Introduction to ARIA and HTML5

Jared Smith & Jonathan Whiting
webaim.org
This session is not…

- An introduction to accessibility
- An introduction to HTML
- An advanced ARIA/HTML5 workshop
HTML5

• Adds new features - primarily for application development and enhanced semantics
• Maintains backward compatibility (mostly)
• Defines error handling for browsers
• Drops all most 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 xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">
  vs
  <html lang="en">
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    vs
    <meta charset="utf-8">
</html>
HTML5 Simplifications

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

vs

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

<script type="text/javascript">

vs

<script>

vs

<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</p>

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 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 Recommendation
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 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 and Code Validation

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

ARIA Support > HTML5 Support
so use both... for now

<main role="main">
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>
</div>
```
ARIA States and Properties

- ARIA states and properties override HTML properties
- Be very careful! You can destroy accessibility by adding one attribute.

```html
<input type="checkbox" aria-checked="false">
```
ARIA Labels and Descriptions

- `aria-labelledby` vs `aria-label` vs `aria-describedby`
- Define *name* and *description* in APIs
- Always use HTML if you can (`<label>`, `alt`, etc.)
HTML Form Labels

First Name: 

<label for="firstname">First Name:</label>
<input type="text" id="firstname"...

Text boxes, text areas, select menus, checkboxes, and radio buttons.
Limitations with `<label>`

1:1 relationship between `<label>` and a form control.

It cannot label more than one form control.

A form control cannot have more than one `<label>`.
One label for multiple controls

id="namelabel"

<input type="text" name="name" aria-labelledby="namelabel"/>
Multiple labels for one control

<input type="text" name="office1" aria-labelledby="cyndi officenum">

<table>
<thead>
<tr>
<th>Name</th>
<th>Office Number</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cyndi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jared</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jonathan</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Password: 

Password must be 8-15 characters and include letters and numbers

```html
<label for="pass">Password:</label>
<input type="password" id="pass" aria-describedby="passworddetails">
<p id="passworddetails">Password must be 8-15 characters and include letters and numbers</p>
```
A screen reader indicates the field is required (that's all)
A screen reader now indicates that the field is invalid or broken (and that's all).

Use ARIA attributes to control styling:
```html
[label for="password"]
Password</label>: 
<input name="password" id="password" aria-invalid="true">
```

```css
[aria-invalid=true] {border : 2px solid red;}
```
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.
Ensure all interactive elements are keyboard focusable and actionable

Enter key on links, Enter and Space on buttons

Ensure a visual indication of interaction and state is available ( :focus in CSS)
Device Independence

Use device independent event handlers

or

combine mouse (e.g, onmouseover) and keyboard (e.g, onkeypress) dependent event handlers
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.
\texttt{tabindex="1+"} defines an explicit tab order

\texttt{tabindex="0"} allows things besides links and form elements to receive keyboard focus. Places element in logical navigation flow.

\texttt{tabindex="-1"} allows things besides links and form elements to receive programmatic focus (by scripting, links, etc.). Removes element from navigation flow.
<div tabindex="0" onclick="submitForm()">Submit Search</div>
is better, but

is even better, yet

or

are best of all!
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.
Keyboard Accessibility is Different When a Screen Reader is Running
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).
Indicates element (and all descendants) are hidden from all users.

You can't unhide a child element.

Can’t hide navigable elements (e.g., links and controls).

Use ARIA attributes to control visual appearance:

```html
[aria-hidden=true] {display:none;}
```
Alert Role

<div role="alert">Read me right now</div>

Also role="alertdialog"
<input type="submit" disabled="disabled">

vs.

<input type="submit" aria-disabled="true">

Disabled HTML buttons are not keyboard focusable and have very poor contrast.
ARIA Design Patterns

ARIA slider design pattern:

<table>
<thead>
<tr>
<th>Description:</th>
<th>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.</th>
</tr>
</thead>
</table>
| 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

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.
ARIA Warnings

- Navigation menus are not ARIA application menus.
- Data tables are not ARIA interactive grids.
- `role=“dialog”` triggers forms/application mode. Use `role=“document”` on non-focusable content.
- Be careful with focusable elements inside `aria-hidden=“true”` or `role=“presentation”` (unless hidden with CSS display: none).
Dynamic Content

- Give users control over content updates.
- Avoid updates on focused element or handle “freak-out” with focus().
- Determine the “intrusiveness” of content updates
  - `aria-live=assertive` - read now
  - `aria-live=polite` - read at a pause
  - `aria-live=off` - read when the user encounters it.
- Consider the amount and importance of content updates. Can the screen reader keep up?
Questions?

Jared Smith & Jonathan Whiting
webaim.org