readers can tell from the code the structure of the page by reading the heading structure (h1 – top level heading, h2 – second level heading etc). In the example below in Figure 14 the Pie chart heading 'Results of the Opinion Poll' is prefixed with the text 'Pie chart heading' to help a screen reader user know what the text relates to.

Figure 14: Pie chart heading added to help orientation
In the example below each section of the pie chart has its own description; Figure 15 shows the text description for the Cycling segment.

Figure 15: Descriptive pie chart segment
Expose state of controls

A key requirement in Flash accessibility is to indicate the change of state of objects on the screen for screen reader users. In the example below in Figure 16 each of four question options initially have a name description of “unselected:...”. When they are selected; you need to modify this text to tell a screen reader it has been selected. This is done in ActionScript by the use of the _accprops (accessibility) property - a way of dynamically changing any of the information in the Accessibility panel.

For example when the first option is selected in the ActionScript the following code is triggered to change the 'Name' description to this:_accProps.name = "selected : option 1 of 4 You should use the ending 'Yours faithfully' if your letter starts Dear Mrs Green."

If the option is then selected again to deselect the choice then accProps.name is called again to reset to the option to 'unselected...'.

To make this work you also need to add the ActionScript statement: 'Accessibility.updateProperties();' once to update the Accessibility panel after the _accProps statements.

In this way a screen reader user receives auditory feedback on an answer option choice they have made by hearing 'selected...' when they read the answer description.
Use color wisely

Conveying information by colour alone

WCAG 2.0 reference 1.4.1 Use of Color - Level A

The most important issue is not using colour alone to convey information in your Flash content; particularly with instructional text such as ‘click on the green button to start’ or ‘click on the red button to stop’.

The reason for this is that people with colour blindness and other vision problems cannot easily distinguish between certain colour combinations.

Similarly do not use any sensory specific language that relies on shape, size or visual location because some people with vision impairments cannot perceive shape or size or use information about spatial location or orientation.

Colour contrast

WCAG 2.0 reference 1.4.3 Contrast (Minimum) - Level AA

Not all users can differentiate between similar colour combinations and tones. Even where colours are different, they may still create issues for users who suffer from colour-blindness, those whose eyesight have deteriorated, e.g. older users and users who use a black and white screen.

Ensure there is sufficient contrast between text and background colours in your Flash movies.

The recommend minimum contrast ratio is 4.5 to 1 for standard text up to the equivalent size of 14pt

For large text (equivalent of 14pt bold+ or 18pt+) a ratio of 3:1 is the minimum required

Note however that logos / brand names are exempt from this requirement under the WCAG 2.0 guidelines

Use a tool such as the free Contrast Analyser, Version 2 from the Paciello Group which can be downloaded form http://www.paciellogroup.com/resources/contrast-analyser.html to check for colour contrast.
Validate for accessibility

However closely you stick to accessibility guidelines in building Flash applications you cannot be sure they are accessible without testing them with adaptive technology such as screen readers.

Below is a testing checklist you should use to test your Flash. Appendix 1 contains a tutorial for using the Jaws screen reader with Flash.

Testing checklist

- Number one thing to do is test with a screen reader:
  - look out for unlabelled movies and buttons
  - confusing reading order
  - information that does not make sense non visually
  - inaccessible interaction screens such as drag and drop
- every feature and task can be completed with the keyboard
- there are no low contrast text/graphics used
- no sensory specific language is used
- the language for clarity and simplicity
- the navigation system is intuitive
- audio can be turned on and off on each page (if used)
- all customisation options work on all screens (if used)
- a glossary is provided for technical terms (if necessary)
Additional issues

Embedding Flash in a webpage
Making a Flash movie accessible can be tricky enough, but before you can allow screen reader users to find their way around your Flash movie, they first have to know it is even there. Setting the window mode (wmode) parameter to transparent or opaque is a common practice when embedding Flash – so it behaves itself on-screen – but the setting also renders the movie completely invisible to screen readers.

Basically, the wmode parameter is used to control how a Flash movie interacts with other web page elements. There are three modes:

- **Window (the default if the parameter isn’t included)**: used to show a Flash movie in its own rectangular window on a web page. The background colour will be whatever is set in Flash when you export the movie. The only mode that is accessible to screen readers

- **Opaque**: with this setting you can use JavaScript to move the movie around the page, or resize it.

- **Transparent**: other content on the page will show through all the transparent parts of the Flash movie – basically, the colour background set when the movie is exported gets ignored.

Transparent wmode is the equivalent of CSS display:none, and screen readers don’t even announce there’s a Flash movie present.

if wmode is set to anything other than ‘window’ the Flash movie will be completely inaccessible to screen readers, undoing any good work you may have done with the accessibility properties.

This means that even if you’d made use of Flash’s built in accessibility features, your efforts will be wasted if you’re embedding the movie with the wmode as transparent.
Appendix A - Testing Flash with Jaws

Overview
This tutorial explains the basics of how the Jaws screen reader works and how it interacts with Flash.

Jaws is the most popular screen reader used by visually impaired people in the UK. Other common screen readers include Window-Eyes and Hal.

You can download a demo version of Jaws from Freedom Scientific

It work for 40 minutes then you need to restart your machine.

Alternative Open Source screen reader
An alternative screen reader is NVDA which is free and open source which can be downloaded from - http://www.nvda-project.org/

There is a good article here on using NVDA and the web - http://www.webaim.org/articles/nvda/

Walk through using Jaws
Because it is easy to make a mistake adding descriptions to elements in a Flash movie it is important to test with the Jaws screen reader. Learning the most commonly used keyboard commands will help you get started using Jaws.

Starting Jaws
When you first load Jaws it will start speaking out information about where it has focus this is typically the desktop. To run Jaws through a Flash movie start the file in Internet Explorer. Note at this point in time Firefox only has very limited screen reader support.

Initial automatic read through
When you bring up a Flash page Jaws will normally say 'Flash movie start' and proceed to read at a rapid pace through the movie - often a user will read through a page like this to orientate themselves and then go back to the beginning.

When the screen reader hits the end of the movie it typically announces this by saying 'Flash movie end'. To go back to the beginning of the Flash movie press Ctrl + Home at which point it will normally read out the page title and then continue with 'Flash movie start'.

Also to go back to the start at any point during the initial rapid reading of the page hit the Ctrl key this will normally take you back to the start of the Flash movie.
Moving through the movie at your own pace

To start moving through the movie at your own pace press the down arrow to go to the first object on the screen, press the down arrow again to go to the next element on screen - this sequence is known as the reading order. To go back press the up arrow which will read the previous element.

You can speed up the process by pressing Ctrl + down arrow to jump blocks on the screen but you should only do this if you are familiar with what is on the screen as it can miss out important information navigating in this way.

When you come across a selectable movie or button such as a multiple choice question check you can select it by hitting the Enter key and also check you get audio feedback to say the button has been selected (Expose state of controls).

Different reading modes

The default reading mode when going through a page is called virtual cursor mode however when Jaws encounters a form field it typically changes to forms mode so you can input data into the field. Once you have input your data you can press Tab to go to the answer or submit button and then press Enter. You should at this point revert to virtual cursor mode but sometimes it does not you have to manually reset to virtual cursor mode by pressing the plus(‘+’) key on the number pad.

You can then continue through the movie using the down arrow

Links list

Also there is a shortcut way of listing all the links and form fields on the page by pressing Insert + F5 which will bring up a list of options use the up and down arrow keys to make your choice and then hit Enter to activate the link or go to the input field. Check that all the options have descriptive labels.
Contents

TOP TIPS FOR ACCESSIBLE JAVASCRIPT ..........................................................3
INTRODUCTION ..................................................................................................4
  JavaScript and Accessibility standards.................................................................4
  Common accessibility issues...............................................................................4
JAVA SCRIPT EVENT HANDLERS ................................................................6
  onMouseOver and onMouseOut (Device dependent)...........................................7
  onFocus and onBlur (Device independent).........................................................9
  onClick and onDblClick (Device dependent).....................................................10
  onChange and onSelect (Device independent)..................................................12
ADDITIONAL ACCESSIBILITY ISSUES.........................................................14
  Dynamic HTML and Accessibility.................................................................14
  Pop-up Windows .............................................................................................15
  Redirecting and Refreshing Browser Windows..............................................16
JAVA SCRIPT ALTERNATIVES ..................................................................17
  Cascading Style Sheet (CSS)......................................................................17
  Server-side Processing..................................................................................18
APPENDIX 1 PROGRESSIVE ENHANCEMENT..............................................19
### Top tips for accessible JavaScript

- Before using JavaScript assess if it is the best solution for what you want to do – can CSS or Server side scripting do the job better?

