HACKS

Jesse Cravens & Jeff Burtoft

HTML5 Hacks

Tips & Tools for Creating Interactive Web Applications

O'REILLY®

WEB DESIGN/HTML

HACKS

HTML5 Hacks

Tips & Tools for Creating Interactive Web Applications

With 90 detailed hacks, expert web developers Jesse Cravens and Jeff Burtoft demonstrate intriguing uses of HTML5-related technologies. Each recipe provides a clear explanation, screenshots, and complete code examples for specifications that include Canvas, SVG, CSS3, multimedia, data storage, web workers, WebSockets, and geolocation.

You'll also find hacks for HTML5 markup elements and attributes that will give you a solid foundation for creative recipes that follow. The last chapter walks you through everything you need to know to get your HTML5 app off the ground, from Node.js to deploying your server to the cloud.

Here are just a few of the hacks you'll find in this book:

- **Make iOS-style card flips with CSS transforms and transitions**
- **Replace the background of your video with the Canvas tag**
- **Use Canvas to create high-res Retina Display–ready media**
- **Make elements on your page user-customizable with editable content**
- **Cache media resources locally with the filesystem API**
- **Reverse-geocode the location of your web app user**
- **Process image data with pixel manipulation in a dedicated web worker**
- **Push notifications to the browser with Server-Sent Events**

ABOUT THE AUTHORS

Jesse Cravens is a Senior Engineer at Trigger where he works with the world's leading companies helping them to design, engineer, and bring to market meaningful products and services. With 12 years of experience in web application development, Jesse is currently focusing on single page web applications, the mobile web and HTML5.

Jeff Burtoft, HTML5 Evangelist for Microsoft Corporation, has built a reputation of pushing the boundaries of HTML5 and JavaScript. With over 10 years of experience in web development, he's held roles in the industry ranging from webmaster of a startup to lead frontend developer for a Fortune 500 company.

Proven techniques from the experts at O'Reilly Media

US $34.99 CAN $36.99
ISBN: 978-1-449-33499-4

O'REILLY®

HTML5 Hacks

Jesse Cravens and Jeff Burtoft

O'REILLY®
Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

HTML5 Hacks

by Jesse Cravens and Jeff Burtoft

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://my.safaribooksonline.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editors: Simon St. Laurent and Meghan Blanchette
Production Editor: Holly Bauer

Copyeditor: Audrey Doyle
Proofreader: Rachel Leach
Indexer: Judith McConville
Cover Designer: Mark Paglietti
Interior Designer: David Futato
Illustrator: Rebecca Demarest

November 2012: First Edition

Revision History for the First Edition:

2012-11-09 First release

See *http://oreilly.com/catalog/errata.csp?isbn=9781449334994* for release details.

ISBN: 978-1-449-33499-4

[LSI]

Table of Contents

Preface

HTML5 is the new catchall term for "the Web." Like Ajax and Web 2.0 before, the term can cause confusion when used in different contexts. HTML5 is technically the fifth revision of the HTML markup language, but you will find the term being used to describe an umbrella of next-generation web technology specifications that include CSS3, SVG, and JavaScript APIs.

In order to understand HTML5 in this context, first it is important to understand that HTML5 is not one technology that is applied or added to a web application. There are more than 30 specifications within the HTML5 umbrella, and each is at a different stage of maturity. Furthermore, each specification is also at a different state of adoption and, potentially, implementation, by the major browser manufacturers.

Depending on an application's business requirements, the app's developer will pick and choose the HTML5 features to take advantage of. It is entirely possible that only a handful of the available specifications will be used for the final implementation of a modern web application.

Critics often proclaim it is necessary to wait until HTML5 is 100% supported before you use it in your application. This is simply not true. Many specifications have already reached maturity and are fully implemented by modern browsers. Other specifications are at an early stage of development, or are poorly supported by some of the major browser manufacturers. It's important to know which specification type you are using. Research is helpful, but the only true way to tell is to thoroughly test your apps in all browsers.

