CSS Mastery

Advanced Web Standards Solutions

Andy Budd
with Cameron Moll
and Simon Collison

friendsof ED™
DESIGNER TO DESIGNER™
an Apress® company

CSS Mastery: Advanced Web Standards Solutions

ISBN-13 (pbk): 978-1-59059-614-2
ISBN-10 (pbk): 1-59059-614-5

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

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

Product numbers for the images used in Tuscany Luxury Resorts are as follows: FAN1003579, FAN1003613, FAN1006983, and DVP0703035.

Credits

Lead Editor
Chris Mills

Technical Reviewer
Molly Holzschlag

Editorial Board
Steve Anglin
Dan Appleman
Ewan Buckingham
Gary Cornell
Jason Gilmore
Jonathan Hassell
Chris Mills
Dominic Shakeshaft
Jim Sumser

Project Manager
Denise Santoro Lincoln

Copy Edit Manager
Nicole LeClerc

Copy Editor
Liz Welch

Assistant Production Director
Kari Brooks-Copony

Production Editor
Kelly Winquist

Compositor and Artist
Diana Van Winkle, Van Winkle Design

Proofreader
April Eddy

Indexer
John Collin

Interior and Cover Designer
Kurt Krames

Manufacturing Director
Tom Debolski

To my beautiful girlfriend Melanie,
for all your love and support over the last 9 months.
—Andy Budd

To Mam, Dad, Emma, Agenzia—
thank you for your support.
—Simon Collison

CONTENTS AT A GLANCE

CONTENTS

FOREWORD

In our wonderful world of web design, there are 3,647 ways to accomplish the same goal. Approximately. And that absurdly fictitious number is increasing every day. Instead of one, correct way of solving a particular problem, we're both blessed and cursed by the abundant choices we have as web designers. It's these choices that make designing for the Web fun and interesting, while at the same time overwhelming. *CSS Mastery* will help cure that *overwhelmingitis* (a word that I've just invented).

Andy Budd has been writing, designing, and speaking about standards-based web design for years, and we're now lucky to see his clear, easy-to-follow way of teaching essential CSS techniques compiled in this very book. The result is a card catalog of indispensable solutions, tricks, and tips that a web professional such as yourself should not be without.

I've always frowned on publications that suggest a *single*, correct way of accomplishing a goal, and Andy does the complete opposite, offering multiple methods for tasks such as styling links, creating tabbed navigation, or creating columned layouts (to name but a few). Armed with these popular and stylish approaches to common design elements, you'll be better prepared to make your own *informed* decisions.

And as if that wasn't enough, Andy's gone ahead and enlisted the help of two imitable designers to help pull all the pieces together, showing how these essential techniques can work *together*. I've long been a fan of Cameron's and Simon's work, and to see two great case studies covering fluid, bulletproof designs as well as flexible style solutions, respectively… well, that's just a gigantic bonus.

So dig in and start chipping away at those 3,647 ways to master your CSS.

Dan Cederholm
Salem, Massachusetts
Author, Web Standards Solutions

ABOUT THE AUTHORS

Andy Budd is a user experience designer and web standards developer living and working in Brighton, England. As the creative director of web design consultancy Clearleft (`www.clearleft.com`), Andy enjoys building attractive, accessible, and standards-compliant websites. His online home can be found at `www.andybudd.com`, where he writes about modern web design practices.

Andy is a regular speaker at international design conferences, workshops, and training events, and organized the UK's first web 2.0 conference (`www.dconstruct.org`). Passionate about the quality of education in the industry, Andy runs SkillSwap (`www.skillswap.org`), a free community training and networking project. Andy also helped set up the Web Standards Awards (`www.webstandardsawards.com`), a project that aims to recognize websites for their use of web standards.

When he's not building websites, Andy is a keen travel photographer. Never happier than when he's diving some remote tropical atoll, Andy is also a qualified PADI dive instructor and retired shark wrangler.

Cameron Moll, recognized as one of the industry's most balanced new media designers, is proficient in functional web design, elegant interfaces, and clean markup. Cameron has been involved in the design and redesign of scores of websites, and his influential techniques have found favor in circles across the Web. A marketing background and a keen eye for design lead him to merge form and function in the form of compelling visual experiences.