- Use the principles of progressive enhancement (see Appendix 1) - use JavaScript to enhance a page rather than for core functionality

- The majority of Adaptive technology used by disabled people fully supports JavaScript so don’t count on them seeing a non-scripted version of your page

- Under the new accessibility guidelines (WCAG 2.0) the requirement is no longer that your site works without JavaScript but that it works with Adaptive technology such as screen readers.

- Be aware of the four most common areas where JavaScript can be an accessibility issue; **Navigation, Hidden content, User control and Confusion/Disorientation.**

- To create accessible JavaScript:
  - Use device independent event handlers (do not need a mouse) or if you use device dependent event handlers pair with independent event handlers or provide the same information another way
  - Use onClick as a device independent control for hypertext and form controls but be aware it is device dependent for non-link and non-control elements, such as plain text or table cells
  - Ensure scripted pages elements can be accessed just using a keyboard.
  - Do not alter the normal browser functionality in a way that may cause confusion or inaccessibility.
  - You can use JavaScript to dynamically write content to the page but if it is constantly changing it will cause accessibility problems
  - Test your code with a screen reader such as Jaws to check how accessible it is
  - Provide an accessible alternative when JavaScript cannot be made natively accessible.
Introduction

JavaScript is the most common scripting language used on the Web. It adds functionality to static html pages, validate forms, detect browsers, and much more.

In simple terms JavaScript allows your web pages to ‘think’ – responding to events triggered by the customer interacting with a page.

JavaScript and Accessibility standards

For a long time there has been a big issue with JavaScript and accessibility on the web. The key problem that is that under version 1.0 of the internationally recognised standards the W3C’s Web Content Accessibility Guidelines (WCAG 1.0) there has been a requirement for web pages to work with JavaScript turned off. This has changed with the introduction of the new version of the guidelines WCAG 2.0 which no longer requires your pages to work if JavaScript is turned off.

Most adaptive technologies such as screen readers and Magnifiers used by people with vision impairments have full support for JavaScript. Most screen readers work primarily with Internet Explorer 6 and above (and some of the more sophisticated with Firefox too). So be aware by default Adaptive technology will be getting a JavaScript-enabled experience too.

However under the new WCAG 2.0 guidelines important functionality should be implemented in a way that is compatible with assistive technology (Accessibility Support of Web Technologies). So if you use JavaScript for core functionality like navigation, search or other interactions they must work with screen readers and other technology such as Magnification software and Voice Recognition technology under WCAG 2.0.

Common accessibility issues

It is possible to create accessible JavaScript if you have a good understanding of what the key issues are and how they should be coded.

Below are the four most common accessibility problems with JavaScript

- **Navigation** - inability or difficulty navigating web pages using a keyboard or assistive technology such as a screen reader.
- **Hidden content** - content or functionality that requires the use of the mouse - for example popup windows that – these are not accessible to assistive technologies.
- **User control** - Lack of user control over automated content changes – particularly if they are not expecting them.
- **Confusion/Disorientation** - Altering or disabling the normal functionality of a browser such as disabling the back button or launching a new browser window without warning.
Content written to the page using JavaScript is accessible if:

- Device independent event handlers are used – script does not rely on a mouse.
- JavaScript enabled content and functionality is accessible to assistive technologies such as screen readers.
- Scripted pages and elements can be navigated just using a keyboard.
- The normal browser functionality is not modified in a way that may cause confusion or inaccessibility.
- An accessible alternative is provided when JavaScript cannot be made natively accessible.

The three appendixes give further information about more advanced JavaScript techniques.

**Appendix 1** explains the progressive enhancement approach – the best practice way to implement JavaScript.

**Appendix 2** explains the key principles of coding AJAX - Asynchronous JavaScript And XML

**Appendix 3** explains a new approach to coding accessible Rich Internet Applications accessibly called WAI-ARIA that addresses the key problems of using AJAX.

The only way to ensure JavaScript accessibility is by evaluating each individual script and devising a unique solution to the accessibility problem it poses.

The rest of this guide explains the key accessibility issues and the recommend coding techniques to address them.
JavaScript Event Handlers

JavaScript can be processed using <script> elements and event handlers. The <script> element can contain JavaScript code directly:

```html
<script type="text/javascript"> <!-- function doit(); --> </script>
```

or can open an external JavaScript (.js) file:

```html
<script type="text/javascript" src="scripts.js">
```

The <script> element is processed when the page is loaded and requires no user intervention.

Event handlers accompany existing HTML code and are triggered by a browser or user event - such as when the page loads, when the user clicks the mouse, or when the time is 8 p.m.

There are two types of JavaScript event handlers. The first call called device dependent handlers and rely upon the use of a mouse or keyboard.

The second type of event handlers are called device independent and are triggered by both the mouse and keyboard or by other means.

Using device dependent event handlers may cause the content to be inaccessible to someone that is not able to use the device needed to trigger it such as a mouse.

Therefore when implementing event handlers, you must either use a device independent event handler or use multiple device dependent event handlers to meet the needs of all users. Below is a list of common event handlers and their accessibility issues.
onMouseOver and onMouseOut (Device dependent)
The onMouseOver event handler is triggered when the mouse cursor is placed over
an item. onMouseOver, and its companion, onMouseOut, can be used as long as
any important content or functionality is also available without using the mouse.

It can be used in combination with the onFocus and onBlur event handlers to allow
keyboard accessibility

**Important:** You need to be provide an accessible alternative if the content or
functionality cannot be made natively accessible

**Example 1**
When the mouse is placed over the image of the word "Accessibility", another image
appears in its place, which presents the definition of the word "Accessibility".

```html
<a href="page.htm"
onmouseover="document.images[\'myimage\'].src='imageoff.gif';"
onmouseout="document.images[\'myimage\'].src='imageon.gif';">
  <img src="media/imageoff.gif" width="289" height="79" id="myimage" alt="Accessibility
  - The quality of being accessible, or of admitting approach; receptibility." />
</a>
```

**Problems**
In this example, additional content is being displayed using onMouseOver. This
content can only be viewed if the user uses a mouse to hover over the image. It is
not available to someone who is using the keyboard to navigate through the page.
The onMouseOver event handler cannot be made directly accessible to keyboard
users. Thus, an alternative must be provided.

**Partial Solution**
Place the additional content in the alternative text (alt text) of the image itself. This
would work for screen reader users - the screen reader would read the alternative
text. But for sighted users whose browser does not display alternative text for images
or who do not know to mouse over the image to see the alternative text, this is not a
viable alternative. Still, the image must have equivalent alternative text.
**Recommended Solution**

In addition to `onMouseOver` and `onMouseOut`, use `onFocus` and `onBlur`. These actions will be triggered when the keyboard is used to navigate to and from a link that surrounds the `<img>` element.

```html
<a href="page.htm"
  onmouseover="document.images['myimage'].src='imageon.gif';"
  onfocus="document.images['myimage'].src='imageoff.gif';"
  onmouseout="document.images['myimage'].src='imageoff.gif';"
  onblur="document.images['myimage'].src = 'imageoff.gif';"> <img
  src="imageoff.gif" width="289" height="79" id="myimage" alt="Accessibility - The quality of being accessible, or of admitting approach; receptibility." /></a>
```

**Example 2**

A common use of `onMouseOver` and `onMouseOut` is for fly-out or drop-down navigation systems. When the mouse is placed over the main menu items (`onMouseOver`), additional sub-menu items appear below. When the mouse is removed from the menu system (`onMouseOut`), then the sub-menus disappear.