For the specifications that are newer or that aren't as strongly supported, some clever developers have produced free and open source code that can be utilized to shim, or *polyfill*, support in older browsers. As defined by Remy Sharp, "A polyfill, or polyfiller, is a piece of code (or plug-in) that provides the technology that you, the developer,

expect the browser to provide natively. Flattening the API landscape, if you will." In our opinion, the best polyfill is one that lets you write your code just as you would if the feature were natively supported, and that does the work in the background when necessary, being transparent to both the user and the developer. In most circumstances, each HTML5 specification has a polyfill, or multiple competing polyfills, and is ready to be used today. You will find references to some of the Web's most effective polyfills within this book.

Why HTML5?

A beginning developer might ask, "Why should I care about HTML5?" Unfortunately, there is not a simple answer to this question. Even the most advanced web developers will answer this question differently depending on the features they are most familiar with.

But overall, there are some common trends that span the feature set and on which most developers would agree. Before HTML5, the Web was not considered to be a rival to native desktop and mobile applications. Nearly since its inception, the Web has been considered to be an easily deployable, cross-platform solution. However, it has been hampered due to its lack of highly important business requirements: namely, performance, security, and graphics. The theory has been that if the modern web browser could mature as an application platform, developers would be able to stop creating platform-specific native applications.

The Ajax revolution took the web application world in the right direction by providing asynchronous, background updates to the server via the `XMLHttpRequest` object, JSON transfer format, and an explosion of JavaScript libraries that stretched the boundaries of application development in the browser, many of which continue to make up the basis for polyfill support. However, HTML5 is about the modern browser providing the necessary support to enable sophisticated application development natively. In order to accomplish this, features such as the ability to maintain browser history and bookmarking during asynchronous interactions, cross-domain communication, local storage, offline support, rich graphics, and even new protocols to improve the speed and efficiency of the connectivity layer still needed to be created and improved.

Browser prefixes are most common in CSS. We urge you to read the introduction to Chapter 2 to get a full explanation of how browser prefixes are implemented in CSS.

HTML5 Implementations

As an eager developer ready to move forward with implementing some of the new features available in this text, it will be important to understand that many of the HTML5 specifications are in experimental stages of development. One major challenge in writing a book about evolving specifications is keeping the information fresh and up to date.

The following topics are important considerations when learning experimental web browser specifications.

Browser-specific prefixes

In order for browser makers to be able to implement experimental features (usually implementing specifications before they were completed), browser makers "prefix" that feature with a shorthand that limits its use to each particular browser. A great example of this is the implementation of `requestAnimationFrame`, which is a JavaScript method in the page that aids in animation within the browser. Each browser originally implemented this feature with browser prefixes as follows:

- `requestAnimationFrame`
- `webkitRequestAnimationFrame`
- `mozRequestAnimationFrame`
- `oRequestAnimationFrame`
- `msRequestAnimationFrame`

Browser prefixes are most common in CSS. We urge you to read the introduction to Chapter 2 to get a full explanation of how browser prefixes are implemented in CSS.

Validation with HTML5 Conformance Checker

An HTML validator is a piece of software that parses your web pages against a set of web standards as defined by a particular Document Type Definition (DTD). If you are unfamiliar with a DTD, think of it as metadata that precedes your HTML markup in order to instruct the browser as to what "flavor" of HTML you will be using.

The HTML validator returns a list of errors found, according to the chosen standard. For our purposes, we will assume that we are using the HTML5 Document Type Definition.

The HTML5 Document Type Definition is more lenient than the most recent XHMTL definition, and the output of the W3C's new validator reflects this difference. After all, a validator should not throw exceptions for stylistic issues. It should be focused on validating your HTML markup against a specification.

HTML5 Lint

This means that developers should also be ready to use a lint tool in order to expose stylistic issues within their code. Some of the more common issues to check for are consistent indentation, lowercase tags, and omission of closing tags.

At the time of this writing, we recommend the HTML5 Lint tool.

References for HTML5 implementation statuses and feature support

We will continue to provide updates as often as possible to the examples provided within this text on our blog.

