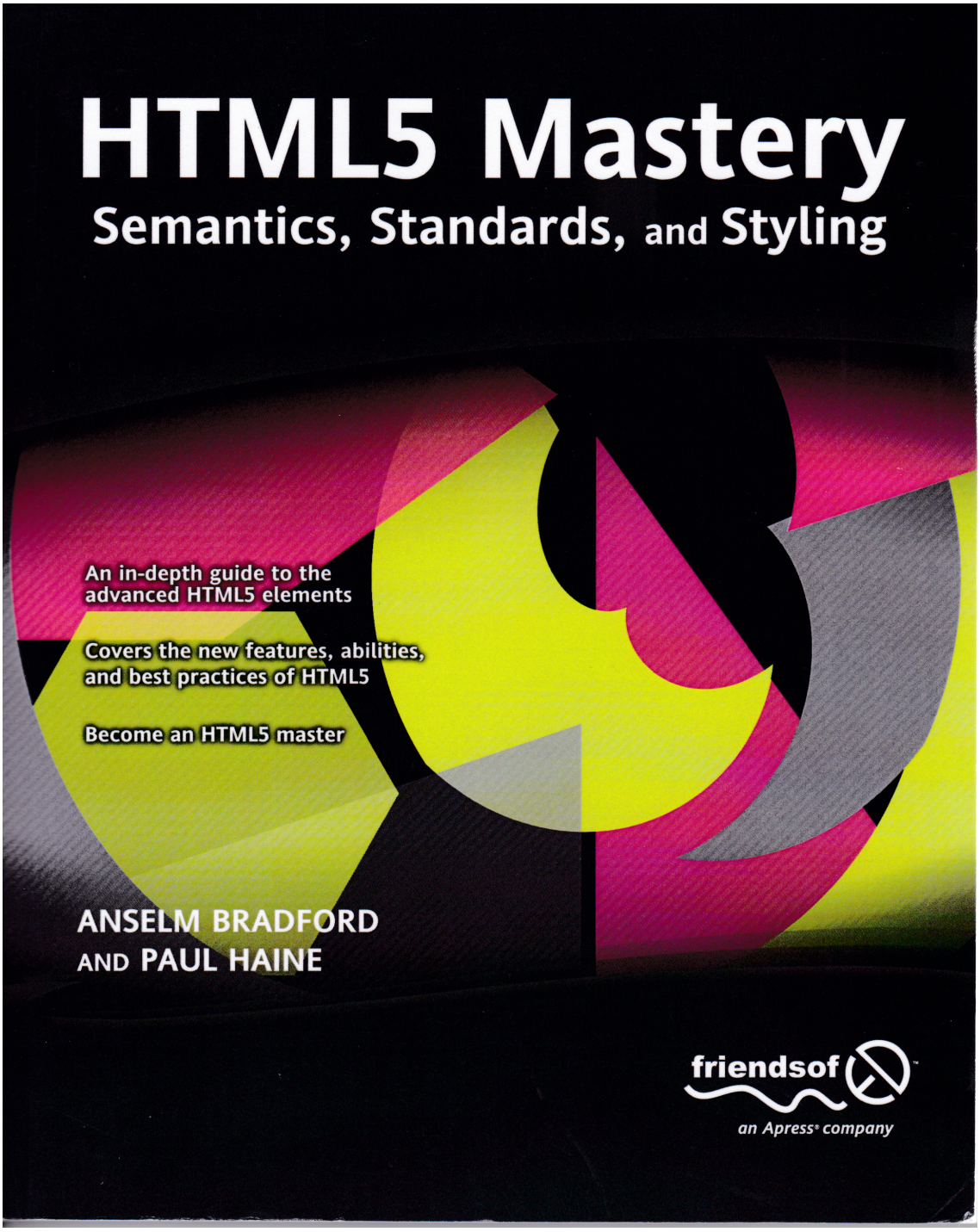

HTML5 Mastery: Semantics, Standards, and Styling

Anselm Bradford
Paul Haine

HTML5 Mastery: Semantics, Standards, and Styling

ISBN-13 (pbk): 978-1-4302-3861-4

ISBN-13 (electronic): 978-1-4302-3862-1

Distributed to the book trade worldwide by Springer Science+Business Media LLC., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com.

For information on translations, please e-mail rights@apress.com or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at www.apress.com/bulk-sales.

The source code for this book is freely available to readers at www.friendsofed.com in the Downloads section.