**Problems**

Additional content and functionality is being displayed using `onMouseOver`. In this example, JavaScript is controlling the display of Cascading Style Sheet elements within the page. The sub-menu items will only be visible if the mouse is placed over the main menu item. These items may not be available if JavaScript is disabled and may not be read by assistive technologies.

**Recommended Solutions**

**When possible, use additional event handlers that allow keyboard navigation.**

When this is not possible, you must provide redundant navigation. This can be done one of two ways. First, provide links within the main content of the page to the pages displayed in the sub-menu navigation. This is often done as a list of categorized links at the bottom of the page. The downside of this approach is that the navigation is only accessible at the bottom of the page, which might be counter-intuitive to most users.

A second method of providing the redundant navigation is to add a standard `href` link from each main navigation items (e.g., Product, Services, etc.) to a separate web page that contains the sub-menu navigation items.

```html
<a href="productlinks.htm"
  onmouseover="SlideOutMenu();"
  onmouseout="SlideInMenu();">Products</a>
```
onFocus and onBlur (Device independent)
These event handlers are typically used with form elements, such as text fields, radio buttons, and submit buttons. onFocus is triggered when the cursor is placed on or within a specific form element. onBlur is triggered when the cursor leaves a form element.

Both of these event handlers are device independent so they can be triggered with the mouse, keyboard, or other assistive technology.

However the actions triggered by these event handlers must be checked to find out if they cause accessibility problems. Typically, these events do not cause accessibility issues unless they are modifying the default behaviour of the web browser or are interfering with keyboard navigation within the web page.

Example 1
When the mouse is placed in an edit box an alert dialog appears.

```html
<input id="fname" type="text" onfocus="alert('Enter your first name only');" />
```

Problems
None. Although the alert window may be distracting and unnecessary, it does not introduce any serious accessibility issues. The alert is triggered when the text box gains focus, either by using the keyboard or mouse. JavaScript alerts are read by most modern screen readers and can be used to increase the usability of forms by providing timely feedback, instructions, or cues. However, if JavaScript is disabled, then the alert will not be available at all.

Example 2
In this example, the onBlur event triggers JavaScript code that changes the functionality of the Web browser. Placing your mouse into a text box and entering a value and then pressing the Tab key or clicking the mouse outside of the text box will trigger the onBlur event, which will validate the value entered to ensure it is correct and present feedback regarding your answer. If it is incorrect, the text box will retain the cursor focus until you enter the correct number.

```html
<input type="text" id="input1" size="5" onblur="checkAnswer();" />
```

Problems
Although the onBlur event is device independent, it is being used to execute JavaScript code that makes the page difficult to navigate. The textbox maintains focus until the correct answer is given. This change in browser default behaviour may be confusing and restricting. Also, the feedback is displayed in a different part of
the page. Because focus is maintained within the text box, a screen reader will not be able to access the feedback text or any other part of the page until the correct answer is entered.

**Solution**
Do not force the focus to remain within the text box. Allow the user to navigate throughout the page. Display the feedback on another page after the form has been submitted (works without JavaScript) or displays the feedback with a JavaScript alert (requires JavaScript).

**onClick and onDblClick (Device dependent)**
The onClick event handler is triggered when the mouse is pressed when over an HTML element such as a hypertext link or form control. onClick is intended to be a mouse dependent event handler. However, if the onClick event handler is used with hypertext links or form controls, then most major browsers and assistive technologies trigger onClick if the **Enter** key is pressed when the link or control has focus.

In these cases, onClick is a **device independent event handler**. Nevertheless, the **Enter** key will not trigger the onClick event if it is used with non-link and non-control elements, such as plain text or table cells. Be aware there are no device independent or keyboard accessible equivalents to these event handlers.

The onDblClick event handler is associated with the double click of a mouse on a selected HTML element. There is no device independent or keyboard equivalent to onDblClick, so this event handler should be avoided.

**Example 1**
The following example is a link that triggers a JavaScript confirmation box, which allows you to confirm whether you want to view the page, or not.

```
<a href="page.htm" onclick="return confirm('Are you sure you want to view this page?');">View this onClick example</a>
```

**Problems**
None. All major screen readers read the confirmation box. By using the return JavaScript parameter, the link action is cancelled if the user selects No in the confirmation dialog box. If JavaScript is disabled, the link will work normally.
Example 2
A very common use of JavaScript is to validate form information. The following form will use JavaScript to allow you to confirm the information you have entered.

This example validates the form information, and feedback is displayed in another place within the page.

```html
<form action="page.htm" onsubmit="checkAnswer2(); return false;">
  <p class="invis" id="answercorrect">Correct, there are 12 items in a dozen.</p>
  <p class="invis" id="answerwrong">Incorrect. There are 12 items in a dozen.</p>
  <p><label for="answerbox">Enter Answer:</label>
  <input type="text" id="answerbox" />
  <input type="submit" onclick="checkAnswer2()" value="Check" />
  </p>
</form>
```

Problems
The feedback is not presented in a manner that would be accessible to some assistive technologies. In this example, JavaScript is modifying the display style parameters for elements within the page to make them visible or invisible based upon the response. The screen reader user would not be aware that additional content has suddenly appeared within the page.

Solutions
Validate the form information with a server side script, and then display the feedback on another page.

Provide the feedback in a way that is accessible, such as a JavaScript alert or another form element.
onChange and onSelect (Device independent)
The onChange event handler is triggered when a form element is selected and changed (for example, when a radio button is initially selected or when the text changes within a text box).

The onSelect event handler is triggered when text is selected within a text field or text area. These event handlers are device independent and can be activated using the mouse, keyboard, or other device.

However the actions triggered by these event handlers must be checked to find out if they cause accessibility problems. For example drop-down menu navigation items that are triggered with onChange are not fully keyboard accessible

Example 1
The following example shows the code for a drop-down menu form element used for navigation. By selecting an item from the list, JavaScript will automatically open the specified page within the browser.

```html
<script type="text/javascript">
<!--
function goto_URL(object) {
    window.location.href = object.options[object.selectedIndex].value; }
//-->
</script>
<form action="page.htm" onsubmit="return false;">
<p><label for="selectPage">Go to:</label>
<select name="selectName" onchange="goto_URL(this.form.selectName)">
<option value="">Select a page:</option>
<option value="page.htm">Page 1</option>
<option value="page.htm">Page 2</option>
<option value="page.htm">Page 3</option>
</select>
</form>
```

Problems
The JavaScript causes the browser to go to a new page using onChange or when the user selects an item from the select list. If the end user is using a keyboard, the onChange event handler is executed for every item within the list. It is impossible for the user to directly select the last item in the list, as each previous item within the list will trigger the page change. The only way the user can select the last menu item is by navigating to the page for the first item in the list, then pressing the Back button,
navigating to the second item, then pressing the Back button, and so forth until the last menu item is accessed.

**Solution**

Rather than using the `onChange` event handler, allow the user to select the item from a list then select a button or submit the form to activate the script. When server-side scripting is used to process the form information and redirect the browser accordingly, there is no need for JavaScript at all. The following code demonstrates one method of fixing the `onChange` problem.

```html
<form action="page.htm" onsubmit="return false;">
  <p><label for="selectPage2">Go to:</label>
  <select name="selectPage2" id="selectPage2">
    <option value="">Select a page:</option>
    <option value="page.htm">Page 1</option>
    <option value="page.htm">Page 2</option>
    <option value="page.htm">Page 3</option>
  </select>
  <input type="button" value="Go!" onclick="goto_URL(this.form.selectPage2);" />
</p>
</form>
```
Additional accessibility issues

Dynamic HTML and Accessibility

Dynamic HTML (DHTML) is a combination of JavaScript and Cascading Style Sheets (CSS). By its very nature, DHTML can present dynamically changing content. DHTML is often triggered by user interactions, such as moving the mouse. When implementing DHTML, you must ensure that the DHTML is triggered in a device independent way and that the content and functionality provided by the DHTML is also accessible.

