FROM HTML+ TO HTML5

Excerpt from:
Learning Web Design, 5e

by Jennifer Robbins

*[**Author's Note:** This article previously appeared as Appendix D in the 5th Edition of Learning Web Design. I have made it available to provide interesting historical context on the evolution of HTML, but keep in mind that some information may not be up-to-date. When the Appendix was first published in 2018, HTML5 was relatively new and had transformed what could be done with markup alone. Since then, the standard as maintained by the WHATWG (whatwg.org) is considered a "living" document and will no longer have numbered releases.]*

IN THIS ARTICLE

The history of HTML

The rise and fall of XHTML

An introduction to XML syntax

What set HTML5 apart

The living HTML specification maintained by the WHATWG

I'm not sure any HTML specification has had such fanfare as HTML5. It offers so many promising possibilities, in fact, that it has become something of a buzzword with connotations far beyond the spec itself. When marketers and journalists use the term "HTML5," they are sometimes referring to any new web technology that replaces Flash. The important thing, however, is that mainstream awareness of web standards is certainly a win and makes our job easier when communicating with clients.

But first, I think it's important to know how we got here and what makes HTML5 a breakthrough. I'll start with a brief history of HTML, then point out some unique qualities of HTML5, including its APIs.

AN ABBREVIATED HISTORY OF HTML

Understanding where we've been provides useful context for where we are going. Our journey to HTML5 passes through the frontier of the early web, the dangerous battlegrounds of the Browser Wars, and a flirtatious fling with XML.

The Wild Frontier

The story of HTML, from Tim Berners-Lee's initial draft in 1991 to the HTML5 standard in development today, is both fascinating and tumultuous. Early versions of HTML (HTML+ in 1994 and HTML 2.0 in 1995) built on Tim's early work with the intent of making HTML a viable publishing option.

But when the World Wide Web (as it was adorably called back in the day) took the world by storm, browser creators, most notably Mosaic Netscape and later Microsoft Internet Explorer, each said, "We ain't waitin' for no stinkin' standards!" (and probably spit in a spittoon, but I'm only guessing). They gave the people what they wanted by creating a slew of browser-specific elements for improving the look of pages on their respective browsers. This divisive one-upping became known as the Browser Wars. As a result, it became common in the late 1990s to create two entirely separate versions of a site that targeted each of the Big Two browsers. Signs on sites reading "Best viewed in Netscape" were the norm. I shudder just thinking about it.

FURTHER READING

For a detailed history HTML from 1989 to 1998, read David Raggett's account from his book ***Raggett on HTML4*** *(Addison-Wesley), available on the W3C site (www.w3.org/People/Raggett/book4/ch02.html).*

A Call for Reason

In 1996, the newly formed W3C put a stake in the ground and released its first Recommendation: HTML 3.2. It is a snapshot of all the HTML elements in common use at the time, and includes many presentational extensions to HTML (such as the `font` and `center` elements) that were the result of the Netscape/IE feud and the lack of a style sheet alternative. HTML 4.0 (1998) and HTML 4.01 (the slight revision that superseded it in 1999) aimed to get HTML back on track by emphasizing the separation of structure and presentation and improving accessibility. All matters of presentation were handed over to the newly minted Cascading Style Sheets standard that was gaining support.

HTML 4.01, along with XHTML 1.0, its stricter XML-based sibling (discussed next), became the cornerstone of the web standards movement (see the sidebar "The Web Standards Project").

The Web Standards Project

In 1998, at the height of the browser wars, a grassroots coalition called the Web Standards Project (WaSP for short) began to put pressure on browser creators (primarily Netscape and Microsoft at the time) to start sticking to the open standards as documented by the W3C. Not stopping there, it educated the web developer community on the many benefits of developing with standards. Its efforts revolutionized the way sites are created and supported. Now browsers (even Microsoft) brag of standards support while continuing to innovate.

In 2013, WaSP declared, "Our work here is done," and disbanded. You can still read its mission statement, history, and reference materials on the WaSP site (webstandards.org).

