Accessible Collapsable Forms/Information Areas

Accessible Expand/Collapse areas

Forms that can be toggled open and close don't work too well by default implementation. When triggering the container of a form to open, resulting in the form being visible, focus is hard to manage on the elements within. For open and close triggers that appear further down in the DOM architecture than the collapsable form it's advised to automatically set focus to the top of the form section, preferably on a descriptive heading or some instruction that indicates they are now within the collapsable area.

We implement a similar technique to the accessible modal/popups by placing focus (using javascript) onto an element after it becomes visible except we don't need to use a focus trap.

An example of Expand and Collapse areas can be found on the FlyBuys Offers page and the FlyBuys Rewards pag
e

aria-expanded

Where possible, an attribute of `aria-expanded` should be added to triggers that expand / collapse areas

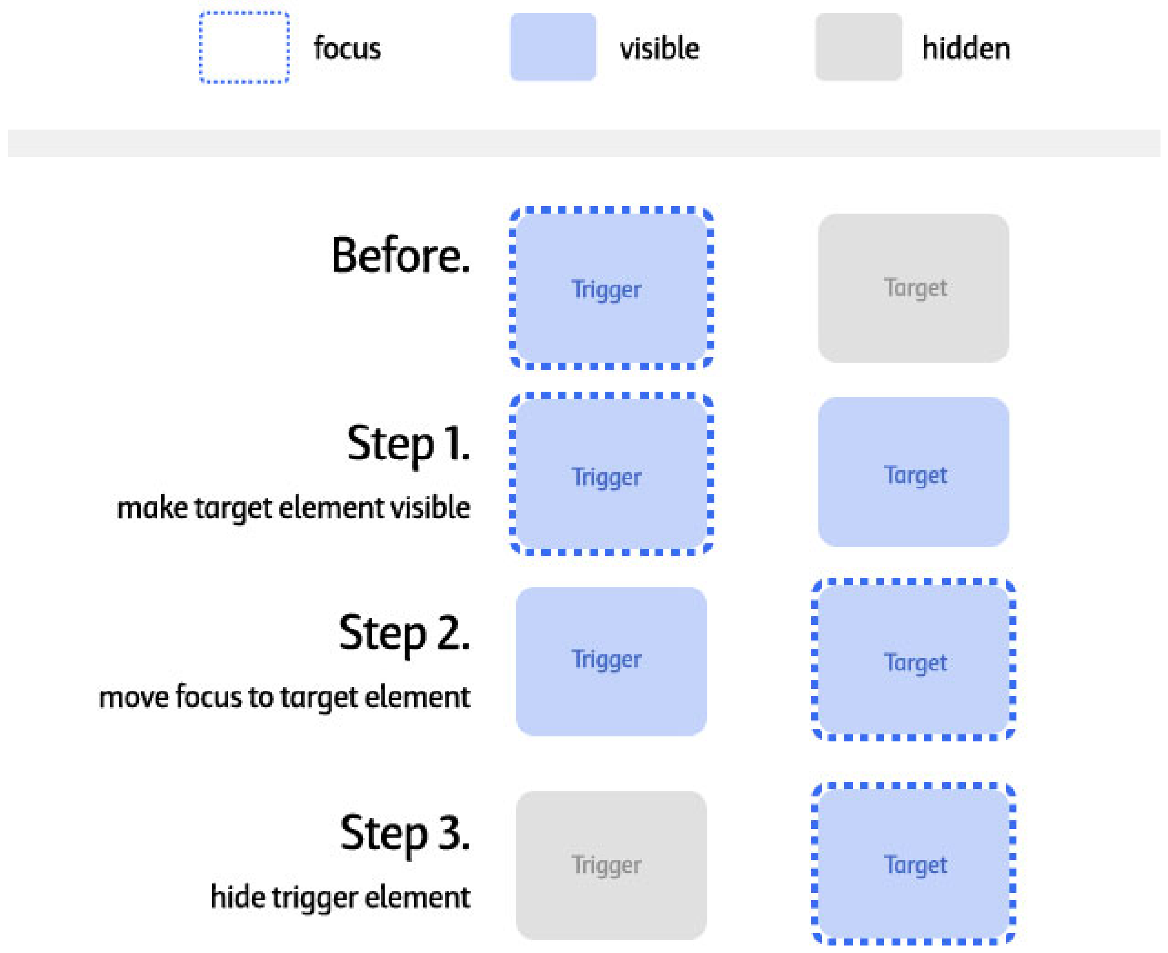
`aria-expanded` should have a value of `true` when the area is visible, and a value of `false` when it's collapsed (or not visible).