JavaScript can also be used to write dynamic content to the web page. In most cases, this content is accessible, unless the content is constantly changing or is in any other way interfering with the accessibility of the page.

Example

Dynamic, constantly changing information can be written to the page using the `document.write` command in JavaScript. In most cases, content that is written to the page using JavaScript is accessible to assistive technologies. However, if the dynamic content is constantly changing or otherwise interferes with navigation or browser functionality, then it may cause accessibility problems.

Problems

Because the content is constantly changing and refreshing, a screen reader would not be able to read it and it may cause some confusion.

Solutions

When using dynamic information, you must first ask yourself if it is necessary and vital to the function or content of the page. If so, there is often a way to provide the content without using inaccessible JavaScript. For instance, you could write the current time to the page when it loads using a server-side script. Though it will not be constantly updating, it may suffice. You may also have the user select a link or button to go to another page that displays the current time from server side processing. Another approach that uses a more accessible implementation of JavaScript would be to provide a link or button that allows the user to display the current time at their request as shown in the code below.

```html
<input type="button" value="Display the current time" onclick="alert('The current time is ' + timeValue);" />
```
Pop-up Windows

JavaScript can also affect the default behaviour of the web browser and certain HTML elements. Pop-up windows can be triggered by JavaScript or by JavaScript event handlers.

If the user is not alerted to the fact that a pop-up window is opening, they may become confused or disoriented by the unnatural behaviour of the Web browser. Further modification of browser windows to remove scroll bars, status bars, menu bars, or tool bars may also cause accessibility problems.

So use pop-up windows with care and if they are used, always alert the user to this fact. The user must also be alerted when JavaScript is used to automatically perform browser functions, such as redirecting, refreshing, or automatic scrolling. In all cases, user testing and testing with assistive technologies can provide valuable feedback regarding accessibility of specific JavaScript implementations.

Example

In the example code below, the link will continue to work normally (i.e., in the same browser window) if JavaScript is disabled.

```html
<a href="popup.htm" onclick="window.open(this.href); return false;" title="Link opens in a new window">Select this...</a>
```

In the example code above, the link will continue to work normally (i.e., in the same browser window) if JavaScript is disabled.

When using JavaScript to open new windows, you can modify the size and position of the new window. You can also add or remove functionality of the window, such as the ability to resize, display scroll bars, show tool bars, etc. Be very careful when changing the default behavior of the browser window.

If a user has low vision and must enlarge the content, a small window that cannot be enlarged and does not display scroll bars would be very inaccessible. Someone with a motor disability may rely upon large tool bars to accurately control the web browser, so removing or modifying them may introduce difficulties for this user.

As you can see, there are many difficulties in both usability and accessibility that arise through the use of pop-up windows. Care must be taken in making the decision to use them. If they are used, thorough user testing of your implementation is vital to ensure accessibility. **Always alert the user to the fact that a pop-up window will be opened.**
Redirecting and Refreshing Browser Windows
When the page the browser is viewing suddenly changes or refreshes, the person viewing that page may become disoriented or confused, especially if that person is using an assistive technology. This is commonly done with page redirects when page content has been moved or updated. Do not automatically change or refresh the browser window without first alerting the user that the change will occur and giving him/her the ability to disable or postpone the change.
JavaScript Alternatives

Cascading Style Sheet (CSS)
Cascading Style Sheet (CSS) parameters are often modified using JavaScript to create dynamically changing DHTML pages. However in CSS version 2, much of the dynamic functionality available with JavaScript is now available within the specifications for CSS itself.

This allows the building of interactive and dynamic navigation and layout elements without the need for JavaScript events. You can create drop-down menus, navigation bars, and interactive images, and other dynamic features in Web sites without worrying about device dependent event handlers. However some older browsers such as Internet Explorer 6 do not fully support CSS standards so any implementation would need to be thoroughly tested.

Also, screen readers do not have great CSS support, especially when presented with content that can be made visible or invisible using either the `display:none` or `visibility:hidden` styles. Many screen readers cannot read content which is formatted with these styles even though the content is still part of the underlying structure of the page. Until there is better CSS support in both web browsers and assistive technologies, using CSS alone to produce dynamic content should only used if you can test in a variety of browsers and screen readers.
Server-side Processing

When JavaScript itself cannot be made natively accessible, you must provide an accessible alternative. Replicating or replacing the JavaScript behaviour with server-side processing can do this.

In many cases, server-side scripting can duplicate the functionality provided by JavaScript. For example, JavaScript is often used to validate form elements before a form is posted. Instead of implementing such JavaScript programming and its accompanying accessibility techniques, you could use a server-side script to validate the form elements. JavaScript is often used to write dynamic information to a web page, such as the current date and/or time. Again, using a server script or server-side include negates the need for additional accessibility implementation.

Note: Providing an accessible alternative within the `<noscript>` element for an inaccessible script will not make the page accessible. The `<noscript>` content will only display if JavaScript is disabled. Most screen reader users have JavaScript enabled, and so will experience your inaccessible script and not the `<noscript>` content.

Example

In the following example, two form buttons are provided within the HTML code. If you have JavaScript enabled, the `document.write` command within the `<script>` element will display the first button. If this button is selected, it triggers a JavaScript function that may be used to validate and submit the form using additional JavaScript within the page. If JavaScript is not enabled, then the button contained within the `<noscript>` element is displayed, which could submit the form to a server-side processing script, which would provide the form validation and feedback.

```html
<script type="text/javascript">
<!--
document.write("<input type="button" value="Submit" onclick="validateForm();"/>");
-->
</script>

<noscript>

<p>
<input type="submit" value="Submit">
</p>
</noscript>
```
Progressive enhancement is a strategy for web design that emphasizes accessibility, semantic markup, and external style sheets and scripting technologies.

Progressive enhancement uses web technologies in a layered fashion that allows everyone to access the basic content and functionality of a web page, using any browser or Internet connection, while also providing those with better bandwidth or more advanced browser software an enhanced version of the page.

Solutions

Below is an example of how progressive enhancement would be built up layer on layer from the most basic to the most sophisticated.

Layer 1: html code for the most basic browsers.

```html
<li><a href="/products/">Products</a></li>
<li><a href="/services/">Services</a></li>
```

Layer 2: enhancement for browsers that support CSS

```html
<li id="products"><a href="/products/">Products</a></li>
<li id="services"><a href="/services/">Services</a></li>
```

Layer 3: The next level of enhancement is for browsers that support JavaScript.

```javascript
function addProducts(){
  // Find the li element to which we will add a sub-menu
  var parent = document.getElementById("products");

  // Make sure it exists (fail silently)
  if (parent) {
    // Create a nested unordered list
    var ul = parent.appendChild(document.createElement("UL"));

    // Add the list items and links
    var items = [['Blue Widgets', 'blue']];
  }
}
```
Layer 4: A separate JavaScript function to ensure cross browser support for Document Object Model

```javascript
function createSubMenus() {
    // Make sure that the DOM functions we will use are supported
    // (fail silently)
    if (typeof document.getElementById != "undefined" &&
        typeof document.createElement != "undefined" &&
        typeof document.createTextNode != "undefined") {
        addProducts();
    }
}
```

Layer 5: Finally adding the functions to the browser onload event.

```javascript
if (window.addEventListener) {
    window.addEventListener("load", createSubMenus, false);
} else if (window.attachEvent) {
    window.attachEvent("onload", createSubMenus);
} else {
    window.onload = createSubMenus;
}
```

For more information see: [http://en.wikipedia.org/wiki/Progressive_enhancement](http://en.wikipedia.org/wiki/Progressive_enhancement)
Camden Web accessibility PDF Guidelines
Draft Version 1.0

July 2011
# Table of Contents

- PDF checklist 3
- Introduction 4
- Issues people with Disabilities have with PDFs 4
  - Low vision/Colour blindness 4
  - Motor disabilities 5
  - Hearing disabilities 5
  - Cognitive disabilities 5
- What is an accessible PDF? 6
- Converting documents to PDF 7
  - Microsoft Word 7
  - Acrobat Professional 13
  - Adobe InDesign 24
- Appendix A Word to PDF conversion Tutorials 31
  - Creating an accessible PDF document from Microsoft Word 2003 using Adobe PDF Maker 31
  - Word 2007 PDF tutorial 34
- Appendix B Accessible PDF checklist 39
PDF checklist

Below are the key issues to consider when creating a PDF. They are provided for easy reference only; you should refer to the full document for detailed guidance.

