Understanding HTML, XML and XHTML

Sep 20, 2006

by Maciej Stachowiak

@othermaciej

OR, Close your `<script>` and `<canvas>` tags!

The relationships among HTML, XML and XHTML are an area of considerable confusion on the web. We often see questions on the `webkit-dev` mailing list where people wonder why their seemingly XHTML documents result in HTML output. Or we're asked why an XML construct like `<b />` doesn't actually close the bold tag.

This article will attempt to clear up some of that confusion.

You may be wondering what the subtitle has to do with the title. Well, the HTML/XHTML distinction may seem like an obscure topic, but it can have significant practical effects. In particular, it is likely to affect Dashboard Widget developers in a huge way in upcoming WebKit versions. I'll explain further at the end.

What are HTML, XML and XHTML?

The original language of the World Wide Web is HTML (HyperText Markup Language), often referred to by its current version, HTML 4.01 or just HTML4 for short. HTML was originally an application of SGML (Standard Generalized Markup Language), a sort of meta-language for making markup languages. SGML is quite complicated, and in practice most browsers do not actually follow all of its oddities. HTML as actually used on the web is best described as a custom language influenced by SGML.

Another important thing to note about HTML is that all HTML user agents (this is a catchall term for programs that read HTML, including web browsers, search engine web crawlers, and so forth) have extremely lenient error handling. Many technically illegal constructs, like misnested tags or bad attribute names, are allowed or safely ignored. This error-handling is relatively consistent between browsers. But there are lots of differences in edge cases, because this error handling behavior is not documented or part of any standard. This is why it is a good idea to validate your documents.

XML and XHTML are quite different. XML (eXtensible Markup Language) grew out of a desire to be able to use more than just the fixed vocabulary of HTML on the web. It is a meta-markup language, like SGML, but one that simplifies many aspects to make it easier to make a generic parser. XHTML (eXtensible HyperText Markup Language) is a reformulation of HTML in XML syntax.

While very similar in many respects, it has a few key differences.

First, XML always needs close tags, and has a special syntax for tags that don't need a close tag. In HTML, some tags, such as `img` are always assumed to be empty and close themselves. Others, like `p` may close implicitly based on other content. And others, like `div` always need to have a close tag. In XML (including XHTML), any tag can be made self-closing by putting a slash before the code angle bracket, for example `<img src="funfun.jpg"/>`. In HTML that would just be `<img src="funfun.jpg">`

Second, XML has draconian error-handling rules. In contrast to the leniency of HTML parsers, XML parsers are required to fail catastrophically if they encounter even the simplest syntax error in an XML document. This gives you better odds of generating valid XML, but it also makes it very easy for a trivial error to completely break your document.

HTML-compatible XHTML

When XML and XHTML were first standardized, no browser supported them natively. To enable at least partial use of XHTML, the W3C came up with something called "HTML-compatible XHTML". This is a set of guidelines for making valid XHTML documents that can still more or less be processed as HTML. The basic idea is to use self-closing syntax for tags where HTML doesn't want a close tag, like `img`, `br` or `link`, with an extra

space before the slash. So our ever-popular image example would look like this: `<img src="funfun.jpg" />`. The details are described in the Appendix C of the XHTML 1.0 standard.

It's important to note that this is kind of a hack, and depends on the de facto error handling behavior of HTML parsers. They don't really understand the XML self-closing syntax, but writing things this way makes them treat `/` as an attribute, and then discard it because it's not a legal attribute name. And if you tried to do something like `<div />`, they wouldn't understand that the `div` is supposed to be empty.

There are also many other subtle differences between HTML and XHTML that aren't covered by this simple syntax hack. In XHTML, tag names are case sensitive, scripts behave in subtly different ways, and missing implicit elements like `<tbody>` aren't generated automatically by the parser.

So if you take an XHTML document written in this style and process it as HTML, you aren't really getting XHTML at all – and trying to treat it as XHTML later may result in all sorts of breakage.

What determines if my document is HTML or XHTML?

You may be a bit thrown off by the last sections talk of treating an XHTML as HTML. After all, if my document is XHTML, that should be the end of the story, right? After

all, I put an XHTML doctype! But it turns out that things are not so simple.

So what really determines if a document is HTML or XHTML? The one and only thing that controls whether a document is HTML or XHTML is the MIME type. If the document is served with a `text/html` MIME type, it is treated as HTML. If it is served as `application/xhtml+xml` or `text/xml`, it gets treated as XHTML. In particular, none of the following things will cause your document to be treated as XHTML:

- Using an XHTML doctype declaration
- Putting an XML declaration at the top
- Using XHTML-specific syntax like self-closing tags
- Validating it as XHTML

In fact, the vast majority of supposedly XHTML documents on the internet are served as `text/html`. Which means they are not XHTML at all, but actually invalid HTML that's getting by on the error handling of HTML parsers. All those "Valid XHTML 1.0!" links on the web are really saying "Invalid HTML 4.01!".

