Adapt Accessibility Standards and Guidelines

HTML Accessibility

Core purpose

The Adapt Accessibility Standards and Guidelines outline the requirements and recommendations necessary for ensuring accessibility to the widest possible audience specifically intended to help users with visual or motor impairments.

The rationale behind these guidelines is to ensure the framework adheres to the W3C2 AA standards where possible.

The goal is to aid users and screen readers by making every piece of content tabbable and readable. This document sets out recommendations on how to achieve this goal in Adapt core.

Audience awareness and component content

Some components will only ever be inaccessible. Those which rely on making sighted judgments or those components which require a level of visual special awareness should be specifically labelled as inaccessible.

Note: It is possible to make an otherwise accessible component inaccessible. This will generally happen when a component's content relies too heavily on the user's visual spatial awareness. Adapt's GMCQ and Graphic can easily present this problem. As a result it is essential to only have images which support the text content, images which could be removed entirely without affecting the content or images which can be described aptly in a brief textual summary.

Landmarks

WAI-ARIA Landmark Roles are used to help keyboard users with assistive technologies such as screen readers to navigate between content.

It is generally important not to embed elements with aria roles inside elements with aria roles. Some screen readers have difficulty processing 'aria-label' attributes within these structures. If any div or span with a role and aria-label is found in Adapt, the role and label will be removed and injected into a transparent span above the DOM element in question.

This allows the aria-label to be processed by every screen reader. The role="region" should be the only role type used in Adapt components. (aside from the drawer and navigation bar which have 'aside' and 'navigation' respectively).

However other roles can be applied where applicable as outlined below:

“button” - to be applied to all buttons.

“list” and “listitem” - applied in *adapt-contrib-resources* and *adapt-pageLevelProgress. “*list*”* role applied to containing div and “listitem” to each item.

“navigation” - applied to navigation.

“main” - applied to page.

“heading” - to be applied to all heading titles.

“menu” and “menuitem” - applied to *adapt-contrib-boxmenu.*

“textbox” - applied to *adapt-contrib-textinput* input text field.

“slider” - applied to *adapt-contrib-slider* handle element.

Headings

‘h’ tags are used in html markup to describe and to style text. The ‘h’ tags are also used by screen readers to allow users to quickly navigate to relevant page headers. Headers are given a ‘level’ to describe their depth to a screen reader user. Adapt has a reasonably complex and nested DOM structure and so, depending on the screen reader, the header tags will often be attributed an inconsistent level. To avoid this issue it is necessary to use an alternate header syntax:

<span class=”h1 accessible-text-block” role=”heading” aria-level=”2” tabindex=”0”>{{title}}

And not

<h1 tabindex=”0”>{{title}}</h1>

Headers are to be labelled according to their hierarchy in the page or popup. Such that a page or menu header will be level 1, article is level 2 etc. A popup title would start from level 1 as it will be the only region available to the screen reader.

The aria-level system does not change an element’s styling however classes .h1-.h6 have been implemented to mimic the styles of tags h1-h6 (*font.less* in adapt-contrib-vanilla theme).

DOM order and Tabindex

It is required to give tabbable elements a tabindex=”0” attribute. Tabindex=”0” allows an element to be tabbable but does not specify the order in which an element receives tab focus. Tabindex=”-1” removes an element from the tab flow and adding the class .a11y-ignore will stop the tabindex changing on popup and popdown operations.

this element will not be in the tab flow

The order of DOM elements must be such that the first tabbable item appears as the first child of its DOM parent. This way the tab flow cascades in content order (Title, body, content, controls, etc). Positive tabindex attribute values must not be used to create a logical tab order.

There are some natively tabbable elements which do not require a tabindex=”0” attribute but which are also not hindered by having one set.

Title attributes

Title attributes are inaccessible to keyboard users without additional Assistive Technology. They are dependent on user settings in Screen Readers and similar Assistive Technology.

Additionally they suffer from discoverability problems and repeating content in visible text and title attributes can lead to content clutter and repeated phrases. Instead *aria-labels* and *aria-labelledby* attributes should be used where appropriate. (example given ahead in **Form Labels**).

Interaction descriptions

Each component has a description of what it represents visually and sometimes how the user should interact with it. These descriptions need to be included at the top of each component template within an ‘aria-label’ attribute. The label will be removed and injected into a transparent span above the DOM element making it processable by every screen reader:

<div class=“accordion-inner component-inner” role=“region” aria-label=“{{_globals._components._accordion.ariaRegion}}”>

The global component labels are stored in the course.json.

Making content readable

Using the helper function *a11y_text* automatically formats text or html into accessible html. This should be used in all adapt templates where appropriate.

