

```
<!DOCTYPE html>
<meta charset="utf-8"><title>HTML5 Now</title>
<hgroup>
  <h1>
```

HTML5 NOW

```
</h1>
<h2>
```

**A STEP-BY-STEP TUTORIAL
FOR GETTING STARTED **TODAY****

```
</h2>
</hgroup>
```

```
<div class="vcard">
  <a href="http://tantek.com" class="url fn">
```

TANTEK ÇELİK

```
</a>
<div>
```

HTML5 NOW

A STEP-BY-STEP TUTORIAL
FOR GETTING STARTED **TODAY**

TANTEK ÇELİK

HTML5 Now
A Step-by-Step Tutorial for Getting Started Today

Tantek Çelik

New Riders
1249 Eighth Street
Berkeley, CA 94710
510/524-2178
Fax: 510/524-2221
Find us on the web at www.newriders.com

To report errors, please send a note to errata@peachpit.com
New Riders is an imprint of Peachpit, a division of Pearson Education

Copyright © 2011 by Pearson Education, Inc.

Senior Editor: Karyn Johnson
Production Editor: Hilal Sala
Copy Editor: Kelly Anton
Technical Editor: Ben Ward
Interior Design and Composition: Andreas deDanaan
Presentation Graphics and Design: Coley Wopperer
Author Photo: Matt Nuzzaco
Cover Design: Mimi Heft
Cover Production: Andreas deDanaan
Video Production and Direction: Mary Sweeney

Notice of Rights

All rights reserved. No part of this book may be reproduced or transmitted in any form by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher. For information on getting permission for reprints and excerpts, contact permissions@peachpit.com.

Notice of Liability

The information in this book is distributed on an “As Is” basis without warranty. While every precaution has been taken in the preparation of the book, neither the author nor Peachpit shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the instructions contained in this book or by the computer software and hardware products described in it.

Trademarks

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and Peachpit was aware of a trademark claim, the designations appear as requested by the owner of the trademark. All other product names and services identified throughout this book are used in editorial fashion only and for the benefit of such companies with no intention of infringement of the trademark. No such use, or the use of any trade name, is intended to convey endorsement or other affiliation with this book.

ISBN-13: 978-0-321-71991-1
ISBN-10: 0-321-71991-3

9 8 7 6 5 4 3 2 1

Printed and bound in the United States of America

ABOUT THE AUTHOR



Tantek Çelik is a computer scientist for Microformats.org and works with numerous clients, such as Mozilla and Revision3, on open web standards leadership and HTML5 upgrades. Previously, as Chief Technologist at Technorati, he led the design and development of new standards and technologies. Prior to Technorati, he was a veteran representative to the World Wide Web Consortium (W3C) for Microsoft, where he also helped lead the development of the award-winning Internet

Explorer for Macintosh. Tantek lives in San Francisco, and has bachelor's and master's degrees in Computer Science from Stanford University, as well as a strong background in human interface and user-centered design from his many years at Apple Inc. He shares his thoughts at tantek.com.

IMAGE CREDITS

John Allsopp johnfallsopp.com

Dmitry Baranovskiy dmitry.baranovskiy.com

Dan Cederholm simplebits.com

Ian Hickson ln.hixie.ch

Ryanne Hodson ryanishungry.com

Jeremy Keith adactio.com

Emily Lewis ablognotlimited.com

Henri Sivonen hsivonen.iki.fi

Neal Stephenson nealstephenson.com

Brian Suda suda.co.uk

Edward Tufte edwardtufte.com

Ben Ward benward.me

Apple apple.com

Google google.com

microformats microformats.org

Mozilla Foundation mozilla.org

Opera Software opera.com

WHATWG whatwg.org

The WebM Project webmproject.org

World Web Web Consortium w3.org

TABLE OF CONTENTS

01	INTRODUCTION	6
02	BACKGROUND: WHERE DID HTML5 COME FROM?	7
03	HTML5 OVERVIEW	11
04	HTML5 BASICS	12
05	HTML5 TRANSITION	15
06	NOTABLE CHANGES TO HTML4 FEATURES	22
07	HTML5 FLEXIBILITY, UNIVERSALITY, AND CONSISTENCY	33
08	ADOPTED FROM XHTML 1.1: RUBY	38
09	CHECKPOINT: VALIDATING HTML5	40
10	NEW HTML5 SEMANTICS	44
11	HTML5 NATIVE VECTOR GRAPHICS	59
12	HTML5 NATIVE AUDIO AND VIDEO	67
13	NEW HTML5 USER INTERFACE ELEMENTS	75
14	THE HTML5 BLEEDING EDGE	88
15	CHECKPOINT: REVALIDATE	91
16	CONCLUSION	93

01

INTRODUCTION

"In times of profound change, the learners inherit the earth, while the learned find themselves beautifully equipped to deal with a world that no longer exists."

—Eric Hoffer

We are in the midst of the biggest change in web development since Cascading Style Sheets (CSS) cleansed our content of presentational markup and inspired us to think outside table layout boxes. Do you work on the web? HTML5 is for you.

We live in the era of the World Wide Web, a globally shared collection of hyper-text and interactive building blocks that have multiplied access to communication and expression more than anything in history, perhaps even more than the printing press.

After the twin failed revolutions of draconian purity and invisible machine-centric semantics, the web's building blocks of interlinked expression and content are finally taking an evolutionary leap forward along paths well worn by the web's working class: authors, designers, and developers, just like you.

HTML5 and related technologies are upgrading the potential of the web with

- simplification, cleanup, and fixes to HTML4 and XHTML
- richer semantic markup
- new forms capabilities
- native multimedia
- programmable vector graphics
- powerful, bleeding-edge APIs

All of this is just potential. For this potential to be realized, it needs people like you, learning and applying these new and evolving building blocks in your practical day-to-day creative acts on the web.

The browsers are implementing these improvements at their fastest rate in 10 years. Huge amounts of HTML5 are ready and reliably usable today. On the



other hand, we must keep in mind that HTML5 is a work in progress — it still has parts that are in flux, have issues, or are downright broken.

HTML5 Now provides you with HTML5 examples, tutorials, and explanations you can start using today along with details about what you can start enhancing, what to be careful with, and what you can start experimenting with to get a head start on future possibilities.

Are you ready? Let's get started.

02 BACKGROUND: WHERE DID HTML5 COME FROM?

Tim Berners-Lee first documented, proposed, implemented, and wrote [HTML in the early 1990s](#). From there, the evolution of HTML slowly accelerated for nearly a decade.

FROM EVOLUTION TO REVOLUTION AND BACK

The delicate and sometimes contentious dance between browser manufacturers and architects of the web helped HTML evolve to version 2, then take the optimistic sidestep of HTML3, which was quickly reality-checked by HTML 3.2. The language finally evolved into HTML4 and capped off with HTML 4.01.

In the trailing years of the web's first decade, the World Wide Web Consortium (W3C) pursued the revolution of [XML](#), with its promise of bringing cleaner, limitless markup, and enabling everyone to make up their own tags for even richer expression. However, while XML did liberate many corporations from proprietary formats and protocols on their intranets, the envisioned XML-based World Wide Web never materialized.

The web is a global communications medium for sharing information and understanding. It turns out that "making up your own tags" leads to Babel — confusion

caused by different languages. To communicate, clearly we need shared vocabularies; on the scale of the web, shared vocabularies are even more essential.

Fortunately, back in 2000 a small group of idealistic pragmatists at W3C combined the immensely successful shared vocabulary of HTML with the clean redesign of XML and produced [XHTML](#), an XML-compatible version of HTML.

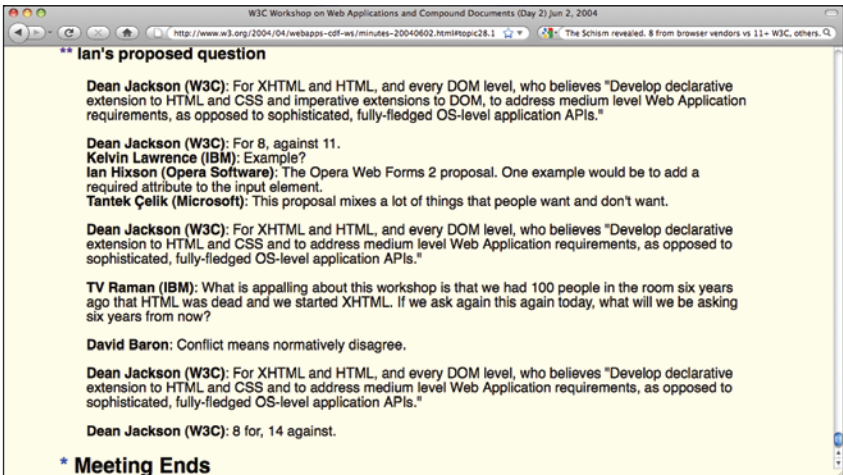
In the early 2000s, the modern web design community adopted XHTML 1.0 in ways compatible with the existing web. Leaders in the community unified and rallied designers and developers around their best practices of standards-based design, using CSS to separate presentation from markup, and maximizing the use of rich semantic markup.

Modern web development leapt forward and brought us fluid and flexible user experiences across a plethora of media and devices. The web design community rediscovered semantic (X)HTML and pushed it to its limits.

THE “GREAT WEB SCHISM” OF 2004

In 2004, the W3C held a two day workshop in San Jose, California on Web Applications and Compound Documents. The conclusion was nothing short of a schism between browser makers and the W3C regarding approaches to web standards. A poll taken at the end of the workshop showed that browser makers unanimously favored an incremental approach based on evolving HTML4/XHTML1 + CSS + DOM, in contrast to the more dominant position led by W3C staff (and a few minor vendors) advocating replacing that stack with non-backward-compatible XHTML2 + XForms + SVG + MathML + RDFa.

Like many critical moments in history, the significance of this poll was not immediately apparent, and has only in recent years come to be known as the “Great Web Semantic(s) Schism” or the “Great Web Standards Schism” of June 2004. That moment inspired the creation of not one, but two mutually complementary efforts — outside the W3C — that drove large evolutionary changes in the web in the latter half of the 2000s. These are changes we’re still in the middle of.

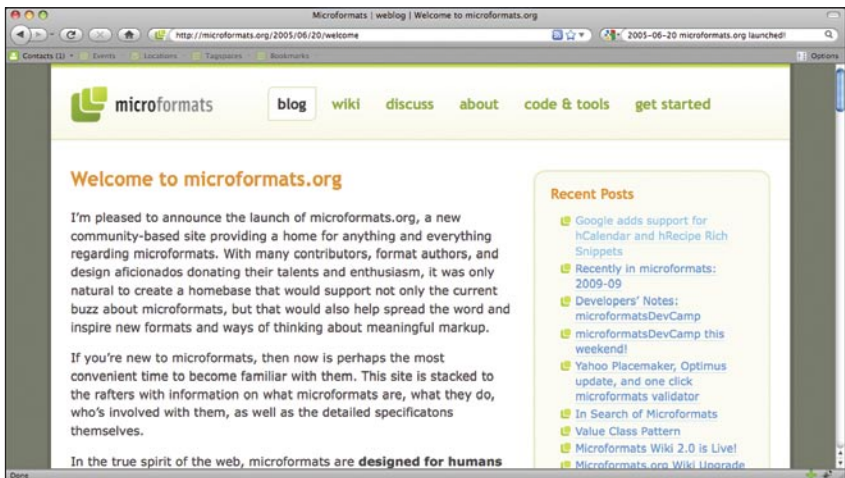


The [minutes from the end of the workshop](#) document the poll that revealed the stark split in opinions.

MICROFORMATS AND THE WHATWG

The concept of microformats — extending web semantics using existing valid semantic HTML4 and XHTML1 (primarily with HTML4's rel attribute, e.g., XFN and rel-license) — would leap forward in just three months following that fateful workshop. The leap came from the introduction of hCalendar for representing events on the web, and hCard for representing people and organizations, subsequently birthing another new standards community in 2005: microformats.org.

Given their obvious unanimity in opinion, in 2004 the browser makers went on to form The Web Hypertext Application Technologies Working Group (WHATWG) to pursue the evolution of HTML itself and new APIs for web applications. (See the sidebar for information about one browser maker that chose not to join the working group.) A bit more than a year later they published the first specifications for "Web Applications 1.0" as well as an update of "Web Forms 2.0." Meanwhile, W3C largely ignored both efforts and continued to pour time and effort into XHTML 2.0 as the future of HTML.



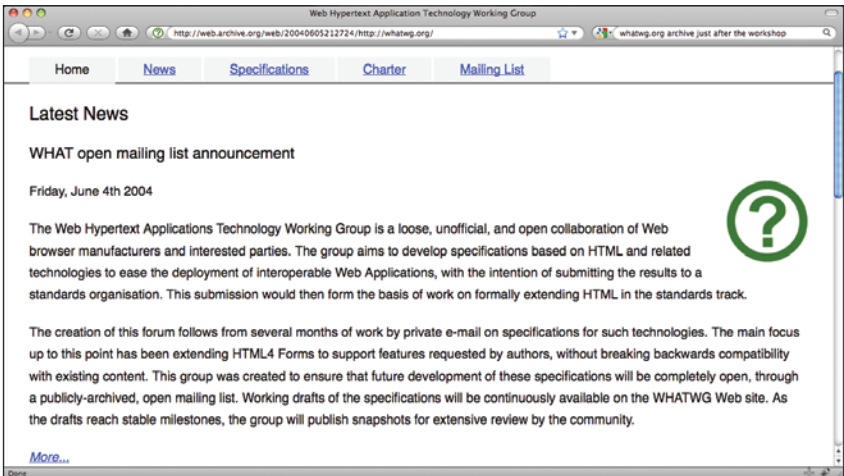
The microformats.org launch and first blog post.

MICROSOFT'S NEUTRAL STANCE

Microsoft, whose delegates agreed with and voted in the poll alongside the other browser makers, chose to stay more neutral, agreeing with them in principle yet declining to join the WHATWG while simultaneously quitting the W3C's XHTML 2.0 effort.

Microformats.org grew as an open independent standards community, producing additional simple formats for marking up reviews (hReview), tags (rel-tag), episodic content (hAtom), products (hProduct), listings (hListing), and even recipes (hRecipe) — nearly all of which are now supported by popular search engines such as Google.

Browsers began to implement portions of Web Apps 1.0 and Web Forms 2.0, which merged and were eventually renamed to HTML5. In 2007 the W3C finally started a new HTML working group to adopt and develop HTML5 in cooperation with the WHATWG, publishing the first W3C HTML5 draft in 2008. In 2009, the W3C finally closed the XHTML2 working group and stopped work on XHTML2 to focus on HTML5.



The whatwg.org open mailing list announcement.

To the benefit of all of us who work on the web, the world of web standards had finally begun to re-converge.

03 HTML5 OVERVIEW

Eager to get started with HTML5? In this tutorial, you'll learn

- how to create your first HTML5 page
- key transitions from HTML4 and XHTML1
- how to validate HTML5
- new semantics
- multimedia: audio, video, and canvas
- new forms features
- bleeding-edge APIs for web applications

ADOPTION STRATEGY

There's a lot to HTML5, from simple to powerful, from reliable to experimental.

To learn HTML5 and become productive with it as quickly as possible, keep the following adoption strategy in mind while you go through each section.

- Start with the basics.
- Update your HTML4 and XHTML.
- Add new HTML5 features incrementally, as needed.

With that in mind, onto the basics and creating your first HTML5 page.

04 HTML5 BASICS

Before we dive into some code examples, let's go over some typographical conventions used in this tutorial.

HTML5 NOW TEXT CONVENTIONS

In this reference guide, we'll be using the following conventions for old code vs. new code:

CONVENTION	DESCRIPTION
monospace	Code in general is shown in monospace type.
<i>bold gray monospace</i>	Old or obsolete code is bold, gray, italicized.
<u>bold orange monospace</u>	Transitional code is orange, bold, dotted-underlined.
bold fuchsia monospace	New and recommended code is fuchsia and bold.
blue text	Hyperlinked text.

Now that you know how to read the code, let's jump straight into using it!



NEW DOCTYPE

Currently you may be using HTML4 or XHTML1 DOCTYPEs like these in your web pages:

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

```
<!DOCTYPE html PUBLIC
  "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

HTML5 has drastically shortened the DOCTYPE:

```
<!DOCTYPE html>
```

That's it — just 15 characters. Short enough to type from memory. What about the rest of that stuff in the previous DOCTYPEs?

Turns out it doesn't matter anymore. Browsers don't do anything with it except for "DOCTYPE switching" between standards and legacy modes. The new HTML5 DOCTYPE puts existing browsers in standards mode, and since we're all using standards mode now, that's all we need.

SIMPLER CHARACTER SET CODE

On the web, people express themselves in numerous languages, with potentially thousands of different characters, glyphs, and symbols. In the past, web developers from different countries had to use different character sets to encode their content for the web.

Unicode has solved that problem, and UTF-8 is the simplest and most backward-compatible encoding for Unicode. Web servers can be configured to send the character set of a web page along with the "text/xhtml" content type with the following HTTP header:

```
Content-Type: text/html; charset=UTF-8
```

However, if you, like most web developers, use a text editor to edit your HTML, you need to indicate in the HTML file itself what its character set is. Before HTML5, here's how you would do so with a `<meta http-equiv>` tag that would emulate that HTTP header:

```
<meta  
  http-equiv="content-type" content="text/html;  
  charset=utf-8">
```

In another nice feat of simplification, HTML5 has shortened this essential meta tag down to 22 characters.

