ARIA and HTML5

Jared Smith & Jonathan Whiting
webaim.org
Device Independence

Use device independent event handlers

or

combine mouse (e.g, `onmouseover`) and keyboard (e.g, `onkeypress`) dependent event handlers
Ensure Interactive Elements are Links or Form Controls

or...

make non-focusable elements focusable with `tabindex`
Avoid Tabindex

... unless you're sure you know what you're doing.

If the default tab order is not logical, fix your source code order.
`tabindex="1+"` defines an explicit tab order

`tabindex="0"` allows things besides links and form elements to receive keyboard focus.

`tabindex="-1"` allows things besides links and form elements to receive programmatic focus (by scripting, links, etc.)
<div tabindex="0" onclick="submitForm()">Submit Search</div>
is better, but

or

are best!
Click events do not always trigger via keyboard for things other than links or form controls...

... even with `tabindex="0"`
Attach an onkeyup event and then check for Enter (13) and Space (32) key presses:

```javascript
if(event.keyCode==13 || event.keyCode==32)
{
    doStuff();
}
```
function allyClick(event) {
    if (event.type === 'click') {
        return true;
    }
    else if (event.type === 'keypress') {
        var code = event.charCode || event.keyCode;
        if ((code === 32) || (code === 13)) {
            return true;
        }
    }
    else {
        return false;
    }
}

$("#fake-button").on('click keypress', function(event) {
    if (allyClick(event) === true) {
        // do stuff
    }
})

tabindex="-1"

- Allows non-focusable elements to receive programmatic focus (by scripting, links, etc.)
- Necessary for focusing dialog boxes, error messages, etc.
- **WARNING:** This removes the element from the default tab order.
Review

• Ensure all interactive elements are links or form controls, or make them focusable with `tabindex="0"`.

• If using `tabindex`, detect Enter and Space key events.

• Ensure non-focusable elements (such as dialog windows) have `tabindex="-1"` before focusing them programmatically.
Modal Dialog

This is content within the dialog window!

Ok
Dialogs

* tabindex="-1" then set focus to dialog (or focus a control inside the dialog)

  Link, button, or tabindex="0" with aria-haspopup="true"

* Maintains keyboard focus if modal

  Closes with ESC key

* Returns focus when dismissed
“Freak-out” Mode

When the currently focused element disappears or is significantly modified

Avoid it or address it with

```javascript
focus();
```
Non-modal Dialogs
Carousels
Carousels

So...
should I use a carousel?

http://shouldiuseacarousel.com/
Q&A by .net magazine

An anti-carousel carousel featured in a carousel

Accessibility expert warns: stop using carousels

by Craig Grannell
Carousels

http://shouldiuseacarousel.com/
Carousel Issues

- Automated carousels violate WCAG 2.0 Success Criteria 2.2.2 (Level A) - Pause, Stop, Hide
- Distracting and confusing
- Difficult interaction model
  - No relationship between controls and content
- “Freak-out mode” when carousel changes
- Allow poor content decisions
Carousel Accessibility Solutions?

- Avoid auto-playing (optimal) or include a visible pause button (preferably) before the carousel
- Pause carousel on mouse hover and on keyboard focus
- Ensure focused items do not disappear, or manage focus when they do
- Provide context for controls
  - Descriptive text
  - ARIA tab panel?
- Ensure accessible content
Roving tabindex

- Useful for controlling focus within interactive widgets (menus, tab panels, tree widgets, etc.).

- Set `tabindex="0"` on currently active item. This places it in the tab order.

- Set `tabindex="-1"` on all other items. This removes them from the tab order and makes them focusable with scripting.

- Use `focus()` to set focus as user navigates within widget (arrow keys, etc.).

- `tabindex="0"` roves or follows the active item allowing users to return directly to it later.
Dogs

The dog (Canis lupus familiaris, is a domesticated form of the wolf, a member of the Canidae family of the order Carnivora. The term is used for both feral and pet varieties. The domestic dog has been one of the most widely kept working and companion animals in human history. The word "dog" may also mean the male of a canine species, as opposed to the word "bitch" for the female of the species.

More information about Dogs on Wikipedia
In an ARIA tab panel, the entire tab group functions as one tab stop, then arrow keys are used to select the active tab.
Dogs

The dog (Canis lupus familiaris, is a domesticated form of the wolf, a member of the Canidae family of the order Carnivora. The term is used for both feral and pet varieties. The domestic dog has been one of the most widely kept working and companion animals in human history. The word "dog" may also mean the male of a canine species, as opposed to the word "bitch" for the female of the species.

