



VISUAL
QUICKPRO
GUIDE

JASON CRANFORD TEAGUE

DHTML AND CSS ADVANCED

*Extend your DHTML and CSS
skills the fast, efficient way!*

*This Visual QuickPro Guide
uses illustrations and in-depth
explanations. You'll be a
master in no time!*

VISUAL QUICKPRO GUIDE

DHTML AND CSS ADVANCED

Jason Cranford Teague

Peachpit Press

1249 Eighth Street
Berkeley, CA 94710
510/524-2178
510/524-2221 (fax)
800/283-9444

Find us on the World Wide Web at www.peachpit.com.
To report errors, please send a note to errata@peachpit.com.
Peachpit Press is a division of Pearson Education.

Copyright © 2005 by Jason Cranford Teague

Editor: Kate McKinley
Production Coordinator: Andrei Pasternak
Proofreader: Elizabeth Welch
Compositor: Danielle Foster
Indexer: Julie Bess
Cover Design: The Visual Group
Cover Production: George Mattingly / GMD

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 Press 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

Visual QuickPro Guide is a registered trademark of Peachpit Press, a division of Pearson Education. All other trademarks are the property of their respective owners.

Throughout this book, trademarks are used. Rather than put a trademark symbol with every occurrence of a trademarked name, we state that we are using the names in an editorial fashion only and to the benefit of the trademark owner 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 0-321-26691-9

987654321

Dedication:

For Tara

There is something within your voice

Something behind your eyes

Something inside your heart

That is beating in time with mine

Special Thanks To...

Jocelyn and Dashiell, who keep my spirits up.
Mom, Dad, and Nancy, who made me who I am.

Uncle Johnny, for his unwavering support.

Pat and Red, my two biggest fans.

Nancy Davis, whose patience kept me from going stark raving mad.

Kate McKinley, who dotted my i's and made sure that everything made sense.

Liz Welch, for her proofreading prowess.

Andrei Pasternak, for keeping this moving through the Peachpit production process.

Danielle Foster, who pulled it all together to make it look this good.

Daniel **Rymer**, who wrote Chapter 10.

Kimberley Blessing, who coded Chapter 8.

Thomas Williams, who listened when I needed to talk

Neil **Salkind** and the good folks at Studio B for representing my best interests.

Charles Dodgson (a.k.a. Lewis Carroll), for writing Alice Through the Looking Glass.

John **Tennet**, for his incredible illustrations of Alice Through the Looking Glass.

Judy, Boyd, Dr. G and teachers everywhere who care. Keep up the good work.

Douglas Adams, Neil Gaiman, and Carl **Sagan** who inspired me to write.

the The, **Siouxsie** and the Banshees, Air America Radio, The Happytones, **Rammstein**, The Hives, Electro Cute, Danielle **Dax**, Morrissey, The Craig Charles Show, Republic of Loose, the **Pogues**, The Pixies, New Model Army, Cocteau Twins, Franz Ferdinand, Kate Bush, Mojo **Nixon**, Bauhaus, Lady Tron, Bad Religion, This Mortal Coil, John Peel, Monty Python, the Dead Milkmen, Johnny Cash, The Sex Pistols, Dead Can Dance, BBC Radio 6, Le Tigre, and ZBS Studios (for Ruby) whose noise helped keep me from going insane while writing this book

Safari

BOOKS ONLINE

ENABLED

This Book Is Safari Enabled

The Safari® Enabled icon on the cover of your favorite technology book means the book is available through Safari Bookshelf. When you buy this book, you get free access to the online edition for 45 days.

Safari Bookshelf is an electronic reference library that lets you easily search thousands of technical books, find code samples, download chapters, and access technical information whenever and wherever you need it.

To gain 45-day Safari Enabled access to this book:

- Go to <http://www.peachpit.com/safarienabled>
- Complete the brief registration form

Enter the coupon code GZFE-NF2E-YJ2C-9XEN-6F23

If you have difficulty registering on Safari Bookshelf or accessing the online edition, please e-mail customer-service@safaribooksonline.com.