- Do not create PDFs from image only scans
- Create Tagged PDFs
- Ensure your PDFs Have a logical reading order
- Provide alternative text for all graphics, links and form fields
- Use navigation aids such as bookmarks
- Specify a document language
- Use Accessible fonts
- If form fields are used they must be accessible
- Make sure security settings do not interfere with screen readers
- As well as screen reader users consider the needs of people with:
  - Low vision/Colour blindness
  - Motor disabilities
  - Hearing disabilities
- Test your PDF with a screen Reader

See Appendix B for a more detailed checklist
Introduction

This guide covers the key accessibility issues to consider when creating accessible PDFs.

The early versions of Adobe PDF were completely inaccessible. Fortunately this began to change with Acrobat 5, when real text was introduced to offer an accessible alternative to images of text. Another key development was the introduction of ‘tags’ to PDFs, a similar concept to tags in HTML, to add extra non visual information to documents that can be understood by adaptive technology such as screen readers used by blind/partially sighted people.

Issues people with Disabilities have with PDFs

As with HTML, you need to know what kinds of issues people with disabilities might encounter when reading PDF files.

When people talk about the accessibility of Adobe Acrobat, or PDF files, they are usually thinking about the accessibility of Acrobat to screen readers, but screen reader users are not the only people who should be considered when creating accessible PDF files.

In addition to those with severe/moderate vision impairment you need to consider the needs of people with motor disabilities, hearing disabilities, cognitive disabilities or low vision / colour blindness.

What follows are some general guidelines for making PDF files accessible to these broad categories of disabilities.

Low vision/ Colour blindness

You need to ensure that there is enough contrast in the PDF document between text and background – there are plenty of tools to help you check this – see the reference section at the end.

Also be careful about using colour only when referring to something in the document like a graph or chart – don’t say ‘look at the red line on the graph’ for example – be more specific because people with colour blindness cannot differentiate certain colours.
**Motor disabilities**
Don't make hot spots around links too small – you can easily change link settings to change the hot spot areas. The key point to be aware of is that the smaller the link, the more difficult it will be for someone with limited fine muscle/motor control to click on it.

**Hearing disabilities**
You can add sound and movie clips to PDFs if you have Adobe Acrobat Professional, 6.0 or later. If you want to do this make sure you provide transcripts for any sound file such as wav.

Also if you embed movie files (such as QuickTime, Flash Player, Windows® Media Player etc) provide synchronized captions for the video, alternatively provide a transcript.

**Cognitive disabilities**
The number one rule is use clear and simple language – keep jargon to a minimum and provide a glossary for any technical terms you use. The better you write, the better you will be understood by everyone, not just those with cognitive disabilities.

Also people with cognitive disabilities or dyslexia might use the read Out Loud feature built into the later version of Adobe Reader – or they might use other software to read out the document, so it is always a good idea to run through a PDF document using the inbuilt Read Out Loud feature to ensure it makes sense.
What is an accessible PDF?

In order for a PDF to be accessible the most important requirement is that it is tagged. PDF tags provide a structured, textual representation of the PDF. All the elements of a document such as titles, headings, body text, links, lists, captions and figures are tagged.

HTML tags and PDF tags often use similar tag names and organisation structures, but they are implemented in different ways.

Figure 1 below shows you an example of how tags appear when you look at a document in Acrobat professional.

![Figure 1 Example tag structure in a PDF](image)

‘Tags’ tell assistive technology which part of a document or page to read and which parts to ignore altogether (footers watermarks) and which parts to present as special elements.

Tagging also makes documents ‘reflow’ correctly on smaller screens such as cell phones and hand-held devices.

There are additional requirements to make a PDF fully accessible such as ensuring images have alt text, the reading order makes sense and a language is specified for the document. A lot of these are depended on the source document the PDF has been created from. It is however possible to edit PDFs in Acrobat professional to make them fully accessible. The next section discusses creating PDFs from the most popular document sources.
Converting documents to PDF

PDF files are not typically created in Acrobat. They are usually created in another program and converted to PDF. There are dozens or probably hundreds of programs that can create PDF files, but very few of them produce tagged PDF files.

If you are using Microsoft Word or PowerPoint, OpenOffice.org Writer, or Adobe tools such as InDesign, you can often create accessible, tagged PDF files without opening Acrobat. Of course, the accessibility of the PDF depends on the accessibility of the original document. Be aware some design software packages like Quark have very little support for creating accessible PDF files and are therefore not recommended as a source from which to create tagged PDFs.

Microsoft Word

The majority of the PDF files on the web are probably created in Microsoft Word. The good news is that it is possible to create accessible PDF files in Office, as long as the following requirements are met:

1. The file must be accessible. That includes providing alternative text for images, proper headings, and appropriate link text. For more information see the Word to PDF Word 2003 and Word 2007 tutorials in Appendix A.

2. Office 2000-2003 users must have Acrobat installed, as well as the add-in. Office 2007 users must have either Acrobat or the Microsoft PDF add-in installed.

3. The file must be exported correctly. If a file is created by printing to PDF, it will not be correctly tagged.
Word 2000-2003

When you install Adobe Acrobat, an add-in for Microsoft Office is installed by default. The add-in allows you to convert Office files to PDF without opening Acrobat. This add-in also installs an Adobe PDF menu should appear in the Menu bar. To convert a Word Document to PDF, Select Adobe PDF > Convert to Adobe PDF. If your document is correctly structured, this should automatically create a tagged PDF. To ensure that files are being converted correctly, go to Adobe PDF > Change Conversion Settings and ensure Enable Accessibility and Reflow with tagged Adobe PDF is selected as shown in Figure 2.

![Figure 2 Acrobat PDF Maker settings](image-url)
**Word 2007 with Adobe add-in installed**

With Adobe the add-in installed, you can export to PDF one of two ways.

Click on the **Office Button**, hover over **Save As**, and select **Adobe PDF**, or press **Alt + F, F, A** as shown in **Figure 3**.

![Figure 3 Selecting Save as Adobe PDF in Word 2007](image-url)
Or you can select **Create PDF** from the **Acrobat** ribbon as shown in **Figure 4**. Either one of these options will open the same dialog box.

![Figure 4 Selecting create PDF from the Acrobat ribbon](image)

Before saving, select **Adobe PDF conversion options** and ensure that **Create Accessible (Tagged) PDF file** is selected as shown in **Figure 5**.

![Figure 5 Adobe PDF Conversion Options](image)
Word 2007 with Microsoft Add-in installed

With Word 2007 you can install the free 2007 Microsoft Office Add-in: Microsoft Save as PDF to create tagged PDFs without installing Acrobat. The process for exporting to PDF is basically the same with each. If you want to verify the accessibility of the PDF or edit the tags that are created, you will still need Acrobat Professional.

To convert to PDF using the Microsoft add-in, click on the Office Button, hover over Save As, and select PDF, or press Alt + F, F, P as shown in Figure 6.

![Figure 6 Selecting Save as PDF in Word 2007](image-url)
Before you save the file, select **Options** and ensure that the **Document structure tags for accessibility** option is selected as shown in **Figure 7**.

**Figure 7 Save PDF options**
**Acrobat Professional**

If you are having trouble converting a document to tagged PDF in Office, or if you want to merge multiple documents into one tagged PDF file, you can convert a file to PDF in Acrobat. There are several ways to do this; one of the easiest is to select File > Create PDF > From File.