More information about Dogs on Wikipedia
Cats

The cat (Felis catus), also known as the domestic cat or housecat to distinguish it from other felines and felids, is a small domesticated carnivorous mammal that is valued by humans for its companionship and its ability to hunt vermin and household pests. Cats have been associated with humans for at least 9,500 years, and are currently the most popular pet in the world. Due to their close association with humans, cats are now found almost everywhere on Earth. This extreme adaptability and their worrying impacts on native animals has led to them being classed as an invasive species. Most of these problems are caused by the large number of feral cats worldwide, with a population of up to 60 million of these animals in the United States alone.

More information about Cats on Wikipedia
If you tab away from the tab panel and later return, “Cats” remains the active and focused tab because it has `tabindex=0`.

```
tabindex=0
```

**Cats**

The cat (*Felis catus*), also known as the domestic cat or housecat to distinguish it from other felines and felids, is a small domesticated carnivorous mammal that is valued by humans for its companionship and its ability to hunt vermin and household pests. Cats have been associated with humans for at least 9,500 years, and are currently the most popular pet in the world. Due to their close association with humans, cats are now found almost everywhere on Earth. This extreme adaptability and their worrying impacts on native animals has led to them being classed as an invasive species. Most of these problems are caused by the large number of feral cats worldwide, with a population of up to 60 million of these animals in the United States alone.

More information about Cats on Wikipedia
Focus on semantics and let CSS do the heavy lifting

```html
[tabindex=0] {
  background:#fff;
  border-color:#ddd;
  border-bottom:none;
}
```
HTML5

• Adds new features - primarily for application development and enhanced semantics
• Maintains backward compatibility (mostly)
• Defines error handling for browsers
• Drops almost all presentational markup
Principles of HTML5

Why make authors do something the browser can do for them?

Things that actually work matter most.
HTML5 DOCTYPE

<!DOCTYPE html>

- The last DOCTYPE you’ll ever use?
- HTML versions vs. HTML the “living standard”
- HTML5 is very much still in progress
HTML5 Simplifications

```html
<html xmlns="http://www.w3.org/1999/xhtml"
     xml:lang="en" lang="en" >
  vs
  <html lang="en">
    <meta http-equiv="Content-Type"
          content="text/html; charset=utf-8">
  vs
  <meta charset="utf-8">
```
HTML5 Simplifications

<link rel="stylesheet" href="styles.css" type="text/css" />

vs

<link rel="stylesheet" href="styles.css" />

<script type="text/javascript">

vs

<script>

<style type="text/css">

vs

<style>
The Smallest Valid Accessible HTML5 Document

<!doctype html>
<html lang=en>
<meta charset=utf-8>
<title>blah</title>
<p>I'm the content

HTML5 allows browsers to do much of the work.

Just because you can, doesn’t mean you should.
New HTML5 Elements

- `<nav>`
- `<header>`
- `<main>` (one per page)
- `<footer>`
- `<article>`
- `<section>`
- `<aside>`
- …and more
HTML5 Sectioning Elements

• `<section>`, `<article>`, `<aside>` and `<nav>`
• Each should generally begin with a heading that describes that section
• `<article>` is self-contained, syndicatable
• HTML5 outlining algorithm is not supported (and may be removed), so do not rely on it. Use proper heading levels!
HTML5 Accessibility Changes

- alt attribute is optional when `<figcaption>` presents an equivalent alternative for an image within a `<figure>`
- Dropped, then reintroduced, longdesc (subject to change?)
- Dropped table summary
- Use `<abbr>`, not `<acronym>`.
- ARIA markup is valid
HTML5 `<video>` and `<audio>`

- Native video and captioning support
- WebVTT captioning format
- Accessible (???) browser controls or you can build your own.

```html
<video id="video" controls preload="metadata">
  <source src="myvid.mp4" type="video/mp4">
  <source src="myvid.webm" type="video/webm">
  <track label="English" kind="captions" srclang="en" src="mycap-en.vtt" default>
  <track label="Deutsch" kind="captions" srclang="de" src="mycap-de.vtt">
  <track label="Español" kind="captions" srclang="es" src="mycap-es.vtt">
</video>
```
New HTML5 Input Types