Cameron's work has been recognized by respected organizations and notable individuals such as National Public Radio (NPR), Communication Arts, and Veer. His personal site, `CameronMoll.com`, delivers design how-tos in the form of engaging conversation, on-topic banter, and downloadable artwork source files.

Simon Collison is Lead Web Developer at Agenzia (`www.agenzia.co.uk`), and has worked on numerous web projects for record labels, high-profile recording artists, and leading visual artists and illustrators, including The Libertines, Black Convoy, and Project Facade. Simon also oversees a production line of business, community, and voluntary sector websites, and passionately ensures everything he builds is accessible and usable, and complies with current web standards. Simon regularly reviews CSS-based websites for Stylegala, and does his best to keep his highly popular blog (`www.collylogic.com`) updated with noise about web standards, music, film, travels, and more web standards.

On those rare occasions away from the computer, Simon can be found in the pub, or trying to con free gig tickets out of his clients. A little too obsessed with music, he is very likely to bore you with his latest musical Top 100, or give you a potted history of the UK indie scene from 1979 to the present day. Simon has lived in many cities, including London and Reykjavik, but now lives happily in Nottingham with Emma and a cat called Ziggy.

ABOUT THE TECHNICAL REVIEWER

Molly E. Holzschlag is a well-known Web standards advocate, instructor, and author. A popular and colorful individual, she is Group Lead for the Web Standards Project (WaSP) and an invited expert to the GEO working group at the World Wide Web Consortium (W3C). Among her 30-plus books is the recent *The Zen of CSS Design*, coauthored with Dave Shea. The book artfully showcases the most progressive csszengarden.com designs. You can catch up with Molly's blog at—where else?—`http://molly.com/`.

ACKNOWLEDGMENTS

Andy Budd

Thanks to everybody who helped make this book possible, both directly and indirectly.

To Chris for guiding me through the writing process and helping turn my ideas into reality. And to everybody at Apress who worked tirelessly to get this book published on time. Your dedication and professionalism is much appreciated.

To my friends and colleagues at Clearleft (`www.clearleft.com`), Jeremy Keith (`www.adactio.com`) and Richard Rutter (`www.clagnut.com`), for providing encouragement and feedback throughout the book-writing process.

To Molly E. Holzschlag for lending your experience and breadth of knowledge to this book. Your support and guidance was invaluable, and I still don't know where you manage to find the time.

To Jamie Freeman and Jo Acres for providing the perfect environment in which to develop my skills. I'll pop around for tea and doughnuts soon. Thanks also to the Brighton web development community at large, and especially everybody on the BNM and SkillSwap mailing lists.

To all my colleagues who continue to share their wealth of knowledge in order to make the Web a better place. This book would not have been possible without the previous work of the following people, to name but a few: Cameron Adams, John Allsopp, Nathan Barley, Holly Bergevin, Douglas Bowman, The BritPack, Dan Cederholm, Tantek Çelik, Joe Clark, Andy Clarke, Simon Collison, Mike Davidson, Garrett Dimon, Derek Featherstone, Nick Fink, Patrick Griffiths, Jon Hicks, Shaun Inman, Roger Johansson, Ian Lloyd, Ethan Marcotte, Drew McLellan, Eric Meyer, Cameron Moll, Dunstan Orchard, Veerle Pieters, D. Keith Robinson, Jason Andrew Andrew Santa Maria,, Dave Shea, Ryan Sims, Virtual Stan, Jeffrey Veen, Russ Weakley, Simon Willison, and Jeffrey Zeldman.

To all the readers of my blog and everybody I've met at conferences, workshops, and training events over the last year. Your discussions and ideas helped fuel the content of this book.

Big thanks to Mel, for proofreading each chapter and putting up with me over the last 9 months.

And lastly, thanks to you for reading. I hope this book helps you take your CSS skills to the next level.