```
<meta charset="utf-8">
```

The XML-compatible version is down to 23 characters:

```
<meta charset="utf-8"/>
```

HTML5 EXERCISE

Here's a simple exercise. Type the following into a text editor from sight, 10 times in a row:

```
<!DOCTYPE html>  
<meta charset="utf-8">
```

When you're finished, try opening a brand new blank document, and typing in that same text from memory. Let your muscle memory take over, and you'll likely find that you've typed in the start of an HTML5 document.

Congratulations! You've started an HTML5 document off the top of your head. Now let's finish it.

YOUR FIRST HTML5 DOCUMENT

You only need two more things to complete your first HTML5 document. First, add a `<title>` element just as you would in previous versions of HTML:

```
<title>Hello</title>
```

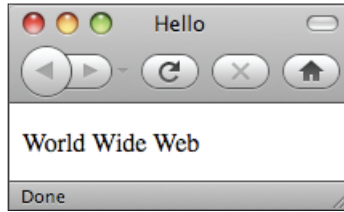


Second, add just a bit of content, perhaps with a paragraph tag:

```
<p>  
World Wide Web  
</p>
```

All put together:

```
<!DOCTYPE html>  
<meta charset="utf-8">  
<title>Hello</title>  
<p>  
World Wide Web  
</p>
```



What it
looks like in
a browser

But where are the `html`, `head`, and `body` tags? It can't possibly be that simple can it? How can you have an HTML document without an `<html>` tag? Or a `<head>` or `<body>`?

In short, you didn't need those tags in HTML4 either — the `head` and `body` tags, both opening and closing, were all optional. They're optional in HTML5 as well.

But what about in XHTML? We'll get to that in the next section.

05 HTML5 TRANSITION

Now that you know how to create an HTML5 document from scratch, let's talk about how to transition your existing HTML4 and XHTML1 documents to HTML5.

REMOVE PRESENTATIONAL MARKUP

If you're a modern web designer who has been keeping up to date, who lives and breathes clean semantic markup, you can almost certainly skip this section.

None of this should apply to you. For everyone else, yes, including those of you still using “Transitional” DOCTYPEs, time is up. If you want to embrace HTML5, you must remove all presentational markup.

Rather than go into details, here’s a simple summary of what to remove, and what you can use instead.

OBSOLETE PRESENTATIONAL MARKUP	CSS REPLACEMENT
<code><basefont></code> <code><big></code> <code></code> <code><tt></code>	font properties
<code><s></code> <code><strike></code> <code><u></code>	text-decoration
<code><center></code> and <code>align=center</code>	text-align:center and margin:auto
<code>align=left</code> , <code>right</code> , or <code>justify</code> on <code><div></code> and other text elements	text-align:left, right, justify respectively
<code>align=left</code> or <code>align=right</code> on <code></code> and other replaced elements	float:left and float:right respectively
<code><body text link alink vlink></code>	color property and :link, :visited, :active pseudo-classes
<code><body background></code>	background-image
<code>bgcolor</code> attribute	background-color
<code>border</code> attribute and <code><table frame rules></code>	border properties
<code><table cellspacing></code>	border-spacing
<code><table cellpadding></code>	padding on the table cells (<code><th></code> and <code><td></code>) themselves
<code><br clear></code>	clear
<code>hspace</code> , <code>vspace</code> , <code>marginheight</code> , <code>marginwidth</code>	margin properties
<code><hr noshade size></code>	border-style:solid and border-width
<code>nowrap</code> attribute	white-space:nowrap
<code>valign</code> attribute	vertical-align
<code>height</code> and <code>width</code> attributes	height and width properties
<code><plaintext></code>	No CSS for this. Use <code><pre></code> or serve that content as text/plain, not HTML



Now would be a good time to take a big fat red marker and write “OBSOLETE” on any presentational markup examples in web design books you have from the 1990s and early 2000s. Or, just toss those books into the recycling bin.

REMOVE FAILED FEATURES

HTML5 has gone beyond purging presentational markup and has removed failed features as well. Thus, you must remove these tags from your HTML as well.

FAILED HTML4 FEATURE	USE THIS INSTEAD
<code></code>	use a visible description or link to one
<code><frameset></code> <code><frame></code>	redesign your content, with <code><iframe></code> if necessary
<code><html version></code>	nothing, just drop the version attribute.
<code><meta scheme></code>	avoid all invisible metadata, add microformats to visible content instead
<code>rev</code> attribute	use rel microformats instead

Very few sites and developers use these features in practice these days. However, if your site happens to, or if you’re maintaining someone else’s old code, now you know to drop them.

JUST FOR COMPATIBILITY

A couple HTML4 attributes have been kept around only for compatibility reasons. You should drop these if you can. However, if your site needs to support the older browsers that need these features, then you may use them as follows in HTML5.

The `border` attribute on the `` tag is perhaps the *only* remaining presentational attribute in HTML5. The only reason it’s there is to undo a default presentational annoyance from certain older browsers (they put a blue border on images inside hyperlinks).

Thus HTML5 permits `border="0"` (no other values) on the `` tag (no other tags) for this purpose:

```
<a href="..."><img border="0"></a>
```

Personally, I think the border attribute is unnecessary and a bit of CSS like this is sufficient:

```
:link img, :visited img { border:0; }
```

However, apparently enough of a case was made to keep that bit of transitional code in HTML5.

The second bit of compatibility markup has to do with scripting. In HTML5, the `type` attribute is no longer necessary on the `<script>` tag. It's 2010 and on the web, JavaScript has won, so there's no need to state the obvious. However, some older browsers may require that you specify the language explicitly. For that case, and that case only, HTML5 permits `type="text/javascript"` on the `<script>` tag:

```
<script type="text/javascript">
```

In this case, not only is the explicit `type` attribute harmless, but it does express a semantic (in contrast to the `border="0"` above). As such, I think it is reasonable to include it.

TRANSITIONING YOUR XHTML

Lots of web designers and developers adopted XHTML 1.0 in the early 2000s along with their transition away from presentational markup to semantic markup. You may wish to maintain XHTML compatibility while upgrading to HTML5, or perhaps you wish to author HTML5 documents that are also well-formed XML. While XML in web browsers never took off, many still use XML-based tools on their web server to process their (X)HTML content. Some developers go the extra mile to serve HTML that is also valid XML to browsers so that their web pages themselves can function as APIs for their site.

Whatever your reason for doing so, the point is that HTML5 permits you to continue to maintain XHTML compatibility. This does require you to be a bit more careful with your code. Here are a few simple rules that will take care of most of the work for you. Since these rules help keep your code clean and consistent in general, it's not a bad idea to keep them in mind even if you don't care about XHTML or XML.



1. SELF-CLOSE EMPTY HTML4 TAGS

A handful of elements in HTML4 never have close tags. In order for them to be properly parsed by XML processors, you need to use XML self-closing syntax.

In short, put a slash (/) just inside the closing angle bracket (>) on these tags:

```
<br/>
<hr/>
<img/>
<input/>
<link/>
<meta/>
<option/>
```

2. ALWAYS USE QUOTED ATTRIBUTE VALUES

XHTML required all attribute values to be quoted, and in HTML5, you can continue to do so. Not only is there no harm in doing so, but quoting attribute values is a good habit that will help you avoid inadvertent errors, such as attribute values that contain spaces or other punctuation. The before-and-after example below demonstrates the use of quoted attribute values.

Before:

```
<img src=photo1.jpg alt=photograph>
```

After:

```

```

3. USE EXPLICIT TAGS FOR A CONSISTENT DOM

In the previous section we constructed a simple HTML5 document:

```
<!DOCTYPE html>
<meta charset="utf-8">
<title>Hello</title>
<p>
World Wide Web
</p>
```

And we noted that it lacked `<html>`, `<head>`, and `<body>` tags — because it doesn't actually need them since they're implied.

In XML (and thus XHTML), however, there are no implied tags. Therefore, you need to write them out yourself. We're going to self-close the meta tag here as well:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>Hello</title>
</head>
<body>
<p>
World Wide Web
</p>
</body>
</html>
```

This is particularly essential for consistent DOM and CSS treatment of the document, whether the document is parsed as HTML or XHTML.

Creating a table is another common situation in which tags are implied in HTML (and therefore you must add them explicitly for XHTML/XML compatibility). HTML permits you to create simple tables with just rows and cells:

```
<table>
  <tr>
    <td> row 1 cell 1 </td> <td> row 1 cell 2 </td>
  </tr>
  <tr>
    <td> row 2 cell 1 </td> <td> row 2 cell 2 </td>
  </tr>
</table>
```

And that implies a `<tbody>` around all the rows. For XHTML/XML compatibility, you must explicitly include a `<tbody>`. Just add the tag along with the opening and closing tags of every `<table>`:



```
<table><tbody>
  <tr>
    <td> row 1 cell 1 </td> <td> row 1 cell 2 </td>
  </tr>
  <tr>
    <td> row 2 cell 1 </td> <td> row 2 cell 2 </td>
  </tr>
</tbody></table>
```

UPDATE OBSOLETE MARKUP

Now that you've removed any presentational markup and failed features, as well as cleaned things up and made all your tags explicit for XHTML/XML, there may be a few remaining bits of obsolete markup in need of transition.

HTML5 has finally done away with a few more deprecated HTML4 tags and attributes that don't fall into any of the above categories. If your code uses of any of these, be sure to use the equivalent instead.

REMOVED HTML4 FEATURE	HTML5 REPLACEMENT
<code><acronym></code>	<code><abbr></code>
<code><applet></code>	<code><object></code>
<code><dir></code>	<code></code>
<code></code>	<code><div id="a1"> ... </div></code>
<code></code>	<code></code>

`<acronym>` to `<abbr>` and `<dir>` to `` are both simple direct conversions; `<applet>` to `<object>` involves a bit more work, and the specific techniques for doing so are lengthy enough that I won't explain them here; I encourage you to consult online resources

Named anchors are finally recognized as unnecessary. Browsers have supported the `id` attribute on all elements for more than 10 years, so you can always use an `id` on a relevant container element instead of an empty anorexic anchor. Similarly, there is no need to use the `name` attribute on images as the `id` attribute works fine for that, too.

06 NOTABLE CHANGES TO HTML4 FEATURES

Several HTML4 features have been refined in HTML5, several former presentational tags received new semantic lives, and some black sheep markup has been welcomed into the fold.

SEMANTIC REFINEMENTS TO HTML4 ELEMENTS

First, let's take a look at what has been improved through refinement.

NESTABLE PHASIS ELEMENTS

The `` element can be nested to indicate even stronger levels of emphasis. In HTML4, you used the `` element to indicate stronger emphasis, but it only provided two levels of emphasis. Now, you simply use another `` element to indicate stronger emphasis and yet another nested `` to indicate even stronger emphasis.

The following examples help you visualize the changes and see how old markup converts to new.

OLD HTML4 SEMANTIC MARKUP	NOW IN HTML5
<code>...</code>	<code>...</code>
<code> </code>	<code> </code>

Some web designers may have been depending on the default styling of `` and ``. As a modern designer, I'm sure you always explicitly style your emphasis.

Either way, when you update your uses of `` to nested `` elements, you will likely want to rewrite your CSS. Since nested `` elements look no different by default, here are a few sets of CSS rules that you can use and adjust as needed for your designs.



```
/* ruleset 1: immitate typical old em/strong styling */
em          { font-style:italic }
em em      { font-weight:bold }
/* end of ruleset */

/* ruleset 2: nested emphasis alternate italics */
em          { font-style:italic }
em em      { font-style:normal }
em em em   { font-style:italic }
em em em em { font-style:normal }
em em em em em { font-style:italic }
/* etc. as needed. end of ruleset */

/* ruleset 3: nested emphasis italic, bold, both, alternate ital-
ics */
em          { font-style:italic }
em em      { font-style:normal; font-weight:bold }
em em em   { font-style:italic }
em em em em { font-style:normal }
em em em em em { font-style:italic }
/* etc. as needed. end of ruleset */
```

Does this mean the `` element has been forgotten? Not forgotten, but repurposed.

THE IMPORTANCE OF ``

The `` element has been redefined to mean “importance” rather than “strong emphasis.” This may sound like semantic nitpicking—in practice important things are usually strongly emphasized, and if something is strongly emphasized it’s typically important. Ironically, that’s exactly why HTML5 is able to redefine `` and still have it “work” with a lot of existing content.

Take a look your existing uses of ``. Did you happen to also markup something that was important? If so, good—no code changes are needed. If not, reflect on the precise semantic you were expressing. If you were expressing strong emphasis as defined in HTML4, go ahead and make the changes noted above to nested `` elements. If you intended some other semantic, you might

need to use a plain `` and a semantic class name that you can then style accordingly.

Just like the new `` element, the new `` element may be nested, and used to indicate increasing levels of importance.

SEMANTIC RECASTING OF HTML4 ELEMENTS

A small handful of presentational HTML4 elements have been recast with semantic meaning in HTML5.

<SMALL> DISCLAIMERS, CAVEATS, AND COPYRIGHTS

The `<small>` element is now used to express a set of semantics present on nearly every web page: little bits of legalese or other items that publishers attempt to explicitly de-emphasize. Things such as

- small print legalese
- disclaimers
- caveats
- copyright statements

You know the kind of text I'm talking about — and all of it is typically styled with a smaller font size. But that's just a visual distinction. If the text is read out loud (like in a TV commercial), it might be spoken quickly in a quieter, monotonic voice.

When you're updating pages, at a minimum wrap copyright statements in the new semantic `<small>` element. As you use it more and more, you'll develop a good intuitive feel for when it applies.

<I>INSTANCES OF IDIOMS AND TAXONOMIC TERMS

The formerly presentational `<i>` for the italics element has been redefined to semantically express any one of the following:

- idioms
- terms from taxonomies (such as names of species)



- technical terms
- ship names
- other obscure semantics you likely haven't used

In essence, the new `<i>` element represents a set of disparate text semantics, all of which were typically styled with italics per common style guidelines. Is such a semantically vague recasting actually worth much? It's not clear at this time. The new `<i>` element may be so ambiguous as to not amount anything more than a heap of *E. coli*. We'll have to wait and see how — and if — authors decide to make much semantic use of the new `<i>` element.

OLD LEADS, KEYWORDS, PRODUCTS, AND OTHER STYLISTIC OFFSETS

The `` element has always meant bold, until now.

Similar to the reverse semantic derivation of the new `<i>` element, the new `` element now represents one or more of the following:

- keyword
- product name
- lead sentence or paragraph
- other text that is stylistically offset for some semantic reason

While at first this set of semantics seems arbitrary, these uses are similar and fairly frequent in practice. In particular, the last use case — that of stylistically offset text — serves to cover the other more specific cases and to illustrate why the new `` makes sense semantically.

By indicating that a portion of text is stylistically offset rather than just bold, browsers have just enough of a hook to know to present the element differently from surrounding text in other mediums as well. When a new `` element is spoken by a screen reader, for example, it might use a different voice to provide a stylistically offset experience for the listener.

In addition to the cases of keywords, product names, and lead sentences/paragraphs — typically styled in bold — I've found a few more use cases where it

makes sense to use the new ``, such as using a spot **color** in text or in a title (as in the title of this work).

**
EAKING LINES FOR FUN AND POETRY**

The `
` element bears a mixed history at best. Introduced in the context of presentational markup, as a no-margin alternative to `<p>` tags, it's largely been used as such. During the rediscovery of semantic HTML4 in the early 2000s, a few creative web designers looked past `br`'s presentational roots and found semantic uses — in particular for poetry and addresses, where placing text on different lines actually expresses different meaning. HTML5 has codified this in-the-wild semantic adoption of the `
` element.

With the introduction and broad adoption of microformats, especially for poetry and addresses, it turns out we no longer need `
` elements to express separation between the different components of an address. For example, addresses might previously have been marked up like:

```
<div class="address">
US Library of Congress<br>
101 Independence Ave. SE<br>
Washington, DC 20540-0002<br>
United States of America
</div>
```

With microformats, in particular hCard and adr, we now mark them up as:

```
<div class="vcard">
  <div class="fn org">US Library of Congress</div>
  <div class="adr">
    <div class="street-address">101 Independence Ave. SE</div>
    <span class="locality">Washington</span>,
    <span class="region">DC</span>
    <span class="postal-code">20540-0002</span>
    <div class="country-name">United States of America</div>
  </div>
</div>
```



This not only expresses the separation between the address components, but also indicates that the address belongs to a specific organization.

That leaves lines of poetry as the primary remaining semantic use for the new `
` element.

<HR>HORIZONTAL RULES AND THEMATIC BREAKS

The last presentational HTML4 element that HTML5 has reclaimed for semantic purposes is the `<hr>` element. From its very name, the horizontal rule `<hr>` element has conveyed nothing but a visual affordance. However, just as the purely presentational `
` found semantic life separating lines of meaning in poetry and addresses, the `<hr>` element has found meaning in separating paragraphs to indicate a thematic break between them (perhaps when the focus shifts from one subject to another).

Books often use ornate breaks and flourishes between paragraphs to indicate a meaningful shift in context, and that's exactly the semantic the new `<hr>` element is intended to convey.

The particular appearance of an `<hr>` element can be controlled with CSS. If you wish to eliminate the "rule" of the `<hr>` element completely, you may create a style rule such as:

```
hr { border:0 }
```

From there, if you want to replace it with an image, you can give the `<hr>` element width, height, and a background-image:

```
hr {  
    width:100%; height:3em;  
    background:url(http://example.com/flourish.jpg) no-repeat;  
}
```

Note that since the image is purely decorative, and the `<hr>` itself conveys the thematic break semantic — which is all that screen-readers should need to provide an appropriate interface/experience — there is no need for alternative text for the image.

HTML BLACK SHEEP ACKNOWLEDGED

Several elements and attributes, both in HTML4 and never before a part of any W3C specification, have been properly recognized and incorporated into HTML5.

THE IFRAME ELEMENT

Omitted from the HTML 4.01 Strict DTD, the `<iframe>` element was long regarded as unnecessary, easily replaced with an `<object type="text/html">`. It turns out that embedding one HTML document inside another is a sufficiently special case that it's worth keeping, and HTML5 recognizes it as such.