This ensures that assistive technology users are made aware of the current state of the expandable areas.

IMPORTANT

The order in which focus is managed is extremely important. Hiding an element that currently has focus is troublesome and focus generally returns to the top of the window. This is not desired.

When an element hides to display another and focus is to be given to the appearing element, the appearing element MUST appear before the hiding element disappears and focus must be moved in between these two actions.



CHECKLIST - How to know when collapsable areas are accessible:

A) If the information is required to be saved/cancelled

- ☐ Ensure the trigger indicates clearly the action to be taken
- ☐ Ensure the trigger has the appropriate value of aria-expanded set
- ☐ Opening the expandable area automatically sets focus on the dominant heading OR first form element within the content.
- ☐ The user should **NOT** be able to use the keyboard to navigate out of and away from the expanded content area.
- ☐ Labelling of "Save" and "Cancel" buttons should be informative and indicate what they are saving or cancelling.
- ☐ Once an action is performed, focus should be returned to the opening trigger and an indication of success should be given.

B) If the information is simply extra content and doesn't require any action.

- ☐ Ensure trigger indicates clearly the action to be taken.
- ☐ Ensure the trigger has the appropriate value of aria-expanded set
- ☐ Opening the expandable area automatically sets focus on the dominant heading within the content.
    - ☐ If no heading is present/required focus should be set to the expanding content container and proper description should be given using the aria-describedby="" attribute
- ☐ The user should be able to use the keyboard to navigate (either upwards or downwards) out of and away from the expanded content area.
- ☐ The trigger element should update it's information to reflect the current state of the expanded area OR the

action that will be performed when triggered again. ie. When open, the trigger should read "Close explain offers to me section". When closed, the trigger should read "Open explain offers to me section".

- ☐ If the expanded area can be closed through a shortcut (such as the 'Esc' key) the original 'opening' trigger should receive focus.

Accessible Modals/Popups

Accessible Modals/Popups

A popup can be closed using a close button or clicking on the "dimmed" area behind the popup content - Or when a button within the content of the popup is clicked and triggers the close.

Herein, the term Modal and Popup are interchangeable as their functionality differs very little when considering accessibility.

Popups require a focus trap. That means they cannot navigate away from the popup without it closing. When a popup is opened focus must be automatically set to either the first heading within the popup content, or the content container if no heading is available or there's indecision on what should be focused.

Upon closing the popup keyboard focus must be returned to the triggering element that opened it. If there was no triggering element, set focus to the <body> element.

The FlyBuys website currently uses MagnificPopup v0.9.9 - An MIT Licenced library reliant on jQuery or Zepto.js - https://github.com/dimsemenov/Magnific-Popup/

The trigger must clearly indicate that the content will open in a popup and give context to the content that will be displayed in the popup. There's an example below which shows how to place visibly hidden text on the trigger link. Please see Hide and Show for Screens and Screen Readers page.

CHECKLIST - How to know when modals and popups are accessible

- [ ] Triggers **MUST** indicate a popup will be opened
- [ ] Triggers should be informative to the information the popup contains
- [ ] When using an <a> tag in absence of a `href=""` attribute, or the trigger is not an <a> tag - you **MUST** use either the the `role="link"` or `role="button"` attribute (which ever is more relevant). Please see Accessible Links page.
- [ ] The first heading **MUST** have an `id=""` attribute (which will be used as the `aria-describedby=""` attribute mentioned below)
- [ ] Popups should be able to be closed using the `ESC` key

AngularJS popup directive process which should be checked by developers...

The guts of focus trapping have been automated - When a popup is opened, focus is automatically given to the heading element, the page behind becomes inaccessible. When closed the reverse happens and focus is returned to the trigger - However it's still important for developers making changes to the popup methods ensure the following:

- [ ] The popup container **MUST** have `role="dialog"` attribute applied.
- [ ] The popup container **MUST** also have `aria-describedby=""` attribute applied and the value should target the first heading within the content.
- [ ] The first heading that is to be focused **MUST** also have a `tabindex="-1"` attribute applied.

For Testers:

- [ ] When the popup finishes rendering - focus should be set to the first heading within the popup content or the popup container.

- ☐ Using the keyboard the user should **NOT** be able to navigate out of and away from the popup container. Only content within the container and a popup close button (when relevant) can be navigated.
- ☐ Closing the popup should return focus to the trigger that opened it.

Example popup trigger and content

Here's a modified snippet taken from the My Offers page on the FlyBuys website which shows a popup trigger and popup content.