TABLE OF CONTENTS

	Introduction	ix
Chapter 1:	Creating a Dynamic Web Site	1
	What Makes a Web Site Dynamic?	3
	Understanding Layout on the Web.	5
	Creating an Accessible Web Site	8
	Understanding Hypertext. Navigation. and Controls	10
	Navigation Dos and Don'ts	12
	Dynamic by Design	15
	The Four Ds of Web Design.	16
Chapter 2:	Mastering CSS	27
	The Myths of CSS	28
	Building Your Style Sheets	30
	Web Typography: Beyond Times and Arial	33
	Creating Web Pages for Print	37
	Building a Master Style Sheet	39
	Understanding CSS Shorthand.	41
	Using Grouping and Context	43
	Fixing CSS Browser Inconsistencies	45
	CSS Beyond Internet Explorer 6.	48
Chapter 3:	Advanced DHTML Techniques	55
	Setting Up and Accessing Arrays.	56
	Changing an Array	61
	Sorting an Array	64
	Working with Data Objects.	69
	Storing Data in Frames.	71
	Storing Data in URLs.	78
	Storing Data in Cookies	84
	Delaying or Stopping an Action.	91
	Handling Errors on the Fly.	94
Chapter 4:	Content	97
	Preloading Images	98
	Adding External Content with iframes	100

Adding External Content Using Server-Side Includes	102
Adding External Content with JavaScript.	104
Viewing Someone Else's External JavaScript or CSS.	106
Adding External Content with PHP.	109
Inserting a New Element	111
Inserting a New Text Element	114
Inserting a New iframe Element.	116
Removing an Element.	118
Including Random Content.	120
Including Multiple Pages in a Single Page.	122
Including a Clock	125

Chapter 5: Layout 129

Creating Simple Columns	130
Creating Balanced Columns	133
Creating Graphic Background Columns	137
Creating Contextual Layouts	141
Centering Layouts Horizontally and Vertically	145
Creating Curved Text Wrapping	147
Creating Curved Borders	150
Creating a Drop Shadow Around an Element.	154
Creating a Frame Drop Shadow.	158
Keeping Pages Framed	162
Switching Layouts on the Fly	166
Highlighting Table Rows	169

Chapter 6: Navigation 173

Working with Link Styles.	174
Creating an HTML Text Graphic Button	182
Creating Tabbed Navigation.	186
Adding a Simple Menu	189
Adding a Fixed Drop-down Menu.	192
Adding a Floating Menu Bar.	200
Adding a Clipping Menu	203
Adding a Jump Menu	212
Adding Pop-up Menus	215
Educating the Browser	220

Chapter 7: Controls 223

Creating Customized Browser Controls	224
Creating a Sortable Table.	228
Adding Font Size Controls.	233

	Creating a Scrollable Area	239
	Animating Scroll Controls	242
	Adding a Calendar Date Picker	249
	Adding QuickTime Video Controls	255
	Opening and Closing Frames	259
Chapter 8:	Forms	269
	Styling Forms	270
	Highlighting Form Fields	276
	Auto-Focusing Form Fields	282
	Performing Form Validation	286
	Creating Contextual Forms	290
	Creating Contextual Form Data	294
	Restricting a Form Field's Content	298
	Disabling Form Controls	301
	Creating Graphic Form Controls	305
Chapter 9:	Special Effects	309
	Creating Transparent Layers	310
	Creating Text Drop Shadows	314
	Floating Objects	317
	Adding a GIF Animation	320
	Adding Ambient Sound	326
	Creating Transparent Graphics in PNG Format	328
	Special Effects in Internet Explorer for Windows	333
Chapter 10:	Databases	339
	Understanding XML	340
	Accessing XML Data in Internet Explorer for Windows	342
	Accessing XML Data with Mozilla and Internet Explorer	345
	Understanding MySQL	349
	Creating Database Objects in MySQL	350
	Understanding PHP	352
	Using PHP with MySQL	353
	Adding Comments to the PHP Blog	356
	Adding a Comments Input Form	360
	Adding an Administrator Page	366
	Creating a Secure Blog Entry Input Form	371
Appendix A:	CSS Quick Reference	375
	Adding Styles	376
	Selectors: HTML. Classes. IDs	378