Its presentational attributes such as *frameborder*, *marginheight*, *marginwidth* have been dropped, deferring to CSS instead. One new presentational attribute has been introduced: the seamless attribute

You can use the **seamless** attribute to undo any default presentation that browsers typically place around iframes. Whether borders, scroll bars, or spacing, the **seamless** attribute tells the browser to include the nested HTML from the iframe in a manner that makes it appear to be part of the document. Inside an `<iframe>` element you may include content for browsers that do not support iframes. Here is a simple example of both:

```
<iframe src="embedded.html" seamless="seamless">
  Your browser doesn't support iframes, otherwise you would see
  embedded.html
</iframe>
```

Currently, the `<iframe>` element supports only text fallback content for non-iframe supporting browsers. This is a shortcoming, as typically you would want to at least link to the iframe content. For example:

```
<iframe src="embedded.html" seamless="seamless">
  Your browser doesn't support iframes,
  <a href="embedded.html">view the embedded HTML content</a>
</iframe>
```

However, HTML5 does not allow this at this time. If you want to provide fallback content with markup, you must use the `<object>` element instead.



The `<i>iframe</i>` element has also been recently enhanced with sandboxing capabilities, potentially useful for when you want to embed an external (and perhaps not trustworthy) chunk of HTML. These capabilities are still quite new, so I recommend staying apprised of them but not using them yet.

In general, if you want to use an `iframe` for embedding HTML content, there's almost always a better solution that integrates the content directly into the document. Despite its newly upgraded status in HTML5, I recommend avoiding the use of `iframes` whenever you can.

THE EMBED ELEMENT

For years browsers have supported the embedding of plug-ins such as Flash and QuickTime via the `<embed>` element, and yet it never made it into any version of any W3C specification — until now.

HTML5 recognizes that browsers already support `<embed>` — and a good portion of the web uses and depends on it for interactive games, advertisements, streaming video, and the like. Here is a simple example from the HTML5 specification:

```
<embed src="catgame.swf">
```

Numerous lengthy texts are written about how to use the `<embed>` element for specific plug-ins such as Flash and QuickTime, so I will not replicate that here. Suffice it to say, if you're a web designer who uses plug-ins in your documents, you now can do so with the `<embed>` element and at least have a chance at validating your documents as HTML5.

THE TARGET ATTRIBUTE

HTML4 deprecated the `target` attribute, presuming it to be either presentational, or exclusively for framesets, which were also deprecated. However, since HTML5 recognizes the `<i>iframe</i>` element, it only made sense to recognize the `target` attribute as well, so that hyperlinks could alter the content of `iframes`.

Thus the `target` attribute is now valid on `a`, `area`, and `base`:

- `` - the hyperlink targets the `iframe` or window named "t1".
- `<area target="t1">` - the `imagemap` area link targets "t1".
- `<base target="t1">` - all links target "t1" by default.

THE LISTS: EMPTY, NUMBERED, AND REVERSED

The last set of black sheep markup that HTML5 has recognized has to do with lists. Perhaps due to its static document focus, the HTML4 specification forbade empty lists; both `` and `` elements were required to have at least one `` inside. In today's world of web applications, it makes sense to allow empty lists that are later filled in with scripts. Therefore, HTML5 now permits them.

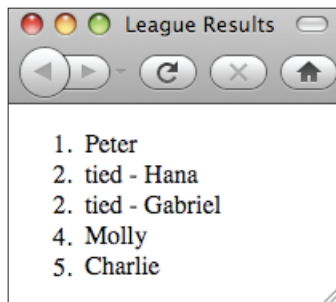
```
<ol>
</ol>
<ul>
</ul>
```

Two key list numbering attributes were mistakenly deprecated in HTML4. HTML5 finally acknowledges that explicitly numbering both specific items, and whole ordered lists, can communicate underlying semantics, such as rankings with ties and paginated ordered results.

Use the **value** attribute on a list item to give it a particular number:

```
<!DOCTYPE html>
<meta charset="utf-8">
<title>League Results</title>

<ol>
  <li value="1">Peter</li>
  <li value="2">tied - Hana</li>
  <li value="2">tied - Gabriel</li>
  <li value="4">Molly</li>
  <li>Charlie</li>
</ol>
```



Results list with a tie for second.

Note that there is no need to explicitly number item 5 because list numbers automatically increment after the last explicitly specified list item value.

For numbered lists that start at a certain number but are otherwise normally numbered, use the **start** attribute on an `` element:



```
<!DOCTYPE html>
<meta charset="utf-8">
<title>League Results p.2</title>

<ol start="6">
  <li>Monica</li>
  <li>Kaito</li>
  <li>Eden</li>
  <li>Micah</li>
  <li>Angela</li>
</ol>
```



The second page of an ordered list that uses the `start` attribute to continue the numbering.

When using the `start` attribute, the entire list is numbered automatically, and there is no need to explicitly number each individual list item.

Lastly, there's one new HTML5 feature that affects list numbering, and that's the `reversed` attribute, which makes ordered lists count down rather than up.

However, as of this writing, two big problems exist with the `reversed` attribute:

1. No browser supports the `reversed` attribute as of this writing.
2. No declarative backward-compatible method exists to use it. Even if some browsers supported it, you would need custom JavaScript to make it work in other browsers.

Thus, for now and in the immediate future, you can safely ignore the `reversed` attribute.

QUESTIONABLE HTML5 RESTRICTION: THE `<CITE>` ELEMENT

One of the few questionable changes in HTML5 is its semantic restriction of the `<cite>` element. HTML4 encourages us to markup works and names of speakers (when there isn't a specific work) with the `<cite>` element. Examples from the HTML 4.01 specification include:

```
<cite>Harry S. Truman</cite> said,
<q lang="en-us">The buck stops here.</q>
More information can be found in <cite>[ISO-8601]</cite>
```

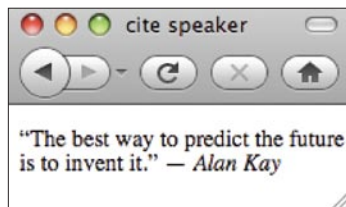

HTML5, on the other hand, only permits the “title of a work” to be marked up with the `<cite>` element.

Given the lengths to which HTML5 has gone to add broad semantics to elements such as `` and `<i>`, and the fact that `<cite>` is currently used by numerous web designers to semantically markup names related to quotes, this cite restriction seems quite odd, and frankly a mistake, perhaps based on bad assumptions, stubbornness, or both.

What do to? Well, the HTML5 Validator (which we’ll get to soon) doesn’t complain about use of the cite element for people’s names as it can’t tell the difference. Thus, I’m encouraging a bit of civil disobedience in this case.

When you’re marking up the name of a speaker, in relation to a quote, or a blog commenter next to their comment, use the `<cite>` element. Be sure to mark it up with the hCard microformat to indicate the more precise semantic (that it is the name of a person):

```
<!DOCTYPE html><meta charset="utf-8">
<title>cite speaker</title>
<p>
  <q cite="#ak">
    The best way to predict the
    future is to invent it.</q>
  - <cite class="vcard" id="ak">
      <span class="fn">
        Alan Kay
      </span>
    </cite>
</p>
```



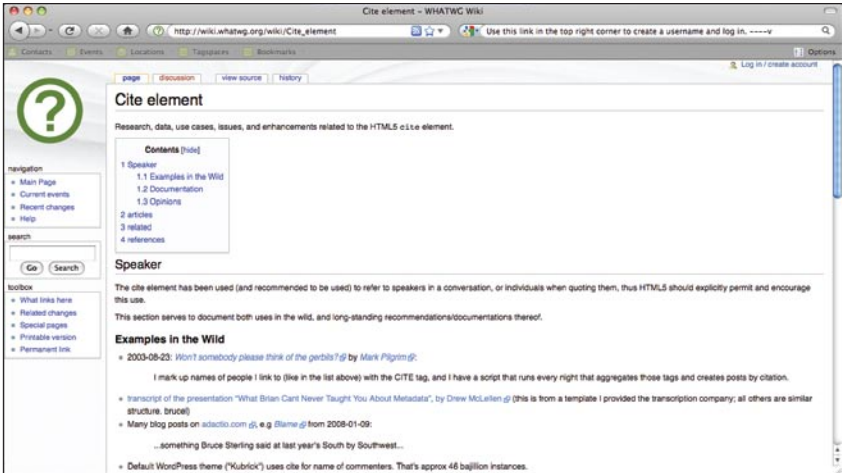
A quote with the name of the speaker marked up with the cite element.

Numerous web designers and developers have spoken out about this apparent mistake in HTML5, and documented millions of instances on the web where the `<cite>` element is used to semantically markup names of speakers/commentors on the WHATWG wiki.

I encourage you to create an account on the WHATWG wiki and edit the Cite_ element page: http://wiki.whatwg.org/wiki/Cite_element. Add your opinions



and links to real-world cases where you used the `<cite>` element to markup names of speakers, and there's a good chance you can help get the spec fixed.



The WHATWG wiki's Cite element page.

Fortunately for us, there are far more areas that the spec has fixed (rather than broken) things from HTML4. That leads us to the next section, where we'll explore a few bits of markup that HTML5 has made more flexible, universal, and consistent.

07 HTML5 FLEXIBILITY, UNIVERSALITY, AND CONSISTENCY

In this chapter we cover the last of the major changes/fixes from HTML4 to HTML5 — and the best part is, all of them make HTML, as a whole, just a bit easier to learn and use.

FLEXIBILITY: HYPERLINKS AND BLOCKS

HTML4 (especially the strict version) has a few seemingly arbitrary rules about what elements can reside inside what other elements. It's hard to keep track of them all, and when you forget, it usually means errors from the validator, some headscratching, and some reorganizing of your markup. One of the most frequent headscratchers is the requirement that hyperlinks contain only "inline" content, that is, HTML4 forbade you from linking entire headings, paragraphs, blockquotes, etc.

HTML5 now lets you do so, because browsers have supported this for years. For example, bloggers typically link their post headlines to their permalinks, and over time most figured out how to put the hyperlinks inside the headings to make them validate.

```
<h3>
  <a rel="bookmark" href="http://example.com/blog/3">Less is
more</a>
</h3>
```

With HTML5, you now have the flexibility of wrapping your links around the heading:

```
<a rel="bookmark" href="http://example.com/blog/3">
  <h3>Less is more</h3>
</a>
```

In addition to fewer odd and arbitrary validation errors, sometimes this will save you some markup.

Often, headings have an associated image that also links to the relevant post or article. These are typically adjacent in the markup — but not contained, as the image isn't semantically part of the heading. Previously, you had to create two hyperlinks, one for the image and one for the heading.

```
<a rel="bookmark" href="http://example.com/blog/4">
  
</a>
<h3>
  <a rel="bookmark" href="http://example.com/blog/4">Moving
forward</a>
</h3>
```



In HTML5, you can now use one hyperlink for both, simplifying your markup, eliminating a potential source of data drift errors, and helping with accessibility by getting rid of the adjacent duplicate hyperlink (especially surrounding an image without alt text).

```
<In the following code block, the highlight code is bold fuchsia>  
<a rel="bookmark" href="http://example.com/blog/4">  
    
<h3>Moving forward</h3>  
</a>
```

Other uses for hyperlinks around blocks exist as well, such as linking an entire blockquote (with perhaps several paragraphs) back to the original source. As more web designers take advantage of this new flexibility, I'm sure we'll see more interesting and creative uses of hyperlinks around blocks.

UNIVERSALITY: GLOBAL ATTRIBUTES

HTML5 finally allows the following attributes on all elements:

- **class**
- **dir**
- **id**
- **lang**
- **style**
- **tabindex**
- **title**

These attributes always *seemed* global — but weren't until now. Previously in HTML4, a few elements here and there didn't allow some or all of these attributes, and whenever you happened upon one of those cases, you'd get an odd validation error.

The new global **class** attribute in particular enables you to mark up an entire page as being a specific microformat, such as an hCard:

```
<!DOCTYPE html>  
<html class="vcard">
```

```

<meta charset="utf-8">
<title class="fn">Tantek Çelik</title>
<p>
  Hi, I'm Tantek and this is my simple home page at
  <a class="url" href="http://tantek.com/">tantek.com</a>.
  You can also find me on Twitter as
  <a rel="me" class="url" href="http://twitter.com/t">t</a>
<p>
</html>

```

Or perhaps an hAtom blog post entry page:

```

<production note: spaces instead of tabs OK?>
<!DOCTYPE html>
<html class="hentry">
<meta charset="utf-8">
<title class="entry-title">Less is more</title>
<p class="entry-content">
  It seems nearly every blogger is
  posting less often on their own blog
  and more often on Twitter.
</p>
<div class="post-info"> written by
  <address class="author vcard">
    <a class="fn url" href="http://tantek.com/">
      Tantek Çelik
    </a>
  </address>
  on
  <span class="published">
    2010-06-09
  </span>
</div>
</html>

```

Or any other microformat that represents the entire page. See microformats.org for a list of microformats.



CONSISTENCY: MEDIA, HREFLANG, REL FOR ALL LINKS

HTML4 introduced the ability to specify which types of media that `<link>` elements applied to — for example, you could indicate a print style sheet:

```
<link media="print" rel="stylesheet" href="default.css">
```

But you could only specify the media on `<link>` elements, not other types of links such as `<a>` and `<area>`.

HTML5 fixes this inconsistency, and adds the `media` attribute to the `<a>` and `<area>` elements.

Nearly every press site links to alternate print versions of their articles, and they can now do so semantically:

```
<a media="print" rel="alternate" href="article-print.html">  
  print version  
</a>
```

Mobile web usage is growing faster than desktop web usage, and as such, while some sites create mobile style sheets or use CSS Media Queries that adjust their content layout for smaller screens, many sites create whole separate mobile sites. They, too, can now link to them semantically:

```
<a media="handheld" rel="alternate" href="http://m.example.com/">  
  mobile site  
</a>
```

Doing so empowers browsers, mobile devices, etc., to automatically detect such alternate versions. So, perhaps when users choose to print a page, you offer to let them print the print version; on a mobile device, give them the option to visit the mobile site instead.

In HTML4, the `<area>` element was even more inconsistent than the `<a>` element as it lacked the `hreflang` and `rel` attributes. HTML5 fixes this, so you may now use the following attributes on all `<a>`, `<area>`, and `<link>` elements:

- **media** (to indicate applicable media)
- **hreflang** (language of the destination)
- **rel** (relation of the destination to the source)

All the changes we've discussed in this chapter — the new flexibility afforded to hyperlinks, the new universally global attributes, and consistent link attributes — reduce the number of rules we have to keep in mind and make using HTML that much simpler. Each HTML5 simplification makes it easier to learn as well, making it more accessible, lowering barriers, and enabling even more people to express themselves on the web.

We've covered a lot about all the cleanup that HTML5 brings us. It's time to start looking at some of the new features, the first of which has been adopted from XHTML 1.1: Ruby.

08

ADOPTED FROM XHTML 1.1: RUBY

Only one major new feature has survived from the otherwise evolutionary dead-end of XHTML 1.1, and that is the `<ruby>` element — along with its children, `<rt>` and `<rp>`. (Note that the `<ruby>` element has nothing to do with the Ruby programming language.) It is the element used for marking up short bits of text with “ruby” annotations, typically used in East Asian typography for pronunciation or other annotations.

Inside a `<ruby>` element, the `<rt>` element is for marking up the **ruby text** annotations themselves, and the `<rp>` element is used to markup the **ruby parentheses** that separate the annotation from the text they are annotating (also called the ruby base).

Here's an example of ruby markup for Japanese with hiragana reading (a Japanese syllabary) and sample rendering:



```
...  
<ruby>  
漢 <rp></rp><rt>かん</rt><rp></rp></rp>  
字 <rp></rp><rt>じ</rt><rp></rp></rp>  
</ruby>  
...
```

かんじ
... 漢字 ...

The two main ideographs, each with its annotation in hiragana rendered in a smaller font above it.

Note that the parentheses in the source markup text will not be drawn in the sample rendering. Browsers that support the `<ruby>` element and place the ruby text near but offset from the ruby base — above for horizontal layout as shown in the example, or to the side in vertical layout — have no need to display the parentheses that separate the ruby text, and thus hide all the `<rp>` elements and their contents. It's still important to include the `<rp>` elements and the parentheses for a decent fallback rendering in browsers that don't support `<ruby>`.

Here's an example of ruby markup for Japanese with hiragana reading and fallback rendering:

```
...  
<ruby>  
漢 <rp></rp><rt>かん</rt><rp></rp></rp>  
字 <rp></rp><rt>じ</rt><rp></rp></rp>  
</ruby>  
...
```

... 漢(かん)字(じ) ...

The two main ideographs, each with its annotation in hiragana rendered inside parentheses immediately following each respective ideograph.

BROWSER SUPPORT

The good news is that the `<ruby>` element is already supported in three major browsers (two rendering engines):

- Internet Explorer 5+ (has supported `<ruby>` for over 10 years!)
- Safari (WebKit)
- Chrome (WebKit)

In addition, experts predict that Firefox (Gecko) will support it soon, and from past experience we can predict that, after that, Opera will likely follow.

In conclusion, go ahead and use the `<ruby>` element — it works in most browsers today and provides a graceful fallback rendering when it doesn't work. Plus, most browsers are likely to support it within the next year.

09

CHECKPOINT: VALIDATING HTML5

You've already learned a lot about HTML5. Whether you've written a new HTML5 document from scratch or made incremental updates to your existing HTML4 or XHTML1 documents, it's time to make sure that your changes are correct.

The W3C Validator at <http://validator.w3.org> has long been an indispensable tool for modern web designers and developers alike. The W3C Validator has been updated to validate HTML5 documents automatically by detecting the HTML5 `<!DOCTYPE html>`. When you run the W3C Validator on your site, if everything is correct, you'll see something like the following results page.

The "Result: Passed, 1 warning(s)" message that displays is not your fault — and it has nothing to do with your HTML5 code. The W3C Validator is merely warning you that the validator *itself* is using an "experimental feature: the HTML5 Conformance Checker."