Cameron Moll

I'd like to give gratitude to all the contributors to my case study. A big nod goes to Ryan Parman, whose TIMEDATE script was used to generate the day/month stamp in the upper-right corner of the Tuscany layout. Download a copy of his script here: `www.skyzyx.com/scripts/`.

And endless thanks to Veer for providing the gorgeous images used in this layout. Without their help, Tuscany Luxury Resorts may have otherwise been visually drab. Somehow, without fail, Veer always delivers unique, phenomenal visual elements—photography, type, merchandise, and more—that are far from commonplace. Access their collections here: `www.veer.com/`.

Simon Collison

I must thank the incredible Jon Burgerman (`www.jonburgerman.com`), Richard May (`www.richard-may.com`), and all my other Black Convoy (`www.blackconvoy.com`) friends for allowing me to use their images and names, and generally skim the cream off their talent for this case study. Huge thanks also to the cool Swede Roger Johansson (`www.456bereastreet.com`) for allowing me to use his rounded corners and for buying me a drink last summer. The More Than Doodles design was built quickly and efficiently thanks to the inspired templating system within the ExpressionEngine (`www.expressionengine.com`) publishing platform—a tool I could not live without. Finally, thanks to the Agenzia (`www.agenzia.co.uk`) boys for turning a blind eye to my fevered book writing of late. Much appreciated all around.

INTRODUCTION

There are an increasing number of CSS resources around, yet you only have to look at a CSS mailing list to see the same questions popping up time and again. “How do I center a design?” “What is the best rounded-corner box technique?” “How do I create a three-column layout?” If you follow the CSS design community, it is usually a case of remembering which website a particular article or technique is featured on. However, if you are relatively new to CSS, or don’t have the time to read all the blogs, this information can be hard to track down.

Even people who are skilled at CSS run into problems with some of the more obscure aspects of CSS such as the positioning model or specificity. This is because most CSS developers are self-taught, picking up tricks from articles and other people’s code without fully understanding the spec. And is it any wonder, as the CSS specification is complex and often contradictory, written for browser manufacturers rather than web developers?

Then there are the browsers to contend with. Browser bugs and inconsistencies are one of the biggest problems for the modern CSS developer. Unfortunately, many of these bugs are poorly documented and their fixes verge on the side of folk law. You know that you have to do something a certain way or it will break in one browser or another. You just can’t remember which browser or how it breaks.

So the idea for a book formed. A book that brings together the most useful CSS techniques in one place, that focuses on real-world browser issues and that helps plug common gaps in people’s CSS knowledge. A book that will help you jump the learning curve and have you coding like a CSS expert in no time flat.

Who is this book for?

CSS Mastery is aimed at anybody with a basic knowledge of (X)HTML and CSS. If you have just recently dipped your toes into the world of CSS design, or if you’ve been developing pure CSS sites for years, there will be something in this book for you. However, you will get the most out of this book if you have been using CSS for a while but don’t consider yourself a master just yet. This book is packed full of practical, real-world advice and examples, to help you master modern CSS design.

How is this book structured?

This book eases you in gently, with two chapters on basic CSS concepts and best practices. You will learn how to structure and comment your code, the ins-and-outs of the CSS positioning model, and how floating and clearing really works. You may know a lot of this already, but you will probably find bits you've missed or not understood fully. As such, the first two chapters act as a great CSS primer as well as a recap on what you already know.

With the basics out of the way, the next five chapters cover core CSS techniques such as image, link, and list manipulation; form and data-table design; and pure CSS layout. Each chapter starts simply and then works up to progressively more complicated examples. In these chapters you will learn how to create rounded-corner boxes, images with transparent drop shadows, tabbed navigation bars, and flickr-style rollovers. If you want to follow along with the examples in this book, all the code examples can be downloaded from `www.friendsofed.com`.

Browser bugs are the bane of many a CSS developer, so all the examples in this book focus on creating techniques that work across browsers. What's more, this book has two whole chapters devoted to hacks, filters, bugs, and bug fixing. In these chapters you will learn about some of the most common filters, when to use them, and when not to use them. You will also learn about bug-hunting techniques and how to spot and fix common bugs before they start causing problems. You will even learn what really causes many of Microsoft Internet Explorer's seemingly random CSS bugs.