- search, number, range, color, url, email, tel, date, month, week, time, datetime, datetime-local
- The browser can (or, more accurately, hopefully will) provide a natively accessible control/interface
- New form attributes - required, pattern, autocomplete, placeholder, autofocus, etc.
When should I start using or transition to HTML5?
Widget Accessibility Questions
What is this thing and what does it do?
Can I interact with it?
How?
Is the interaction standardized?
Does it change dynamically?
Is it compatible with the user agent and technologies I’m using?
ARIA

- Accessible Rich Internet Applications
- W3C Recommendations
ARIA Paves the Cow Paths
ARIA Paves the Cow Paths
ARIA Paves the Cow Paths
ARIA Paves the Cow Paths
ARIA Paves the Cow Paths
You can only make things more accessible by implementing ARIA now

... if you do it correctly
Rule #1 of ARIA: Don’t use ARIA
Accessibility APIs

- Role
- Name
- Description
- Value
- State
- etc.
Accessibility APIs

- Role=button (or pushbutton, etc.)
- Name=Search
- Description=
- Value=
- State=default, focusable
Accessibility APIs

<input type="checkbox" value="subscribe" id="sub">
<label for="sub">Subscribe?</label>

☑ Subscribe?

- Role=checkbox (or checkbutton, etc.)
- Name=Subscribe?
- Description=
- Value=subscribe
- State=checkable, checked
ARIA and Code Validation

If your (X)HTML is valid and your ARIA is valid...
valid + valid = ???
ARIA Roles

Enhance or change default roles to expand the vocabulary of HTML to include standard API values which screen readers already understand.

<slider>???
ARIA Roles

- Avoid duplicating default roles
  (<button role="button">)

- ARIA roles override HTML native roles, but do NOT change functionality.

- Be very careful! You can destroy accessibility by adding one attribute.

<input type="checkbox" role="radio">
ARIA Landmark Roles

- banner, complementary, contentinfo, main, navigation, and search
- Allows easy access to major page components
- The end of "skip" links? Oh yeah... browser keyboard accessibility still sucks.
- Called “regions” in JAWS (press R to navigate them)
What is WAI-ARIA?

In practice, WAI-ARIA gives us more attributes to assign to elements. There are two kinds of attributes, the role attribute and the aria-.* attributes (".*" meaning that what follows "aria." is variable).

Those new attributes seek to increase the semantics of our documents, facilitate the development of Rich Internet Applications and improve Accessibility.

The aria-.* attributes and the values they can have gives us information about the state of an element, and are more geared toward Rich Application Development.

The role attribute and the values it can have gives us information about the purpose of an element in question (is it navigation? Main data? Or tangential content?). Assistive technologies can use this information to jump directly to the main content, immediately detect the main navigation of the document, etc.

Done with a lotta love by DrummerHead, you can find me on Twitter and also drop me a line if you see any errors or just wanna say hi :)

http://mcdlr.com/wai-aria-cheatsheet/
Landmark Roles

You can add `aria-label` to differentiate multiple landmarks of the same type.
HTML5 and Landmark Roles Mapping

<main> - role="main"
<article> - role="article"
<footer> - role="contentinfo"
<header> - role="banner"
<nav> - role="navigation"
<aside> - role="complementary"

Can combine both for older browsers…

<footer role="contentinfo">
Modifying and Enhancing Roles

Bad:
<img src="submit.jpg" onclick=...>

OK:
<a onclick="..."><img src="submit.jpg"...

Better:
<a role="button" onclick="..."><img src="submit.jpg"...

Best:
<button onclick="...">
ARIA States and Properties

- Always start with “aria-”
- State = current condition. Usually change dynamically
- Properties are usually static

```html
<input aria-checked="true"/>
<a aria-expanded="false">View details</a>
<a aria-haspopup="true">Open dialog</a>
<input aria-labelledby="foo">
<div role=slider aria-valuemin=0 aria-valuemax=100 aria-valuenow=10>
```
ARIA States and Properties

- ARIA states and properties should not override HTML properties if there’s a conflict.
- Be very careful!