Credits

President and Publisher: Paul Manning

Lead Editor: Ben Renow-Clarke

Development Editor: Susan Ethridge

Technical Reviewer: Jeffrey Sambells

Editorial Board: Steve Anglin, Mark Beckner, Ewan Buckingham, Gary Cornell, Jonathan Gennick, Jonathan Hassell, Michelle Lowman, James Markham, Matthew Moodie, Jeff Olson, Jeffrey Pepper, Frank Pohlmann, Douglas Pundick, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh

Copy Editor: Kim Wimpsett

Compositor: Bytheway Publishing Services

Indexer: SPI Global

Artist: SPI Global

Cover Image Artist: Corné van Dooren

Cover Designer: Anna Ishchenko

Coordinating Editor: Jennifer L. Blackwell

environment), so I don't cover this sort of usage in great detail here. Simply be aware that when embedding plug-in-based media, the `object` element is what you usually need to use.

To wrap up this discussion on embedded media, it's important to mention a feature of the `object` element that came up earlier: the fallback content mechanism. How exactly is this done? Well, it's quite intuitive and easy to implement. You can nest `object` elements (or other elements), allowing the web browser to display alternative content if it cannot render the preferred choice. For instance, you can nest a video, an image, and finally some text like so:

```
<object data="video.mpg" type="application/mpeg">
        <object data="picture.jpg" type="image/jpg">
                Some descriptive text, and <a href="video.mpg">a download link</a>.
        </object>
</object>
```

The user agent should first try to display the video, but if it can't, it should then try to display the image, and if it can't do that, it displays the text—no need for `alt` attributes here! This concept of nesting alternative content within an element is particularly important to the elements to come later in this chapter, such as `video`, so keep it in mind.

■ **Note** Although `object` can theoretically be used to embed many types of media, the reality is that many new elements added in HTML5, as well as existing elements, have usurped `object` on several fronts. `video` and `audio` should be used in place of `object` for embedding those types of media, and `img` has long been the ubiquitous answer to embedded images, despite `object`'s attempt to supersede it. Also, `iframe` (discussed in the next section) is a better alternative for nesting one web page inside another. The `applet` element for Java applets, however, is obsolete in HTML5, and for this type of content `object` should be used instead. It should also be used for plug-in-based media such as Adobe Flash content.

Embedding HTML: iframe

Netscape Navigator 2.0 introduced the ability to create frames, which seemed pretty innovative at the time. The ability to construct an HTML page out of numerous other pages meant, for instance, that in a page that had a frame for content and a frame for a navigation bar, the content of the page could be loaded without the need to reload the navigation bar. Also, the contents of the page could be scrolled while leaving the navigation bar stationary. Along with these advantages came a number of problems, however, which led to frames falling out of favor. Among these were difficulties search engines had navigating a frameset, leading to situations where a particular frame outside of its frameset context might turn up in search results. Since the days of Netscape Navigator, advances in web browser and web server technology have made frames obsolete (CSS, for instance, can fix the position of content so it does not scroll). In HTML5 the `frame` and `frameset` elements are obsolete and must not be used. What remains in HTML5 is the `iframe` element, known as an **inline frame**. This element allows an entire HTML page (or snippets of HTML code) to be embedded in another page. The ability to embed one page into another is useful for incorporating third-party HTML code into your own web page. For example, this element is often used to embed third-party ads (for instance, Google's AdSense ads). Other uses include Facebook's "Like" button and Twitter's "tweet" button, which are often seen embedded in blog posts or similar and are used to share the article on those social networks. An inline frame might also be

used to embed more extensive third-party content on your site, such as a news feed or articles from another website.

Handling content in the iframe element

The `iframe` element creates a box with a set width and height into which the external document is loaded. It loads its content via a URL supplied to the `src` attribute. Like `object`, content can be added between the opening and closing tags as a fallback mechanism should the page be viewed by a browser that does not support inline frames. While all major browsers support the `iframe` element, from an accessibility perspective, it is a good practice to add fallback content anyway. If nothing else, it can be used like a comment to remind you what the linked resource is. Consider also including a link to the embedded document so that the user has access to the document even if the `iframe` is not supported. Here is an example:

```
<iframe src="embed.html" width="300" height="150">
        <a href="embed.html">View embedded web page.</a>
</iframe>
```

If the previous snippet were viewed in a browser that did not support `iframe`, the link would be displayed; otherwise, the contents of `embed.html` would be rendered within the boundaries of an inline frame, which is a 300 pixel wide by 150 pixel tall box (in this case, as set by the `width` and `height` attribute). Horizontal and vertical scrollbars appear as needed to fit the content, as shown in Figure 5-3.