The screenshot shows the W3C Markup Validation Service interface. At the top, it says "W3C Markup Validation Service" and "Check the markup (HTML, XHTML, ...) of Web documents". Below this, there are navigation links: "Jump To: Notes and Potential Issues" and "Congratulations - Icons". A green banner states "This document was successfully checked as HTML5!". Below the banner, a table displays the validation details:

Result:	Passed, 1 warning(s)
Address:	http://tanteq.com/
Encoding:	utf-8 (detect automatically)
Doctype:	HTML5 (detect automatically)
Root Element:	html

At the bottom, there is a blue "I ♥ VALIDATOR" button and a message: "The W3C validators rely on community support for hosting and development. Donate and help us build better tools for a better web."

W3C Validator results with a warning.

The screenshot shows the "Notes and Potential Issues" section of the W3C Validator. It contains the following text:

The following notes and warnings highlight missing or conflicting information which caused the validator to perform some guesswork prior to validation, or other things affecting the output below. If the guess or fallback is incorrect, it could make validation results entirely incoherent. It is *highly recommended* to check these potential issues, and, if necessary, fix them and re-validate the document.

◆ Using experimental feature: *HTML5 Conformance Checker*.

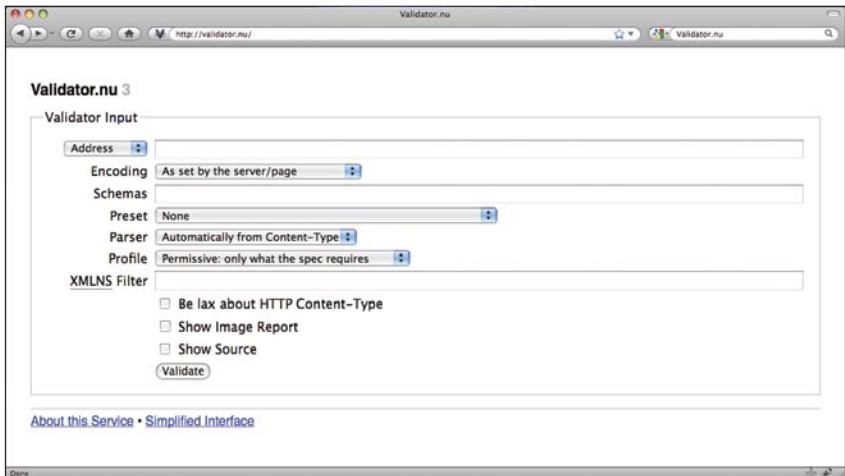
The validator checked your document with an experimental feature: *HTML5 Conformance Checker*. This feature has been made available for your convenience, but be aware that it may be unreliable, or not perfectly up to date with the latest development of some cutting-edge technologies. If you find any issues with this feature, please [report them](#). Thank you.

Congratulations

The document located at `<http://tanteq.com/>` was successfully checked as HTML5. This means that the resource in question identified itself as "HTML5" and that we successfully performed a formal validation using an SGML, HTML5 and/or XML Parser(s) (depending on the markup language used).

W3C Validator issue that triggers the warning.

In addition to the W3C Validator, a new validator dedicated to checking HTML5, called Validator.nu, has a few more options for custom validating your HTML5 documents.



The Validator.nu home page.

I recommend that you continue to use the W3C Validator as part of your web authoring workflow. When your documents validate there, go ahead and check them with Validator.nu as well.

I've created browser buttons (aka bookmarklets or favelets) that enable a one-click validation of your web pages using the W3C Validator or Validator.nu. Check out <http://favelets.com> for the latest.

Drag the links in the right column (the highlighted "V.nu HTML5 Validator" and "W3C HTML Validator") to the links toolbar in your browser. Then, when you view a web page in the browser, you can click the Validator favelet links in your toolbar to validate the page.

While you're at favelets.com, grab the "microformats validator" favelet at the top of the right column. Whether you're using microformats already or you want to try some of the enhanced HTML5+microformats markup we've discussed, it's a good idea to make sure your microformats are working properly.



Favelets

Add any of these Favelets to your Favorites, or drag them to your Favorites Toolbar or Bookmarks Bar. Then load a page, click the favelet in your toolbar and go.

Combine Internet Explorer's Toolbar Favorites feature and mini applications (*applets*) written in Jscript, stamped with a "javascript:" URI scheme, and you have *Favelets* - a way to add features built from DHTML to your browser.

I have found Favelets useful for:

- **Authoring:** Validating the HTML, CSS and HREFs in pages I've written, and testing how my layouts reflow to various screen sizes.
- **Learning:** Viewing the CSS and scripts on a page in addition to the HTML.
- **Enhanced User Interface (UI):** Adding features like a simple Alternate Style Sheets UI.

Questions? [Answers](#).

Updated! Get the new **microformats** favelets that use the H2VX Contacts and Events services.

Authoring

microformats

- microformats validator
- Richsnippets tester

Multivaldator: HTML, CSS & HREFs all in one.

- Multivaldator
- Multivaldator Window

HTML

- **V.nu HTML5 validator**
- **W3C HTML validator**
- WDG HTML validator
- W3C link checker
- application/xml proxy

CSS

- W3C CSS validator
- Toggle CSS style sheets

Screen sizes

- 320x480 (iPhone portrait)

Favelets.com home page with the V.nu HTML5 Validator and W3C HTML Validator highlighted.

The microformats validator favelet uses the Optimus Microformats Transformer, open source code to validate or transform your microformats into other formats — such as JSON, JSON-P, or XML — useful for quickly creating APIs for your web site from your semantic HTML5+microformats markup.

OPTIMUS—MICROFORMATS TRANSFORMER

WHAT IS IT?

Optimus—is a microformats transformer. Easily transform your microformatted content to nice, clean, easily digestible, XML, JSON or JSON-P. You can also easily set filters to only receive particular formats.

Now your web site could really be your **API** with goodness of microformats and power of Optimus.

HOW TO USE IT?

```
http://microformatique.com/optimus/?url=web site URI [&format=Format] [&function=Function] [&filter=Filter]
```

Where:

- web site URI: An address of the page you want to transform
- format: Format should be either "xml", "json" or "rss". Default is "xml".
- function: Callback function for JSON
- filter: filter

TRANSFORM IT NOW

URI:

Format: XML JSON RSS

Callback:

Filter: hcalendar hcard hentry hresume

VALIDATOR

URI:

TWITTER

Best source of updates, so far.

IT IS OPEN

This project is distributed under MIT License.

The Optimus Microformats Transformer home page.

HTML5 validators are useful for finding general markup problems and identifying code that may need to be upgraded to be valid and proper HTML5. We've already covered all the major changes from HTML4 to HTML5. For more detailed changes, take a look at the W3C document outlining the latest HTML5 differences from HTML4 at <http://dev.w3.org/html5/html4-differences>.

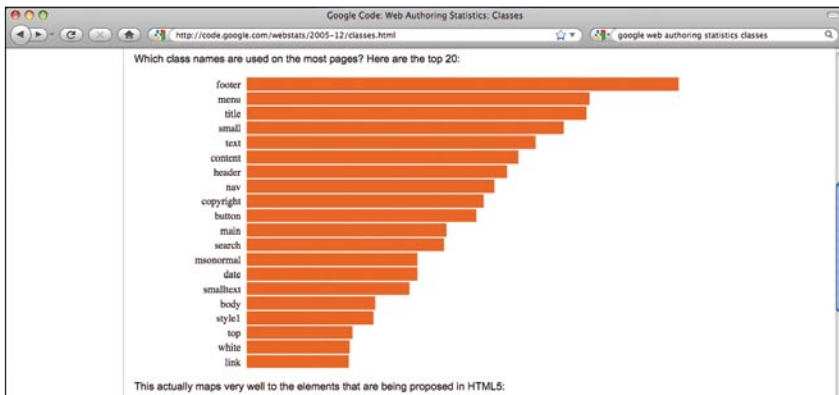
Now that you're all set with validation tools, have checked your HTML5 updates so far, acquired a good reference for changes from HTML4 to HTML5, and fixed any problems in your code, it's time to start learning new HTML5 semantics.

10 NEW HTML5 SEMANTICS

HTML5 is finally upgrading the HTML4 tags with semantics commonly used across the web.

MARKUP STUDIES

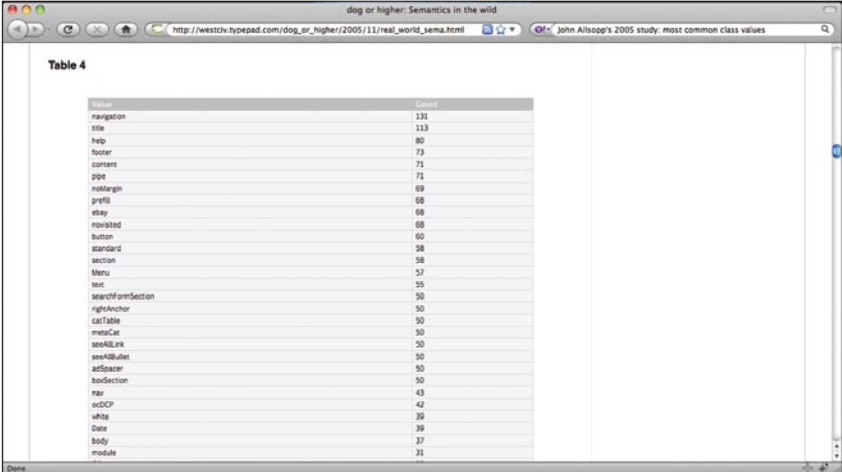
In December 2005, Google published a study of Web Authoring Statistics that documented the [class names used on the most pages](#):



This bar chart shows which class names are used on the most pages.



In many ways, Google’s study was an expansion of earlier studies, such as John Allsopp’s “Semantics in the wild” study: http://westciv.typepad.com/dog_or_higher/2005/11/real_world_sema.html.



Value	Count
navigation	131
top	113
help	80
footer	73
content	71
page	71
noMargin	69
prefill	68
tbody	68
mainText	68
button	60
standard	58
section	58
Menu	57
text	55
searchFormSection	50
rightAnchor	50
catTable	50
metaCat	50
seeAllLink	50
seeAllAlert	50
adBanner	50
boxSection	50
nav	43
ecDCP	42
white	39
Date	39
body	37
module	31

This table displays which class names are used on the most pages.

For the first time in the history of HTML, studies such as these have helped drive the choice of what semantic elements to add — relying on actual data rather than the intuition of experts and standards committee compromises.

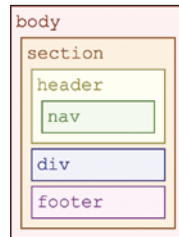
STANDARDIZED PAGE STRUCTURE

Page-level structures consisting of multiple sections — including a header section, navigation tools (usually), a main content area, and a footer section — are perhaps the most common of the widely used semantics. According to the class name studies mentioned previously, many pages even used common terms for these sections. Now, HTML5 provides the corresponding elements.

```

<DOCTYPE html><meta charset="utf-8">
<title>HTML5 structural elements</title>
<body>
  <section>
    <header>
      <nav>
        ...
      </nav>
    </header>
    <div>
      ...
    </div>
    <footer>
      ...
    </footer>
  </section>
</body>

```



This outline demonstrates new HTML5 structural elements.

Strictly speaking, the top-level `<section>` element is unnecessary in the previous document, as the `<body>` element is enhanced by HTML5 to also be a “sectioning” element. A sectioning element sets a scope for various elements such as headings `<h1>` thru `<h6>` and the `<address>` element.

However, it is still good practice to use such a top-level section as it enhances the portability of your markup by turning your page into a building block. Sections can be nested to express the overall structure of more complex pages. Thus, if you always author your pages as sections, you can easily rearrange sections from page-level to nested within one another, perhaps combining multiple sections to create larger compound documents.

The new `<header>` element is for exactly what it sounds like: introductory material, navigation, and headings. Speaking of navigation, if your section has navigation links (which web designers typically organize into unordered lists of hyperlinks), you can simply wrap them in the new `<nav>` element to help identify navigation areas for HTML5 browsers.

Note that there is no new “main” or “content” element that goes between the new `<header>` and `<footer>` elements — that is why the previous example uses



a plain old `<div>` for that purpose. In HTML5, everything between the `<header>` and `<footer>` elements is considered to be the “main” content area of a `<section>`. However, I recommend using an explicit `<div>` for the main/content area with a class name of your choice. At a minimum, use the `<div>` as a styling hook for that chunk of stuff between the header and footer.

This brings us to the new `<footer>` element for completing the section. In the past couple of years, web designers have popularized “fat footers,” which often contain information about the section such as authorship and copyright notices along with additional navigation links. Wherever you include clusters of navigation links, including inside a footer, wrap them in a `<nav>` element.

ARTICLES OF INDEPENDENCE

In addition to the hierarchy of sections with headers and footers, many web pages are themselves self-contained posts, entries, news items, or comments. Semantically, all of these can be considered sections, but their self-standing nature merits a more specific semantic.

HTML5 introduces the `<article>` element, a special type of `<section>` element, to represent potentially independent content components that may be distributed or syndicated as items in feeds. Cases exist where `<article>` elements may be nested. An example in the HTML5 specification uses a page representing a blog post, which would be marked up as an `<article>` with comments on the post represented by nested `<article>` elements:

```
<article>
```

```
  I think HTML5 is pretty neat! -Tantek
```

```
  <article>
```

```
    But how useful is it really? -anonymous
```

```
  </article>
```

```
  <article>
```

```
    You can use HTML5 to express new semantics, provide  
    native multimedia like audio, video, vector graphics,  
    and there's lots of potential for building powerful  
    web applications as well. -Tantek
```

```
  </article>
```

```
</article>
```


This example is greatly simplified to illustrate the basic utility of the new `<article>` element. Any time you mark up a blog post or comments, you should also use the [hAtom microformat](#); be sure to mark up the authors of the post and comments with the [hCard microformat](#) as well.

Finally, note that `<article>` elements are just as portable and modular as `<section>` elements. If you find yourself using a `<section>` element for a piece of content that you would be willing to place on its own page, or syndicate as an independent item in an Atom feed, go ahead and upgrade it to the more semantically specific `<article>` element.

TANGENTIAL ASIDES

All content cannot be considered as “main” content or an article. Quite often, longer articles will contain a few paragraphs of related material that didn’t quite fit or flow in the main text (such as a sidebar to a magazine story). To represent such tangential topics, side comments, or even pull quotes, HTML5 introduces the new `<aside>` element.

`<aside>`

```
<h1>After HTML5?</h1>
<p>What happens <em>after</em> HTML5?
  Will there be an HTML6?
  Or will the web standards community repeat history,
  and fork off an attempt to generalize semantics instead?
</p>
```

`</aside>`

Just like the `<article>` element, the `<aside>` element is a special type of `<section>` element and thus can include its own headings (as the previous example demonstrates) and header and footer sections.

On the web, most tangential content tends to be linked to rather than included with the main content. However, there are certainly online magazines and other such publications that include sidebars and other loosely related content. The `<aside>` element helps to clearly separate such content.



HEADING GROUPS: TWO AT A TIME OR PERHAPS MORE

In print, you can find numerous examples where headings of various levels are clustered together for greater effect. One of my favorite examples is the title of a novel by Neal Stephenson:



Part of the cover of Neal Stephenson's book, *Diamond Age*, showing the title heading and subhead.

The full name of the book is split into a heading and a subhead with obvious stylistic differences. Newspapers and magazines often use such heading/subhead pairings for articles as well (referred to as heds and deks). In HTML5, you can now semantically indicate such pairings (or more) of headings using the new `<hgroup>` element:

```
<hgroup>
  <h1>The Diamond Age</h1>
  <h2>or, A Young Lady's Illustrated Primer</h2>
</hgroup>
```

In the actual cover, notice the break between “The” and “Diamond Age” (never mind the ALL-CAPS effect, which can be achieved with a trivial `text-transform: uppercase`). Your first instinct might be to use a `
` for that line break. However, HTML5 gives us a more semantic way of doing so. *Diamond Age* is clearly stylistically offset (with a larger font size for starters) from the “The,” thus it makes sense to use the new `` element, which we can style as its own block:

```
<hgroup>
  <h1 style="text-transform:uppercase">The
    <b style="display:block">Diamond Age</b>
  </h1>
  <h2>or, A Young Lady's Illustrated Primer</h2>
</hgroup>
```

The remainder of restyling this `<hgroup>` example to recreate the original cover is left as an exercise for the reader.

STYLING NEW SEMANTICS: INTRODUCING BULLETPROOF HTML5

As of this writing, all modern browsers, except Internet Explorer, support styling the new HTML5 semantic elements. Even modern smart-phone browsers, most of which use Webkit (iPhone, iPad touch, Palm Pre, Android), and modern BlackBerry browsers, all support the styling of new HTML5 semantic elements!

However, Internet Explorer versions 8 (IE8) and earlier — as well as much older versions of Firefox, Safari, Opera, and older smart-phone browsers — all simply ignore these new elements, or technically speaking, create empty DOM nodes for them.

Why would they do such a thing?

Because that's what HTML has specified since the first paragraph of the first documentation of HTML (*emphasis* and **stronger emphasis** added).

The WWW system uses marked-up text to represent a hypertext document for transmission [sic] over the network. The hypertext mark-up language is an SGML format. *WWW parsers should ignore tags which they do not understand, and ignore attributes which they do not understand of tags which they do understand.*

— From "[HyperText Mark-up Language](#)," by Tim Berners-Lee, published on the W3C's web site in 1992.

As of HTML5, unknown elements are no longer ignored and are parsed into the hierarchy of a document, just as if they were `div`s or `span`s by other names.



That's all fine when we live in a world where everyone's browser handles elements both known and unknown. However, we want to use new HTML5 semantics now and have them work in new browsers as well as old.