Enter XML and XHTML

Around the same time that HTML 4.01 was in development, folks at the W3C became aware that one limited markup language wasn't going to cut it for describing all the sorts of information (chemical notation, mathematical equations, multimedia presentations, financial information, and so on) that might be shared over the web. Their solution was XML (eXtensible Markup Language), a metalanguage for creating markup languages. XML was a simplification of SGML (Standardized Generalized Markup Language), the big kahuna of metalanguages that Tim Berners-Lee used to create his original HTML application. But SGML itself proved to be more complex than the web required.

The W3C had a vision of an XML-based web with many specialized markup languages working together—even within a single document. Of course, to pull that off, everyone would have to mark up documents very carefully, strictly abiding by XML syntax, to rule out potential confusion.

Their first step was to rewrite HTML according to the rules of XML so that it could play well with others. The result is XHTML (eXtensible HTML). The first version, XHTML 1.0, is nearly identical to HTML 4.01, sharing the same elements and attributes, but with stricter syntax requirements (see the "XHTML Markup Requirements" sidebar).

But the W3C didn't stop there. With a vision of an XML-based web in mind, they began work on XHTML 2.0, an even bolder attempt to make things work "right" than HTML 4.01 had been. The problem was that it was not

XHTML Markup Requirements

XHTML syntax follows the strict markup requirements of XML, including:

- Element and attribute names must be lowercase. In HTML, element and attribute names are not case-sensitive.
- All elements must be closed (terminated). Empty elements are closed by adding a slash before the closing bracket (for example, **
**).
- Attribute values must be in quotation marks. Single or double quotation marks are acceptable as long as they are used consistently. Furthermore, there must be no extra whitespace (character spaces or line returns) before or after the attribute value inside the quotation marks.
- All attributes must have explicit attribute values. XML (and therefore XHTML) does not support attribute minimization, the SGML practice in which certain attributes can be reduced to just the attribute value. So, while in HTML you can write **checked** to indicate that a form button be checked when the form loads, in XHTML you need to explicitly write out **checked="checked"**.
- Proper nesting of elements is strictly enforced. Some elements have new nesting restrictions.
- Start tags and end tags are required.
- Special characters must always be represented by character entities (e.g., **&** for the & symbol).
- Scripts must be contained in a CDATA section so they will be treated as simple text characters and not parsed as XML markup. Here is an example of the syntax:

```
<script type="type/javascript">
  // <![CDATA[
  ... JavaScript goes here...
  // ]]>
</script>
```

Unlike HTML parsers, which are forgiving of incorrect markup, errors in XHTML syntax stops the parser in its tracks. Running your XHTML code through a validator is a good idea to catch syntax errors before pages get launched.

backward-compatible with old standards and browser behavior. The writing and approval process dragged on for years with no browser implementation. Without browser implementation, XHTML 2.0 was dead in the water.

Hello HTML5!

HTML5 aimed to make HTML more useful for creating web applications.

Meanwhile...

In 2004, members of Apple, Mozilla, and Opera formed the Web Hypertext Application Technology Working Group (WHATWG, *whatwg.org*), separate from the W3C. The goal of the WHATWG was to further the development of HTML to meet new demands in a way that was consistent with real-world authoring practices and browser behavior (in contrast to the start-from-scratch ideal that XHTML 2.0 described). Their initial documents, Web Applications 1.0 and Web Forms 1.0, were rolled together into HTML5.

The W3C eventually established its own HTML5 Working Group (also led by Hickson) based on the work done by the WHATWG. HTML5 reached formal Recommendation status in October of 2014.

For a while, work on the HTML5 specification happened at both organizations in tandem, with slight inconsistencies. In 2019, the W3C officially handed over control of HTML development to the WHATWG under a Memorandum of Understanding (MoU). This agreement established WHATWG as the maintainer of the "HTML Living Standard", which evolves continuously based on the needs of the development community rather than having numbered releases. It has been adopted by the W3C as the official HTML specification.

And XHTML 2.0? At the end of 2009, the W3C officially put it out of its misery, pulling the plug on the working group.

So that's how we got here. Now let's get to know HTML5 a little better.