The last two chapters are the *piece de resistance*. Simon Collison and Cameron Moll, two of the best CSS designers around, have combined all of these techniques into two fantastic case studies. So you learn not only how these techniques work, but also how to put them into practice on a real-life web project.

This book can be read from cover to cover, or kept by your computer as a reference of modern tips, tricks, and techniques. The choice is up to you.

Conventions used in this book

This book uses a couple of conventions that are worth noting. The following terms are used throughout this book:

- (X)HTML refers to both the HTML and XHTML languages.
- Unless otherwise stated, CSS relates to the CSS 2.1 specification.
- IE 5.*x*/Win means Internet Explorer versions 5.0 and 5.5 for Windows.
- IE 6 and below on Windows refers to Internet Explorer 5.0 to 6.0 on Windows.

It is assumed that all the (X)HTML examples in this book are nested in the `<body>` of a valid document, while the CSS is contained in the `<head>` of the document for convenience. Occasionally, (X)HTML and CSS have been placed in the same code example for brevity. However, in a real document, these items need to go in their respective places to function correctly.

```
p {color: red;}

<p>I'm red</p>
```

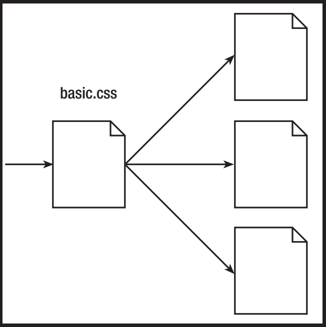
Lastly, for (X)HTML examples that contain repeating data, rather than writing out every line, the ellipse character (…) is used to denote code continuation:

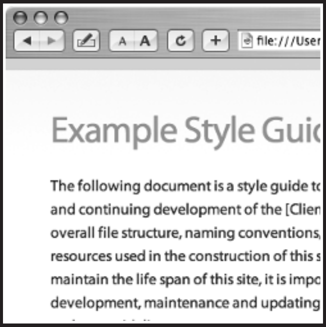
```
<ul>
<li>Red</li>
<li>Yellow</li>
<li>Pink</li>
<li>Green</li>
…
</ul>
```

So, with the formalities out of the way, let’s get started.

1 SETTING THE FOUNDATIONS

```
<!--------------- 11 MAIN C
 <td colspan="2" width="42
<table width="417" cellspa
<tr><td width="417" valign
<a href="/sections/politic
<img
src="http://a4.g.akamaitec
width=200 height=150 vspac
align=left></a>
<font face= geneva,arial,h
<a href="/sections/politic
Passing the Torch
</a>
</b></font><br>
```





Example Style Guid

The following document is a style guide to
and continuing development of the [Clien
overall file structure, naming conventions,
resources used in the construction of this s
maintain the life span of this site, it is impo
development, maintenance and updating

The human race is a naturally inquisitive species. We just love tinkering with things. When I recently bought a new iMac G5 I had it to bits within seconds, before I'd even read the instructions. We enjoy working things out ourselves, creating our own mental models about how we think things behave. We muddle through and only turn to the manual when something goes wrong or defies our expectations.

One of the best ways to learn Cascading Style Sheets (CSS) is to jump right in and start tinkering. However, if you're not careful you may end up misunderstanding an important concept or building in problems for later on. In this chapter, I am going to review some basic, but often misunderstood, concepts and show you how to keep your (X)HTML and CSS clear and well structured.

When we use the term XHTML, we are referring to Extensible Hypertext Markup Language, and when we use the term (X)HTML, we are referring to both XHTML and HTML.