BRINGING BACK THE BULLETPROOFING

In his book *Bulletproof Web Design*, Dan Cederholm introduced the concept of writing HTML/XHTML and CSS that gracefully handles variations in content, text sizes, window sizes, etc. Jeremy Keith expanded the notion of "bulletproofing" to include JavaScript as well in his book *Bulletproof Ajax*.

As professional web workers, we must now also design (and thus bulletproof) for the varying levels (and features) of HTML5 support in browsers, both today and in the near future.

Since unknown or new tags are ignored in IE8 and other older browsers, styling new elements in particular requires a bulletproofing technique that we will likely need for several years, until IE8 fades from view. It starts with a time-honored workaround technique: nesting an extra `<div>` element. This simple example uses the new `<article>` element, but could apply equally to any new HTML5 element.

```
<article>
  <div>
    ...
  </div>
</article>
```

To which we now add a class name the same as the new element:

```
<article>
  <div class="article">
    ...
  </div>
</article>
```

And now, whenever you want to reference an `<article>` element in your style sheet, instead of using "article," use ".article" as follows:

```
.article { margin:3em 0; /* more article styling etc. */ }
```

By styling the `<div>` that is immediately nested inside the `<article>`, you are essentially styling the same element — and doing so in a way that is compatible with existing browsers. HTML5 browsers will gain the semantic benefits of your use of new elements such as `<article>` while your CSS will continue to work in older browsers.

BEWARE OF JAVASCRIPT SHIMS

As some folks have pointed out, it is possible to use a bit of JavaScript with `document.createElement` with the new elements to get them recognized, parsed, and styled by IE8. This technique is too fragile to be reliable and must be avoided for a few reasons. (And if you've never heard of this technique, you may skip this aside.)

Your CSS must not depend on your JavaScript. Your styling should never depend on your scripting, your CSS should stand on its own, and in fact (this is part of the original philosophy in the book *Bulletproof Web Design*), each piece should work as well as it can without the other pieces.

You cannot assume the presence of JavaScript. Up to 10 percent of users browse the web without JavaScript. Either their devices don't support it or they have it turned off for performance or security reasons. Again, another element of bulletproof web design is that your site should work without JavaScript.

There are non-HTML5 browsers for which `document.createElement` does not do the trick. IE8 (and earlier) is not the only non-HTML5 browser out there. Numerous devices, such as older BlackBerries, have their own browsers, which don't yet support HTML5 but do offer very good HTML4 and CSS support. The `document.createElement` trick won't work on them.

To really indicate that tight coupling between the `<article>` element and its styling surrogate `<div>`, place their start/end tags immediately adjacent to each other on the same line.

```
<article><div class="article">  
    ...  
</div></article>
```



Some new elements, such as `<hgroup>`, disallow `<div>` elements as immediate children. For such cases — or, if you, prefer in general — you can place the extra `div` outside the element instead. (The validator will quickly alert you of issues such as this.)

```
<div class="hgroup"><hgroup>
    ...
</hgroup></div>
```

Finally, if you have other classes you want to add to the new element, add them to the extra `<div>` instead. For example, you'll likely be using the hAtom microformat in addition to the `<article>` element. In that case, add the hAtom entry class name `hentry`.

```
<article><div class="article hentry">
    ...
</div></article>
```

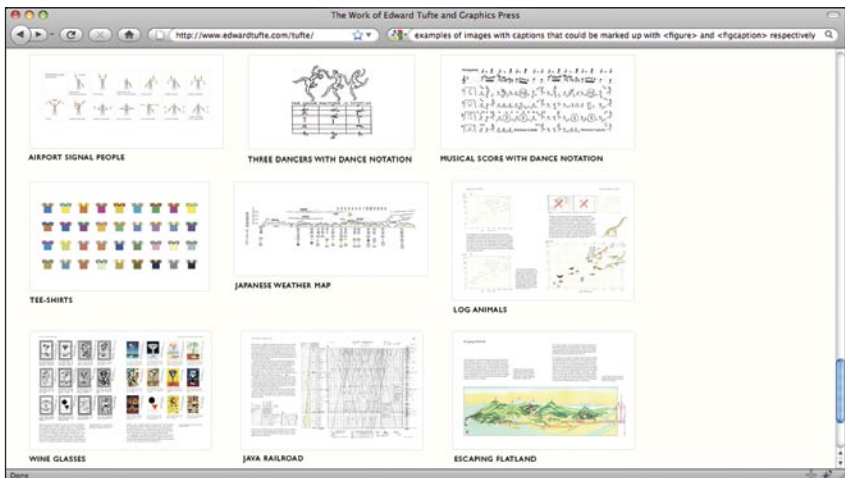
This is just the first of many bulletproof HTML5 techniques that are likely to be developed as HTML5 is adopted and deployed on real-world web sites that must continue working in browsers that lack support either for HTML5 or for specific features.

FIGURES, MARKS, AND DATES

The last set of new HTML5 semantic elements are the `<figure>`, `<mark>`, and `<time>` elements.

PICTURES ARE WORTH A FEW ASSOCIATED WORDS

Another very common semantic that web authors have expressed for years is the association between images and their captions. Some particularly beautiful examples are on Edward Tufte's home page:



Part of [Edward Tufte's home page](http://www.edwardtufte.com/tufte/) showing a series of figures with associated captions.

HTML5 introduces the `<figure>` and `<figcaption>` elements to explicitly semantically connect images (or other types of figures) with their captions. They're fairly easy to use: simply wrap the caption for an image with the `<figcaption>` element and then put a `<figure>` element around both the image and the `<figcaption>`.

`<figure>`

```

```

```
<figcaption>Figure 1</figcaption>
```

`</figure>`

One warning, though: The `<figure>` element has undergone quite a few changes in the HTML5 specification. Recently, the decision made to wrap the caption text in the `<figcaption>` element. Previously, attempts were made to reuse the `<caption>`, `<legend>`, and even `<dt>` and `<dd>` elements — all of which turned out to have too much legacy baggage in current browser implementations to be usable inside a new element.



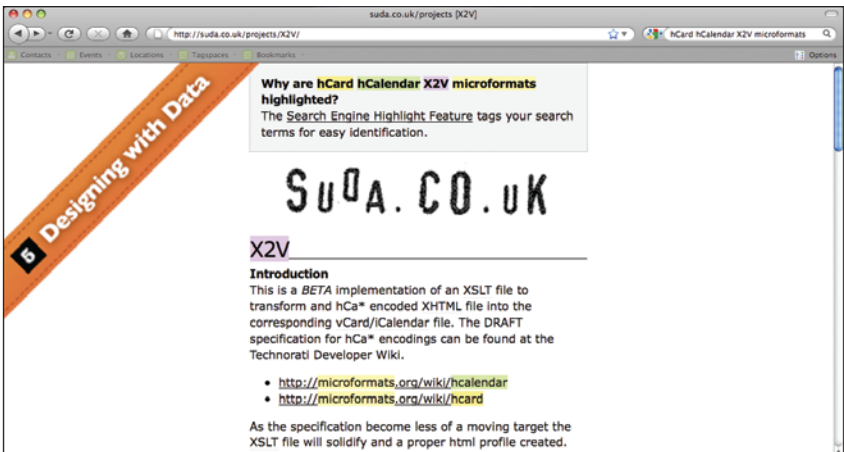
The `<figure>` and `<figcaption>` elements may still be subject to change — at least more so than the other new semantic elements. Thus, it may be worth waiting until it is more clear that they have stabilized.

MARKED TEXT

HTML5 introduces the `<mark>` element to represent text that is highlighted for some reason outside its current context or by someone other than the author. A few cases where you might do this include:

- Highlighting portions of a quote to draw attention to it but not imply emphasis
- Pointing out errors, perhaps in source code
- Arriving at a page via a search engine then highlighting the terms that brought the user to the page

The following example shows a page found through a web search, where instances of the search terms are highlighted:



Brian Suda's X2V project page with highlight on the search terms used to find the page.

The highlights at the top of the page might be marked up with the `<mark>` element:


```
<p>Why are  
  <mark class="t1">hCard</mark>  
  <mark class="t2">hCalendar</mark>  
  <mark class="t3">X2V</mark>  
  <mark class="t4">microformats</mark>  
highlighted?  
</p>
```

Note the use of class names `t1` `t2` `t3` `t4` to indicate the different search terms, which can then be styled with different background color highlights to help distinguish them on the page.

DATES AND TIMES

Time and date information is published all over the web, perhaps most often in event listings and date/timestamps of articles and blog posts.

You can use the new `<time>` element to mark up either 24-hour time, a date, or date and time (with optional time zone).

```
<p>  
  When I said <time>13:37</time>,  
  I meant on <time>2010-03-03</time>,  
  as in <time>2010-03-03T13:37</time>,  
  in particular <time>2010-03-03T13:37-0800</time>.  
</p>
```

The first two uses of the `<time>` element are quite readable and understandable because more people worldwide understand 24-hour time and ISO dates than any other single (and often culture-specific) date or time format. The latter two examples use the ISO8601 datetime format (the last with time zone offset) and are not very human friendly (except to programmers who deal with dates/times on a daily basis).

To address this problem, the `<time>` element has a `datetime` attribute where you can specify the machine readable ISO8601 date or datetime, while placing a more human-friendly equivalent in the contents of the element.



We can update the previous example with `datetime` attribute for the latter two instances:

```
<p>
  When I said <time>13:37</time>,
  I meant on <time>2010-03-03</time>,
  as in <time datetime="2010-03-03T13:37">1:37pm on March 3rd,
  2010</time>,
  in particular <time datetime="2010-03-03T13:37-0800">1:37pm
  PST</time>.
</p>
```

We've seen several examples where the `<time>` element works well to represent date and time information. Several instances of dates and times, however, don't work with the `<time>` element.

The `<time>` element does not work for

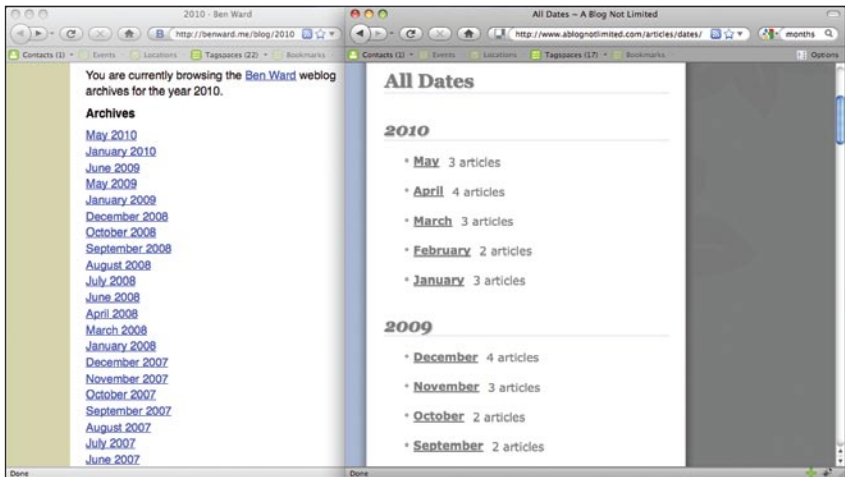
- imprecise times, such as "sometime this afternoon"
- imprecise dates, such as "the most recent Ice Age"
- dates before the introduction of the Gregorian calendar
- dates in countries before they adopted the Gregorian calendar

Almost all of these examples are related to *historical* dates and times, which often have additional semantics or information associated with them — such as different possible dates, sources of evidence for the dates, etc. The `<time>` element does not attempt to solve the larger problem of how to represent historical dates, and it frankly doesn't make sense to use it for those cases.

Unfortunately, the `<time>` element currently has several seemingly artificial limitations that could be fixed. The `<time>` element does not currently allow usage in the following real-world web situations:

- whole years, such as blog archives, years of birth
- years and months, such as blog archives once again
- month and day of a year, such as birthdays given without a year
- week of a year, not commonly seen on the web

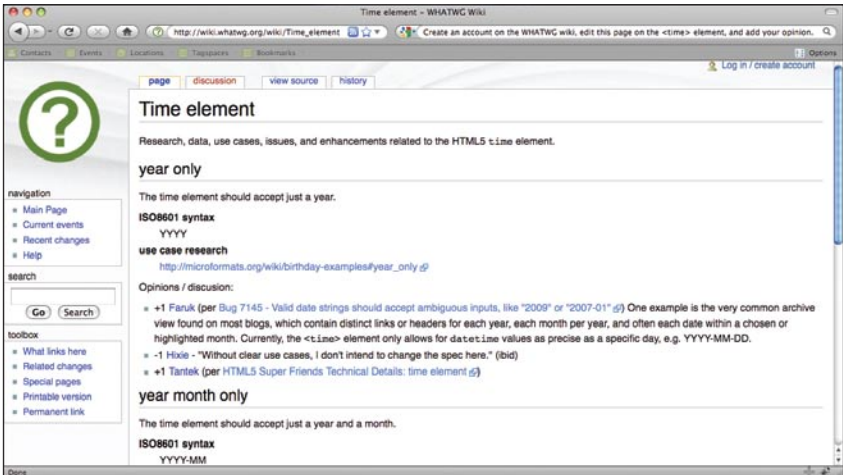
The seemingly obvious and widespread use case of links to blog archives should justify the first two cases mentioned, which have corresponding ISO8601 date formats: YYYY and YYYY-MM.



Two blog archives with year and month archive links, and just month archive links with the year implied.

Most social networks and online profiles permit the display of only the month and day of a person's birthday without the precise year. This, too, has a standard ISO8601 date representation: --MM-DD. I've documented the use cases and numerous examples in the wild for all three of these additional uses on the WHATWG wiki: http://wiki.whatwg.org/wiki/Time_element.

As HTML5 is still a working draft, this is one limitation/problem that we may be able to fix. If you agree with the additional use cases outlined here, I encourage you to visit the WHATWG wiki Time element page, create an account, and add your use cases and opinions as well.



The WHATWG wiki page documenting additional use cases for the time element.

11

HTML5 NATIVE VECTOR GRAPHICS

The introduction of inline image support in the Mosaic web browser in the mid-1990s breathed life into otherwise staid pages of hypertext. Yet other than a few changes in bitmap formats, from GIF to JPEG and PNG, very few truly revolutionary advances have occurred in web graphics. In recent years, SVG (W3C's Structured Vector Graphics format) has slowly been gaining adoption — for example, for external images in Wikipedia pages.

HTML5 takes a major step forward with vector graphics support by incorporating two approaches:

- Inline SVG: the ability to place SVG markup directly into your HTML
- The `<canvas>` element: a graphics API for JavaScript

INLINE SVG

HTML5 defines how to natively embed and parse `<svg>` elements and their children. For example:

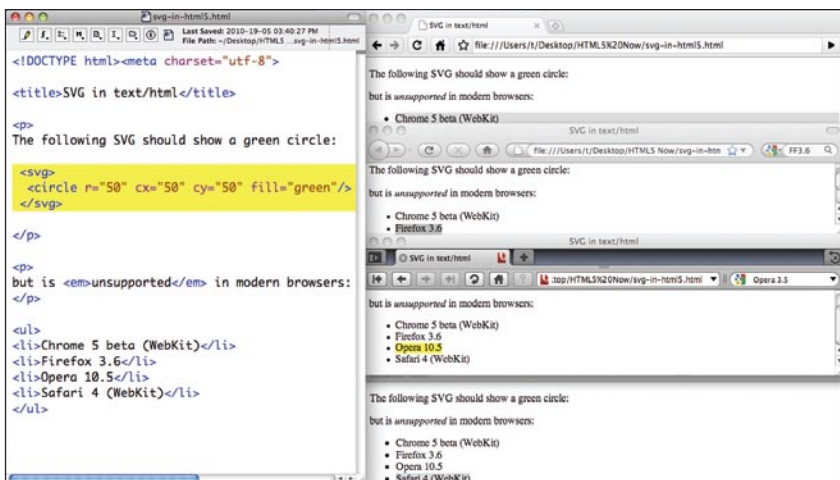
```
<p>
```

The following SVG should show a green circle:

```
<svg>
  <circle r="50" cx="50" cy="50" fill="green"/>
</svg>
```

```
</p>
```

However, as of this writing, no browser supports this.

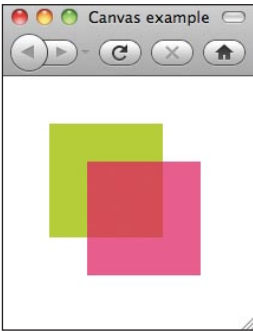


Note that HTML5 has a provision for inline MathML that's not supported by any of today's HTML5 browsers either. This provision will be very useful for scientists and mathematicians when browsers do support it. However, just like inline SVG, it's not something we can use on the web today.

Maybe someday inline SVG will work. Until then, I recommend only using SVG as Wikipedia does — for external decorative images that are better represented by vector graphics than bitmapped images.



CANVAS



HTML5 introduces the new `<canvas>` element for drawing arbitrary vector graphics using JavaScript code in contrast to SVG's approach of using XML markup. Here's a simple example.

```
<!DOCTYPE html><meta charset="utf-8">
<title>Canvas example</title>
<script>
function draw()
{
    var canvas = document.getElementById("c1");
    if (canvas.getContext) {
        var c = canvas.getContext("2d");

        c.fillStyle = "rgb(171,213,16)";
        c.fillRect(32, 32, 96, 96);

        c.fillStyle = "rgba(255,0,102,0.75)";
        c.fillRect(64, 64, 96, 96);
    }
}
</script>
<body onload="draw();">
    <canvas id="c1" width="128" height="128">
        two overlapping squares.
    </canvas>
</body>
```

Let's break this example down step-by-step. First, we'll start with the `<canvas>` element itself:

```
<canvas id="c1" width="128" height="128">  
two overlapping squares.  
</canvas>
```

In this example, notice three important aspects of the `<canvas>` element:

- An `id` attribute, which provides a simple hook for JavaScript code to refer to it
- The `width` and `height` attributes that indicate the dimensions of the canvas's drawing area
- Text fallback content inside for nonvisual browsers, and visual browsers that either have scripting disabled or do not support the `<canvas>` element

Next let's take a look at the `<body>` element surrounding the canvas:

```
<body onload="draw();">  
  <canvas id="c1" width="128" height="128">  
    two overlapping squares.  
  </canvas>  
</body>
```

The only purpose the `<body>` element serves in this example is to call the JavaScript function `draw()` after the document has finished loading.