HTML5: MORE THAN MARKUP

Prior HTML versions concerned themselves mainly with elements for marking up content to be viewed on web pages. HTML5 is a bundle of new methods for accomplishing tasks that previously required special programming or proprietary plug-in technology such as Flash or Silverlight. Solutions include both markup and scripting components, including APIs for things like putting audio and video on the page, the ability to store data locally, work offline, take advantage of location information, and more. With HTML5 for common tasks, developers can rely on built-in browser capabilities and not reinvent the wheel for every application.

Much of what's new in HTML5 requires advanced web development skills, so it is unlikely you'll use them right away (if ever), but as always, I think it is

beneficial to everyone to have a basic familiarity with what can be done. And "basic familiarity" is what I'm aiming at here. For more in-depth discussions of HTML5 features, I recommend the following books:

- *HTML5 for Web Designers, 2e*, by Rachael Andrew and Jeremy Keith (A Book Apart)
- *HTML5: Up and Running* by Mark Pilgrim (O'Reilly and Google Press)
- *Introducing HTML5, 2e* by Bruce Lawson and Remy Sharp (New Riders)

HTML5 Markup Component

HTML5 is based on HTML 4.01 Strict, the version of HTML that did not include any presentation-based or other deprecated elements and attributes. That means the vast majority of HTML5 is made up of the same elements that were used for years, and browsers know what to do with them.

HTML5 introduced a number of new elements, form input types, and global attributes. It also made many deprecated elements and attributes in HTML 4.01 officially obsolete.

One departure from previous HTML versions is that HTML5 is the first specification that includes detailed instructions for how browsers should handle malformed and legacy markup. It bases the instructions on legacy browser behavior, but for once, there is standard protocol for browser makers to follow when browsers encounter incorrect or non-standard markup.

A DTD-Free DOCTYPE

HTML documents should begin with a Document Type Declaration (DOCTYPE declaration) that identifies which version of HTML the document follows. The declaration we use today (established for HTML5) is short and sweet:

```
<!DOCTYPE html>
```