If the file format is supported (i.e. the file is created in a Microsoft or Adobe product), the file should be tagged as it is converted. If no tags are present, select Edit > Preferences > Convert to PDF, choose the correct format, select Conversions Settings, and ensure that **Enable accessibility and reflow** is selected as shown in Figure 8.

![Figure 8 Selecting Enable accessibility and reflow](image)

If you want to combine multiple files into one PDF, select File > Create PDF > From Multiple Files. A dialog box will appear. Select Options and then select **Always enable accessibility and reflow** – see Figure 9.

![Figure 9 Always enable accessibility and reflow option](image)
Tags Panel

Note: Starting with Acrobat 8, Adobe has started calling tabs "panels." If you are using Acrobat 7 or earlier, you can replace the word "panel" with "tab" every time it appears.

The Tags panel allows you to view, reorder, rename, modify, delete and create tags. To view the Tags panel, select View > Navigation Panels > Tags. It may be easier to view the tags window if it is docked in the side bar. To do this, click View > Navigation Panels > Dock all Panels or drag the Tags panel into the side bar. For some reason, dragging the entire window will not cause it to dock. It must be dragged by the panel.

After expanding <Tags> and <Sect>, a long list of tags should be visible – see Figure 10. The list can be navigated, expanded, and collapsed using a mouse or keyboard.

Figure 10 Tags panel
Many of the tags are similar, if not identical to HTML. Some of these familiar tags are described in the table below:

<table>
<thead>
<tr>
<th>Tag</th>
<th>Stands for</th>
<th>Description/Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;H1&gt; to &lt;H6&gt;</td>
<td>Heading</td>
<td></td>
</tr>
<tr>
<td>&lt;P&gt;</td>
<td>Paragraph</td>
<td></td>
</tr>
<tr>
<td>&lt;L&gt;</td>
<td>List</td>
<td>Similar to &lt;ul&gt; or &lt;ol&gt; in HTML</td>
</tr>
<tr>
<td>&lt;LI&gt;</td>
<td>List Item</td>
<td>Should be nested just as in HTML, which can get very confusing.</td>
</tr>
<tr>
<td>&lt;Table&gt;&lt;TH&gt;&lt;TR&gt;&lt;TD&gt;</td>
<td>Table, Table Row, Table Header, Table Data</td>
<td></td>
</tr>
<tr>
<td>&lt;Figure&gt;</td>
<td>Figure</td>
<td>similar to &lt;img&gt; tag in HTML</td>
</tr>
</tbody>
</table>

**Highlight content**

One of the first things you should do in the tags panel is to select the **Highlight Content** in the **Options** panel – see **Figure 11**. When this option is selected, selecting a tag should highlight the corresponding text, image, or other element in the PDF file.

![Figure 11 select the Highlight Content in the Options panel](image)
Find Tag From Selection

Just as selecting a tag highlights the content in the body of the PDF file, there is a way to accomplish the opposite effect of highlighting the tag that corresponds to selected content. First, click on the Select Tool. Next, select a portion of text, an image, or a table. Finally, select Options ›Find Tag From Selection from the dropdown list in the Options menu at the top of the Tags panel. This will highlight the tag or tags that contain the content you previously selected.

Change tags

At times, you will encounter a PDF file that contains incorrect tags. You can modify these by right-clicking the tag you want to change by selecting Properties – see Figure 12, then the Tag tab, and then selecting the appropriate new tag type from the dropdown list labeled Type.

Figure 12 selecting tag Properties

For example, to change a tag to <H1> instead of <Normal> select the <Normal> tag you want to change and do the following: Right click the tag and select Properties and select Heading Level 1 from the list labeled Type.
Add tags to an untagged document

If you do not see any tags in the tags panel, your document is untagged, and you will need to tag the file. To add tags to an untagged document, choose Advanced > Accessibility > Add Tags to Document. This process can sometimes be extremely time-consuming, and you will almost certainly have to edit the new tags manually. Still, it is a start and will probably be faster than doing all the work manually. This is especially true if the document contains tables.

TouchUp Reading Order

Note This feature is only available in Acrobat 7 and later.

The TouchUp Reading Order feature is a new addition to Acrobat 7 and later. This is the first real feature to offer a fast way to tag a PDF. Although it can speed up the tagging process, it does not take the place of the other tools mentioned previously.

To use the TouchUp Reading Order tool, select Advanced > Accessibility > TouchUp Reading Order. When this feature is selected, the view on the screen will change. All of the content will be enclosed in numbered boxes. Each of these boxes represents a tag and the number corresponds with the tag number in the Order panel. The TouchUp Reading Order window will also open as shown in Figure 13.

Figure 13 TouchUp Reading Order window
Note If you have TouchUp Reading Order open and don't see any boxes, your PDF document is probably untagged.

You will notice a group of buttons with the names of several common tags. You can use these buttons to assign tags to a selected block of text or an object.

TouchUp Reading Order is not perfect. Selecting text is difficult and many tags are not supported. It is also somewhat difficult to assign alternate text to images. Still, it is easier to mark up a PDF file using this feature than any other way.

Adding/Changing tags

There are two ways to select an area of text, image, table or other element using TouchUp Reading Order.

1. Drag a box around an element using the crosshairs that have replaced the default pointer. It is sometimes difficult to select exactly the right area, but it is a little easier if you try drawing a box that is slightly larger than the element. Once you have selected a new element, you can assign some of the most common tags to that element by clicking on one of the several buttons.

2. You can also select everything within a box by clicking on the number at the top left corner.

Now that you have selected the text, you can assign a tag by clicking on the corresponding button. Once you select a button, Acrobat will place the selected content in the appropriate tag. If you have the Tags panel open, you can view these changes instantly.

While in TouchUp Reading Order, you can also assign alternate text to images by Right-clicking on the image and choosing Edit Alternate Text.
## TouchUp Reading Order Options

<table>
<thead>
<tr>
<th>Button</th>
<th>Adobe Tag</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>&lt;P&gt;</td>
<td></td>
</tr>
<tr>
<td>Form Field</td>
<td>&lt;Form&gt;</td>
<td></td>
</tr>
<tr>
<td>Heading 1</td>
<td>&lt;H1&gt;</td>
<td></td>
</tr>
<tr>
<td>Heading 2</td>
<td>&lt;H2&gt;</td>
<td></td>
</tr>
<tr>
<td>Heading 3</td>
<td>&lt;H3&gt;</td>
<td></td>
</tr>
<tr>
<td>Figure</td>
<td>&lt;Figure&gt;</td>
<td></td>
</tr>
<tr>
<td>Figure/Caption</td>
<td>&lt;Figure&gt;</td>
<td>If you select the image and the nearby caption image will be tagged as a figure and the text will be tagged as its caption.</td>
</tr>
<tr>
<td></td>
<td>&lt;Caption&gt;</td>
<td></td>
</tr>
<tr>
<td>Table</td>
<td>&lt;Table&gt;</td>
<td>Acrobat attempts to assign rows, columns, and headings. Sometimes it does this correctly, but this should still be checked with the table inspector.</td>
</tr>
<tr>
<td></td>
<td>&lt;TR&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;TH&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;TD&gt;</td>
<td></td>
</tr>
<tr>
<td>Cell</td>
<td>&lt;TD&gt;</td>
<td>Can be used to merge cells if they are incorrectly split</td>
</tr>
<tr>
<td>Formula</td>
<td>&lt;Formula&gt;</td>
<td></td>
</tr>
<tr>
<td>Background</td>
<td>none</td>
<td>Also called an artifact, this will hide an item completely from a screen reader.</td>
</tr>
</tbody>
</table>
Order Panel

Note This feature is only available in Acrobat 7 Professional and later.

Although the reading order of a PDF can be changed by dragging tags around within the Tags panel, there is an easier way in Acrobat 7 and newer. To open the Order panel, select or View > Navigation Panels >Order or select Show Order Panel in the TouchUp Reading Order tool as shown in Figure 14.

Figure 14 Show Order Panel

At first glance, it resembles the Tags panel, but there are a few differences.

- The document is divided into pages.
- Each element is numbered, and the numbers start over on each page.
- There is no hierarchy of elements; everything is on the same level.