.jsp file located at: opencms//elements/site/offers/myOffers.jsp)**

```
<a data-ng-href="#terms-and-conditions" data-popup data-popup-options="popupOpts" role="link">
   <span class="visuallyhidden">Read Terms and conditions for Coles offer. This will open in a popup.</span>
   <span aria-hidden="true">Terms and conditions</span>
</a>

<div id="terms-and-conditions">
   <h3 id="terms-and-conditions-title" tabindex="-1">Terms and Conditions</h3>
   <div class="visuallyhidden" id="terms-and-conditions-dialog-title">Press escape to exit this popup</div>
   <div> Terms and Conditions content here ... </div>
</div>
```

Accessible Links/Buttons

Accessible Links/Buttons

CHECKLIST

How to know when links are accessible

- ☐ Links **MUST** be descriptive and not rely on the content around them.
- ☐ If no `href=""` attribute is placed on the <a> tag, a `role="link"` attribute **MUST** be added.
- ☐ If the <a> tag is used to submit a form or process a form in any way, a `role="button"` attribute **MUST** be added. Do not have `role="link"` and `role="button"` on the same element.
- ☐ If an element other than an <a> tag is used to perform an action it **MUST** include a `tabindex="0"` attribute - to make it focusable - and `role="link"` or `role="button"` depending on it's functionality.
- ☐ If using an <a> tag with role="button" you MUST detect the SPACE key and trigger "click" event accordingly. Please see https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/ARIA_Techniques/Using_the_button_role
- ☐ **DON'T** use role="button" for link where it points to external link. Otherwise screen reader will read "visited" at the end if link has been visited. This may leads to confusion for user as it speaks button + visited. Only links can have state visited or unvisited state.

Example of Non-accessible link:

```
<p>
To visit rewards page and redeem a reward, <a
href="/rewards">click here</a>
</p>
```

Same example that is considered accessible:

```
<p>
<a href="/rewards">Visit the rewards page</a>
to redeem a reward.
</p>
```

Accessible Form Elements

Accessible Form Elements

The most important aspect of any site's interaction is forms. Special attention must be given to the usability and accessibility of forms. Below are a few guidelines in making different form elements accessible.

For additional information on making forms accessible within a collapsable/popup area - please visit the Accessible Collapsable Forms/Information Areas wiki page

CHECKLIST - How to know when forms are accessible

- ☐ If required, the `aria-required="true"` attribute must be added.
- ☐ If the form was submitted and no value was supplied or the supplied value did not meet validation requirements `aria-invalid="true"` must be added.
- ☐ If a label is available for the field, it must have a `for=""` attribute specified and the value of the attribute must be the ID of the input element.
- ☐ Any field that needs a value should NOT be declared as `readonly=""`
- ☐ If no label is available for the field, the `aria-label=""` attribute must be added to the element indicating what the user needs to enter – Even if the field isn't required.
- ☐ If additional information is available for the element but not required as part of the instruction to the element, `aria-describedby=""` attribute **SHOULD** be added. The value of the attribute specifies the ID of the element in which additional information is present.
- ☐ If additional context can be given to the element that IS important or provides hierarchical labelling to a section the `aria-labelledby=""` attribute **SHOULD** be added.
- ☐ Checkboxes and radio buttons give context to the field question. Provide more information than just "Yes" or "No", "Male" or "Female"
- ☐ Any special formatting should be vocalised in a very easy to understand mannor. ie. "Please enter your date of birth in the format of, two digit day, slash, two digit month, slash, four digit year."
- ☐ Ensure each form element can be accessed by keyboard only, preferably using only the TAB key.
- ☐ Submit buttons should clearly indicate they are submitting a form. Using appropriate labels to vocalise this as opposed to just "Next" or "Submit".

All fields

A number of `aria` attributes should be added (when appropriate) to provide additional instruction to a screen reader on the current state of a form element. These include;

aria-required	Omitting this attribute is the same as declaring the field is NOT required. Adding `aria-required="true"` indicates to the screen reader this field is required and "required" will be vocalised.

