CS7026: Authoring for Digital Media

Structural Elements and the HTML Shiv
HTML5 Shiv

- Every browser has a list of HTML elements that it supports.

- E.g. Mozilla Firefox’s list of elements is stored in nsElementTable.cpp.

- Elements not on this list are treated as “unknown elements”.

- There are 2 fundamental questions regarding these “unknown elements”
  - How should the element be styled?
  - What should the element’s DOM look like?
The Document Object Model (briefly)

- The Document Object Model (DOM) is an application programming interface (API) for valid HTML and well-formed XML documents.

- It defines the logical structure of documents and connects web pages to scripts or programming languages.

- In the DOM, documents have a logical tree-like structure.
Document Object Model (DOM)

- Root element: `<html>`
  - `<head>`
    - `<title>`: "My title"
  - `<body>`
    - `<h1>`: "A heading"
    - `<a>`
      - `href`
      - "Link text"
Each browsers list of allowed elements includes information about what kinds of other elements each element can contain.

E.g. If you include markup like `<p>`<p>`, the second paragraph element implicitly closes the first one, so the elements end up as siblings, not parent and child.

But if you write `<p>`<img>`, the image does not close the paragraph, because Firefox knows that `<p>` is a block element that can contain the inline element `<img>`. So the `<img>` ends up as a child of the `<p>` in the DOM.
Remember 2 important questions:

- How should the element be styled?
- What should the element’s DOM look like?

However, when it comes to “unknown elements” different browsers answer these questions in different ways.

Of the major browsers, Microsoft Internet Explorer’s answer to both questions was the most problematic.

The first question should be relatively simple to answer: don’t give any special styling to unknown elements. Just let them inherit whatever CSS properties are in effect wherever they appear on the page, and let the page author specify all styling with CSS.
Unfortunately, Internet Explorer (prior to Version 9) does not allow styling on unknown elements. For example, if you had this markup:

```html
article {display: block; border: 1px solid red }
...
<article>
  <h1>Welcome to TCD</h1>
  <p>This is your <span>first day</span>.
</article>
```

Internet Explorer (up to and including IE 8) will not put a red border around the article.
The second problem is the DOM that browsers create when they encounter unknown elements.

If IE doesn’t explicitly recognize the element name, it will insert the element into the DOM as an empty node with no children.

All the elements that you would expect to be direct children of the unknown element will actually be inserted as siblings instead.
So this is the DOM that HTML5 dictates:

```html
article
 | +--h1 (child of article)
 | | +--text node "Welcome to TCD"
 | +--p (child of article, sibling of h1)
 | | +--text node "This is your "
 | | +--span
 | | | +--text node "first day"
 | | | +--text node "."
```
But this is the DOM that Internet Explorer actually creates:

```
article (no children)
h1 (sibling of article)

|  |  |  +---text node "Welcome to TCD"
p (sibling of h1 and article)

|  |  |  |  +---text node "This is your "
|  |  |  |  |  |  |  +---span
|  |  |  |  |  |  |  |  |  |  +---text node "first day"
|  |  |  |  |  |  |  |  |  |  |  |  |  +---text node "."
```
HTML5 Shiv

- Shiv (or Shim): an application compatibility workaround

- However, if you create a dummy `<article>` element with JavaScript before you use it in your page, Internet Explorer will magically recognize the `<article>` element and let you style it with CSS.
<html>
<head>
<script>
document.createElement("article");
</script>
</head>
<body>
<article>
<h1>Welcome to TCD</h1>
<p>This is your <span>first day</span>.</p>
</article>
</body>
</html>
HTML5 Shiv

- This works in all versions of Internet Explorer, all the way back to IE 6.

- We can extend this technique to create dummy copies of all the new HTML5 elements and then just start using them without having to worry too much about non-HTML5-capable browsers.
HTML5 Shiv

- Remy Sharp has done just that, with his aptly named “HTML5 enabling script” or HTML5 shiv

- The script has gone through several revisions, but this is the basic idea:

```html
<!-[if lt IE 9]>
<script>
var e = ("abbr,article,aside,audio,canvas,datalist,details," +
"figure,footer,header,hgroup,mark,menu,meter,nav,output," +
"progress,section,time,video").split('',');
for (var i = 0; i < e.length; i++) {
document.createElement(e[i]);
}
</script>
<!-[endif]-->
```
HTML5 Shiv

- The `<!--[if lt IE 9]>` and `<![endif]-->` bits are conditional comments.

- Internet Explorer interprets them like an if statement: “if the current browser is a version of Internet Explorer less than Version 9, then execute this block.”