These differences help make the Order panel a much easier way to reorder tags. To change the order of a tag, just click and drag the tag to the location that reflects the correct reading order. This new order will be reflected in the Tags panel and when the document is viewed in Reflow mode.

This feature is also available in the TouchUp Reading Order tool.
Alternative text

The easiest way to add alternative text is with the TouchUp Reading Order tool. When an image is tagged as an image (or figure), the alternative text will appear next to the image. If the image has no alternative text, the caption will read "Figure - No alternate text exists."

To add alternative text, right click on the image and select Edit Alternate Text. Enter the appropriate alt text in the dialog box. If the image is just decoration you can add in

Making an Image a Background Element

Images that do not add value to the document and are purely decoration should be set as a background object. These objects will not be announced by screen readers.

To make an image a part of the background from the main menu bar click on Advanced > Accessibility > TouchUp Reading Order.

The “TouchUp Reading Order” window will be displayed as shown in Figure 15.

Select the object that is to be made “Background” by selecting it with the mouse.
Click the “Background” button or press **Alt + B**.

Click on Close or Press **Alt+ C**

The image has now been set as background.

**Table Inspector**

**Note** This feature is only available in Acrobat 8.

The table inspector allows you to easily identify and assign scope to table headers.

With the TouchUp Reading Order tool open, select a table and then select **Table Inspector**. You can now select table cells that should be headers, **right click** on a selected cell or cells and choose **Table Cell Properties**. A dialog box will appear as shown in **Figure 16**.

![Table Cell Properties dialog box](image)

**Figure 16 Table Cell Properties**

If the selected cell(s) needs to be tagged as a header, select the **Header Cell** option and assign a scope of either **Row** or **Column**. After selecting **OK**, you will notice that the table header cells will be highlighted in red and the data cells will be highlighted in gray as shown below.

<table>
<thead>
<tr>
<th>Header</th>
<th>Header</th>
<th>Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>Data</td>
<td>Data</td>
</tr>
<tr>
<td>Data</td>
<td>Data</td>
<td>Data</td>
</tr>
<tr>
<td>Data</td>
<td>Data</td>
<td>Data</td>
</tr>
</tbody>
</table>
Artifacts

Artifacts are elements that are ignored by a screen reader, much like an image with empty alt text in HTML is ignored. Important text should never be labeled as an artifact. As with alt text, the easiest way to change an element to or from an artifact is with TouchUp Reading Order tool. Select the element and press the **Background** button to make it an artifact.

To search for artifacts, within the Tags panel select **Options > Find**. A window will appear with several search options. Artifact is the default search, so click Find. If there are any artifacts, the search will identify them and allow you to change them to other elements.

To change a tag to an artifact in the tags panel, **right click** on the item and select **Change Tag to Artifact**.
Adobe InDesign

Adobe InDesign CS generates tagged accessible PDF files. You can improve the accessibility and reuse of Adobe PDF documents by adding tags to the InDesign document before you export. If you do not get the results you want in the exported Adobe PDF file, you can use tools in Acrobat Professional to edit the structure of tagged PDF documents.

When you apply tags to a document for PDF export, the tags do not control which content is exported to PDF, as is the case with XML export. Instead, the tags give Acrobat more information about the document’s structural contents.

Preparing an Adobe InDesign Document for Making Accessible PDF

Tagging

There are a variety of ways to work with tags in InDesign CS. This example will illustrate the easiest method for adding tags to an unstructured InDesign CS2 file. InDesign CS2 allows you to add tags to an untagged InDesign document. Choose Window > Tags to display the tags palette. To determine if your InDesign document is structured, select View > Structure > Show Structure, or Alt + Ctrl + 1, to display the InDesign structure pane. At this point your untagged document will look something like the example shown in Figure 16.

![Figure 16 InDesign Tags Pane and Palette Prior to Adding Tags](image-url)

October 2011  AbilityNet Charity No. England and Wales 1067673 – Scotland SC039866
Telephone: 0800 269545 - email: enquiries@abilitynet.org.uk - web: www.abilitynet.org.uk
The next step is to add tags to the InDesign document. To do this, choose “Add Untagged Items” from the Structure Pane Menu - See Figure 17

Figure 17 Selecting Add Untagged Items from the InDesign Structure Pane.
This causes a structure tree to appear in the structure pane and colour coded tags to appear in the tags palette which correspond to coloured items on the document view.

Figure 18 Result of Adding Untagged Items Command in InDesign
To determine which graphic images on the page correspond to individual figure/image tags appearing in the structure pane, highlight the figure/image tags and right click to select the “Go to Item.” The graphic object will be highlighted with a blue border and small square “handles” will appear on the corners and the middle of the graphic – see Figure 19.

![Figure 19 Relating a Figure Tag to a Page Graphic in InDesign](image)

**Figure 19 Relating a Figure Tag to a Page Graphic in InDesign**

**Note:** You can also relate an item on a page with its associated tag by using the InDesign Selection Tool, or the Direct Selection tool to highlight the item of interest. Right click on the selected item and choose Select in Structure. The tag representing that page item will be highlighted in the Structure Pane – see Figure 20 below.

The ability to relate items in the page view to items in the structure view is helpful for establishing a logical read order for items intended to be exported to PDF and XML from InDesign CS.
There are other ways to incorporate tags into InDesign CS2 documents. If you’ve styled your document using paragraph and character tags, you can map those styles to XML tags, and let InDesign apply the tags to the styled content. You can map more than one style to the same tag, and edit the mappings at any time. Refer to the InDesign documentation for detailed information about these other techniques.
Adding ALT Text to Graphics in InDesign

In InDesign the ActualText attribute lets you substitute an image that is part of a word, such as when a fancy image is used for a drop cap. In this example, the ActualText attribute allows the drop cap letter to be read as part of the word.

With Adobe InDesign you add the Alt attribute to the Figure / Image tags. Right click on the desired figure / image tag and select “New Attribute” – See Figure 21.

![Figure 21 Adding an Attribute to a Figure Tag in InDesign CS](image)

If necessary, choose View > Show Structure to display Structure pane, and choose Window > Tags to display the Tags palette.

In the Name field of the New Attribute Dialog, type either ‘Alt’ or ‘ActualText’ (this feature is case-sensitive).

For Value field of the New Attribute Dialog, type the text that describes the image – see Figure 22

![Figure 22 New Attribute Dialog in InDesign CS](image)
Generating Accessible PDF from Adobe InDesign

Once the Adobe InDesign CS file has been set up to generate accessible PDF by using structure tags and associating an Alt attribute or ActualText attribute for each Figure/Image tag appearing in the document, you are ready to export the InDesign document as an accessible PDF file.

To do this, you select **File > Export** from the InDesign menu or type **Control +E**. Select Save and the Export Adobe PDF dialog will appear. Be sure that the check box labelled Create Tagged PDF is checked. Also have Bookmarks and Hyperlinks checked- see **Figure 23**.

Select export and an accessible tagged PDF will be generated from the Adobe InDesign document.

![Figure 23 Export Adobe PDF Dialog in Adobe InDesign CS](image)

For more details of creating accessible PDFs from Adobe products see **http://www.adobe.com/accessibility/**
Appendix A Word to PDF conversion Tutorials

Creating an accessible PDF document from Microsoft Word 2003 using Adobe PDF Maker

One of the most common ways PDF are created is from Microsoft Word documents. To do this you need to have the Adobe’s PDF Writer Plug-in installed. Please note there are many other tools to create PDFs but using Adobe’s PDF Writer Plug-in is the easiest way to make accessible PDFs.

To create an accessible PDF from a Word document you must first ensure it has been well structured.

To do this:

1. Add alternative-text (e.g. alt-text) to all images in the Word document via the format picture option - the ‘Web’ tab is where you put the alt text.

2. Use styles for headers and lists. Specially use the Headings – Heading 1 to Heading 5 to mark up your section heading –these can be accessed via the Styles and Formatting menu in the Task Pane as shown in Fig 1. This structure can then be easily converted to the equivalent in the PDF document.