There are also many great resources around the web to reference HTML5 implementation statuses and feature support of specific browsers.

For all modern browsers:

http://caniuse.com/
http://html5test.com/

For Chrome:

http://www.chromium.org/developers/web-platform-status

For Internet Explorer:

http://msdn.microsoft.com/en-US/ie/ff468705.aspx

For Mozilla Firefox:

https://wiki.mozilla.org/Features

For Apple Safari:

https://developer.apple.com/technologies/safari/html5.html

Why HTML5 Hacks?

The term *hacker* carries a negative connotation within the media, but the term has evolved to describe a number of different technical people. Wikipedia provides three very different definitions for the term *hacker*:[1]

[1] *http://en.wikipedia.org/wiki/Hacker*

1. Hacker (computer security), someone who accesses a computer system by circumventing its security system
2. Hacker (hobbyist), who makes innovative customizations or combinations of retail electronic and computer equipment
3. Hacker (programmer subculture), who shares an anti-authoritarian approach to software development now associated with the free software movement

It is in the context of definition 2 that we are using the term *hack*. Among these types of hacks, the term refers to a self-contained proof of concept, similar to agile spikes, or recipes. These quick solutions exercise or validate an API, feature, or technology, and can also serve a very important role, not only in educating the software team, but also in driving the direction of development within a project's life cycle.

Who This Book Is For

HTML5 Hacks introduces readers to the umbrella of HTML5 specifications through 90 hacks. For beginners it can serve as a starting point for building browser-based applications. For intermediate to advanced developers it can serve to quickly fill in the gaps for specifications they have yet to be exposed to.

Nevertheless, this book will be what you make of it.

Contents of This Book

This book consists of 10 chapters, organized as follows:

Chapter 1, Hacking the Semantic Way
: Introduces new key HTML5 markup elements and attributes

Chapter 2, Hacking with Style
: Covers visual expression and behaviors with CSS3

Chapter 3, Multimedia Hacking
: Discusses HTML5 audio and video tags

Chapter 4, Hacking Your Graphics with Canvas and SVG
: Covers working with Canvas and SVG

Chapter 5, User Interactions
: Introduces HTML5 drag-and-drop, editing elements, and other interactions

Chapter 6, Client-Side Data Storage Hacks
: Discusses storage and HTML5 application cache

Chapter 7, Geolocation Hacks
: Teaches how to work with geolocations

Chapter 8, WebWorker API
: Covers taking advantage of the WebWorker API

Chapter 9, Hacking HTML5 Connectivity
: Discusses web sockets, cross-document messaging, server-side events, and more

Chapter 10, Pro HTML5 Application Hacks with Node.js
: Teaches how to build professional HTML5 applications with Node.js

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, email addresses, filenames, file extensions, pathnames, directories, and Unix utilities

`Constant width`
: Indicates commands, options, switches, variables, attributes, keys, functions, types, classes, namespaces, methods, modules, properties, parameters, values, objects, events, event handlers, XML tags, HTML tags, macros, the contents of files, or the output from commands

`Constant width bold`
: Shows commands or other text that should be typed literally by the user

`Constant width italic`
: Shows text that should be replaced with user-supplied values

---

This formatting signifies a tip, suggestion, general note, warning, or caution.

Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example,

writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*HTML5 Hacks* by Jesse Cravens and Jeff Burtoft (O'Reilly). Copyright 2013 Jesse Cravens and Jeff Burtoft, 978-1-449-33499-4."

All of the code examples are located at *https://github.com/html5hacks*.

You can also keep up with the authors and any updates at *http://html5hacks.com*.

We'd Like to Hear from You

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-9938 (in the United States or Canada)
(707) 829-0515 (international or local)
(707) 829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at:

oreil.ly/HTML5_Hacks

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about our books, courses, conferences, and news, see our website at *http://www.oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Safari® Books Online

Safari Books Online is an on-demand digital library that delivers expert content in both book and video form from the world's leading authors in technology and business.

Technology professionals, software developers, web designers, and business and creative professionals use Safari Books Online as their primary resource for research, problem solving, learning, and certification training.