	Grouping Styles.....	379
	Properties and Values.....	380
Appendix B:	DHTML Quick Reference	389
	The DOM.....	390
	Events.....	391
	Properties and Values.....	392
	Reserved Words.....	395
Appendix C:	WAI Accessibility Checklist	397
	Priorities	398
	Priority 1 Checkpoints	399
	Priority 2 Checkpoints.....	401
	Priority 3 Checkpoints.....	403
Appendix D:	Browser-Safe Fonts	405
	Index	411

INTRODUCTION

When a page loads in my browser window, the first thing I want to figure out is "How did they do that?" Whether it's tabs using HTML text, a menu bar floating down the page, or drop shadows behind text, I am constantly discovering new ideas for Web design and user experience. Almost invariably, though, the answer to my question involves cascading style sheets or dynamic HTML.

CSS and DHTML (along with HTML and JavaScript) form the core skill set of every Web designer, yet mastering these two technologies takes more than knowing the basics of how they work. The problem is that, though both HTML and JavaScript have been relatively stable for several years now, CSS and DHTML have only in the last couple of years become standardized technologies that can be counted on to behave in a reliable manner on different browsers.

Today's Web designers, whether their approach is that of a programmer or a visual designer, need to draw on more than just a bag of tricks. They need ideas for treating designs that look great and can be easily updated. They need ideas for creating a superior user experience that puts the visitor in control. They need ideas for linking complex data streams to their Web pages that will turn data into knowledge.

If that sounds like you, then have I got the book for you.

Who This Book Is For

It may sound like an oversimplification, but there are two basic groups on the stage when creating Web sites: designers and programmers. I know, I know, there can be a lot of gray in between, and depending on the size of the project many people may be wearing many different hats. But it's my experience that these two groups will approach the problems at hand with different mind-sets.

Designers are concerned with the users' needs and how visitors will interact with the Web page. Designers look for ideas that will enhance the visitor's experience even if those ideas take them beyond what is possible.

Programmers, on the other hand, are most often concerned with what can be done. They start from a basis of the possibilities (and limitations) of the system, and want to find the simplest, yet most versatile, solutions to the problems at hand.

So who is this book for? Designers are often put off by books that are intended for programmers, and programmers are often not interested in the elementary approach of some books intended to teach designers. However, there is a common ground between them. Regardless of which side of the fence you are on, the end goal is to make sure that the visitor can get the information they need when they need it as simply as possible.

This book strives to strike a good balance between the needs of designers and programmers: to provide ideas you can incorporate into your design and improve the user experience, as well as solutions that you can apply to a wide variety of situations.

What This Book Is About

In many ways this book is Part II to *DHTML and CSS for the World Wide Web*. In fact, when I started revising that book for its third edition, I realized that I had too much material to fit it all in. So Peachpit Press and I decided to take the advanced material out of that book, add the new material, and put it all in a new book. You are now holding the result.

The first three chapters in this book will help you to become a better Web designer. Whether you come from the programming or design side of the World Wide Web, these chapters will help you understand not just the *how* of Web design but also the *why*.

The last seven chapters are broken down into broad categories of user interface design:

- **Content:** Content may be king, but you are still the boss. Learn how to get the content you want when you want it.
 - **Layout:** Everything from columns to curved corners is covered.
 - **Navigation:** There's more than one way to get around the Web. Learn how to create different menu types and get a few pointers on jazzing up buttons.
 - **Controls:** Give visitors the power to create the Web page that best suits their needs.
 - **Forms:** Why stick with boring, lifeless form fields? Instead, you can create forms that always let visitors know what they're doing and what they've done.
 - **Special effects:** Some fun stuff, some useful stuff, and one or two stupid tricks.
 - **Databases:** Turn data into knowledge using XML, MySQL, PHP, and a little CSS and DHTML.
-

Values and Units

Throughout this book, you'll need to enter different values to define different properties. These values come in various forms, depending on the need of the property. Some values are straightforward—a number is a number—but others have special units associated with them.