In this chapter you will learn about

- The importance of a well-structured and meaningful document
- Coding best practices
- Common coding mistakes
- Document types, DOCTYPE switching, and browser modes
- Ways to target your styles
- The cascade, specificity, and inheritance

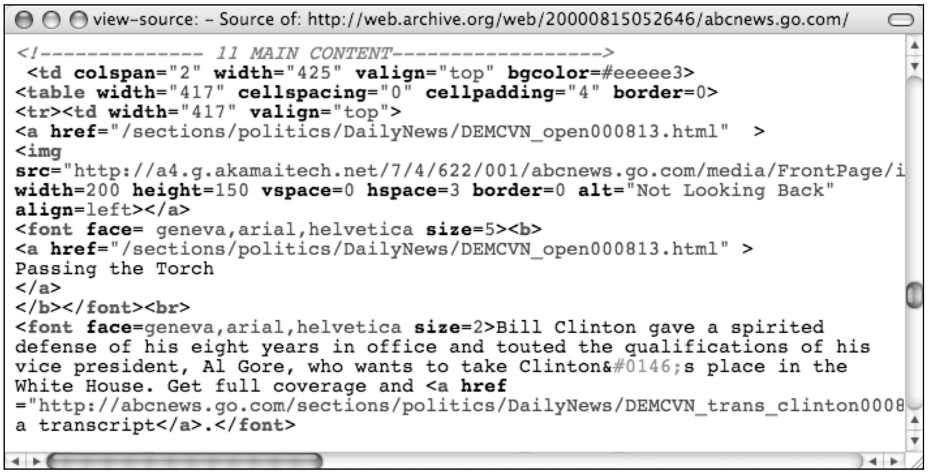
Structuring your code

Most people don't think about the foundations of a building. However, without solid foundations, the majority of the buildings around us wouldn't exist. While this book is about advanced CSS techniques, much of what we are going to do would not be possible (or would be very difficult) without a well-structured and valid (X)HTML document to work with.

In this section you will learn why well-structured and meaningful (X)HTML is important in CSS development. You will also learn how you can add more meaning to your documents, and by doing so, make your job as a developer easier.

Use meaningful markup

The early Web was little more than a series of interlinked research documents using HTML to add basic formatting and structure. However, as the World Wide Web increased in popularity, HTML started being used for presentational purposes. Instead of using heading elements for page headlines, people would use a combination of font and bold tags to create the visual effect they wanted. Tables got co-opted as a layout tool rather than a way of displaying data, and people would use blockquotes to add whitespace rather than to indicate quotations. Very quickly the Web lost its meaning and became a jumble of font and table tags (see Figure 1-1).


view-source: – Source of: http://web.archive.org/web/20000815052646/abcnews.go.com/

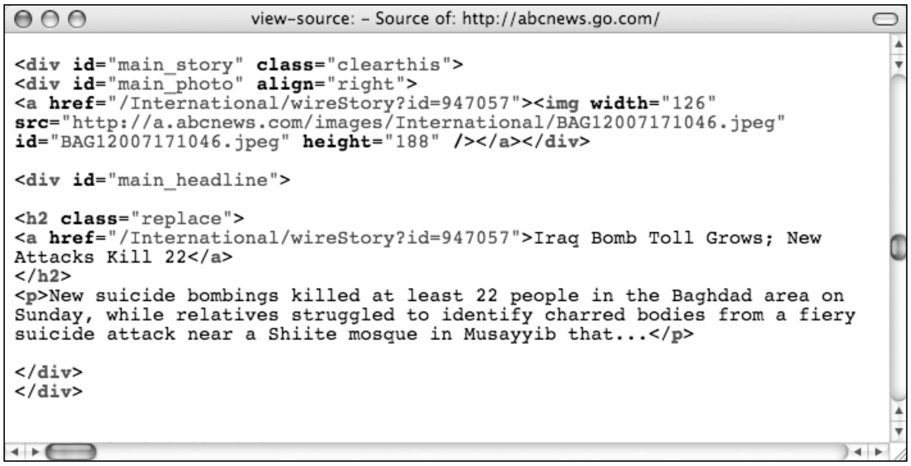
```
<!-------------- 11 MAIN CONTENT------------------>
 <td colspan="2" width="425" valign="top" bgcolor=#eeeee3>
<table width="417" cellspacing="0" cellpadding="4" border=0>
<tr><td width="417" valign="top">
<a href="/sections/politics/DailyNews/DEMCVN_open000813.html"  >
<img
src="http://a4.g.akamaitech.net/7/4/622/001/abcnews.go.com/media/FrontPage/i
width=200 height=150 vspace=0 hspace=3 border=0 alt="Not Looking Back"
align=left></a>
<font face= geneva,arial,helvetica size=5><b>
<a href="/sections/politics/DailyNews/DEMCVN_open000813.html" >
Passing the Torch
</a>
</b></font><br>
<font face=geneva,arial,helvetica size=2>Bill Clinton gave a spirited
defense of his eight years in office and touted the qualifications of his
vice president, Al Gore, who wants to take Clinton&#0146;s place in the
White House. Get full coverage and <a href
="http://abcnews.go.com/sections/politics/DailyNews/DEMCVN_trans_clinton0008
a transcript</a>.</font>
```