Safari Books Online offers a range of product mixes and pricing programs for organizations, government agencies, and individuals. Subscribers have access to thousands of books, training videos, and prepublication manuscripts in one fully searchable database from publishers like O'Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology, and dozens more. For more information about Safari Books Online, please visit us online.

Acknowledgments

We would like to extend a special thank you to both of our families. This book required a lot of evening and weekend hours to complete, and our wives and children are the ones who sacrificed the most.

Jeff would like to thank his wife Carla who encouraged him to step out and write a book, and for allowing their lives to be turned upside down while he worked to complete it. He would also like to thank his children, Chloe, Maddy, and Jude, for being his inspiration for creativity, and for being proud of him in everything he does. Jeff would also like to thank Jason, his partner and coauthor, for dreaming up ideas like this book, and making them a reality.

Jason would like to thank his wife Amy for all the support she gave him through the long and late hours he spent on this book; his children, Carter and Lindsey, for trying to understand when Daddy had to work weekends and nights; his brother and sister-in-law for providing a quiet place to write and encouragement that it was worth the effort; and his parents for the continued onslaught of inspiration to check another item off the bucket list. And finally, he'd like to thank Jeff for having the shared determination to champion this book, overcome the adversity associated with an ambitious project, and make it across the finish line.

Guest Hackers

John Dyer is the executive director of Communications and Educational Technology at Dallas Theological Seminary. He has been a web developer and technology writer for more than 10 years, and he loves creating tools that make complex tasks easier for other developers. He lives in the Dallas area with his two amazing kids and his lovely wife, Amber.

Alex Sirota cofounded and was the CTO and Head of Product at FoxyTunes, an Internet startup acquired by Yahoo!, where he spent more than four years building media and entertainment web products. Previously, he cofounded Elbrus Ltd., a company that provided software solutions to Philips Medical Systems, IBM, and others. Prior to Elbrus, he was the head of a computer facility in the Israel Defense Forces, and he coauthored a book (published by Wiley) on Mozilla and web technologies. He holds a bachelor's degree in computer science from Technion–Israel Institute of Technology.

Raymond Camden is a senior developer evangelist for Adobe. His work focuses on web standards, mobile development, and ColdFusion. He's a published author and presents at conferences and user groups on a variety of topics. He can be reached through his blog, via Twitter (**@cfjedimaster**), or via email (*raymondcamden@gmail.com*).

Phil Leggetter is a Real-Time Web Software and Technology Evangelist. He has been developing and using real-time web technologies for more than 10 years, and his focus is to help people use these technologies to build the next generation of interactive and engaging real-time web applications.

Alexander Schulze is the founder of the jWebSocket project, as well as an IT consultant and trainer for IT professionals. He is a speaker at various conferences and author of several articles and books.

1

Hacking the Semantic Way

The spirit of HTML5 is simplicity. HTML5 has made it easy to implement web standards that in the past have been difficult to implement. Instead of trying to reinvent the Web, visionary consortiums such as the WHATWG (Web Hypertext Application Technology Working Group) and the W3C (World Wide Web Consortium) looked at the web standards that had evolved and built upon them.

In essence, HTML5 is primarily an update to the HyperText Markup Language (HTML). In this chapter we will start with the basic building blocks of HTML, the semantic elements, to provide a foundation for the simple yet powerful new web browser technologies exposed within this book.

So, open up your favorite code editor, brew a pot of coffee, and get ready to code in the most powerful language the Web has ever seen: HTML5!

HACK 01 Simplify Your Doc with the Right <doctype>

If there's an emblem representing the simplicity HTML5 brings to the markup world, it's the `<DOCTYPE>` tag. The HTML5 `<doctype>` tag is easy to use.

When you open an XHTML document the first thing you see, the first line of the document, is a mess:

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

The `<DOCTYPE>` tag of HTML past, inherited from its SGML foundations, consisted of three main components: the tag name, the public identifier string, and the DTD (Document Type Definition) URL. It's a strange mix of uppercase and lowercase letters, quote marks and slashes, and a URL that brings up an even less readable file. To make it even stranger, the `<DOCTYPE>` tag is unique, as it is the only HTML tag since HTML 4.01 that is in all caps.