Length values

Length values come in two varieties:

- ◆ Relative lengths, which vary depending on the computer being used (**Table i.1**).
- ◆ Absolute values, which remain constant regardless of the hardware and software being used (**Table i.2**).

I generally recommend using pixels to describe font sizes for the greatest stability between operating systems and browsers.

Color values

You can describe color on the screen in a variety of ways (**Table i.3**), but most of these descriptions are just different ways of telling the computer how much red, green, and blue is in a particular color.

Percentages

Many of the properties in this book can have a percentage as their value. The behavior of this percentage value depends on the property being used.

Table i.1

Relative Length Values			
NAME	TYPE or UNIT	WHAT IT IS	EXAMPLE
em	Em dash	Width of the letter M for that font	3em
ex	x-height	Height of the lowercase x of that font	5ex
px	Pixel	Based on the monitor's resolution	125px

Table i.2

Absolute Length Values			
NAME	TYPE or UNIT	WHAT IT IS	EXAMPLE
pt	Point (1pt. = 1/72 in.)	Generally used to describe font size	12pt
pc	Picas (1pc. ff 12pt.)	Generally used to describe font size	3pc
mm	Millimeters		25mm
cm	Centimeters		5.1cm
in	Inches (1 in. = 2.54 cm)		2.25in

Table i.3

Color Values		
NAME	WHAT IT IS	EXAMPLE
#RRGGBB	Red, green, and blue hexadecimal value of a color (00–99, AA–FF)	#CC33FF or #C3F
rgb(R#,G#,B#)	Red, green, and blue numeric values of a color (0–255)	rgb(104,51,255)
rgb(R%,G%,B%)	Red, green, and blue percentage values of a color (0–100%)	rgb(81%,18%,100%)
name	The name of the color	Purple

URLs

A uniform resource locator (URL) is the unique address of something on the Web. This resource could be an HTML document, a graphic, a CSS file, a JavaScript file, a sound or video file, a CGI script, or any of a variety of other file types. URLs can be local, simply describing the location of the resource relative to the current document; or global, describing the absolute location of the resource on the Web and beginning with `http://`.

In addition, throughout the book, I use links in the code examples. I used the number sign (#) as a placeholder in links that can be directed to any URL you want:

```
<a href="#">Link</a>
```

The number sign is shorthand that links to the top of the current page. Replace these with your own URLs as desired.

However, in some links, placing any URL in the `href` will interfere with the DHTML functions in the example. For those, I used the built-in JavaScript function `void()`:

```
<a href="javascript:void('')">Link</a>
```

This function simply tells the link to do absolutely nothing.

Reading This Book

For the most part, the text, tables, figures, code, and examples should be self-explanatory. But you need to know a few things to understand this book

The code

For clarity and precision, I have used several layout techniques to help you see the difference between the text of the book and the code.

Code looks like this:

```
<style>
p { font-size: 12pt; }
</style>
```

All code in this book is presented in lower-case. In addition, quotes in the code always appear as straight quotes (" or '), not curly quotes (“ or ’). There is a good reason for this distinction: Curly quotes (also called smart quotes) will cause the code to fail.

Important lines of code that I discuss in the accompanying section are emphasized by appearing in red in the code sample.

When you type a line of code, the computer can run the line as long as needed, but in this book, lines of code have to be broken to make them fit on the page. When that happens, I use this gray arrow () to indicate that the line of code is continued from above, like this:

```
.title [ font: bold 28pt/26pt times,
+serif; color: #FFF; background-color:
#000; background-image: url(bg_
title.gif); }
```

I often begin a numbered step with a line of code. This is intended as a reference to help you pinpoint where that step applies in the larger code block that accompanies the task.

