

HTML5 Training for Web Developers

HTML5 - How We Got Here

Lesson 1, Activity 2: The Problems HTML 4 Addresses

HTML 4 was introduced in 1997 (yeah, a long time ago). The biggest gain in HTML 4 was the separation of formatting into CSS. Formatting tags like ``, `<s>` and `<u>` and attributes like `bgcolor`, `height` and `width` were deprecated in favor of corresponding CSS properties. This allowed for big gains in accessibility and much easier, more semantic mark up (e.g, tableless layout).

There were also additional accessibility improvements, including requiring the `alt` attribute on `` tags and allowing for the `title` attribute on almost all tags.

It took browsers awhile to conform, but eventually they managed and now web designers can safely take advantage of most HTML 4 / CSS features.

Lesson 1, Activity 3: The Problems XHTML Addresses

HTML is an SGML-based language and SGML-based languages are not easy to extend or consume generically. The major issue is that such languages are too flexible. The two major problems are:

1. Not all tags are closed.
2. Boolean attributes take no values.

As such, the tool consuming these languages (e.g, a browser or an editor), must be aware of every aspect of the language.

On the other hand, XML-based languages are stricter:

1. All tags must be closed.
2. All attributes must have values.

XML also enforces case sensitivity and use of quotation marks to enclose attribute values, but it is the two issues above that make XML so easily extensible.

A nice side effect was that it provided freedom **from** choice. And with this freedom comes the knowledge that you can look at any valid XHTML document and know what to expect. No need for a style guide to tell authors when to and not to put quotation marks around their attributes or to write in lowercase or uppercase letters.

As it turns out, XHTML wasn't adopted as whole-heartedly as had been anticipated and, as such, didn't add as much value as we all had hoped it would. The W3C stopped work on XHTML 2 in 2009.

Lesson 1, Activity 4: The New More Flexible Approach of HTML5 - Paving the Cowpaths

HTML5 takes a much more flexible approach than XHTML did.

HTML5 is designed with the idea that authors have been writing HTML in many different ways over the years and there are zillions of web pages out there that don't adhere to the strict XHTML standards. Rather than render those page useless, let's just relax the standard a bit (well, a whole lot). They call this "paving the cowpaths."

As an example of this flexibility, all of the following are permitted in HTML5:

1. `<link type="text/css" href="style.css"/>`
2. `<LINK TYPE="text/css" HREF="style.css"/>`
3. `<link type=text/css href=style.css>`
4. `<LINK TYPE=text/css HREF=style.css>`
5. `<LiNk TyPe=text/css hReF="style.css">`

As the above shows:

1. HTML5 is case-*ins*sensitive.
2. HTML5 allows for unclosed tags, but you can use the XML-style shortcut close tag if you want.
3. HTML5 does not require quotes around attribute values (unless the values have spaces in them).

This new flexibility could lead to a bit of chaos on your development team. Different HTML authors will take different approaches. Our recommendation is that you choose one approach and stick to it. In this course, for example, we use the following guidelines:

1. Write tags and attributes in all lowercase letters (even event handlers like `onclick`).
2. Do not use short-cut close tags for void/empty elements.
3. Put all attribute values in quotes. (Why? Because attribute values that have spaces in them have to be in quotes. And I do not like the idea of having some attributes in quotes and some not.)
4. Minimize attributes when you can.

Again, it doesn't matter so much which guidelines you choose, but it'll make your life easier if you specify some.

Lesson 1, Activity 7: New Features of HTML5

The table below shows the new elements that HTML5 has introduced. We will cover most of these in this course.

New HTML5 Elements

#	Tag	Description
1	<article>	For a standalone article on a page. Articles can be nested within other articles; for example, a blog post would be contained in <article> tags and each comment would be contained within a nested <article> tag.
2	<aside>	For content contained in an aside. Often used for navigation elements or for a list of articles or categories (e.g., in a blog).
3	<audio>	For embedding audio files.
4	<canvas>	For creating drawings natively in the browser.
5	<command>	For command buttons similar to what you might see in the Microsoft Office 2010 ribbon. <command> must be nested in <menu>.
6	<datalist>	For a dropdown list providing built-in functionality similar to a JavaScript autocomplete boxes.
7	<details>	For expandable (usually initially hidden) content to provide more information about an element.
8	<embed>	For backwards compatibility with the non-standard (but well supported) <embed> tag in HTML 4.
9	<figcaption>	For captions on images. (In HTML 4, there was no way to semantically associate a caption with an image.
10	<figure>	For wrapping embedded content (e.g, an image) with a <figcaption>.
11	<footer>	For the footer of a page or a section.
12	<header>	For the header of a page or a section.
13	<hgroup>	For grouping <h1>...<h6> tags on a page. For example, the title and subtitle of a page could be an <h1> and <h2> grouped in an <hgroup> tag.
14	<keygen>	For a generated key in a form