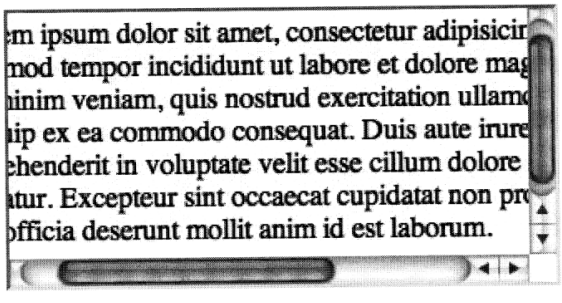

Figure 5-3. *Appearance of an inline frame with scrollbars*

Since it provides a fixed area to display the content within, an `iframe` can also be used to display a large amount of content (text generally) in a confined area on the page. For example, perhaps your website has a "terms and conditions" statement, which generally would be a long document filled with legalese text. The document is accessible through a link in the footer at the bottom of your pages, but you also want to embed this same document in a registration form for when a user registers on your site. Instead of having a duplicate of the terms and conditions page on your site—both linked in the footer and on a user registration form—you could embed the existing document into the registration page, such as shown in Figure 5-4.

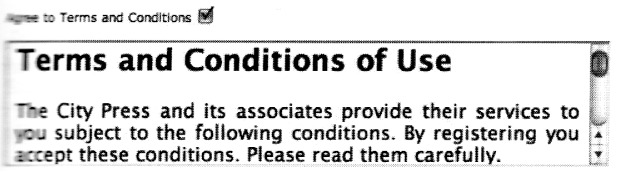

Figure 5-4. Example of using an inline frame to embed an existing text-heavy document in a small area

The source code for Figure 5-4 might look like this:

```
…
<p>
        <label>Agree to Terms and Conditions <input type="checkbox" title="Agree to Terms
        and Conditions" /></label>
</p>
<iframe src="tnc.html" width="500" height="100">
        <a href="tnc.html" target="_blank">View Terms and Conditions</a>
</iframe>
…
```

New iframe element attributes

As far as the attributes of `iframe` go, there has been the usual practice of presentational attributes being marked obsolete in HTML5. The following attributes should not be used: `frameborder`, `marginheight`, `marginwidth`, and `scrolling`. Additionally, the `longdesc` attribute has been dropped, which shouldn't be a loss, because it never had adequate browser support anyway.

HTML5 has added a small number of attributes related to the ability to embed HTML code snippets and pages that are **sandboxed**, meaning they are prevented from performing certain operations that could be used for nefarious purposes by third-party pieces of code. The idea is to use inline frames as a way to secure pieces of code added dynamically to a web page from a third-party source, as might be done as part of the commenting system on a blog or forum. The three attributes are `srcdoc`, `seamless`, and `sandbox`.

■ **Note** You may find that the new `iframe` attributes don't work in your preferred web browser, because they have yet to gain much support among major browsers. As of this writing, `sandbox` is the only one of the three new attributes to have support (in Safari and Google Chrome). We hope further support will be quick to arrive, but as with any new feature in HTML5, keep an eye out for changes to these attributes as the HTML5 specification is refined further in the months and years to come.

The `srcdoc` attribute allows snippets of HTML to be entered directly into the attribute's value, unlike `src`, which needs to be given a URL that points to another HTML file, so for example:

```
<iframe srcdoc="<p>This text is in an inline frame</p>"><p>This is regular text</p></iframe>
```

This code will create a snippet of HTML that will be inserted into an inline frame. If the `srcdoc` and `src` attributes are present, `srcdoc` will override `src`, but `src` can still be included and given a value so as to provide fallback content for when the `srcdoc` attribute is not supported (which currently is the case in all major browsers).

The `seamless` attribute, which is a Boolean attribute, makes the included content appear to be part of the containing document, meaning, for example, that links clicked in the inline frame will load in the parent document instead of loading inside the `iframe` by default.