Web Sites for This Book

All of the code presented in this book can be viewed online and downloaded from www.webbedenvironments.com/dhtml_css_advanced/.

You have DHTML questions? I have DHTML answers. You can contact me at vqp-dhtml@webbedenvironments.com.

CREATING A DYNAMIC WEB SITE

1

A line is defined by two points; call them A and B. All you can do with a line is travel straight from point A to point B, without deviation or interruption. Traditionally, most media that communicate information are linear. You can fast-forward through a video or flip through a book, but these media are best suited for presenting ideas in a linear order that is set by the author, one idea following the next.

Human beings, however, tend to reason in more chaotic and intricate ways than mere lines can ever describe. We learn by exploring, searching, and finding; by making mistakes and trying again. We rarely think in a straight line; we think dynamically (**Figure 1.1** on next page).

The Web has forever changed the way we look at information and how we structure knowledge. Despite the great leap forward that hypertext represents, we still have a way to go before mastering this new medium. Before it can more accurately mimic the way humans think, the Web must become far more dynamic.

In order to create the best dynamic Web sites, though, you need to know more than how to create code; you need to understand not only "how" but also "why" and "when." In this chapter, we'll look at what it means to create a dynamic Web site, how to think about navigation (a key component of any Web site), and then how to design a dynamic Web site including a four-step process that you can follow to make sure you're covering all of your bases before you go live on the Web with your work.

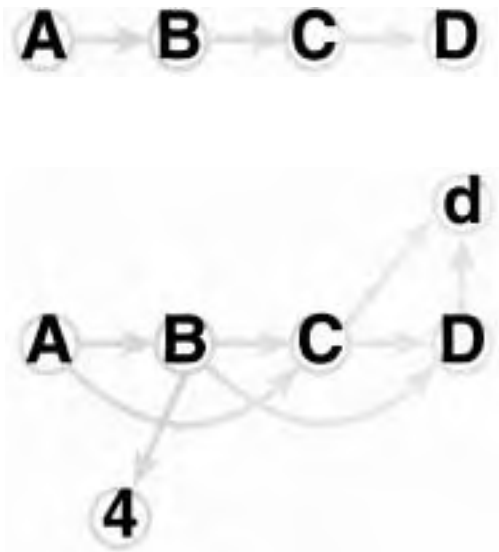


Figure 1.1 The linear path from A to B to C to D is fairly typical of how traditional media, such as books and movies, work. The dynamic model lets you jump to any point without passing through the intervening points, and can offer alternative versions or link to outside information.

What Makes a Web Site Dynamic?

Dynamic site can mean many things, depending on whom you ask (and maybe even on when you ask them). However, at its simplest, it means that information is presented actively, so the viewer participates in some way rather than merely being a passive spectator.

The Web is a naturally dynamic medium, because of hypertext. With hypertext, Web designers can link information and facilitate a more natural human learning environment. With the addition of CSS, JavaScript, and the Document Object Model (DOM), designers can make a Web site still more dynamic.

Unfortunately, more than a decade since the first Web sites started popping up on the Internet, most sites are still relatively static. To be truly "dynamic," a Web page should meet one or more of the following criteria (Figure 1.2):

- ◆ **Interactivity.** A dynamic Web site should allow the visitor to find their own path to the information they are looking for, rather than simply sitting passively receiving information. By taking actions and performing tasks, the visitor becomes engaged on multiple levels, thus creating a more robust experience.
- **Synchronicity.** A dynamic Web site should bring together relevant information and activities from a variety of sources—either directly or through linking—at the time the visitor needs them. An example is when a visitor chooses from a list of countries and is immediately presented with a list of cities in that country.