Figure 1-1. The markup for the lead story from abcnews.com on August 14, 2000, uses tables for layout and large, bold text for headings. The code lacks structure and is difficult to understand.

HTML was intended to be a simple and understandable markup language. However, as web pages became more and more presentational, the code became almost impossible to understand. As such, complicated WYSIWYG (What You See Is What You Get) tools were needed to handle this mass of meaningless tags. Unfortunately, rather than making things simpler, these tools added their own complicated markup to the mix. By the turn of the millennium, the average web page was so complicated it was almost impossible to edit by hand for fear of breaking the code. Something needed to be done.

Then along came Cascading Style Sheets. With CSS it became possible to control how a page looked externally and to separate the presentational aspect of a document from its content. Presentational tags like the font tag could be ditched, and layout could be controlled using CSS instead of tables. Markup could be made simple again, and people began to develop a newfound interest in the underlying code.

Meaning started to creep back into documents. Browser default styles could be overridden so it became possible to mark something up as a heading without it being big, bold, and ugly. Lists could be created that didn't display as a series of bullet points, and blockquotes could be used without the associated styling. Developers started to use (X)HTML elements because of what they meant rather than how they looked (see Figure 1-2).



```
<div id="main_story" class="clearthis">
<div id="main_photo" align="right">
<a href="/International/wireStory?id=947057"><img width="126"
src="http://a.abcnews.com/images/International/BAG12007171046.jpeg"
id="BAG12007171046.jpeg" height="188" /></a></div>

<div id="main_headline">

<h2 class="replace">
<a href="/International/wireStory?id=947057">Iraq Bomb Toll Grows; New
Attacks Kill 22</a>
</h2>
<p>New suicide bombings killed at least 22 people in the Baghdad area on
Sunday, while relatives struggled to identify charred bodies from a fiery
suicide attack near a Shiite mosque in Musayyib that...</p>

</div>
</div>
```

Figure 1-2. The markup for the lead story on abcnews.com from earlier this year is well structured and easy to understand. While it does contain some presentational markup, the code is a significant improvement on the code in Figure 1-1.

Meaningful markup provides the developer with several important benefits. Meaningful pages are much easier to work with than presentational ones. For example, say you need to change a quotation on a page. If the quotation is marked up correctly, it is easy to scan through the code until you find the first `blockquote` element. However, if the quotation is just another paragraph element tag, it will be a lot harder to find.

As well as being easy for humans to understand, meaningful markup—otherwise known as semantic markup—can be understood by programs and other devices. Search engines, for instance, can recognize a headline because it is wrapped in `h1` tags and assign more importance to it. Screenreader users can rely on headings as supplemental page navigation.

Most importantly for the context of this book, meaningful markup provides you with a simple way of targeting the elements you wish to style. It adds structure to a document and creates an underlying framework to build upon. You can style elements directly without needing to add other identifiers, and thus avoid unnecessary code bloat.

(X)HTML includes a rich variety of meaningful elements, such as

- `h1`, `h2`, etc.
- `ul`, `ol`, and `dl`
- `strong` and `em`

- blockquote and cite
- abbr, acronym, and code
- fieldset, legend, and label
- caption, thead, tbody, and tfoot

As such, it is always a good idea to use an appropriate meaningful element where one exists.