Fig 1 Styles and formatting pane
3. Create hyperlinks (e.g. links) in the text.

Then to create an accessible PDF from a Word document:

4. Go to: Adobe PDF (menu) > Change conversion settings. As shown in Fig 2 below [You only have to set it up once and save your new settings] There, verify that the settings of each of the tabs are as explained below:

**Fig 2 Adobe PDF Writer menu in Word**

**Settings tab:**

- Set conversion settings to “standard”
- Check mark all the options but “Attached source file to Adobe PDF”

**Security tab:**

- Uncheck the first two options “require a password to open the document” and “use a password to restrict printing and editing of the document and its security settings”.
- Check the options “Enable to copying of text, images and other contents” and “enable text access from screen reader devices for the visually impaired.”

**Bookmarks tab:**

- Check the option “Convert Word Headings to bookmarks” as shown in Fig 3 below.
- Uncheck the option: “Convert Word styles to bookmarks and click on the OK button.”
5. In Word, go to the Adobe PDF menu and select Convert to Adobe PDF

Save the file and the conversion process will then take a few minutes and you will have created an accessible PDF.
Word 2007 PDF tutorial
Please note the plug-in does not work for previous versions of Word including 2003.

First download and install the plug-in and then start Word 2007.


Create word document
Follow the principles of a well structured document using headings and bulleted lists and adding alt text to images.

Use heading styles
Ensure the section headings are coded correctly using heading styles – for example see Fig 1 below. These option are found under the ‘Home’ tab. Click on the Home tab or press ‘Alt’ then ‘H’ if on a different tab such as ‘Insert’ or ‘View’.

Fig 1 coding headings with heading styles
Alt text for images

If there are images in the Word document you need to add alternative text to them via the ‘Size’ option.

To access this option right click on the image to bring up the options menu as shown in Fig 2 and then click on ‘Size’ or press ‘Shift’ + ‘Z’.

Fig 2 Image options

You then need to click on the ‘Alt Text’ tab or press the right arrow key once. Next, write in alt text description by clicking in the ‘Alternative’ text box or by pressing ‘Alt’ + ‘A’. An example is shown in Fig 3. When you have finished click on Close or tab to the ‘Close’ button and press ‘Enter’.
Having created your accessible Word document you then need to save it as a PDF. To do this click on the Office button or press ‘Alt’ then ‘F’ then ‘F’ again. Then, as shown in Fig 4 below select the PDF or XPS option or press ‘P’.

**Fig 4 Selecting the Save as PDF option**
PDF Options

From the dialog box in Fig 5 click on the ‘Options’ button or press ‘Alt’ + ‘A’ to bring up the Options dialog box as shown in Fig 6.

![Fig 5 Publish as PDF dialog box](image_url)
**Fig 6 Options dialog box**

In **Fig 6** make sure that you select ‘Create bookmarks using:’ checkbox by clicking on it and or by pressing ‘Alt’ + ‘C’ if not already selected.

Then if not selected click on Headings radio button or press ‘Alt’ + ‘H’.

Then if the ‘Document structure tags for accessibility’ checkbox is not checked click on it or press ‘Alt’ + ‘M’ to select it.

Then click ‘OK’ or tab to the ‘OK’ button and press ‘Enter’ to create your PDF.

You will then need to check your PDF in Adobe Acrobat for any other accessibility tweaks you will need to add such as specifying a reading order through the document.
Appendix B Accessible PDF checklist

This checklist summarises the key requirements to make PDFs accessible to people with disabilities. For detailed technical guidance see the Creating Accessible PDF Documents with Adobe Acrobat 7.0 which can be downloaded from http://www.adobe.com/accessibility/

Do not create PDFs from image only scans

This type of PDF is the least accessible, Adaptive technology such as screen readers will not be able to access any information in this type of document. To make it assessable you would need to run it through OCR software to make it text searchable.

Create Tagged PDFs

A tagged PDF contains text (e.g. real-text) and it contains tags which help to identify order and structure of the document such as headings, figures and text. PDF tags are a similar concept to HTML tags in Web pages. You can either structure documents to enable tagging during PDF conversion such as Word documents or you can use Acrobat Professional or Acrobat standard to add tags after the conversion to PDF.

Have a logical reading order

This is important for Adaptive technology like screen readers that rely on making sense of the document by the order it is read out for example the correct heading should be read out before the relevant section text or a figure caption read out just after the related figure. Tagging a PDF normally means a logical reading order is created by default; however on more complex PDFs such as Newsletters with multiple columns or those containing financial data Adobe Acrobat Professional will be needed to fix the reading order.

Provide alternative text for all graphics, links and form fields

Also known as descriptive text, alternative text is very important for screen readers and other adaptive technology that cannot make sense of graphical elements by themselves. You can add Alternative text in some documents such as Word but often you will need to use Acrobat Professional to add descriptions to non text elements.
Use navigation aids such as bookmarks
Providing simple and intuitive ways to navigate a PDF are very helpful feature for many people with disabilities and are of benefit to non disabled people as well. Bookmarks are an easy way to navigate the sections of a document and are normally created automatically when converting a well structured document (such as a document with proper headings) to a tagged PDF. Other navigation aids include links in the pages where appropriate and clear and logical headings as well as a logical tabbing order through form fields if present. Many of these features can be created/modified in Acrobat Professional.

Specify a document language
This is important for screen readers which need to switch to different language engines if a document is not in their default language. In Acrobat professional you can specify multiple languages in a PDF.

Use Accessible fonts
Not only must the fonts you use be legible but they must contain enough information for Acrobat or Adobe reader to render them as understandable text which is particularly important for screen readers and the inbuilt Read Out Loud tool. Also when users cut and paste or convert PDFs to text some fonts do not convert properly and you will get strange characters such as black rectangles, question marks etc. Most popular fonts do not have this problem; it is often more to do with specialised/customised fonts. You need to use Acrobat professional to check if your document has inaccessible fonts.

If form fields are used they must be accessible
Form fields must be interactive – in other words it must be possible to put content in them and the form field descriptions must be readable and make sense to screen readers. The form fields must also have a logical tab order.

Make sure security settings do not interfere with screen readers
Setting the default security option to not allow copying can cause problems for screen readers trying to read the document so you must ensure the permissions security option ‘Enable text access for screen reader devices for the visually impaired’ is selected.
Low vision/ Colour blindness accessibility

You need to ensure that there is enough contrast in the PDF document between text and background colours. Use a tool such as Colour Contrast Analyser: http://www.paciellogroup.com/resources/contrast-analyser.html

Also be careful about using colour only when referring to something in the document like a graph or chart – don’t say ‘look at the red line on the graph’ for example – be more specific because people with colour blindness cannot differentiate certain colours.

For Motor disabilities accessibility

Don't make hot spots around links too small – you can easily change link settings to change the hot spot areas. The key point to be aware of is that the smaller the link, the more difficult it will be for someone with limited fine muscle/motor control to click on it.

For hearing disabilities accessibility

You can add sound and movie clips to PDFs if you have Adobe Acrobat Professional, 6.0 or later. If you want to do this make sure you provide transcripts for any sound file such as wav.

Also if you embed movie files (such as QuickTime, Flash Player, Windows® Media Player etc) provide synchronized captions for the video, alternatively provide a transcript.

For cognitive disabilities accessibility

The number one rule is use clear and simple language – keep jargon to a minimum and provide a glossary for any technical terms you use. The better you write, the better you will be understood by everyone, not just those with cognitive disabilities.

Also people with cognitive disabilities or dyslexia might use the Read Out Loud tool built into the later version of Adobe Reader – or they might use other software to read out the document, so it is always a good idea to run through a PDF document using the inbuilt Read Out Loud feature to ensure it makes sense.

Test your PDF with a screen Reader

If possible run your PDF through a screen reader such as Jaws. Instructions for reading PDFs are included within the Jaws help system. You can download a 40 minute demonstration version of Jaws from http://www.freedomscientific.com/fs_downloads/jaws.asp