- Every other browser will treat the entire block as an HTML comment. The net result is that Internet Explorer (up to and including Version 8) will execute this script, but other browsers will ignore it altogether.

- This makes your page load faster in browsers that don’t need this hack.
HTML5 Shiv

The JavaScript code itself is relatively straightforward:

1. The variable `e` ends up as an array of strings like "abbr", "article", "aside", and so on.
2. Then we loop through this array and create each of the named elements by calling `document.createElement()`.

This is enough to get Internet Explorer to treat these elements the way we want them to be treated when we actually use them later in the page.
HTML5 Shiv

- That “later” bit is important. This script needs to be at the top of your page - preferably in your `<head>` element - not at the bottom.

- That way, Internet Explorer will execute the script *before it parses your tags and attributes*.

- *If you put this script at the bottom of your page, it will be too late. Internet Explorer will have already misinterpreted your markup and constructed the wrong DOM, and it won’t go back and adjust it just because of this script.*
HTML5 Shiv

- Remy Sharp has “minified” this script and hosted it on Google Project Hosting.

- The script itself is open source and MIT-licensed, so you can use it in any project.

- If you like, you can even “hotlink” the script by pointing directly to a hosted version, like this:

  `<head>
  <meta charset="utf-8" />
  <title>My HTML5 Page</title>
  <![if lt IE 9]>
  <![endif]-->
  </head>`
HTML5

- We are going to:
  - Look at the structural elements a bit more closely by constructing a blog page
HTML5 – Marking up a Blog

- We have seen how a document’s outline can be structured, now we will look deeper to show how you can further structure your main content.

- Let’s mark up a typical blog with HTML5. It’s a good archetype of websites with headers, footers, sidebars, multiple navigation areas, and a form, whether it’s a blog, a news site, or a brochure site (with products instead of news pieces).
Structuring Main Content Areas

- Take a look at the main content area of a blog. There may be multiple articles, each containing “metadata” and the actual textual content of that article.
Structuring Main Content Areas

- Here’s some typical markup (simplified from a default WordPress theme)

```html
<div class="post">
<h2>Memoirs of a Dublin delinquent</h2>
<small>December 1st, 2016</small>
<div class="entry">
<p>Molly Malone’s poignant autobiography is this winter’s must-read.</p>
</div>
</div>

<p class="postmetadata">Posted in <a href="/?cat=3">Books category</a> | <a href="/?p=34#respond">No Comments</a></p>
```
Structuring Main Content Areas

- There is nothing major wrong with this markup. It will work fine in “HTML5” browsers.

- But apart from the heading for the blog post, there is no real structure - just meaningless `<div>`s and paragraphs.
Structuring Main Content Areas

- XHMTL1.0 gave us generic structures to mark up content. `<div>`, for example, is just a generic “box” that tells the browser “here’s some stuff, it all belongs together,” but it doesn’t mean anything; there’s no semantic value beyond “these belong together.”

- Where possible, we’ll replace generic boxes with new HTML5 elements, while still using `<div>` where there isn’t an appropriate element.
Structuring Main Content Areas

- As we have already seen you can replace the outer `<div class="post">` with `<article>`.

- But you can go further. The HTML5 `<header>` and `<footer>` elements can be used multiple times on a page, each time referring to the section it’s in.

- The heading and the time of posting is “introductory matter” and thus the job for `<header>`.

- Similarly, the metadata about the post that is currently in a paragraph with `class=postmetadata` is better marked up in HTML5 as a `<footer>`, which “typically contains information about its section, such as who wrote it, links to related documents, copyright data, and the like.”
Diagrammatically, the revised structure is:

```html
<article>
  <header>
    <h2>Memoirs of a Dublin delinquent</h2>
    <time datetime=2016-12-01>December 1st, 2016</time>
  </header>
  <p>Molly Malone’s poignant autobiography is this winter’s must-read.</p>
  <footer>
    Posted in <a href="/?cat=3">Books category</a>. <a href="/?p=34#respond">No Comments</a>
  </footer>
</article>
```
The `<time>` element

- `<time>` is a new element for unambiguously encoding dates and times for machines, while still displaying them in a human-readable way.

- The uses of this in web pages aren’t hard to imagine:
  - a browser could offer to add future events to a user’s calendar;
  - content aggregators could produce visual timelines of events;
  - a Thai-localised browser could offer to transform dates into Thai Buddhist era dates, which are numerically 543 years greater than their corresponding Western-style years.

- The spec says “The time element represents either a time on a 24-hour clock, or a precise date in the proleptic Gregorian calendar, optionally with a time and a time-zone offset.”
The `<time>` element

- The machine-readable part of the `<time>` element is usually encapsulated in the element’s `datetime` attribute.