IDs and class names

Meaningful elements provide an excellent foundation, but the list of available elements isn't exhaustive. (X)HTML was created as a simple document markup language rather than an interface language. Because of this, dedicated elements for things such as content areas or navigation bars just don't exist. You could create your own elements using XML, but for reasons too complicated to go into, it's not very practical at this time.

The next best thing is to take existing elements and give them extra meaning with the addition of an ID or a class name. This adds additional structure to your document, and provides useful hooks for your styles. So you could take a simple list of links, and by giving it an ID of `mainNav`, create your own custom navigation element.

```
<ul id="mainNav">
  <li><a href="#">Home</a></li>
  <li><a href="#">About Us</a></li>
  <li><a href="#">Contact</a></li>
</ul>
```

An ID name is used to identify an individual element on a page, such as the site navigation, and must be unique. IDs are useful for identifying persistent structural elements such as the main navigation or content areas. They are also useful for identifying one-off elements—a particular link or form element, for example.

Across a site, ID names should be applied to conceptually similar elements in order to avoid confusion. Technically, you could give both your contact form and your contact details the ID name of `contact`, assuming they were on separate pages. However, you would then need to style each element based on its context, which could be problematic. Instead, it would be much simpler to use distinct ID names such as `contactForm` and `contactDetails`.

While a single ID name can only be applied to one element on a page, the same class name can be applied to any number of elements on a page. Classes are very useful for identifying types of content or similar items. For instance, you may have a news page that contains the date of each story. Rather than giving each date a separate ID, you could give all of them a class name of `date`.

When naming your IDs and classes, it is important that you keep the names as meaningful and "un-presentational" as possible. For instance, you could give your section navigation an ID of `rightHandNav` as that is where you want it to appear. However, if you later choose to position it on the left, your CSS and (X)HTML will go out of sync. Instead, it would make more sense to name the element subNav or secondaryNav. These names explain what the element is rather than how it is presented. The same is true of class names. Say you want all your error messages to be red. Rather than using the class name red, choose something more meaningful like error or feedback (see Figure 1-3).

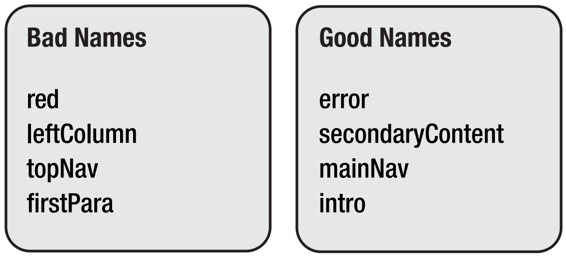


Figure 1-3. Good and bad ID names

When writing class and ID names, you need to pay attention to case sensitivity. CSS is generally a case-insensitive language. However, the case-sensitivity of things that appear in the markup, such as class and ID names, depends on the case sensitivity of the markup language. If you are using XHTML, class and ID names are case sensitive, whereas with regular HTML they are case insensitive. The best way to handle this issue is simply to be consistent with your naming conventions. So, if you use camel case in your (X)HTML class names, carry this through to your CSS as well.

Due to the flexibility of classes, they can be very powerful. At the same time, they can be overused and even abused. Novice CSS authors often add classes to nearly everything in an attempt to get fine-grained control over their styles. Early WYSIWYG editors also had the tendency to add classes each time a style was applied. Many developers picked up this bad habit when using generated code to learn CSS. This affliction is described as *classitis* and is, in some respects, as bad as using table-based layout because it adds meaningless code to your document.

```
<h3 class="newsHead">Zeldman.com turns 10</h3>
<p class="newsText">
Another milestone for Jeffrey as zeldman.com turns 10 today
</p>
<p class="newsText"><a href="news.php" class="newsLink">More</a></p>
```

In the preceding example, each element is identified as being part of a news story by using an individual news-related class name. This has been done to allow news headlines and text to be styled differently from the rest of the page. However, you don't need all these extra classes to target each individual element. Instead, you can identify the whole block as a news item by wrapping it in a division with a class name of news. You can then target news headlines or text by simply using the cascade.

```
<div class="news">
<h3>Zeldman.com turns 10</h3>
<p>Another milestone for Jeffrey as zeldman.com turns 10 today</p>
<p><a href="news.php">More</a></p>
</div>
```

Removing extraneous classes in this way will help simplify your code and reduce page weight. I will discuss CSS selectors and targeting your styles shortly. However, this overreliance on class names is almost never necessary. I usually only apply a class to an element if an ID isn't suitable, and I try to use them sparingly. Most documents I create usually only need the addition of a couple of classes. If you find yourself adding lots of classes, it's probably an indication that your (X)HTML document is poorly structured.