<input type="checkbox" aria-checked="false">
Labelled Links

<a href="application.pdf" aria-label="PDF file">Download the employment application</a>

- Role=link
- Name=PDF file
- Description=
- Value=
- State=linked

Screen reader reads “link PDF file”! Link text is lost.
href-less Links

<a id="applink">Download the employment application</a>

- Role=text
- Name=
- Description=
- Value=
- State=

Treated identical to a <span>. No keyboard interactivity.
role="application"

... disables standard A.T. keystrokes to allow the page to function like a desktop application

Be very careful with

<body role="application">

Some ARIA elements (tree view, slider, table, tabs, dialog, toolbar, menus, etc.) have an assumed application role.

role="document" enables reading mode.
role="presentation"

<ul role="presentation">
  <li>menu item</li>
  ...
</ul>

Hides native roles of elements (and all required descendants) from assistive technology. Useful on layout tables, lists, etc.

Is ignored if the element is navigable (e.g., links and controls).
You can't unhide a child element.

Ignored on navigable elements.

Use ARIA attributes to control visual appearance:

```html
[aria-hidden=true] {display:none;}
```
Dynamic Content

- When possible, give users control over content updates
- Avoid updates on focused element or handle “freak-out” with focus().
- Determine the “intrusiveness” of content updates with ARIA live regions or roles.
- Consider the amount and importance of content updates. Can the screen reader keep up?
ARIA Live Regions

- `aria-live=assertive` - read now
- `aria-live=polite` - read at a pause
- `aria-live=off` - read when the user encounters it.
- `aria-busy`
- `aria-atomic` - read the entire region or only what has changed
- `aria-relevant` - additions, removals, text, or all
- `aria-controls`
- Special live regions: `alert` (important), `status` (not important), `timer` (always changing), `marquee` (same as `aria-live="polite"`), and `log` (updates added to the bottom)
Alert Role

<div role="alert" aria-live="assertive">Read me right now</div>

Element must be present in the DOM at page load, then updated.

Also role="alertdialog"
What is the air-speed velocity..." <p id="question">What is the air-speed velocity..." <a aria-haspopup="true">Open dialog</a> <a aria-expanded="false">View details</a>
Odds & Ends

<ul role="menu">

<div role="tablist | tab | tabpanel">

<table role="grid">

etc.

</ul>
ARIA Design Patterns

ARIA slider design pattern:

| Description: | A slider is user input where the user selects a value from within a given range. Sliders typically have a button such that when moved will change the current value within the current range of the slider. The button must be keyboard accessible. It is typically possible to add or subtract to the current value by using directional keys such as arrow keys. |
| Keyboard Interaction: | • Right Arrow and Up Arrow increase the value of the slider.  
• Left Arrow and Down Arrow decrease the value of the slider.  
• Home and End move to the minimum and maximum values of the slider.  
• Tab into and out of the slider.  
• Page Up and Page Down optionally increment or decrement the slider by a given amount.  
Focus is placed on the slider. (The visual object that the mouse user would move, also known as the thumb.)  

Localization for right to left languages may wish to reverse the left and right arrows. |
| WAI-ARIA Roles, States, and Properties: | • The slider control has the role `slider`.  
• Sliders support the `aria-valuemin`, `aria-valuemax`, and `aria-valuenow` properties representing the minimum possible value of the slider, the maximum possible value, and the current value. All of these are decimal numbers. The minimum and maximum are typically fixed and do not change.  
• Sometimes the value is not user readable, such as a number for the day of the week, e.g., "1". In those cases, use the `aria-valuetext` property to provide a human readable string for the slider’s value, e.g. "Monday".  
• It is recommended that authors provide a visible label for the slider, referencing it using `aria-labelledby`. |

Example:  
• Open Ajax Alliance Sliders  
• MINDTROVE Rating Widget  
• Paciello Group slider

Please note that not all examples work in all browser and version combinations. For example, note the compatibility statement.
Slider Example

```html
<div role=slider
    aria-labelledby=maxbid
    aria-valuemin=10 aria-valuemax=99
    aria-valuenow=72
    aria-valuetext=“72 dollars”>

Maximum bid: $72

$10 $99

</div>
```

This provides semantics, but you must still ensure proper keyboard interactions and attribute values.
Tabpanel Example

This provides semantics, but you must still ensure proper keyboard interactions.
Single Page Apps

- Document structure
- Use structural elements (<main>) for content updates
- Update page titles to reflect content/state
- Keyboard navigation
  - Ensure only visible elements are navigable
  - Set focus() when necessary
    - $('main').attr("tabIndex", -1).focus();
- Use live regions for messaging, if necessary
Questions?

Jared Smith & Jonathan Whiting

webaim.org