aria-invalid	Omitting this attribute is the same as declaring the field IS valid. Adding `aria-invalid="true"` would indicate that no value was supplied or the supplied value did not meet a validation requirement.
aria-label	If a form field doesn't require a `<label>` element to become a labelled control, an `aria-label` attribute can be added to the input element instead. This works well along side `placeholder` attributes in JAWS and can be used to construct very space limiting accessible forms.
aria-describedby	This attribute can be used to provide vocalisation of information that is important but not necessary to the required value. The describing element should also be relevant/comprehensive when not being referenced. For example; `<input type="email" placeholder="Your email address" class="validation-required" aria-required="true" aria-describedby="email-notice" />` `<p id="email-notice">Flybuys will send communications such as statements, special offers, and program news to your email address, subject to the flybuys Privacy Policy.</p>`
aria-labelledby	This attribute can be used to provide a number of relevant labels to an element. Multiple labels can be added to an element using space separated values of label ID's. For example; `<h[x] id="baby-club-join-label">Would you like to join the Baby Club?</h[x]>` `<input type="radio" value="YES" id="babyClubYes" aria-labelledby="baby-club-join-label" />` `<label class="baby-club-yes" for="babyClubYes">Yes</label>`

Notes on `label` elements

The `label` element should only be used in conjunction with form elements (input, select, textarea etc), and when used **MUST** have a `for` attribute that matches the `id` of the form element it is labeling.

In the event that something other an a form control needs to be labelled, use whichever of `aria-labelledby` or `title` is appropriate in the context.

Accessible Lists of Content - ie. Offers and Rewards

Accessible Lists of Content (Offers and Rewards pages)

Repeated sections of the site that contain large amounts of content (ie. Offers and Rewards on the respective pages) - each segment of content **MUST** utilise the ARIA `role="article"` attribute. The `role="article"` attribute provides the same functionality as an `<article>` HTML5 tag, used in situations the `<article>` tag is not warranted or appropriate.

An article, as per the W3 states:

> *It is* independent *in that its contents could stand alone, however, the element is still associated with its ancestors. Nesting articles represent content that is related to the content of the parent article. A good example of this is on the Offers page where grouped offers are observed. A group of offers is itself an article, with each offer within also being an article.*

JAWS and VoiceOver indicate the beginning and end of an article. Using articles also provides the benefit of advanced navigation of a webpage using "R" on JAWS or "CTRL+ALT, Left/Right" on VoiceOver. Read more about quick navigation through regions.

An article **MUST** also have a concise and descriptive label which JAWS and VoiceOver use to describe the area.

For example;

```
<div class="reward-item" data-ng-repeat="...."
role="article" aria-label="Reward {{$i}} of
{{$total}} on page {{$p}}">
```

or

```
<div class="reward-item" data-ng-repeat="...."
role="article" aria-label="Reward from
{{$provider}}">
```

Accessible Pagination

Accessible paged content

An article on Mike West's blog sums up pagination quite well. Here's an excerpt that highlights a potentially inconvenient issue;

> *For pagination, it seems like it would make perfect sense to use an ordered list rather than the unordered list I've chosen here. It's almost certainly semantically correct, as the list of pages is indeed ordered, and that order is indeed meaningful.*
>
> *In this case, however, I think it's the wrong choice. NVDA (which is the only screen reader I have access to at the moment) reads ordered lists as "One. [List item content] Two. [List item content] …" An unordered list, on the other hand, doesn't number the items as they're read. Since I'm explicitly including the page number in the link, an `ol` simply sounds strange and repetitive: "One. Example Page one. Link. Two. Example page two. Link. …" Assuming other readers like Jaws and WindowEyes behave similarly, an unordered list simply* sounds*better.*

The FlyBuys site has pagination set up as unordered lists. However we didn't feel this was acceptable enough. Hiding the pagination from the screen reader and creating a custom, limited set of links was a better solution... Less is more.

We give the user options to back and forth (previous and next pages) and first to last. If the limited pagination was visible it would look like this: << First | < Prev | Next > | Last >>

No page numbers are supplied because that introduced problems when there was a limited set of pages displayed. ie. < Prev | 1 ... 5 6 7 8 ... 14 | Next >

This was a decision based on the fact that where there's pagination, there's also search, sort and filter functionality available. If there's no filtering available, all page links should be made available.

The current implementation of pagination on the FlyBuys website handles the replacement with limited pagination options automatically. All that needs to be ensured in the future is that the pagination links abide the same rules as specified on the Accessible Links page.

Range Results

When a back, next, first or last link is clicked, focus is automatically set to the range results (ie. "Showing 1 - 5 of 15").

This is handled automatically now by the shared angular directives.

For Testers:

When a next, previous, first or last link is clicked, focus should be set to the range results which will be vocalised by JAWS.

When clicking a next, previous, first or last link in pagination that appears at the BOTTOM of the page, focus should be set to the top pagination's range results.