Divs and spans

One element that can help add structure to a document is a div element. Many people mistakenly believe that a div element has no semantic meaning. However, div actually stands for *division* and provides a way of dividing a document into meaningful areas. So by wrapping your main content area in a div and giving it an ID of mainContent, you are adding structure and meaning to your document.

To keep unnecessary markup to a minimum, you should only use a div element if there is no existing element that will do the job. For instance, if you are using a list for your main navigation, there is no need to wrap it in a div.

```
<div id="mainNav">
  <ul>
    <li>Home</li>
    <li>About Us</li>
    <li>Contact</li>
  </ul>
</div>
```

You can remove the div entirely and simply apply the ID to the list instead:

```
<ul id="mainNav">
  <li>Home</li>
  <li>About Us</li>
  <li>Contact</li>
</ul>
```

Using too many divs is often described as *divitus* and is usually a sign that your code is poorly structured and overly complicated. Some people new to CSS will try to replicate their old table structure using divs. But this is just swapping one set of extraneous tags for another. Instead, divs should be used to group related items based on their meaning or function rather than their presentation or layout.

Whereas divs can be used to group block-level elements, spans can be used to group or identify inline elements:

```
<h2>Where's Durstan?</h2>
<p>Published on <span class="date">March 22nd, 2005</span>
by <span class="author">Andy Budd</span></p>
```

It's generally less common to need to group or identify inline elements, so spans are seen less frequently than divs. Where you will see spans used are effects such as image replacement, which use them as extra hooks to hang additional styles on.

Although the goal is to keep your code as lean and meaningful as possible, sometimes you cannot avoid adding an extra nonsemantic `div` or `span` to display the page the way you want. If this is the case, don't fret too much over it. We live in a transitional period and hopefully CSS 3 will give us much greater control of our documents. In the meantime, real-world needs often have to come before theory. The trick is knowing when you have to make a compromise and if you are doing it for the right reasons.

CSS comes in various versions, or "levels," so it's important to know which version to use. CSS 1 became a recommendation at the end of 1996 and contains very basic properties such as fonts, colors, and margins. CSS 2 built on this and added advanced concepts such as floating and positioning to the mix, as well as advanced selectors such as the child, adjacent sibling, and universal selectors. At the time of writing, CSS 2 was still the latest version of CSS, despite becoming a recommendation as long ago as 1998.

Time moves very slowly at the World Wide Web Consortium (W3C), so while work on CSS 3 started before the turn of the millennium, the final release is still a long way off. To help speed development and browser implementation, CSS 3 has been broken down into modules that can be released and implemented independently. CSS 3 contains some exciting new additions, including a module for multicolumn layout. However, the selectors module is nearest completion and could possibly become a recommendation as early as 2006.

Because of the expected length of time between the release of CSS 2 and CSS 3, work started in 2002 on CSS 2.1. This revision of CSS 2 intends to fix some errors and provide a much more accurate picture of CSS browser implementation. CSS 2.1 is slowly nearing completion but probably won't be finished until late 2006. But it does provide a much more accurate representation of the current state of CSS and is the version I currently use.

Document types, DOCTYPE switching, and browser modes

A document type definition (DTD) is a set of machine-readable rules that define what is and isn't allowed in a particular version of XML or (X)HTML. Browsers will use these rules when parsing a web page to check the validity of the page and act accordingly. Browsers know which DTD to use, and hence which version of (X)HTML you are using, by analyzing the page's DOCTYPE declaration.

A DOCTYPE declaration is a line or two of code at the start of your (X)HTML document that describes the particular DTD being used. In this example, the DTD being used is for XHTML 1.0 Strict:

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