EMPLOYING UNOBTUSIVE JAVASCRIPT

Note that for the simplicity's sake, the canvas example uses the inline event handler attribute `onload`. In practice, however, you should assign all event handlers dynamically, using the modern web developer practice of unobtrusive JavaScript. For more on unobtrusive JavaScript, see the "Behavioral Separation" article at A List Apart and the book *DOM Scripting*, both by Jeremy Keith:

<http://www.alistapart.com/articles/behavioralseparation>

<http://domscripting.com>



Finally we get to where the action happens: the JavaScript that draws on the canvas:

```
<script>
function draw()
{
    var canvas = document.getElementById("c1");
    if (canvas.getContext) {
        var c = canvas.getContext("2d");

        c.fillStyle = "rgb(171,213,16)";
        c.fillRect(32, 32, 96, 96);

        c.fillStyle = "rgba(255,0,102,0.75)";
        c.fillRect(64, 64, 96, 96);
    }
}
</script>
```

Let's break down the `draw()` function even further:

```
var canvas = document.getElementById("c1");
```

Here's where the `id` attribute on the `<canvas>` element comes in handy: The first thing the script does is get a reference to the canvas.

```
if (canvas.getContext) {
    var c = canvas.getContext("2d");
```

The `if` test for `canvas.getContext` is effectively checking to see if the browser supports the `<canvas>` element. If so, next it retrieves the canvas's two-dimensional ("`2d`") drawing context. Which is the object that responds to JavaScript drawing commands?

```
c.fillStyle = "rgb(171,213,16)";
c.fillRect(32, 32, 96, 96);
```

The 2d drawing context offers many commands, two of which set the `fillStyle` and draw a rectangle with the `fillRect` function. You can set the `fillStyle`

with an = assignment statement — in this case, to a solid green color using the `rgb()` syntax same as in CSS.

The `fillRect()` function takes four parameters, in order: the left, top, width, and height of a rectangle. Combined, these two statements draw a solid green square:

```
c.fillStyle = "rgba(255,0,102,0.75)";  
c.fillRect(64, 64, 96, 96);
```

These next two lines of code do almost exactly the same thing except:

- The `rgba()` function sets a partially transparent (75% opaque) pink `fillStyle`. The syntax for `rgba()` is reused from CSS color (<http://w3.org/TR/css3-color>).
- The `fillRect()` is nearly identical, with its top left offset down and to the right.

Altogether, the example draws a solid green square, and then a slightly transparent pink square on top of it.

One of the best ways to learn is the time-honored web development practice of:

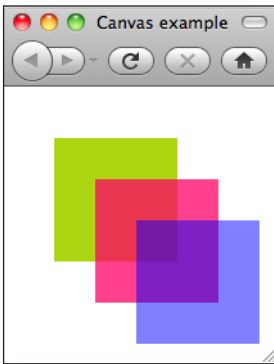
1. View source
2. Copy/paste
3. Make some changes
4. See what happened
5. Repeat steps 3-4 as necessary to understand how the code works

Let's copy/paste the `fillStyle/fillRect` lines of code to create another square of a different color:

```
<script>  
function draw()  
{  
  var canvas = document.getElementById("c1");  
  if (canvas.getContext) {  
    var c = canvas.getContext("2d");  
  
    c.fillStyle = "rgb(171,213,16)";  
    c.fillRect(32, 32, 96, 96);  
  }  
}
```



```
c.fillStyle = "rgba(255,0,102,0.75)";  
c.fillRect(64, 64, 96, 96);  
  
c.fillStyle = "rgba(0,0,255,0.5)";  
c.fillRect(96, 96, 96, 96);  
}  
}  
</script>
```



The new code we added sets the `fillStyle` to a 50% transparent blue, and then draws another square 32 pixels down and to the right of the second square.

But we're not quite finished. One more bit to update remains: the fallback text.

```
<canvas id="c1" width="128" height="128">  
  three overlapping squares.  
</canvas>
```

Our complete updated example:

```
<!DOCTYPE html><meta charset="utf-8">  
<title>Canvas example</title>  
<script>  
function draw()  
{  
  var canvas = document.getElementById("c1");  
  if (canvas.getContext) {
```

```
var c = canvas.getContext("2d");

c.fillStyle = "rgb(171,213,16)";
c.fillRect(32, 32, 96, 96);

c.fillStyle = "rgba(255,0,102,0.75)";
c.fillRect(64, 64, 96, 96);

c.fillStyle = "rgba(0,0,255,0.5)";
c.fillRect(96, 96, 96, 96);
}
}
</script>
<body onload="draw();">
  <canvas id="c1" width="128" height="128">
    three overlapping squares.
  </canvas>
</body>
```

Numerous JavaScript drawing commands exist for the `<canvas>` element to perform all the usual graphics primitives, drawing and/or filling lines, paths, circles, arbitrary polygons, etc. To learn more I recommend the Mozilla Developer Center online canvas tutorial: https://developer.mozilla.org/en/Canvas_tutorial.

BROWSER SUPPORT

As with any graphical element, you must make sure to provide good fallback content for browsers that don't support `<canvas>`, for text browsers, for accessibility, and of course, for search engines.

I recommend that you limit your use of `<canvas>` on production web sites to decorative images only. This way, you can provide reasonably equivalent fallback content — for example by nesting an `` element with `alt` text.

```
<canvas id="c1" width="128" height="128">
  
</canvas>
```



Finally, do go ahead and experiment with the potential of `<canvas>`, building more complex interactive pages that push the limits of what's possible. My one request is that while you're designing such rich interactive experiences, keep in mind — and simultaneously develop — similar interactive functionality with semantic markup and forms as well.

Most of the beautiful canvas experiments to date have not bothered with any kind of fallback or accessibility, and so consider this your opportunity to create both beautiful and accessible experiences, because it is possible, especially when you do so from the start. These new vector graphics capabilities in HTML5 are just one part of its new multimedia features, which also include native audio and video support.

12

HTML5 NATIVE AUDIO AND VIDEO

With the exception of low-quality animated GIFs and the proprietary `<bgsound>` tag, HTML's native multimedia support has been limited to static images. Nearly all of the audio and video on the web has been handled by plug-ins, typically Flash or QuickTime, until now. HTML5 introduces the `<audio>` and `<video>` elements, bringing rich declarative sound and motion to our otherwise static text and graphics.

AUDIO AND VIDEO BASICS

In their simplest forms, both new tags work like the trusty `` tag, with a `src` attribute for the URL of the audio or video file respectively — except that both new tags have end tags as well.

```
<audio src="chess.wav"></audio>  
<video src="chess.avi"></video>
```

Where the `` tag has an `alt` attribute for a simple text alternate, the `<audio>` and `<video>` elements are containers, thus wrapping their fallback alternate content.

```
<audio src="chess.wav">  
  Joshua asks in a synthetic voice:  
  <samp>HOW ABOUT A NICE GAME OF CHESS?</samp>  
</audio>
```

```
<video src="chess.avi">  
  Video of Joshua simultaneously displaying on a terminal  
  while asking in a synthetic voice:  
  <samp>HOW ABOUT A NICE GAME OF CHESS?</samp>  
</video>
```

One advantage of fallback content inside an element is that you're able to use additional markup, while attributes such as `alt` are limited to plain text content. Note that the examples above use the `<samp>` rather than `<q>` tags as Joshua is a computer program.

SUPPORTING MULTIPLE FORMATS

The biggest challenge facing both `<audio>` and `<video>` elements is that there is currently no single format (either audio or video) that is natively supported across all browsers.

To demonstrate this, we're going to code another example, this time using some freely available open source/community video from the Internet Archive (archive.org). To get started, download the following video files:

- <http://www.archive.org/download/Ryanne-BarCampSF816/Ryanne-BarCampSF816.ogv>
- http://www.archive.org/download/Ryanne-BarCampSF816/Ryanne-BarCampSF816_512kb.mp4

Rename your local copies to just `barcampsf.ogv` and `barcampsf.mp4`, respectively. Doing so will keep our examples shorter/simpler.



Note: As the earlier examples demonstrate, audio and video examples are nearly identical. In the remaining examples, we're going to focus on video — although they all apply to audio with just a change in formats. Where there are differences in browser support between H.264 (typically .mp4 files) and Ogg Theora (.ogv files) formats in video, there are essentially the same differences in browser support between MP3 (.mp3) and Ogg Vorbis (.ogv) in audio.

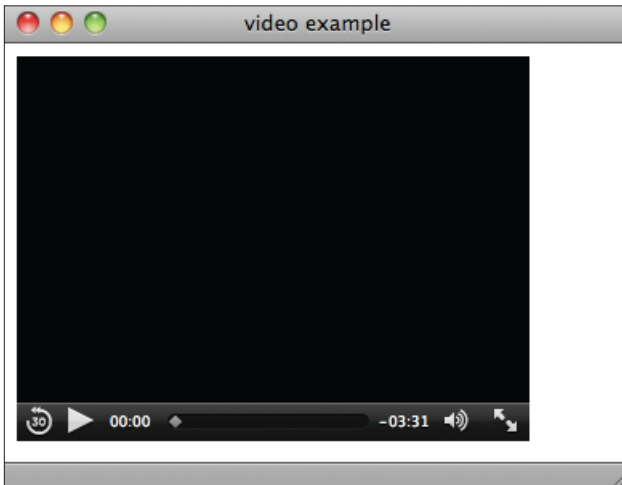
Now back to a simple video example using these new files.

```
<!DOCTYPE html><meta charset="utf-8">
<title>video example</title>
<video src="barcampsf.mp4">
  A video showing BarCampSanFrancisco.
</video>
```

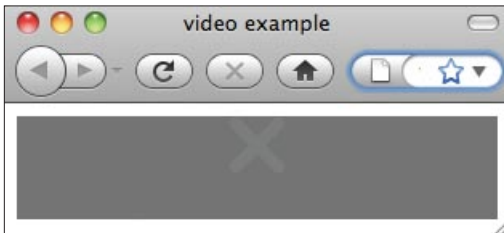
By default no controls are shown for video or audio. The expectation is that you, the web designer, will build your own controls from HTML elements styled with CSS, and wire them up to your audio and video elements with JavaScript. The full DOM APIs for controlling the `<audio>` and `<video>` elements are listed in the HTML5 specification itself (and potentially in a bit of flux). We're going to use the `controls` attribute to instruct the browser to give us a default set of `controls` so that we can focus on the content itself.

```
<!DOCTYPE html><meta charset="utf-8">
<title>video example</title>
<video src="barcampsf.mp4" controls>
  A video showing BarCampSanFrancisco.
</video>
```

If you post this example to your web site along with the video file, you can then view it using Safari, Chrome, or an iPad, as all of them use WebKit, which supports HTML5 video and also supports the H.264 format. Here is how Safari displays default video controls:



If you take a look at this same example in Firefox, you'll see something a little different, because Firefox does not support the H.264 video format.

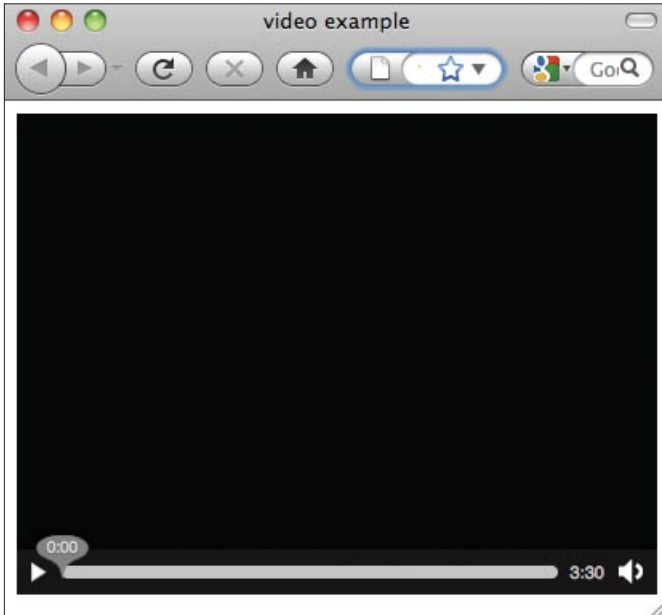


However, Firefox does support the Ogg Theora video format, and thus if we change the example to use Ogg instead of H.264:

```
<!DOCTYPE html><meta charset="utf-8">
<title>video example</title>
<video src="barcampsf.ogv" controls>
  A video showing BarCampSanFrancisco.
</video>
```

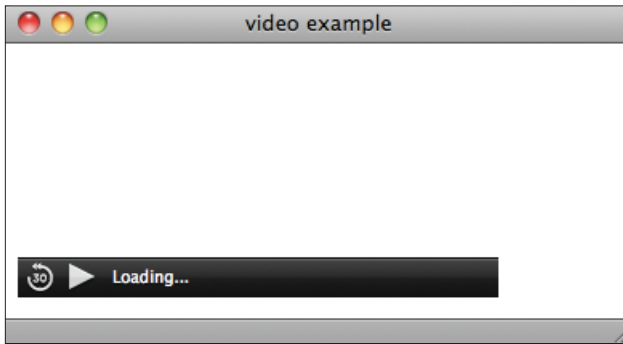


Now it looks much better in Firefox.



Note how the default controls in Firefox look different from the default controls in Safari. Expect different browsers — or even different devices with the same browser (or browser engine), such as desktop Safari vs. the iPad — to all have different default controls. The differences typically are designed to fit better with the overall user experience offered by the platform or browser.

With the Ogg Theora (.ogv) file, however, now the video example doesn't work in Safari:



Safari handles failure worse than Firefox. By providing a play button and a “Loading...” status indicator, Safari indicates that if we just wait, the video will load and play, which is, of course, not true — Safari does not support Ogg Theora. At least with Firefox, you get immediate feedback (if a bit cryptic with a medium gray X on a darker gray background) that the video format is not supported.

Now the challenge is to create an HTML5 video element that works in both Firefox and Safari, which are reasonably representative of HTML5 video supporting browsers.

HTML5 introduces a `<source>` element for just this purpose. It has an `src` attribute just like the `<audio>` and `<video>` elements themselves. The key to using `<source>` elements is that you place them inside `<audio>` and `<video>` elements, in the order in which you want the browser to check their compatibility, followed by the fallback content for browsers that don't support the `<video>` element .

```
<!DOCTYPE html><meta charset="utf-8">
<title>video example</title>
<video controls>
  <source src="barcampsf.ogv">
  <source src="barcampsf.mp4">
  A video showing BarCampSanFrancisco.
</video>
```



If you reload this example in both Safari and Firefox, you'll see that it now works in both and properly displays the video with default controls. For interactive demonstrations of the `<video>` element in various browsers, please see the HTML5 Now video.

UPDATE ON BROWSER SUPPORT AND FORMATS

Right now, the following video formats are supported by the following browsers:

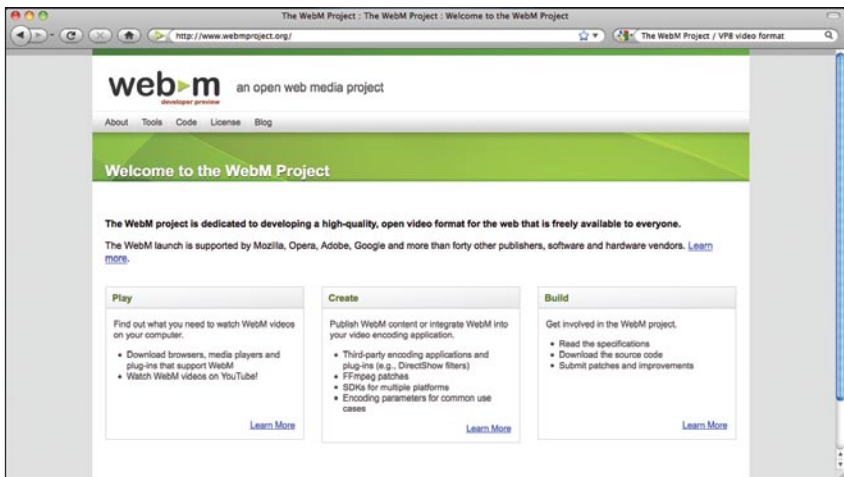
- **H.264:** Safari, Chrome
- **Ogg Theora:** Firefox, Opera, Chrome

Two recent developments have changed the support scenario a bit. Microsoft has announced that Internet Explorer 9 (IE9) will support HTML5 video, and, in particular, the H.264 format. You can download the IE9 Preview and see for yourself that it does support HTML5 video with H.264.

Second, Google and Mozilla have open sourced a new video format, VP8, which Firefox and Opera have either announced or already shipped support for. Thus, in the near future — or potentially already shipping by the time you're reading this — you'll be able to depend on more:

- **H.264:** Safari, Chrome, IE9
- **Ogg Theora:** Firefox, Opera, Chrome
- **VP8:** Firefox, Opera, Chrome

In addition, Microsoft has said that IE9 will support VP8 through an optional user-installable extension, and YouTube has already started converting their videos and serving VP8 video. For more on VP8, I recommend that you keep an eye on The WebM Project at <http://www.Webmproject.org>. There's a very good chance that VP8 will quickly eclipse Ogg Theora. However, due to widespread support across mobile devices, H.264 is here to stay for at least a few years and web sites will need to support both.



MULTIPLE VIDEO FORMATS FOR SEVERAL YEARS

Despite the open source appeal of the newly released VP8, web sites will likely need to support H.264 for at least a few years.