Complex content such as body and instruction text must become tabbable and appropriately wrapped. They should be written as follows:

{{a11y_text body}}

```
<div>{{a11y_text instruction}}</div>
```

Any other text which must be accessible but which does not require complex handling:

```
<div role="region" tabindex="0">Duration: {{duration}}</div>
```

Focusable controls

When creating controls for user interaction in JavaScript, for example previous and next controls, the controls must be implemented with elements that provide natively focusable elements with click, keydown, and focus events so they are accessible to keyboard as well as pointing device users.

Sometimes it becomes necessary to control the focus. It is possible to explicitly set focus to a DOM element (or any available+tabbable child, or any available+tabbable+subsequent element) using:

```
$("selector to element").a11y_focus();
```

This is especially useful at the beginning or end of a child item list whereby the previous and next buttons are removed or disabled accordingly.

To set the focus to the first tabbable element on the page use:

```
$.a11y_focus();
```

To enable and disable a link, button or any other form element; in JavaScript use:

```
$("selector for element").a11y_cntrl_enabled(true/false);
```

This will handle the element's enabled states and its tab indexes.

Content alerts - selections

As some screen readers are unable to perform selection alerts automatically, inform that user that a selection has been made by using the JavaScript below:

```
$('selector to element with text to read').a11y_selected(true/false);
```

This will force the browser to read the text contained in the element, prepended with the word "Selected ";

Content alerts - popups

There is a small change to the Adapt popup architecture. The popup:opened trigger now requires the parent element of the popup. This is so that that jquery.a11y popup manager can disable all other tabbable elements and restrict the tab index to the popup. This action is revered on popup:close.

```
Adapt.trigger('popup:opened', this.$el);
Adapt.trigger(‘popup:closed');
```

As these processes produce changes in the tab indexes it is essential to remember to make ‘popup:opened’ your last DOM changing call in a sequence. This is so the DOM beneath the popup will not change until the popup is closed.

Likewise on ‘popup:closed’ it is important to remember that to restore the tabindexes in the DOM beneath the popup, you must trigger ‘popup:closed’ before effecting any other changes in the DOM. Failing to do so will likely prevent the restoration of the tabindexes and focus to its pre-popup state.
The notify and notifyPush templates should have an ‘aria-label’ saying ‘popup opened’ and a role of ‘region’.

Control Styles

Links that are part of general content must be self evident, identifiable by their visual style, and distinguishable from the surrounding content.

Text links must have a mouse-over state change.

To aid discoverability all links must be made self-evident with their visual style.

Appropriate styles are:

- underlined.
- a different colour, meeting contrast standards, to the surrounding text.

When hovered over links should have a change in style as confirmation that they are links.

Appropriate styles are:

- adding an underline to previously not underlined content.
- a change in colour, with a sufficiently different contrast to meet the contrast standard compared with the default state colour.

Focus Styles

All focusable elements must have a clearly identifiable visual style when they have focus.

Sighted keyboard users do not have the explicit association between pointer and content that pointing device users have, so it is important that they are aware at all times what element currently has focus and which element keyboard interactions will operate on.

The currently focussed element should therefore have a visual style that makes it clearly distinguishable from the surrounding content.

Adapt Core (*base.less*) has a default focus outline variable set for no-touch devices, *@focus-outline*. This variable requires to be set in the *variables.less* in *adapt-contrib-vanilla* theme.

Colour contrast

All colour combinations must pass the WCAG 2.0 AA-compliant colour contrast check.

This is a ratio of 4.5:1 for text 18pt or less in size, and 3:1 for text larger than 18pt or text that is bold and larger than 14pt.

Where it cannot be adapted, stylised text used in pre-existing logos and branding is exempt from this requirement.

If there is sufficient contrast between foreground and background colours, particularly between text and its background then users with moderately low vision or with colour deficiencies that affect contrast to a minor degree are more likely to be able to access content without requiring additional assistive technologies.

Colour and meaning

Information conveyed with colour must also be identifiable from context or markup. When colour is used to enhance visual display, people who can't access colour will not necessarily suffer from reduced usability. They will still be able to access the materials on the site and to operate the site functions. However, when colour is an integral part of the user interface, people who cannot access colour may encounter difficulties.

For example: "Please select the red button to move on".

Image alternatives and Aria Labels

All content graphics must have a text alternative either as the value of the alt attribute or in the immediately surrounding text content. This includes either a non-null alt attribute value, or a text alternative in the accompanying content.

Assistive technologies such as screen readers will provide a text alternative for images that do not have alt attributes based on the image file name. For images that do not have content significance, such as background graphics, or are described in the surrounding text content it is appropriate to use a null *alt=""* value for the image alt attribute to avoid this.

The alt attribute should only be used to provide text descriptions of images. Where other text descriptions are required it is recommend to use ‘aria-labels’ in Adapt to keep consistency of descriptive text implementations throughout, to display a clear intent of use to other programmers.