HTML is probably what you want

Perhaps you're just now realizing that your lovingly crafted valid XHTML document is actually invalid HTML. You have a couple of choices:

1. **Serve your content as `application/xhtml+xml`.** That's probably not such a good idea though.

Microsoft Internet Explorer will not handle XHTML at all, and serving it such a MIME type will lead it to download. Unless you're willing to completely lock out IE users, you probably don't want to take this option.

2. **Serve as `text/html` to IE, but as `application/xhtml+xml` to other browsers.** This way your content at least has a chance of working in IE, and uses HTML-compatible XML for its original intended purpose, as a fallback compatibility hack. However, there are still downsides. Your documents will be processed in entirely different ways in IE vs other browsers. A construct that may be perfectly valid HTML could totally break XML parsing, due to the strict error handling rules. Or conversely, some kinds of valid XHTML changes might result in an HTML document that looks wrong. Furthermore, the XHTML modes of the browsers that support it are not nearly as mature or well tested as the HTML modes. This is definitely the case for Safari. And Mozilla also discourages this practice due to lack of support for incremental rendering. And they have a list of some of the many differences in processing XHTML vs HTML.
3. **Stick with the status quo.** Another option is to just stick with the status quo – generate XHTML content but serve it as HTML. The disadvantages here are mainly that you are losing out on HTML validators, which will validate the document in a way that matches how browsers parse it; and that you run the risk of subtle incompatibilities if your document is

ever actually processed as XHTML. But this also raises the question: what do you think you are getting out of using XHTML? You may have heard a lot of hype about it, experts may have told you it's the next big thing, but what kind of benefits do you get if in the end it's just treated as HTML tag soup?

4. **Serve valid HTML.** This is the option I recommend – serve valid HTML documents with a `text/html` MIME type. This way you'll be using the best-tested mode of web browsers, won't have to worry as much about weird compatibility issues, and will get the most benefit out of HTML-based toolchains.

So overall it seems best to go with HTML, and follow through consistently. But don't just take my word for it. Leading web standards experts like Ian Hickson, Anne van Kesteren and Mark Pilgrim have all pointed out the pitfalls of serving XHTML as HTML.

Best practices

On today's web, the best thing to do is to make your document HTML4 all the way. Full XHTML processing is not an option, so the best choice is to stick consistently with HTML4. Here's the best way to do that:

1. Use an HTML4 doctype declaration, ideally one that will trigger "standards mode" in web browsers. One example of an HTML4 standards mode doctype is

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

2. Serve your content with the `text/html` MIME type, or for local content give it a `.html` or `.htm` suffix. This will lead browsers, search engines, and other apps to properly process your content as HTML.
3. Validate your content as HTML, not as XHTML. One handy way is using a validation service, such as the W3C Validator. (But beware, the validator looks at your doctype instead of the MIME type, unlike browsers.)

Unfortunately, sometimes you are not fully in control of the content you produce. For example, this very blog, published with WordPress tags. If you find yourself in this same boat, encourage your tools vendors to provide support for generating valid HTML.

About those <script> and <canvas> tags

I promised at the start of this post to tell you all what this had to do with closing script and canvas tags in Dashboard widgets. Well, the upshot is that XML-style self-closing syntax in HTML is not always so innocuous.

The Safari 2.0 version of WebKit had a special quirk for treating `script` elements with the self-closing syntax (like this: `<script src="myscript.js" />`) as if they were actually properly closed. At the time Gecko-based browsers like Firefox had a similar quirk, and we decided to copy it for compatibility with particular web sites. However, future versions of Firefox will remove this quirk, and this kind of behavior is going to be explicitly outlawed by future standards that build on HTML, such

as Web Apps 1.0. So we will probably remove this quirk in future versions of WebKit as well. Unfortunately, HTML relying on this parsing quirk has crept into a lot of Dashboard widgets. A WebKit that didn't support this quirk would lead to broken widgets – the external script code would never run.

There is a similar issue with the `canvas` alement, as it makes its way through the standardization process. `canvas` was originally implemented in Safari as an empty tag like `img`, but standards and other browsers have all gone with making it require an explicit close tag, to support fallback content. Widgets hit two different pitfalls here – many use the XML self-closing syntax (`<canvas />`), while others have just a plain old unclosed (`<canvas>`) tag. Either way, you need to change to using an explicit close tag (`<canvas></canvas>`, or future WebKit versions will think all the rest of your document is inside the `canvas` element and won't render it.

Conclusion

It's easy to get confused about HTML and XHTML, and many of the experts out there give misleading advice on the subject. Fortunately, most of the time it doesn't matter. But sometimes it does, and can badly break your content. So make sure you understand the difference, and serve up some good clean markup. ■

Older Post
Strange Medium

Newer Post
Mitz Pettel is a WebKit reviewer