continues on next page

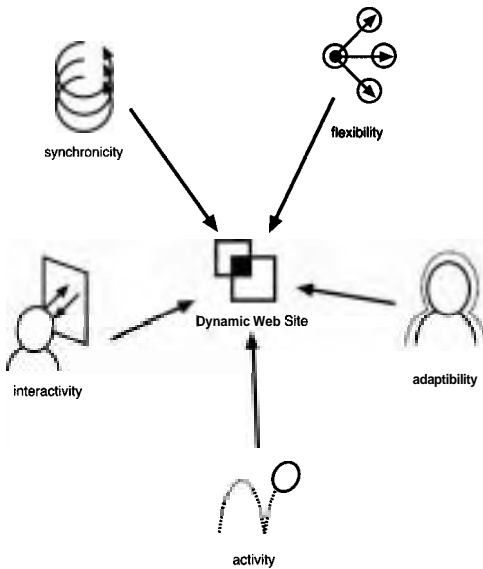


Figure 1.2 Key elements of a *dynamic* Web site.

- ◆ **Flexibility.** A dynamic Web site should give the visitor a variety of ways to find information or accomplish tasks so that they can choose the method that best suits their needs. For example, you may present menus and search fields to navigate your site, allowing visitors to choose the method that works best for them.
- ◆ **Adaptability.** A dynamic Web site adjusts to cater to the individual visitor's needs. Sometimes this adjustment is made on the server through customization of content, but Web designers can do much to accommodate visitors without requiring them to load a new Web page. For example, menus that slide in and out from one side of the screen are always available to visitors who are constantly moving around, but can be hidden to maximize the content display area.
- ◆ **Activity.** A dynamic Web site uses motion and sound to draw attention to changes on the screen. Often presented in the form of animation, activity is one of the most difficult dynamic principles to apply affectively. Everyone is familiar with animated GIFs or Flash animations used in advertisements to draw the visitor's attention. This kind of "dancing baloney" frequently disrupts the Web experience and actually creates a more passive experience for the visitor, who sits and watches the activity without taking any action (unless it is to click the Skip button). However, activity can be an integral part of the dynamic experience if used to reinforce interactivity, for example, if you use animation in a link's rollover state to create a more pronounced effect.

Understanding Layout on the Web

Because of the Web's expandable windows, unpredictable screen resolutions, and variable font sizes, it may feel like you have a better chance of predicting the price of Internet stocks than the final appearance of your Web design. You know that your design must fit into a rectangle (the browser window), but will that rectangle be wide and long enough?

All Web layouts have two basic parts. The first part is the content area, which features navigation, titles, graphics, and text—in other words, the stuff visitors are interested in. The other part is filler. Whether the filler is just empty space or a design that fills the void, it is there simply to absorb extra space in the browser window. The balance between content and filler is crucial to creating an attractive Web layout.

Based on this balance, there are four broad categories of Web layout, and all Web designers must start their designs by deciding which of the four to use.

- Unrestrained. The content is allowed to stretch horizontally from the left edge to the right edge of the window and vertically down to the bottom edge of the window (Figure 1.3). This design eliminates the need for filler. But wide columns of text, often a symptom of unrestrained layout, can be difficult to read. The online store Powell's Books (www.powells.com) uses unrestrained layout (Figure 1.4). The content in the center and right columns is allowed to grow as wide as necessary to fill the empty space.

continues on next page

Content

Figure 1.3 The unrestrained Web-page layout.



Figure 1.4 Powell's Books uses an unrestrained layout that stretches to fill the entire browser window.

◆ **Fixed width.** The content is given a set margin on the left side, the right side, or both, restraining the content horizontally (**Figure 1.5**). Salon (www.salon.com) restrains the width of its content area, centering it in the window with a fixed width of 736 pixels. This is roughly the width that an 800-pixel-wide monitor can comfortably display without horizontal scrolling if the window fills the entire screen. The design uses white space to fill additional areas to the left and right (**Figure 1.6**).