Results Per Page

Some pages allow the user to show 10, 40 or 80 results per page. These links should be accessible via Down Arrow and TAB keys in JAWS.

For Testers:

When arriving on the link that's currently active (ie. 10 results are showing and virtual focus is on the "10" link) JAWS should vocalise *"Currently showing 10 results per page"*

When arriving on a link that's not active, it should read more than just the digits that are visible. ie. the number "40" should be vocalised as "Show 40 results per page"

Hide and Show for Screens and Screen Readers

Visibility & Readability

Elements can be hidden from the screen, and hidden from a screen reader. We CAN'T detect which though. For more info on how that works: http://css-tricks.com/places-its-tempting-to-use-display-none-but-dont/

Below are two code examples that indicate the syntax required to implement either of the scenarios listed above;

```
<div class="visuallyhidden">
    <!-- Anything within this tag will NOT be visible on the screen
         and WILL be accessible through a screen reader -->
</div>
```

```
<div role="presentation" aria-hidden="true">
    <!-- Anything within this tag will NOT be accessible through a
         screen reader -->
</div>
```

Notes on focusable elements

Do not use `role="presentation"` or `aria-hidden="true"` on a focusable element. If an element is focusable, most browsers currently still allow it to receive focus, even if `role="presentation"` or `aria-hidden="true"` has been set. Therefore, using either of these on a focusable element will result in some users focusing on 'nothing'.

Do **not** do this:

```
<button role=presentation>press me</button>
```

Do **not** do this:

```
<a href="http://www.example.com.au" aria-hidden="true">take me to example.com.au</a>
```

Skip Links

Skip links

The idea behind skip links is provide a very direct route to content regions within a page.

Skip links must be placed at the top of the page (ie. before the logo) as the first focusable links. They must be visually hidden, short and meaningful.

Example skip links

```
<a class="visuallyhidden skip-link" role="link"
data-ref-id="main">Skip to main content</a>
<a class="visuallyhidden skip-link" role="link"
data-ref-id="search">Skip to search</a>
```

The data-ref-id attribute is used to find a region with that name. If no region is found, It uses an element with an ID of that name. If neither region or element ID are found, the skip link is hidden and shouldn't be focusable.

Consideration for the important parts of a page should be done on a per-page basis and possible consultation with project managers/stakeholders to agree unanimously on what areas can be skipped to.

For Testers:

- ☐ Skip links must be first set of focusable links and must appear before the logo or navigation
- ☐ Triggering the link, focus should be given to the region indicated.
- ☐ There's no way to return to skip links unless you go back to the top of the page.

Tree Views

Tree & Tree items

One of the more complicated components in accessibility.

Here's some resources in case you get really stuck;

http://oaa-accessibility.org/examples/role/106/

http://www.w3.org/wiki/TreeView

http://accessibleculture.org/articles/2013/02/not-so-simple-aria-tree-views-and-screen-readers/

How it works

Placing focus on the Treeview item should vocalise the currently selected item - If no item is selected it should default to the first available.

- Pressing ENTER key enables treeview mode and allows navigation within using the UDLR Arrow keys.
- Pressing ENTER key selects the currently focused item and should disable treeview mode.
- Pressing ESC key disables treeview mode.

Once in treeview mode, tree's should be usable by a keyboard alone - Using UDLR Arrow keys to navigate in the respective direction.

UP & DOWN arrow keys

The UP arrow moves the focus upward one item

- When at the very top of the treeview - Focus should be limited to the top-most item.
- When focus is nested within an item (ie. Level 2 or greater) and the top-most item of the current level is focused - focus should move to the parent item. *See figure 1.*

The DOWN arrow moves the focus downward one item

- When at the very bottom of the treeview - Focus should be limited to the bottom-most item.
- When focus is nested within an item (ie. Level 2 or greater) and the bottom-most item of the current level is focused, AND, there is another outer level item following - Focus should move to the outer level item following the currently 'opened' item. *See figure 1.*