HTML5 says farewell to all that, and keeps it simple:

```
<!doctype html>
```

The browser uses the **<doctype>** to know how to render the web page. Most browsers didn't download the DTD from the URL, but they did change their behavior based on the **<DOCTYPE>**. If a browser encountered the preceding code, it would switch to standards mode (as opposed to quirks mode) and apply XHTML transitional formatting.

Given all that, how can HTML5 get away with a basic **<doctype>** such as **html**? The simple answer is that the new **<doctype>** is a "simple answer." The new **<doctype>** was made to trigger a simplified approach to document rendering, not to meet old expectations. Browser makers reached a consensus on how browser-specific functionality should be handled, so there is no need for "quirks mode" page rendering. If all browsers render in a standard manner, the DTD is unnecessary; thus a simple declaration of **html** states that the browser should set aside any DTD and simply render the page.

HTML5 is a simplified version of HTML. The tags are less complex, the features are less complex, and most importantly, the rules are less complex.

However, in most applications you write, you will not yet be servicing a user base that consistently supports HTML5. So how can you switch between **<doctype>**s when the **<doctype>** is supposed to be the first line of the document? This doesn't leave much room for JavaScript trickery or fancy hacks. Well, good news; there is a backward-compatible HTML5 **<doctype>** as well:

```
<!DOCTYPE html>
```

"But wait," you say. "Isn't that the same simple **<doctype>** presented earlier?" Yes, it is! The only key difference is that "doctype" is now in all caps. The HTML5 specification states that the **<doctype>** is case-insensitive; however, previous versions of HTML require an all-caps version of the **<doctype>**. You will find that much of HTML5 is backward-compatible with earlier versions. The vast majority of browsers on the market today will see the new **<doctype>** and recognize it as simply being "standards mode" for page rendering.

Using the backward-compatible version of the **<doctype>** will allow you to start using HTML5 today, while continuing to support browsers of the past!

HACK 02 Adopt Common Structures

Many web documents have similar structures. Take advantage of markup that makes it easier to share styles and expectations.

Web designers and developers have long conformed to structural components on a page. A common high-level page structure may look something like the following:

```
    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
    <html>
    <head>
    <meta http-equiv="content-type" content="text/html;charset=UTF-8" />
        <title>...</title>
    </head>
    <body>
        <div id="header">...</div>
        <div id="nav">...</div>
        <div id="article">...</div>
        <div id="footer">...</div>
    </body>
    </html>
```

Take note of the "structural" `id`s in the page. This reflects well-organized content and a clean structure for the page. The problem with the preceding code is that almost every element in the markup is a `div`. `Div`s are great, but they are of little use in page definition without associating them with an `id`. The problem with using `id`s for role association is that when you want to use them for another purpose—say, to identify a doc tree—you run into problems: as soon as you add a tool such as YUI Grids or WordPress to a page that actually uses the `id` of a `div`, it conflicts with your `div` "roles," and the next thing you know you are adding layers of `div`s just to satisfy your structural needs. As a result, the clean page shown earlier may now look something like this:

```
    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
    <html>
    <head>
    <meta http-equiv="content-type" content="text/html;charset=UTF-8" />
        <title>...</title>
    </head>
    <body>
        <div id="header">
        <div id="nav">
            <div id="doc2">
                <div id="wordpress-org-2833893">...</div>
            </div>
        </div>
        <div id="article">
            <div id="doc2">
```

```
            <div id="wordpress-org-887478">...</div>
        </div>
    </div>
    <div id="footer">...</div>
</body>
```