- The content inside the element is what gets presented to end users.
  
  ```html
  <time datetime="2016-11-14"> 14 November 2016</time>
  ```
  
  ```html
  <time datetime="2015-11-14">14<sup>rd</sup> November last year</time>
  ```
  
  ```html
  <time datetime="2016-11-14">Molly’s 21st birthday</time>
  ```
  
  ```html
  <time datetime="2016-11-14T02:00Z">8PM on Molly’s birthday</time>
  ```
  
  ```html
  <time datetime="20:00">8 PM</time>
  ```
The `<time>` element

- If you’re happy to have the machine-readable format visible to the end user as well, you don’t need to use a separate `datetime` attribute.

- User agents should then simply pick the content of the element and interpret it: `<time>20:00</time>`
Machine-Readable Dates and Times

- To be machine-readable, dates must be in the format YYYY-MM-DD and may also include a time, prefixed with “T” to separate the date and time, in the format HH:MM.

- Optionally you can append seconds (separated from the minutes with a colon).

- Fractions of a second are allowed after a full stop mark.

- As you’ve seen above, you can give a time on the 24-hour clock with no date information.
The `<time>` element

- If you’re giving time and date together, you need to show the time zone: that’s either “Z” for Coordinated Universal Time (UTC), or an offset from UTC in hours and minutes, prefixed with a plus or minus.

- Putting that all together: “1979-10-14T12:00:00.001-04:00” represents one millisecond after noon on October 14th, 1979, in Eastern Standard Time during daylight saving time (UTC - 4 hours).
The **pubdate** attribute

- **pubdate** is a Boolean attribute to indicate that this particular `<time>` is the publication date of an `<article>` or the whole `<body>` content.

- Why not just assume that any `<time>` element in an `<article>`’s `<header>` is its publication date?
The `pubdate` attribute

- Consider this example:

```
<article>
  <header>
    <h1>Come to my party on <time datetime=2016-12-01>1 December</time></h1>
    <p>Published on <time datetime=2016-06-20 pubdate>20 June 2016</time></p>
  </header>
  <p>I’m throwing a party for my birthday</p>
</article>
```

- There are two dates within the `<header>`: the date of the actual party and the publication date of the article. The `pubdate` attribute is required to remove any ambiguity.
Styling Multiple Headers and Footers

- This main change with our article makeover is that each article can have its own `<header>` and `<footer>`.

- This means that, in addition to the “main” header and footer on a page, each article can have its own headers and footers.

- They can be separately styled with CSS: `body header` and `body footer` target the “main” headers and footers (assuming that they’re direct descendants of `<body>`), whereas `article header` and `article footer` target the inner structures.
Styling Multiple Headers and Footers

- Define generic header and footer styles, and then redefine/override them for article header and article footer:

  ```css
  header {display:block; color:red; text-align:right;} /*page header */
  article header {color:blue; text-align:center;} /*article header */
  ```

- Note that so far, you’ve introduced no ids or classes as hooks for CSS.
Using multiple `<footer>`s on the same element

- The spec says “Footers don’t necessarily have to appear at the end of a section, though they usually do,” and it allows an element to have two or more footers.

- A simplified version of the example in the spec is

```html
<body>
<footer><a href="/">Back to index..."</a></footer>
<h1>Lorem ipsum</h1>
<p>Lorem ipsum</p>
<footer><a href="/">Back to index..."</a></footer>
</body>
```
Using multiple `<footer>`s on the same element

- The reason for this is that the elements are supposed to be non-presentational.

- If “back to index” is the footer below the article, and you choose to have “back to index” above the article, too, you should use the same element for the same content, regardless of where it appears.
Adding Blogposts and Comments

- If we add this to our page, we have a page with a header, footer, navigation, content area containing several articles (blog posts), each with its own header and footer.

- But what about comments?

- The specification recommends the use of nested `<article>`s: “When article elements are nested, the inner article elements represent articles that are in principle related to the contents of the outer article. For instance, a blog entry on a site that accepts user-submitted comments could represent the comments as article elements nested within the article element for the blog entry.”
Adding Blogposts and Comments

- So let’s do that. Blog comments are typically shown in chronological order and have information such as author’s name and URL—in short, header information:
Come to my party on 1 December

Published on 20 June 2016

I’m throwing a party for my birthday

Published in the Party category by Nina

Comment from Mary B at 8.45 on 1 May 2016

I’ll be there. I very much enjoy parties

Comment from John B at 10.45 on 2 May 2016

Sorry. Am washing my hair.