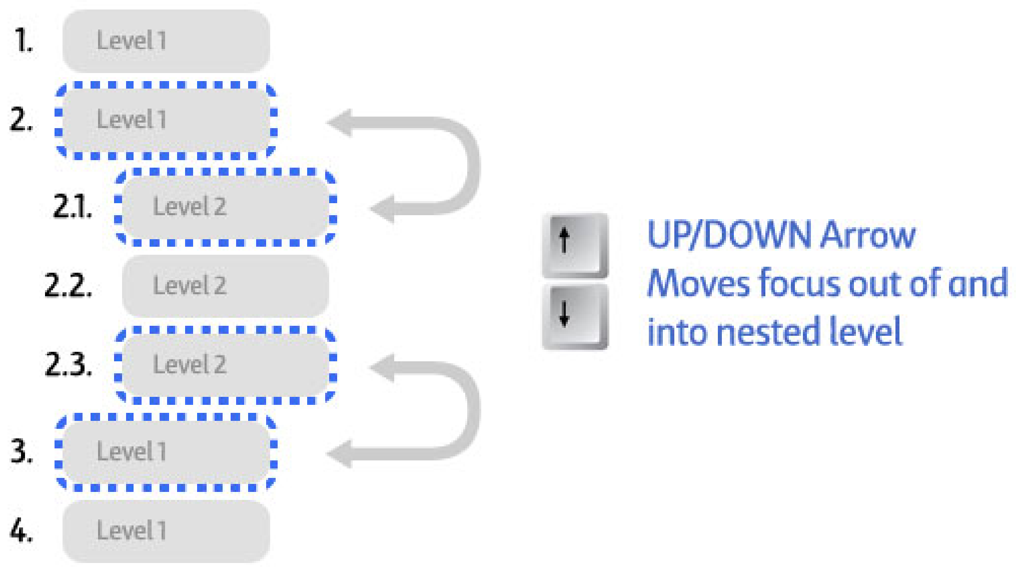


Figure 1.
Moving in and out of tree item levels.

RIGHT arrow key

The RIGHT arrow activates the currently selected item.

Treeview spec. recommended behaviour indicates the RIGHT arrow should place focus on the first available child item - This was possible, however it caused a number of issues with the current implementation of JAWS and VoiceOver. They don't handle tree views well.

- If the item selected has sub categories they should be made visible.
- If the item selected has sub categories the item should be set as "open" using `aria-expanded="true"`
- If the item selected has sub categories the sub item container (`<ul>`) should be visible using `aria-hidden ="false"`