One of the big reasons H.264 has become particularly important in the past few years is the popularity of smart phones that have built-in hardware decoding for H.264. Hardware decoding is much more efficient, consuming less power and thus being much more battery friendly, which is a very important consideration for all mobile devices.

VP8 hardware decoders don't exist yet, as the hardware makers are first waiting to see if VP8 gains sufficiently popularity. Even if such decoders were to arrive in a year from now and were incorporated into new mobile devices after another year, there will still be millions of users of existing smart phones (iPhones, Androids) tied to 2-year contracts who won't be able to upgrade until their contracts expire. That adds up to at least four years at best that web sites will have to support H.264 if they want to support the growing number of video supporting mobile devices out there.

Thus for the foreseeable future, we'll need to code our HTML5 video to support multiple formats, VP8, and H.264.



13

NEW HTML5 USER INTERFACE ELEMENTS

When HTML was introduced in the early 1990s, the only semblance of interactivity it gave the user was the hyperlink — with a single click you could leap to any other page on the fledgling web. While hypertext certainly changed our perspective of self-contained linear text documents, all you could do really was jump around.

In 1995, [HTML 2.0 introduced forms](#) and suddenly the web became interactive. Forms enabled e-commerce, adding profit-motive fuel to the fire of self-expression, and incredibly accelerating the growth of the web. The addition of a small number of user interface elements (eight types of `<input>`, `<select>` and `<option>`, and `<textarea>`) gave us the building blocks for web applications, and a platform was born.

NEW FORM INPUTS

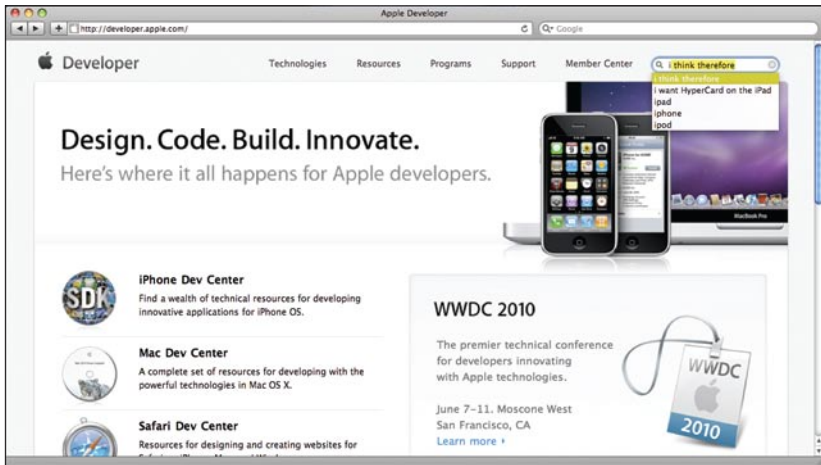
HTML5 adds more new `<input>` elements than all previous versions of HTML — 13 in total. The new elements fall into a few different categories.

SEARCH

Nearly every site on the web has a search box for searching the site. The new **search** input type is a special kind of text input that conveys the additional semantic that the user will be entering search terms.

```
<input type="search">
```

Browsers that understand this additional semantic can display the search input in a manner consistent with platform search interfaces, as well as perhaps offer search term suggestions. Safari, for example, renders the search input on developer.apple.com similar to its own input box, and offers a drop-down menu of recent searches.



The nice thing about the search input is that non-supporting browsers treat it like a normal text input. Browsers that support the search input, such as Safari, typically customize its appearance only a bit. Thus, you can start using `<input type="search">` on your sites today and on supporting browsers, your users will experience a subtly improved experience.

TEL, URL, EMAIL

The new input types of `tel`, `url`, and `email` also represent special text inputs, but with different structures as well as semantics.

```
<input type="tel">
```

```
<input type="url">
```

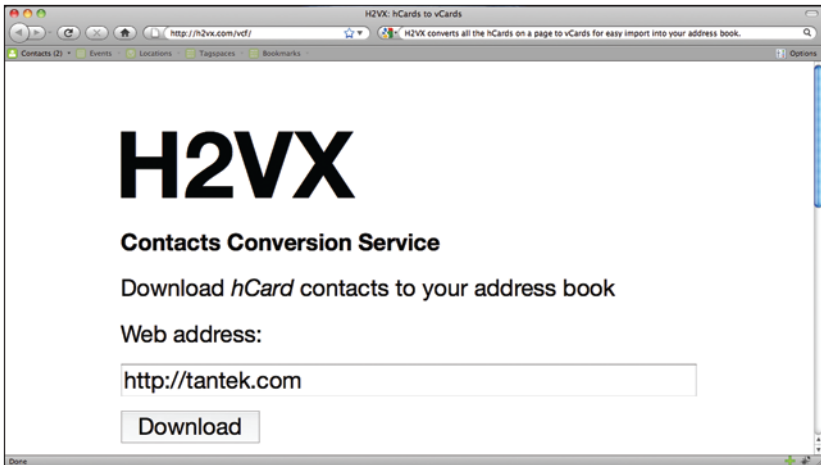
```
<input type="email">
```

Browsers can choose to present custom interfaces for more easily entering phone numbers, web addresses, and email addresses — such as providing the option to pick from an address book or recent history. In particular, mobile browsers on touch devices can choose to present touch keyboards customized for the keys frequently used for each: numbers and hyphens for phone numbers, slashes (/), dots (.), .com for URLs, @ for email addresses, etc. As of this writing, the



iPhone offers such support, and it's expected that other mobile devices such as the Android and Palm Pre already do or will soon support these elements as well.

For now, browsers typically treat these elements the same as text inputs. Of these three, I frequently use the URL input. Many sites take a web address (URL) as input and do something with it:



[H2VX.com's hCard to vCard conversion service.](http://h2vx.com)

H2VX, for example, converts the hCards on a page to vCards for easy import into your address book. It's also a relatively simple design, so if the browser does something custom or special with the URL input, it's unlikely that the site design will be affected much.

NUMBER

Many web forms ask users to input numbers (such as quantities on shopping sites). The `<input type="number">` allows the browser to provide a custom interface for entering numbers, perhaps with up/down arrows or a simpler numeric keypad on a touch interface.

BEWARE OF POTENTIAL NUMBER INPUT ABUSE

Examples of when NOT to use `<input type="number">` include:

ZIP and postal codes: In the United States, ZIP codes are five- or nine-digit numbers, but in other countries, such as England, postal codes include letters as well.

Numbered street names: Many street addresses include a street number as well as a street name, and many of these street numbers actually end with a letter, such as 37A. (Not to mention street numbers that use fractions such as 15 1/2.)

ISBN numbers: ISBN numbers can end with the letter X.

Driver's license numbers: Many states (inappropriately) reuse Social Security numbers for driver's license numbers. California driver's license numbers, however, include a leading letter.

In general, any type of data that seems like a number — and perhaps even is a number most of the time — but is actually a label, categorization, or identifier of some sort should likely just be a good old trusty `<input type="text">`, not a number input.

COLOR

Many sites, such as Twitter, allow you to choose various colors for your profile page, including background, text, hyperlink, and other colors. For this purpose and for graphical web applications in general, you now can use the color input:

```
<input type="color">
```

As of this writing, no browsers are known to support the color input. We can only hope that they'll come up with some way of presenting the platform user interface control(s) for easily picking a color.

DATE AND TIME INPUTS

In stark contrast to the limitations of the `<time>` element, there are numerous date and time inputs for specifying the date and time (with or without each other, local or not), a month, or a week.



```
<input type="date">  
<input type="time">  
<input type="datetime">  
<input type="datetime-local">  
<input type="month">  
<input type="week">
```

HOLES IN DATE AND TIME

Conspicuously missing from the list of date and time inputs are the same two real-world scenarios mentioned regarding the `<time>` element — that is, the ability to pick a year, or the ability to pick just a month and a day, for example, for birthdays or anniversaries. These would be logical additions:

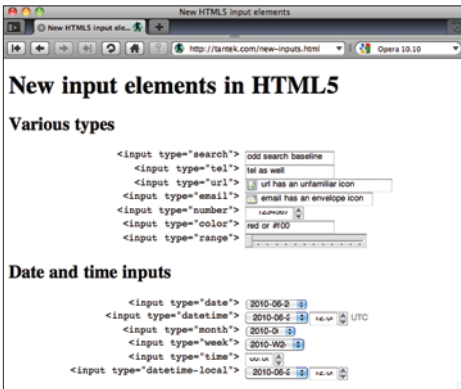
```
<input type="year">  
<input type="month-day">
```

What's also interesting about the new date and time inputs is that for the last two, `month` and `week`, there is no equivalent for semantically marking them up using the `<time>` element. Presumably, any form that lets you pick a month or a week will later display those values (perhaps on a confirmation screen), and thus it makes sense to be able to mark them up semantically as such. This mismatch is odd to say the least, and is hopefully something that will be fixed in HTML5.

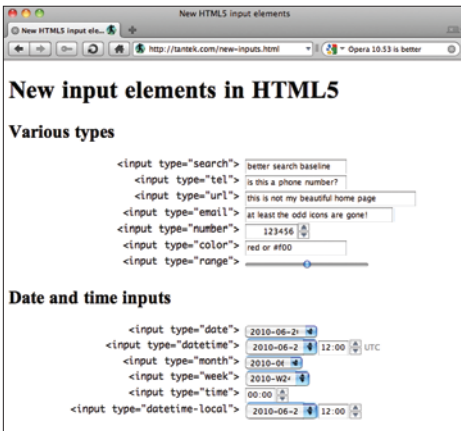
Today's browsers offer inconsistent support for the date and time inputs (as noted in the next section). While support for them does appear to be improving, at this point, I recommend the date and time inputs only for experimentation, not for production sites.

BROWSER CHALLENGES

In many ways, Opera has pioneered the implementation of the HTML5 inputs. As a result, we're able to see just how big a challenge the implementation is. Opera 10.10 showed strange icons for the email and url inputs, and clipped the top and right edges of some values.

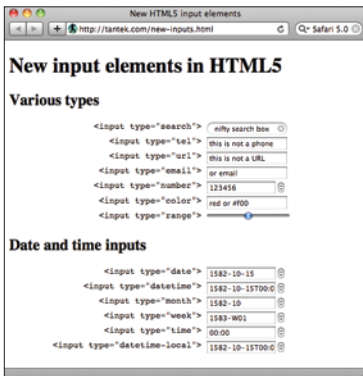


Opera 10.53 (the latest as of this writing) has removed the strange icons, and fixed the clipping of the top parts of text values. It still has problems, however, sizing the inputs wide enough to display their values without clipping, and has new problems with the up/down arrow indicators on the right edges of most of the date and time inputs.





Other browsers have taken a more cautious approach, rendering very little (if anything) differently for the new input types. Version 5 of Safari has started to provide a hint of customization with little up/down buttons next to some of the new input types.



And yet Safari has its own quirks — for example, if you click the little up button next to the date and time inputs, all the inputs with years give you “1582” (and the week input gives you “1583”). A very strange year to start with, except it turns out that 1582 was the year of the Gregorian Calendar Switch according to Wikipedia (<http://en.wikipedia.org/wiki/1582>), and in fact October 15 (as shown in the screenshot) was the date the switch occurred. This is perhaps why October 15, 1582 is the earliest date Safari will let you enter with the new date and time inputs.

In summary, many challenges exist when using the new HTML5 input types. I recommend the following:

- Start with simple uses in pages, for example, the search and URL inputs.
- Download the latest versions of Opera, Safari, and Firefox, and then try the “New input elements in HTML5” page (<http://tantek.com/new-inputs.html>) for yourself to get an idea of how browser support is evolving.

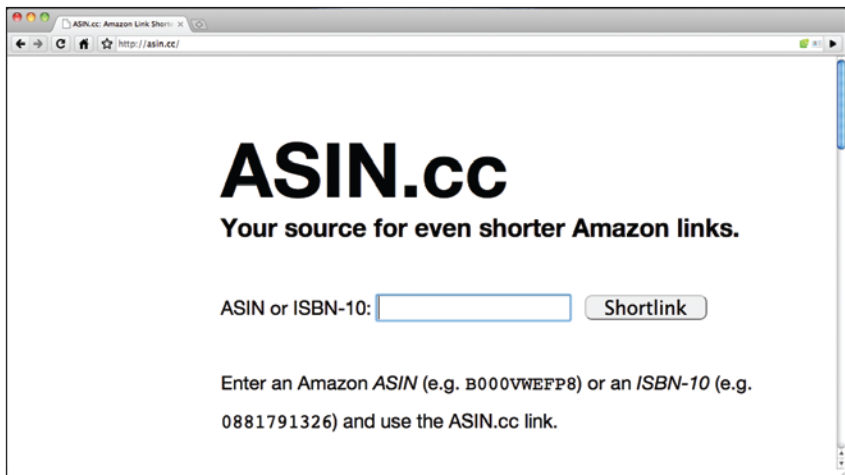
- Try out the new inputs on mobile browsers to see how they customize and perhaps enable better user input. For mobile use, the benefits may outweigh the risks. You'll need to decide on a case-by-case basis.
- Beware that the look, feel, and dimensions of the new HTML5 inputs are changing. If you decide to use them in pages, especially in pages for clients, be prepared for the look and feel of the pages to change when browser updates are released (thus possibly requiring you to go back and tweak your styling and scripts).

THE AUTOFOCUS, REQUIRED, AND PLACEHOLDER ATTRIBUTES

HTML5 adds more features for all input types, old and new alike. The first is the new **autofocus** attribute. On an `<input>` element, this attribute indicates to the browser that this input should be focused upon loading the page, so the user can start typing without having to first activate that input.

```
<input type="text" autofocus>
```

The ASIN.cc web site uses the **autofocus** attribute to automatically activate its one text input upon page load.





Examples of other sites that could benefit from auto-focus include search engines and simple status update sites (such as Twitter), where the user interface is designed around a primary input box. Be sure to use only one **autofocus** per page.

Many web forms have fields that users are required to fill in. Required fields often are indicated by an asterisk, a red border, or some other form of styling. HTML5 introduces the **required** attribute to make this semantic explicit:

```
<input type="text" required>
```

BEWARE THE BEVERLY HILLS PROBLEM

Regardless of how you implement required semantics in your web forms, you are highly likely to run into what I call the “Beverly Hills problem.” In short, do you know the ZIP code that attracts the most users to web sites?

Answer: 90210.

Naïve interaction design and pop culture contribute to this phenomenon. Many (if not most) instances of “required” inputs on web forms are not actually required to make the forms or sites work. Typically, they’re required only because some overzealous marketing person thought they might have a chance at forcing users to divulge some demographic information — like their ZIP code.

Of course, this is the World Wide Web, and requiring a ZIP code makes little sense (unless you are operating an e-commerce site, in which case, make it clear to users which countries you can ship to, and don’t assume everyone has a ZIP code in your billing/shipping forms). Since people worldwide — and yes, that’s a good thing — want to use your web site, they’ll put something into any form field that seems to be required. The most well-known ZIP code in the world is 90210 thanks to the well-known and worldwide-syndicated TV show called “Beverly Hills, 90210” and its subsequent follow-ons. So if you have a form that requires a ZIP code, but you don’t actually need one for site functionality, be prepared to have a lot of people appear to show up from 90210.

“Requiring” a user to fill in a field that is unnecessary for site functionality will likely result in lots of bad/noisy data for those supposedly required fields. So don’t bother wasting time coding unnecessary required fields, nor wasting the user’s time asking for them. And tell your overzealous marketing person that most of your users will likely come from Beverly Hills.

When browsers support the **required** attribute, they automatically alert users when they fail to complete one or more of the required fields in a form.

Until all browsers support the **required** attribute, you still need to write JavaScript to check that required form fields are properly filled in on forms. Regardless, you always need to check the form submissions on your server to make sure required fields have been filled in, as someone could always fake a form submission without using a browser.

The bottom line is that you can't depend on the **required** attribute alone to guarantee that any particular form field will be filled. It may help improve the user experience in some cases, but you always need to do work on the server side.

When creating forms, it can be quite useful to suggest to users what kind of information you're expecting in a specific field, and it may even make your user interface friendlier.

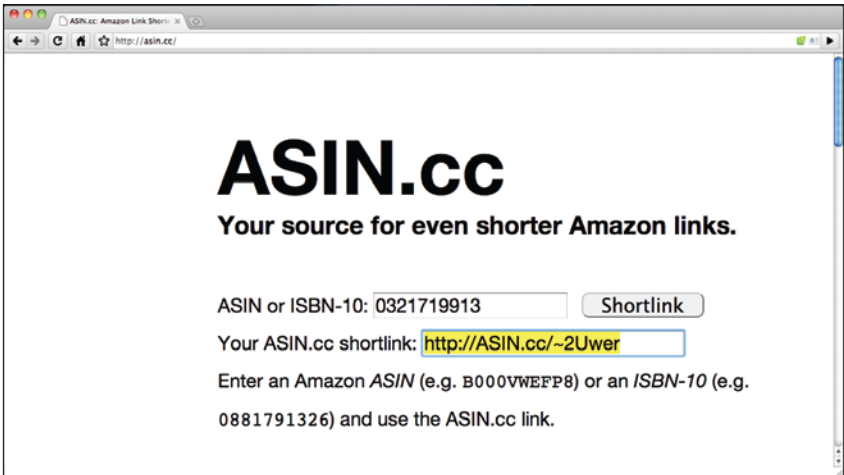
HTML5 introduces the **placeholder** attribute for providing such hints in form fields:

```
<input type="text" name="locality" placeholder="San Francisco">  
<input type="text" name="region" placeholder="California">  
<input type="text" name="country" placeholder="USA">
```

Currently, Webkit browsers (such as Safari and Chrome) and Opera support these new attributes, and Firefox will be supporting them soon. So you can start using them immediately, but be sure to still write your JavaScript with the expectation that not all browsers may support them.