The last attribute, `sandbox`, may be treated as a Boolean attribute but does not necessarily need to be (as you'll see momentarily). When treated as a Boolean attribute, it adds a number of security restrictions to the `iframe`'s source content. These restrictions are as follows:

- **Restricted local access**: The content is treated as being from a different server, which prevents access to local server content such as cookies and other web storage options tied to the local server domain.
- **No form submission**: Form submission from the inline content is disabled.
- **No JavaScript**: Scripts in the inline content are disabled.
- **No external link targets**: Links in the inline content are prevented from targeting other browsing contexts, such as the containing document through the use of target="_parent", for example.
- **No plug-ins**: Inline content requiring plug-ins, such as for Adobe Flash content, are disabled.

As a Boolean attribute, `sandbox` just needs to be added to the `iframe` to enable these restrictions, like so:

```
<iframe src="external.html" sandbox><!-- Fallback content --></iframe>
```

If the `sandbox` attribute is not treated as a Boolean attribute, a number of text keywords values can be set that will negate almost all of the previous restrictions. Table 5-1 shows the available keywords. More than one keyword can be added to negate more than one restriction, with each separated by a space. Here's an example:

```
<iframe src="external.html" sandbox="allow-forms allow-top-navigation">
    <!-- Fallback content -->
</iframe>
```

The previous code would allow form submissions and external link targets in the embedded content but would have the other sandbox restrictions in effect.

Table 5-1. The sandbox attributes

Sandbox attribute value	Description
`allow-same-origin`	Allow content to be treated as if it is from the same origin as the local server's content
`allow-scripts`	Allow the execution of scripts
`allow-forms`	Allow form submissions
`allow-top-navigation`	Allow links to target other browsing contexts (such as the containing HTML page)

■ **Note** Notice that there is not a keyword to override disabling plug-ins. The reason for this is a plug-in could be built that circumvented some of the other restrictions, presenting a vulnerability to any selective restrictions in place. Therefore, plug-ins are always disabled as long as the `sandbox` attribute is present.

Targeting inline frames

The `iframe` has one last attribute—which isn't new—the `name` attribute, which can be used to identify a particular inline frame among other inline frames used on a page. Doing so means a particular inline frame can be used as a target for links when the `target` attribute is set to a particular inline frame's name. Here's an example:

```
<iframe name="terms" src="tnc.html">
          <a href="tnc.html" target="_blank">View Terms and Conditions</a>
</iframe>
<a href="page2.html" target="terms">Next page</a>
```

In this case, when the "Next page" hyperlink text is clicked, the `page2.html` page is loaded into the "terms" inline frame.

Video

More than any other element, the `video` element is what brought HTML5 into the public consciousness. To the wider public that just uses the Web, it became known that videos on YouTube could not be viewed on the iPhone/iPad because YouTube used Adobe Flash for displaying video.[6] However, there

[6] YouTube still predominantly uses Flash, but it has an HTML5 player at www.youtube.com/html5, and videos viewed with a device that does not support Flash will fall back to using this HTML5 video player.

existed a new solution called "HTML5" that could display video on the iPhone/iPad. Exactly what that meant was no doubt lost on the majority of users of the Web, but for you and others like you, it means there is a new element, `video`, that allows video files to be embedded into a web page without the use of any third-party plug-ins. It means native video playback in the browser. This differs from using `object` for video playback, because `object` just hands the video content off to a plug-in.

This is a huge addition to the HTML specification, because video can now be integrated anywhere other HTML elements can be used. For instance, CSS and JavaScript can interact with the video. However, implementing `video` has not been without its hurdles. Notably, a standard video format has not emerged, which leads to the need to encode video in different formats to handle all major web browsers. Let's begin our discussion there.

■ **Note** The `video` element has quite good support across major video browsers, but if you want to provide support for older browsers, consider a solution such as that found at `http://html5media.info`, which provides JavaScript code that emulates the `video` (and `audio`) element's functionality for older browsers.

Video formats

A video file is potentially a very large (in file size) piece of content to include on a web page. Outside of the Web, depending on the length and quality a video, a particular clip could easily stretch into gigabytes in size. Therefore, to display on the Web, a suitable form of compression needs to be applied to the video to reduce the size to make delivery over the Internet feasible. The form of compression applied to a video is referred to as a **codec**. Since video is a multimedia format, in that it can contain video *and* audio, the codec is only part of what makes up a video file. The video and audio are placed together in what is known as a *container* format, which is what the video file actually is. There are three major container formats for video on the Web: WebM, Ogg, and MPEG-4. The container format will contain a codec for video compression and a codec for audio compression, plus any metadata about the video, such as subtitles. The codecs used inside a particular container format can vary, but the commonly used ones on the Web are shown in Table 5-2, along with browser support for these formats.