◆ **Fixed height.** The content area is given a set margin on the top, the bottom, or both, restraining it vertically (**Figure 1.7**). Therefore, the content is forced to scroll horizontally. This design is rarely used, however, because it usually is advantageous to maximize the height in which the content can appear. The experimental Web site CSS Zen Garden (**Figure 1.8**) includes several fixed-height designs, including one by Clement Hardouin (www.csszengarden.com/?cssfile=048/048.css).

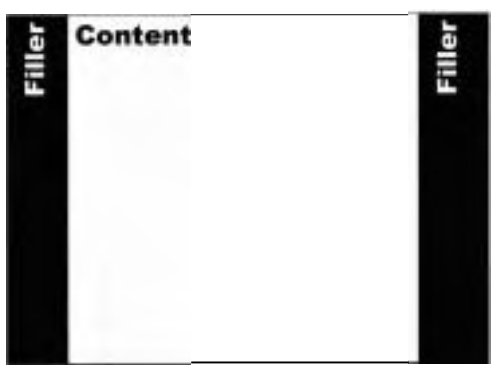


Figure 1.5 The fixed Web-page layout.



Figure 1.6 Salon uses a fixed-width layout to keep the content in one central column with filler on either side.



Figure 1.7 The fixed-height Web-page layout.



Figure 1.8 Clement Hardouin uses a horizontal layout for his contribution to the CSS Zen Garden.

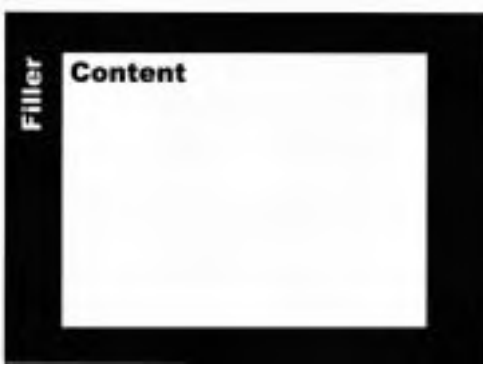


Figure 1.9 The fixed-size Web-page layout.



Figure 1.10 WonderWorld uses a fixed-size layout.

Fixed size. Both the height and width of the display area are restrained to a fixed size (Figure 1.9). You can create this layout in two ways. The first way is to surround one central frame with other frames that expand around it. The second is to create a layer and center it in the page (see "Centering Layouts Horizontally and Vertically" in Chapter 5, "Layout"). I used a fixed width and height with a site I created for WonderWorld (www.wonderworldfilmvideo.com), a film and video company (Figure 1.10). This provides the greatest control of the dimensions that the design will fill, but means the page displays at the same size regardless of the size of the visitor's screen.

Where in the World...

They don't call it the *World Wide Web* for nothing. As if you don't have enough to worry about, yes, anyone on the entire planet can view your Web site. In fact, astronauts on the space station have Net access, and there is even talk of establishing domain names for Mars (no, really!)

The good news is that as long as you follow the W3C standards, you don't have to worry unduly.

The W3C has established several Internationalization Task Forces to ensure that existing W3C recommendations work across geographic borders.

Creating an Accessible Web Site

Unless you have a specific obstacle to using the Web, you may not think about making your Web site accessible. But for some visitors, the Web can be as frustrating as trying to climb a flight of stairs in a wheelchair. For example, blue/green color-blindness may make it impossible to locate links if you rely solely on color to distinguish them. Visitors with more severe visual impairment may rely on electronic screen readers, which read every word on the page but not graphic elements—leaving them stranded if your navigation is graphics-based.

Accessibility is an increasing concern for many Web site designers, especially those creating sites for a wide audience. In fact, the Section 508 legislation (www.section508.gov) says that federal agency Web sites and any companies receiving federal funds for their Web sites must be accessible (Figure 1.11).

So what is a Web designer to do? Turn to our good friends at the World Wide Web consortium (W3C), of course (www.w3c.org).