OUTPUT AND DETAILS ELEMENTS

Many programs perform some sort of computation or calculation based on input from the user. For this purpose, HTML5 provides the **<output>** element. Obvious examples are calculators, financial projection applications, and other sites that take a bunch of numbers and provide some sort of result. However, you can use the **<output>** element for any kind of computed or algorithmic result, for example, an algorithmic shortlink:



The `<output>` element seems simple enough to start using right away, for the few situations where it applies. In contrast, the new `<details>` element applies to more use cases yet presents more challenges.

The `<details>` element is for representing and presenting a piece of content that offers a summary label with further details provided through a progressive disclosure interface. The new `<summary>` element marks up the summary or label inside the `<details>` element.

`<details>`

```
<summary>shopping list</summary>
```

```
<ul>
```

```
<li>spinach</li>
```

```
<li>tomatoes</li>
```

```
<li>carrots</li>
```

```
<li>avocado</li>
```

```
<li>broccoli</li>
```

```
<li>tofu</li>
```

```
<li>red wine</li>
```

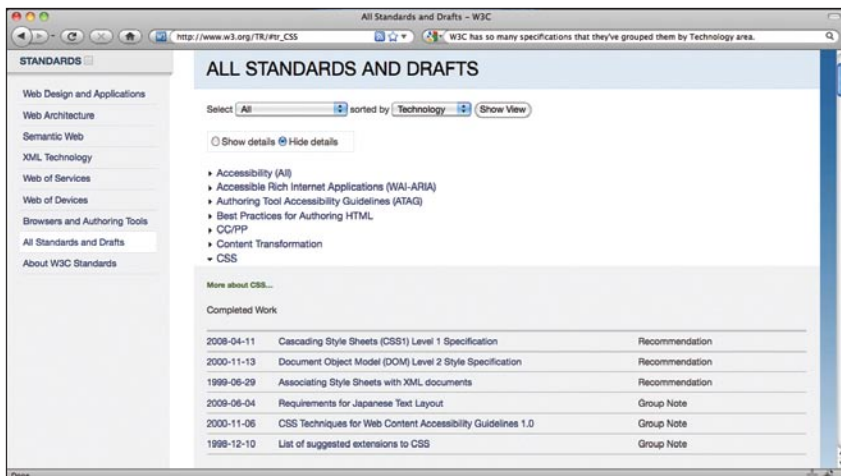
```
<li>dark chocolate</li>
```

```
</ul>
```

```
</details>
```

No current browsers support the interactivity portion of the `<details>` element, that is, they all show the summary and contents all the time. There's good reason for this — these new elements have been among the more unstable in the HTML5 specification. In particular, the `<summary>` element has gone through a number of naming revisions. Earlier attempts were made to reuse the `<label>`, `<caption>`, and even `<dt>` and `<dd>` elements instead. Given the instability of this feature, it's no surprise that no browsers have implemented it yet.

If you want to implement a disclosure interface that only shows the summary until the user clicks on it to reveal the rest, you'll have to build it yourself in JavaScript. The W3C's Technical Reports page (<http://w3.org/TR>) is a good example:



Note the areas of standards work being used as summary labels, and the detailed list of specifications in each area only being revealed when the user clicks the disclosure triangle or the summary labels.



METER, RANGE, AND PROGRESS

HTML5 introduces a few elements for displaying and inputting ranges of values. The new `<meter>` element can be used to display measurements such as a score, a rating, a countdown, or donations towards a goal. The new element comes with `min` and `max` attributes to set the bottom and top ends of the range, as well as a `value` attribute.

```
<meter value="74" max="100"> 74% </meter>
<meter value="0.75"> 3/4 </meter>
<meter min="0" max="250" value="185"> 74% </meter>
```

On the input side, there is also an input type for entering a measurement along a range of values, `<input type="range">`, also with `min`, `max`, and `value` attributes:

```
<input type="range" min="0" max="100" value="74">
<input type="range" min="0" max="250" value="185">
```

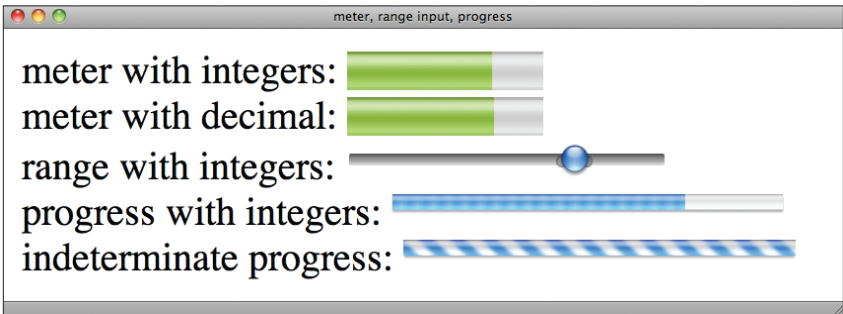
For the special case of progress towards completion of a task, HTML5 introduces the `<progress>` element. This element also has `max` and `value` attributes, but has a fixed minimum of zero (0):

```
<progress value="75" max="100"> 75% complete </progress>
```

The one additional special use of the `<progress>` element is to indicate indeterminate progress — that is, when there is no estimate available as to how far along a task is. Use the `<progress>` element on your sites to indicate how far along various user tasks are, such as uploading, online checkout, or filling out multipage forms. To indicate indeterminate progress, simply omit the `value` attribute:

```
<progress> working... </progress>
```

For both the `<meter>` and `<progress>` elements, be sure to always include fallback text like the previous examples for browsers that don't support those elements. Speaking of which, although browser support for the `<meter>`, `<input type="range">`, `<progress>` elements is currently quite sparse, recent improvements in particular in Safari show promise:



Opera has implemented a similar range input and we can expect these elements to be implemented soon in Firefox as well.

A common theme throughout all these new HTML5 user interface elements is the variance in levels of implementation across browsers, which change rapidly with each release. It's both exciting to see, and challenging to design for.

14 THE HTML5 BLEEDING EDGE

With all the immediate practical utility that HTML5 brings, a larger concept of HTML5 is broadly championed and cheered on by web developers worldwide and promoted by companies as large as Apple and Google.

As marketed, the concept would be more accurately framed as “The Open Web Applications Platform.” While HTML5 is a big part of how we build web Apps today and for the foreseeable future, other key pieces to learn go beyond HTML5:

1. **Microformats** for additional semantics and data portability
2. **CSS** for beautiful design and emerging effects such as animation
3. **JavaScript** and DOM for additional behaviors
4. Numerous **Web APIs** and other related specifications



The first three concepts are immediately practical, dependable, and fairly well-established; they have entire books written about them.

The fourth concept is a set of working drafts, some of which were spun off from HTML5 and others of which were never part of HTML5 and yet have sometimes been included in the HTML5 marketing umbrella. All are fairly experimental in nature, and you should keep this in mind when considering their use.

For the most part, these related drafts are enabling new classes of applications on the web while also unstable and volatile enough to require close attention to depend on them. Following is a brief summary with pointers to their drafts.

Geolocation API (<http://www.w3.org/TR/geolocation-API>). Allows a web page to ask the browser for geolocation information about where the user is. Quite powerful and already implemented in Firefox and Safari. Note: DO NOT tell your marketing person about this API because they might just try to use it to get more accurate demographic information (see “Beware the Beverly Hills problem” in Chapter 13).

HTML Device (<http://dev.w3.org/html5/html-device>). For accessing various hardware capabilities, such as built-in cameras or microphones. Just an editor’s draft currently, it’s not clear when an official working draft will be published.

Microdata (<http://www.w3.org/TR/microdata>). A way to annotate content in HTML with custom vocabularies, in many ways much simpler than RDFa. Use microformats that are already well supported to mark up common semantics such as people, organizations, events, reviews, etc., and take a look at microdata if you want to create your own vocabulary.

Web Sockets API (<http://www.w3.org/html5/websockets>). Defines an API for two-way communication between web pages and servers.

Web SQL Database (<http://www.w3.org/TR/webdatabase>). Defines an API for storing data in a database in the web browser that can be queried using a variant of SQL. There have also been discussions about using non-SQL databases as alternatives.

Indexed Database API (<http://www.w3.org/TR/IndexedDB>). One of those non-SQL alternatives is the Indexed Database API, which defines a database of records holding simple values and hierarchical object records of key/value pairs.

Web Storage (<http://www.w3.org/TR/webstorage>). Simpler than a database, Web Storage defines an API for storing key/value pairs in the browser, a logical step forward from using cookies.

Web Workers (<http://www.w3.org/TR/workers>). Web Workers is an API for creating additional JavaScript threads that are able to run in the background. They're useful for any script that performs computations or other tasks that should not hold up the user interface. If your pages or scripts feel slow or cause wait cursors such as the infamous spinning rainbow beachball, look into offloading some of your processing onto Web Workers.

Web Messaging (<http://dev.w3.org/html5/postmsg>). Defines mechanisms for communicating between different browser windows, tabs, and iframes. Although currently only an editor's draft, Web Messaging, also known as `postMessage`, has actually been implemented in modern browsers for some time: Internet Explorer 8+, Firefox 3+, Opera 9+, and Safari 4+.

All these "specifications" are W3C public working drafts or editor's drafts, which means that they all issue the following warning about their status (emphasis from source): "Implementors should be aware that this specification is not stable. **Implementors who are not taking part in the discussions are likely to find the specification changing out from under them in incompatible ways.**"

While HTML5 itself is a working draft as well, enough of it is either based on what worked in HTML4, what has worked in browsers for many years, or has been interoperably implemented, that HTML5 is much more dependable than these related bleeding-edge specifications.

Plenty of ever-changing opportunities are available to push the boundaries of web applications experiments. Just keep in mind that as the web APIs change, any web apps using them may break. Also be sure to keep up with the editor's drafts of these specifications (noted in their headers) as well as their latest official working drafts.

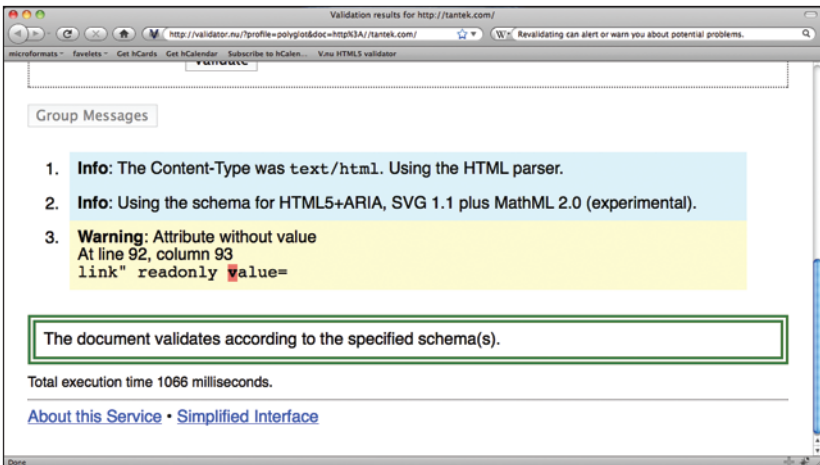


15

CHECKPOINT: REVALIDATE

By now you've added at least a few new HTML5 features to your pages, from new semantics to perhaps some native multimedia or new forms support. It's time to revalidate your pages and see what we can learn.

Using the same HTML5 validator browser button/favelet used in Chapter 9, revalidate and note any errors to fix.



Even if your document validates, pay close attention to any warnings and see if there is something you can improve. In this case, the validator is warning that I have an "Attribute without value."

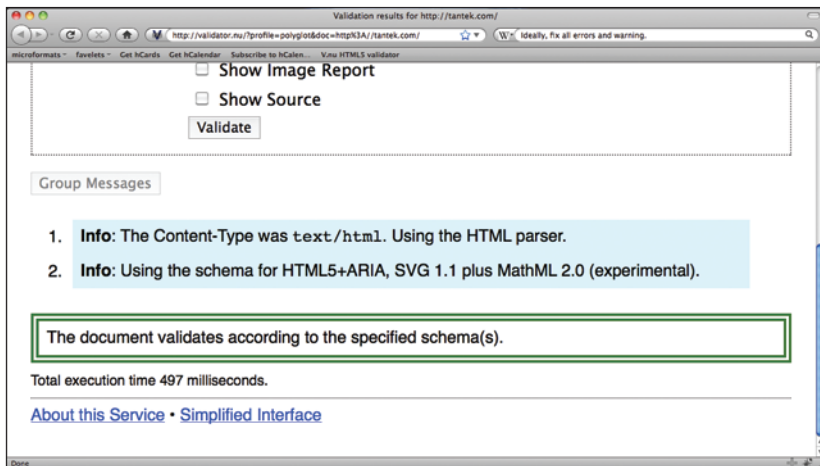
In "Transitioning your XHTML" in Chapter 5, I noted that all attribute values must be quoted for XHTML compatibility. When checking my documents for both HTML and XHTML validity (what some call polyglot or biglot documents), the validator noted a particular case of an unquoted value not having an explicit value at all! Here's the errant tag:

```
<input type="url" id="link" readonly value="http://tantek.com" />
```

HTML5 has several such “standalone” attributes such as the **autofocus** and **required** attributes mentioned in Chapter 13. However, XHTML compatibility requires that all attributes have explicitly quoted values. The fix is simple—use the name of the attribute as a value for itself:

```
<input type="url" id="link" readonly="readonly" value="http://tantek.com" />
```

Applying that fix, let’s take a look at what Validator.nu reports:



Repeat this process of validating, finding the sources of errors or warnings, fixing them, and revalidating until you have fixed all errors/warnings and the validator returns a clean green validation report.

Congratulations! You’re now using HTML5.

Now, find out where you can learn more, and help make a difference in HTML5.



16 CONCLUSION

We've covered a lot about HTML5, and no doubt you will have further questions, want to both keep up with the spec as it evolves, and perhaps even help it do so.

KEEPING UP WITH HTML5 EVOLUTION

You now have a good idea of what HTML5 currently does and doesn't do. I recommend following updates to the these resources for keeping up with changes.

HTML5 differences from HTML4 (<http://w3.org/TR/html5-diff>)

HTML: The Markup Language (<http://w3.org/TR/html-markup>)

The HTML5 Now Wiki (<http://html5now.pbwiki.com>)

The HTML5 Now Twitter (<http://twitter.com/html5now>)

The Differences specification will help serve as a good reminder of what's changed (and continues to change) from HTML4, and The Markup Language specification provides an excellent index and overview of *all* the tags and attributes, new and old, in HTML5. Finally, I've setup an HTML5 Now Wiki and Twitter account where I'll be posting updates both to the HTML5 Now video and booklet, and about HTML5 in general.

MORE HTML5 RESOURCES

There are a few good HTML5 books that have been recently released or soon will be that will help provide additional depth and perspective on HTML5.

HTML5 for Web Designers by Jeremy Keith

HTML5: Up and Running by Mark Pilgrim

Introducing HTML5 by Bruce Lawson and Remy Sharp

While reading anything about HTML5, whether on the web or in books, keep in mind what you've learned in this video and booklet, especially what we talked about in Chapter 14, specifically, what is *actually* HTML5, and what is *marketed* or otherwise lumped in with HTML5. If you're not sure, you can always search the table of contents of the HTML5 specification itself:

HTML5: A vocabulary and associated APIs for HTML and XHTML

(<http://www.w3.org/TR/html5>)

If it's not in that table of contents, it's not part of HTML5, no matter how many CEOs or well-meaning books may claim otherwise.

For additional perspectives on HTML5, I recommend the following blogs:

WHATWG blog (<http://blog.whatwg.org>)

Ian Hickson (<http://ln.hixie.ch>)

Jeremy Keith (<http://adactio.com/journal/tag/html5>)

Bruce Lawson (<http://www.brucelawson.co.uk/category/html5>)

Jeffrey Zeldman (<http://www.zeldman.com/category/html5>)

HELP IMPROVE HTML5 WHILE LEARNING

HTML5 is still a working draft, and needs your feedback, input, and suggestions. In late summer 2009, a group of colleagues (myself included) concerned about the direction and specifics of HTML5, gathered to share our understandings and write up a statement of feedback. The result was the following

HTML5 Super Friends statement (<http://www.zeldman.com/superfriends>)

Guide to HTML5 Hiccups (<http://www.zeldman.com/superfriends/guide>)

Much of the feedback and hiccups listed in the guide have already been fixed in HTML5.

You too can use your practical experience to help improve HTML5. Here are some resources to participate in the evolution of HTML5:



WHATWG wiki (<http://wiki.whatwg.org>): A very good resource for tracking and contributing research and proposals for changes to HTML5. As with any other wiki, create an account and first help out with minor fixes and contribute to existing proposals.

#whatwg IRC channel on Freenode IRC (<irc://irc.freenode.org/whatwg>): Set up an IRC client (such as Colloquy.info on the Mac), connect to the irc.freenode.net server and join the #whatwg channel. A lot of informal communication takes place on the IRC channel, and HTML5 experts are there nearly every hour of the day. This is a good place to start if you think you've found a problem in HTML5 and/or have suggestions for improvement.

W3C HTML Working Group (<http://www.w3.org/html/wg/#join>): The HTML Working Group is open for anyone to join. All you need to do is fill out a few forms, agree to share your contributions freely (e.g. that you're not going to assert any patents), and you can join the HTML Working Group mailing list and participate directly in HTML5 discussions at W3C.

HTML5 NOW

You've learned everything you need to know to write your first HTML5 document, and update your HTML4/XHTML1 documents to HTML5. Enhance your pages with new semantics, graphics, multimedia support. Validate and fix your pages as necessary and publish your HTML5 site!

Follow updates to HTML5, especially if you choose to use any of the bleeding edge features, give feedback and help move the web forward.

Thank you for reading [this book](#) and watching the accompanying video. You have what it takes to make a difference, in your pages, on the web, and for the web. I look forward to seeing your work.

Tantek Çelik, <http://tantek.com>