15	<code><mark></code>	For showing marked (or highlighted) text. Unlike <code></code> or <code></code> , <code><mark></code> doesn't give the text any special meaning. The best example is marking a word or phrase that a user has searched on within the search results.
16	<code><meter></code>	For a measurement within a set range.
17	<code><nav></code>	For holding a group of navigation links.
18	<code><output></code>	For holding output (e.g., produced by a script).
19	<code><progress></code>	For a progress indicator (e.g., for a loading).
20	<code><rp></code>	Used within <code><ruby></code> tags to tell browsers that cannot render the East Asia characters properly what extra characters (usually parentheses) to display.
21	<code><rt></code>	Used within <code><ruby></code> tags to show how to pronounce East Asia characters.
22	<code><ruby></code>	For ruby annotations. (See http://www.w3.org/TR/ruby for examples.)
23	<code><section></code>	For creating a <code><section></code> on the page. This helps the browser (user agent) determine the page outline.
24	<code><source></code>	For indicating media sources within <code><video></code> and <code><audio></code> .
25	<code><summary></code>	For the header of a <code><detail></code> element. The <code><summary></code> would show by default.
26	<code><time></code>	For holding a machine-readable date and/or time while displaying a friendly date and/or time.
27	<code><video></code>	For embedding video files.
28	<code><wbr></code>	An opportunity to insert a line break within a word. (e.g, super <code><wbr></code> califragilistic <code><wbr></code> expialidocious)

HTML5 and JavaScript

The HTML5 specifications show an API for each HTML5 element, which gives instructions on how to access an elements methods and properties via JavaScript.

JavaScript Cowpaths

Some JavaScript practices long supported by browsers but not officially in the HTML 4 specification have been specified in HTML5:

1. `innerHTML`
2. `XMLHttpRequest`
3. `JSON`
4. `element.getElementsByClassName()`

Additional Changes

1. Native `audio` and `video` - covered in **HTML5 Audio and Video**.
2. Huge advances with forms - covered in **HTML5 Forms**.
3. New ways to store data in the client - covered in **HTML5 Web Storage**.
4. Canvas for creating drawings natively in the browser - covered in **HTML5 Canvas**.
5. HTML5 introduces the new `contenteditable` attribute, which makes the content of a tag editable in the browser:
 1. Open html5-how-we-got-here/Demos/html5-layout.html in your browser.
 2. Click on the content in the HTML5 Training section and start editing.

Modernizr

Modernizr is a relatively small JavaScript file that checks the user's browser for HTML5 feature support. The name is a bit of a misnomer as it doesn't actually modernize the browser. It doesn't add any missing features, it just gives developers an easy way of figuring out if the browser supports a given feature, so they can write conditional code like this:

```
if (Modernizr.canvas) {
  //use canvas to create awesome drawing application
}
```



```
} else {  
  alert("Go get yourself a browser that supports canvas.");  
}
```

We use Modernizr in this course (see html5-common/modernizr.min.js in your class files. For the latest version, check <http://www.modernizr.com/>.

If your curious what features your browser supports, check out <http://www.html5test.com>.

Lesson 1, Activity 9: The HTML5 Spec(s)

There are two official HTML5 specifications:

1. [WHATWG](#)
2. [W3C](#)

The history is a bit complex - a lot of fighting and bickering. But the end result is that we have two specifications, both with the same editor: Ian Hickson. WHATWG seems to be sort of a playground, which the editor and his contributors use to innovate. We expect that the W3C specification will ultimately be considered authoritative.

The HTML5 specs are incredibly long, but most of it is describing how user agents should deal with HTML5. Don't be afraid to use it as a reference. It can be intimidating at first, but can be very useful once you get used to it.

Lesson 1, Activity 10: **Current State of Browser Support**

Browser support is coming along surprisingly quickly considering Ian Hickson has said that HTML5 is unlikely to reach Candidate Recommendation until 2012 and full Recommendation before 2022. However, the reality is that browsers that don't support HTML5 very well make up a significant share of the market. For example, as of this writing, IE6 and IE7, which have virtually no support for HTML5, still have more than 20% of the market share. See the chart below ([source](#)):



That said, we can start playing with HTML5 now and even developing full-fledged applications. We just need to be aware of browser limitations and attempt to degrade gracefully.