You can see pretty quickly where this gets messy, yet we don't want to abandon the idea of structural elements that declare page segments—many code readers, such as screen readers and search bots, have come to rely on structural conventions. As with many parts of HTML5, the new structural tags have provided a simple solution to the problem of added complexity. Let's build our page with structural elements:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>...</title>
</head>
<body>
    <header>...</header>
    <nav>...</nav>
    <article>...</article>
    <footer>...</footer>
</body>
</html>
```

Once again we have a simple, clean HTML5 solution that keeps our page easy to work with, and easy to consume by screen readers and search bots. This same code can meet the needs of our third-party products as well, as shown in the following solution:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>...</title>
</head>
<body>
    <header data-yuigrid="doc2" data-wordpress="2833893">...</header>
    <nav>...</nav>
    <article data-yuigrid="doc2" data-wordpress="887478">...</article>
    <footer>...</footer>
</body>
</html>
```

We'll get into the `data-` attributes later in this chapter, but for now you just need to understand that this solution allows you to keep the structural elements of the page and let third-party components apply identifiers to the nodes, while freeing up the `id` attributes for the page author to control. Take note, third-party developers: never assume that the `id` of an element is yours to consume!

All That and More

HTML5 didn't stop at the new tags discussed in the preceding section. Here's a partial list of some of the new HTML5 markup tags to take note of:

`<article>`	`<aside>`	`<figcaption>`	`<figure>`	`<footer>`	`<header>`	`<hgroup>`
`<mark>`	`<nav>`	`<section>`	`<time>`	`<keygen>`	`<meter>`	`<summary>`

A lot of these tags grew out of common use by web developers. The W3C smartly decided to "pave the cow paths" instead of trying to change the behavior of web developers. This way, the tags are generally useful for immediate adoption.

In most cases each tag's intent is pretty obvious. The `<header>` and `<footer>` tags do exactly what they say: they outline the header and footer of the page (or app). You use `<nav>` to wrap your navigation. The `<section>` and `<article>` tags give you options to the overused `<div>` tag; use these to break up your page according to the content (e.g., wrap your articles in the `<article>` tag). The `<aside>` tag acts in a similar way to the `<article>` tag, but groups the content aside the main page content. The `<figure>` tag refers to a self-contained piece of content, and so on and so on. Note that this list is not conclusive and is always changing. Visit the w3schools website for the most complete list I could find.

HACK 03 Make Your New HTML5 Tags Render Properly in Older Browsers

Don't wait for full HTML5 adoption across the Web. Make HTML5 structural tags render properly in all browsers.

So, now you have this whole new world of HTML5 elements that will let you be both expressive and semantic with your markup. You've been freed from the shackles of `div`s and can show your face at parties again!

> Semantic markup *is the use of markup in a meaningful way. Separation of structure and presentation leads us to define our presentation (look and feel) with CSS, and our content with meaningful or semantic markup.*

You're feeling pretty good about yourself until you remember that some of your visitors are not using HTML5 browsers, and being the web standards elitist that you are, your page has to be backward-compatible. Don't throw those HTML5 tags out the window just yet. This hack will teach you how to write your code once, and use it on all the browsers out there.

Any browser made in the past 10 years will see your HTML5 tags in one of 3 ways:

1. See the HTML5 tag and render it appropriately (congratulations, you support HTML5!).
2. See the HTML5 tag, not recognize it, and consider it an unstyled (which defaults to inline) DOM (Document Object Model) element.
3. See the HTML5 tag, not recognize it, and ignore it completely, building the DOM without it.

Option 1 is a no-brainer: you're in an HTML5 browser. Option 2 is likewise pretty easy to address, as you simply have to set your default display parameters in your CSS. Keep in mind that with option 2, you have no functional DOM APIs for these new tags, so this is not true support for the tags. In other words, using this method to create a `meter` element does not create a functional meter. For our use case of semantic markup elements, however, this should not be an issue.

So, focusing on option 3, you're using IE 6, 7, or 8 and you're loading a page that contains new HTML5 semantic tags. The code will look something like this:

```
    <!DOCTYPE html>
    <html>
    <head>
        <meta charset="UTF-8">
        <title>My New Page with Nav</title>
</head>
<body>
<div>
    <nav class="nav">
    <p>this is nav text</p>
    </nav>
</div>
</body>
</html>
```

There are basically two different ways to handle this lack of support.