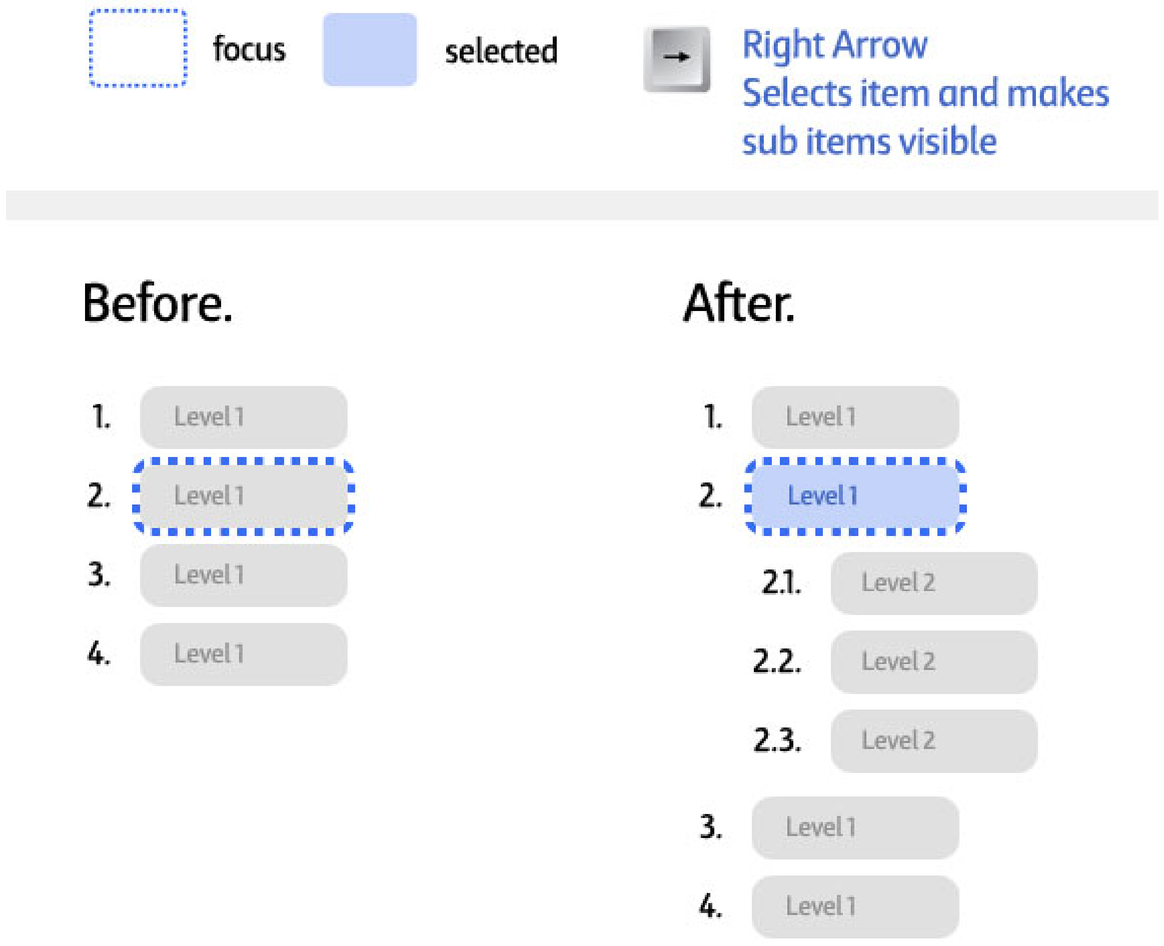
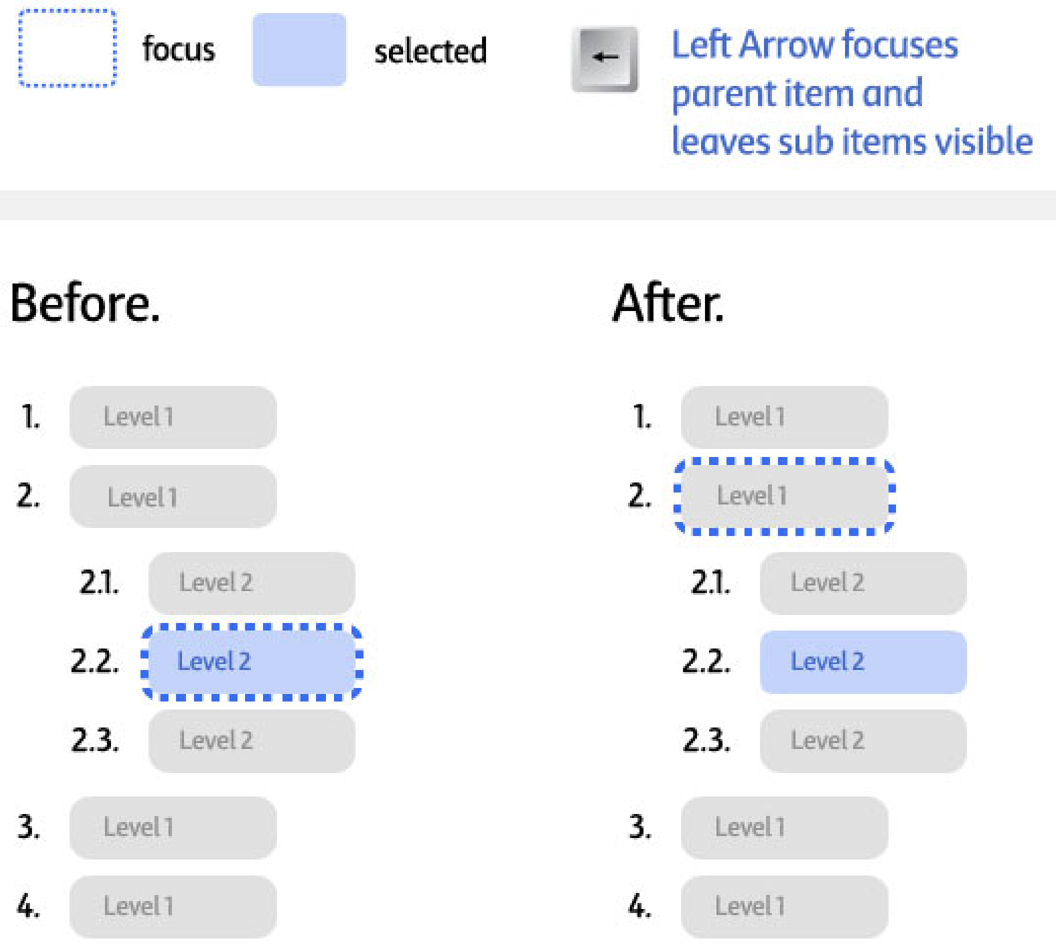


Figure 2.
Activating a tree item that has sub categories

LEFT arrow key

The LEFT arrow focuses the parent item, leaving the current item selected and subitems visible.

Treeview spec. recommended behaviour indicates the LEFT arrow should place focus on the parent item and select it - Closing the sub items - This wasn't possible given the current implementation.