The ‘aria-label’ attribute is a description that is never displayed on screen but is relayed to a screen reader user. It defines a string that labels that current element. ‘aria-labels’ are recommended as ‘alt’ tags are often read in inconsistent styles by various screen readers whereas ‘aria-labels’ are generally read very well.

So only use ‘alt’ attributes for content graphics. ‘aria-labels’ to be used for elements.

Form Labels

Form labels are important for all users in order to understand what the form field is however they are essential for speech output users who cannot easily infer what the form element is by looking at the surrounding content.

Label and form input elements are associated with each other and must have an associated label, such that the text in the label is read for the focus on the input tag. Unfortunately the aria and html systems of association are in opposition and sometime neither work. Both must be included creating duplicate readings with some screen readers and the label text should also be included on the input element. This will guarantee that the correct label is read across all screen readers:

```
<input id="input-id" aria-labelledby="label-id" title="{{ labelText }}”/>

<label id="label-id" for="input-id">{{ labelText }}</label>
```

Component state partial

A file called ‘state.hbs’ is required in the theme/templates folder *adapt-contrib-vanilla*. It is similar to the ‘components.hbs’ file, in that it is common to all components.

The ‘state.hbs’ file renders a small and transparent text block describing the state completion and correctness of the component.

Currently the my/model answer button functionality doesn’t meet the desired level of accessibility for screen reader users as it does non-impaired users. The addition of the component state however offers impaired users an alternative access to their marking / current progress. This functionality is still a working progress.

It should look something like this:

```
<div class="accessibility-state">
    {{#if _isComplete}}
        <span tabindex="0" role="region" class="aria-label a11y-ignore-focus prevent-default ">
{{a11y_normalize displayTitle}}
{{_globals ._accessibility._ariaLabels.complete}}
{{#if _isQuestionType}}
{{#if _canShowFeedback}}
{{#if _isCorrect}}
{{_globals._accessibility._ariaLabels.correct}}
{{else}}
{{_globals._accessibility._ariaLabels.incorrect}}
{{/if}}{{/if}}{{/if}}
</span>
    {{else}}
        <span tabindex="0" role="region" class="aria-label a11y-ignore-focus prevent-default"
id="buttons-aria-label-incomplete">
{{a11y_normalize displayTitle}} {{_globals._accessibility._ariaLabels.incomplete}}
</span>
    {{/if}}
</div>
```

The class “.accessibility-state” is required to allow for removal on state changes.

The class “.no-state” can be applied to any component which doesn’t require the state to be rendered such as *adapt-contrib-blank* or even some text / graphic components.

Mobile Accessibility

Adapt content is accessed different depending on device. When accessibility if first enabled navigational instructions are provided based on user device. This includes touch, no touch and iPad.

Touch gestures include, swipe left for next, swipe right for previous, double tap to select and two finger slide up to go to the top of the page.

No touch gestures include, tab for next, shift tab for previous, enter to select and escape to return to the navigation.

iPad instructions provides both instructions to include touch and keyboard access.

iPad and iPhone are best supported at present and are the target devices of choice. (See page **10**).

Screen Reader Testing Guidelines

As with normal browser testing guidelines it is possible to provide general rules for which browsers to test with which screen readers. Because screen readers and other assistive technologies work in combination with browsers it is important to restrict the number of versions that are used during routine testing so that this does not take undue resources. Therefore for browsers and assistive technologies with good and practical upgrade paths the latest versions are sufficient testing. For browsers and assistive technologies with more problematic upgrade paths, for example due to ties to a particular operating system or expense, then a broader range must be considered, but should still be as few in number as possible.

JAWS

JAWS is expensive software and therefore has many users with older versions. Testing the latest and latest -1 versions provides reasonable coverage. It is also the most widely corporately supported screen reader which ensures it remains popular despite its price tag.

JAWS works best with Internet Explorer and Firefox so the majority of JAWS users will use one of these two browsers.

Testing carried out - Internet Explorer 8 - 11 / Windows 7.

NVDA

NVDA is free software, therefore it is reasonable to expect users to upgrade regularly.

NVDA works best with Internet Explorer and Firefox so the majority of NVDA users will use one of these two browsers.

Testing carried out - Internet Explorer / Firefox / Safari.

Google Chrome not recommended.

VoiceOver OSX

VoiceOver is bundled with OSX and works best with Safari and Firefox so the majority of VoiceOver users will use one of these two browsers.

OSX 10.9 (Mavericks) is a free upgrade so it is reasonable to expect users to have an up to date version.

Testing carried out - Google Chrome / Safari / FireFox.

VoiceOver iOS

VoiceOver is bundled with iOS and works best with Safari.

iOS7 is a free upgrade available on iPhone 4 onwards, iPod Touch 5th generation, iPad 2 onwards, and iPad Mini 1st generation onwards so it is reasonable to expect users to have an up to date version.

Testing carried out - to be updated.

Android

- Currently not supported.