Compare that to a declaration for a Strict HTML 4.01 document:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/HTML4.01/strict.dtd">
```

Why so complicated? For documents written in HTML 4.01 and XHTML 1.0 and 1.1, the declaration must point to the public DTD (Document Type Definition), a document that defines all of the elements in a markup language as well as the rules for using them. DTDs are a remnant of SGML and proved to be less helpful on the web than originally thought, so the authors of HTML5 simply didn't use one. As a result, the DOCTYPE declaration is much more simple.

HTML5 in XML

HTML5 can also be written according to the stricter syntax of XML (called the XML serialization of HTML5). Some developers have come to prefer the tidiness of well-formed XHTML (lowercase element names, quoted attribute values, closing all elements, and so on), so that way of writing is still an option, although not required. In edge cases, an HTML5 document may be required to be served as XML in order to work with other XML applications, in which case it can use the XML syntax and be ready to go.

Validators—software that checks that all the markup in a document is correct (see Note)—use the DOCTYPE declaration to make sure the document abides by the rules of the specification it claims to follow.

NOTE

To check whether your HTML document is valid, use the online validator at the W3C (validator.w3.org). An HTML5-specific validator is also available at html5.validator.nu.

Both HTML 4.01 and XHTML 1.0 had three separate DTDs (for Traditional, Strict, and Frameset versions of each spec), so there were a lot of little details to keep track of. For a full list of DOCTYPE declarations (including DTD references) for HTML 4.01, XHTML, SVG, and other document types, go to *www.w3.org/QA/2002/04/valid-dtd-list.html*.

Meet the APIs

HTML specifications prior to HTML5 included only documentation of the elements, attributes, and values permitted in the language. That's fine for simple text documents, but the creators of HTML5 had their minds set on making it easier to create web-based applications that require scripting and programming. For that reason, HTML5 also defines a number of new APIs for making it easier to communicate with an application.

An API (Application Programming Interface) is a documented set of commands, data names, and so on, that lets one software application communicate with another. For example, the developers of Twitter (now X) documented the names of each data type (users, tweets, timestamps, and so on) and the methods for accessing them in an API document (dev.twitter.com/docs) that lets other developers include feeds and elements in their programs. That is why there are so many Twitter/X programs and widgets available. Amazon.com also opens up its product data via an API. In fact, publishers of all ilks are recognizing the power of having their content available via an API. You could say that APIs are hot right now.

But let's bring it back to HTML5, which includes APIs for tasks that traditionally required proprietary plug-ins (like Flash) or custom programming. The idea is that if browsers offer those features natively—with standardized sets of hooks for accessing them—developers can do all sorts of nifty things and count on them working in all browsers, just as we count on the ability to embed an image on a page today. Of course, we have a way to go before there is ubiquitous support for some cutting-edge features, but we're getting there steadily. Some APIs have a markup component, such as embedding multimedia with the new HTML5 `video` and `audio` elements. Others happen

entirely behind the scenes with JavaScript or server-side components, such as creating web applications that work even when there is no Internet connection (Offline Web Application API).

The W3C and WHATWG are working on *lots and lots* of APIs for use with web applications, all in varying stages of completion and implementation. Most have their own specifications, separate from the HTML5 spec itself, but they are generally included under the wide HTML5 umbrella that covers web-based applications. HTML5 includes specifications for these APIs (see Note):

Media Player API

For controlling audio and video players embedded on a web page, used with the new **`video`** and **`audio`** elements.

Session History API

Exposes the browser history for better control over the Back button.

Editing API

Provides a set of commands that could be used to create in-browser text editors, allowing users to insert and delete text; format text as bold, italic, or as a hypertext link; and more. In addition, there is a new **`contenteditable`** attribute that allows any content element to be editable right on the page.

Drag and Drop API

Adds the ability to drag a text selection or file to a target area on the page or another web page. The **`draggable`** attribute indicates the element can be selected and dragged.

The following are just a handful of the APIs in development at the W3C with specifications of their own:

Canvas API

The **`canvas`** element adds a dynamic, two-dimensional drawing space to a page. The **`canvas`** element is discussed in Chapter 10, Embedded Media.

Service Workers API

This specification describes a method that enables web applications to work while offline.

Web Storage API

Allows data to be stored in the browser's cache so that an application can use it later. Traditionally, that has been done with "cookies," but the Web Storage API allows more data to be stored. It also controls whether the data is limited to one session (**`sessionStorage`**: when the window is closed, the data is cleared) or based on domain (**`localStorage`**: all open windows pointed to that domain have access to the data).

NOTE

The following list is from the 2018 version of this article and has not been updated. For a list of current APIs, see the "Web APIs" page on MDN Web Docs (developer.mozilla.org/en-US/docs/Web/API). The W3C lists all the documents they maintain, many of which are APIs, at www.w3.org/TR/tr-title-all.

Geolocation API

Lets users share their geographical location (longitude and latitude) so that it is accessible to scripts in a web application. This allows the app to provide location-aware features such as suggesting a nearby restaurant or finding other users in your area.

Web Sockets API

Creates a "socket," which is an open connection between the browser client and the server. This allows information to flow between the client and the server in real time, with no lags for the traditional HTTP requests. You can think of a web socket as an ongoing telephone call between the browser and server compared to the walkie-talkie, one-at-a-time style of traditional browser/server communication. (A hat tip to Jen Simmons for this analogy.) It is useful for multiplayer games, chat, or data streams that update constantly, such as sports or stock tickers or social media streams

WHERE WE GO FROM HERE

The WHATWG maintains the living (unnumbered) HTML specification that is continually updated and forms the basis for most browser vendors' implementation. All indications are that the HTML specification will continue to undergo minor revisions in order to keep up with the changing demands of how we use the web and the devices we use it on. Minor revisions generally include attributes, elements, and APIs and putting to rest features that were never implemented. In other words, there is no indication that there will be another seismic shift such as the switch from XHTML to HTML5.

So know that HTML is always changing, and as a web designer or developer, you will need to keep your ear to the ground. There is always something new to learn..