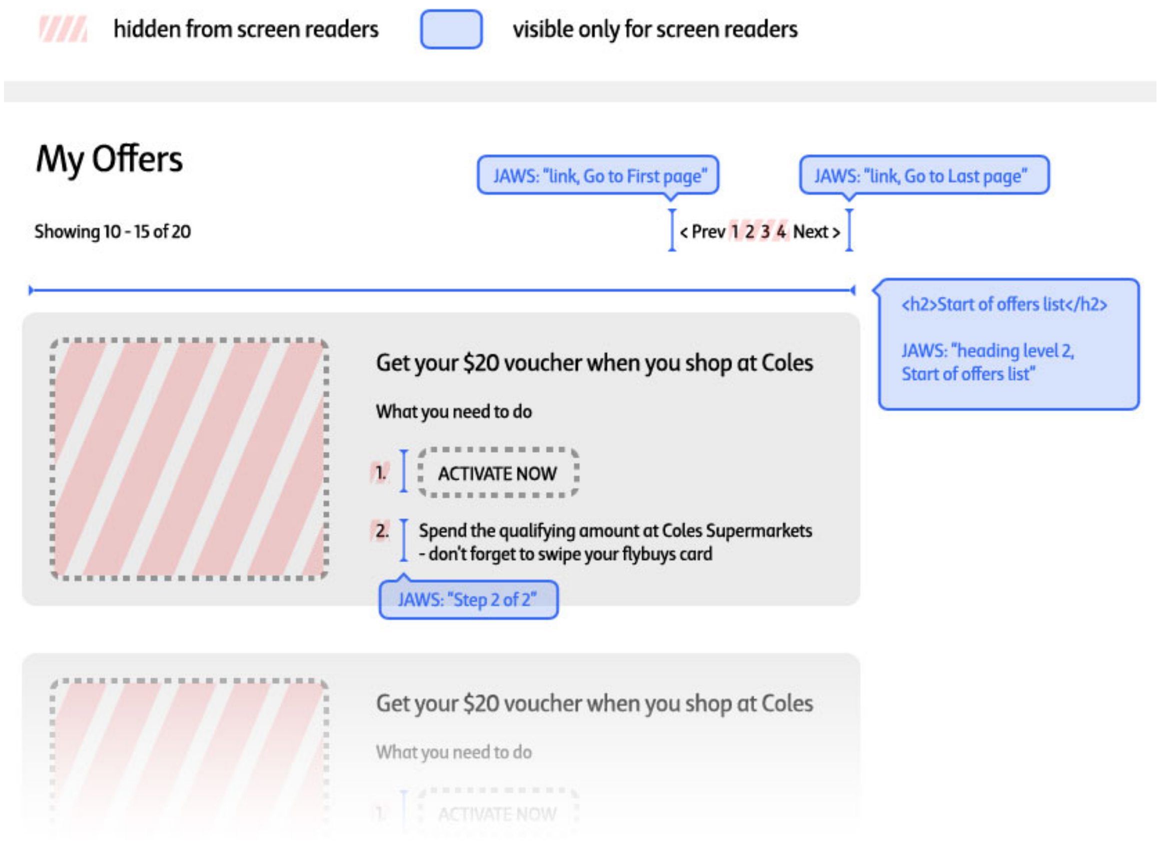

focus
selected
Left Arrow focuses parent item and leaves sub items visible
Before.
1. Level 1
2. Level 1
2.1. Level 2
2.2. Level 2
2.3. Level 2
3. Level 1
4. Level 1
After.
1. Level 1
2. Level 1
2.1. Level 2
2.2. Level 2
2.3. Level 2
3. Level 1
4. Level 1

Website - Accessibility - How Tos

Visually Hidden content for Screen Readers Only

It's sometimes very useful to include additional information or instruction to screen reader users.

Other times it's useful to remove information from screen readers to avoid clutter and repetitive content.


hidden from screen readers
visible only for screen readers
My Offers
Showing 10 - 15 of 20
JAWS: "link, Go to First page"
JAWS: "link, Go to Last page"
< Prev 1 2 3 4 Next >
<h2>Start of offers list</h2>
JAWS: "heading level 2, Start of offers list"
Get your $20 voucher when you shop at Coles
What you need to do
1. ACTIVATE NOW
2. Spend the qualifying amount at Coles Supermarkets - don't forget to swipe your flybuys card
JAWS: "Step 2 of 2"
Get your $20 voucher when you shop at Coles
What you need to do
1. ACTIVATE NOW