Table 5-2. Major video formats used in HTML5 video

Container format	Video codec	Audio codec	Internet Explorer 9	Firefox 5	Google Chrome 13	Safari 5	Opera 11
WebM	VP8	Vorbis		■	■		■
OGG	Theora	Vorbis		■	■		■
MPEG-4	H.264 (also called MPEG-4 AVC)	AAC	■		□[7]	■	

As you can see in Table 5-2, currently there is not a single format that can be used across all major web browsers. It's a shame that Microsoft and Apple have not rallied behind WebM or Ogg,[8] but the reality of the current landscape of native browser video is such that web developers have to deal with delivering multiple formats for the same video file, if broad browser compatibility is a concern. We'll see how to do this after we've had a look at licensing issues.

The licensing issue

In addition to browser support issues, there is another issue to be aware of in regard to codecs. The codecs in MPEG-4, H.264, and ACC contain patented technology. This means certain uses of these formats are subject to royalty fees by the MPEG LA consortium, which holds the rights to these patents. You'll be fine streaming these videos for free on the Web, but it will be a different scenario if you wanted to charge users to view videos in this format or provided technology that decoded or encoded video in these formats. WebM and Ogg, on the other hand, are both open formats, unencumbered by any known patents.

Handling video sources

If you're delivering only one video, the markup is about as easy as placing an image on the page. The video file is supplied in the `src` attribute. Like `img`, a `width` and `height` are also advisable to set, but make sure it matches the aspect ratio of the video source:

[7] Don't lose all faith in the information I'm providing if you launch Google Chrome and find it doesn't run MPEG-4 video. Support has traditionally been included in Chrome, but Google has since pledged to remove support for H.264-encoded video in favor of the other two open formats, so going forward it can't reliably be said to support this format.

[8] There are ways to make Internet Explorer 9 and Safari support at least WebM, but it requires installing additional software that does not ship with the browsers, which is by no means an option to rely on when deciding how to deliver your video content to users in these browsers.

are more powerful than dedicated workers because they can communicate with multiple scripts, while a dedicated worker responds only to the script that spawned it in the first place.

Web Sockets API

The Web Sockets API[5] is a specification that defines a protocol for providing two-way communication with a remote host. The Web's roots have traditionally been essentially one-way. A server sends a page to a client web browser, and then nothing happens between the two until the user clicks a link and requests another page. What a web socket provides is an open connection over which data can be sent from the client to the server at any time after the page has loaded, and vice versa. This could be used , for example, to create multiplayer online games or applications, because data can be sent to the server from one client and distributed to all other clients connected to the same server.

Video conferencing and peer-to-peer communication

A project is underway to create a specification for video conferencing between two web browsers. This is a major area of difference between the W3C HTML5 and WHATWG HTML specifications, because it is included in the WHATWG version but omitted from the W3C specification. Instead, the W3C has a separate specification named "WebRTC 1.0: Web Real-time Communication Between Browsers."[6] Since both specifications are in draft status, it is not inconceivable that the version included in the WHATWG HTML draft may well be spun off into a separate specification in the future, as has happened at the W3C.

Anyway, administration issues aside, the actual technology for enabling video conferencing requires that two separate web browsers gather video and audio and stream it over a peer-to-peer connection to each other. Specifically, the following steps need to happen:

1. Gain access to a webcam or other video/audio input device.
2. Record the video/audio locally so that it can be streamed to a remote web browser.
3. Connect and send the video/audio to a remote web browser.
4. Display a video/audio stream in a video or audio element on the local and remote web browsers.

An API called the Stream API, which defines an interface called `MediaStream`, would be used with JavaScript to handle parsing and displaying of the streaming media. In terms of sending the media stream, another API, called the Peer-to-peer Connections API, would be used. This API describes a `PeerConnection` JavaScript interface that defines methods for connecting and sending a media stream to a remote peer.

[5] See `http://dev.w3.org/html5/websockets/`.
[6] See `http://dev.w3.org/2011/webrtc/editor/webrtc.html`.

WAI-ARIA

WAI-ARIA,[7] the
Initiative), aim
with disabilitie
describe how
values for desc
large number
attributes can
dragged objec
specification i
www.w3.org/W

File API

There are thr
the file syste
FileList, Fil
name and las
interfacing w
File API deal
even through
the File API
methods of i
so the file sy
into a user's
data can be
files in conju

Useful

The followi

-
-
-
-

[7] See www.
[8] See www.
[9] See www.
[10] See www.