Web content accessibility checklist

The Web Accessibility Initiative (WAI) was established by the W3C to create guidelines for Web designers struggling to create accessible Web sites (Figure 1.12). Some of these guidelines may seem a little unreasonable and may cramp your design, but remember; they are guidelines, not laws. On the next few pages you will find a synopsis of the guidelines, while Appendix C provides the full checklist to use for evaluating how well your site conforms to these guidelines.

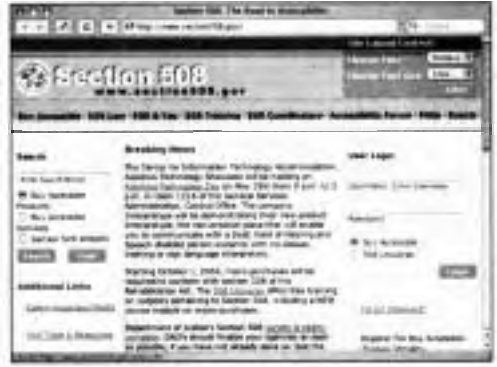


Figure 1.11 The U.S. Government's Section 508 Web page may become a common stop if you are designing anything that has to be universally accessible.

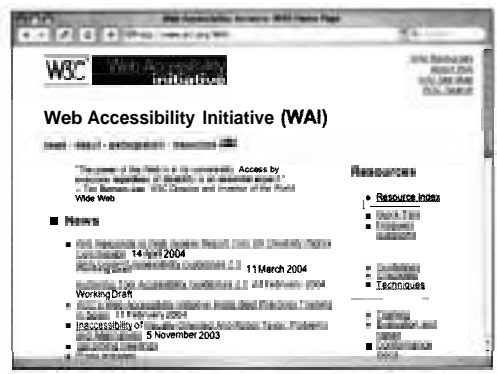


Figure 1.12 The W3C's Web Accessibility Initiative section offers guidelines for how to ensure that your Web site is accessible.

- 1. Provide equivalent alternatives to audio and visual content.** Always include an `alt` attribute in image tags and `desc` or `longdesc` attributes in your tables and frames.
- 2. Don't rely on color alone.** If you are using color in your design to convey information, make sure the information is also conveyed in some other manner. It's also important that your background and foreground colors provide enough contrast for color-blind visitors (no green text on blue backgrounds!).
- 3. Use markup and style sheets and do so properly.** Separate your styles (CSS) from your structure (HTML) and stick to the standards (see guideline 11). In addition, the guidelines recommend that you use relative rather than absolute values (in other words, *em* and % rather than *pt* and *cm*). This allows the visitor to enlarge pages (especially text) for easier reading.
- 4. Clarify natural language usage.** Identify the language the page is using and use the `<abbr>` and `<acronym>` tags to define abbreviations and acronyms.

```
<acronym title="Hypertext Markup  
· Language">HTML</acronym>
```
- 5. Create tables that transform gracefully.** Define table headers using `<th>` rather than `<td>` tags and avoid using tables for layout. Use CSS instead.
- 6. Ensure that pages featuring new technologies transform gracefully.** If your design relies on a specific technology, such as CSS or JavaScript, it should still be accessible if these technologies are not available to the browser.
- 7. Ensure user control of time-sensitive content changes.** Some visitors will have trouble reading moving, blinking, or scrolling content. Provide controls to pause or stop this content.
- 8. Ensure direct accessibility of embedded user interfaces.** If you are using the `<embed>` tag to include content such as QuickTime movies or Flash, make sure that these interfaces also conform to these guidelines.
- 9. Design for device-independence.** The Web is now accessed by more than just computers. Your content may be viewed with mobile phones, PDAs, and other devices and you need to ensure that your interface will work in these.
- 10. Use interim solutions.** Keep in mind that older browsers and screen readers behave differently from their modern counterparts, and you may need to adjust your design for these legacy technologies.
- 11. Use W3C technologies and guidelines.** This one is a little redundant (these guidelines are set by the W3C, after all) and may not always be practical since even the most current version of Internet Explorer is not fully W3C compliant. But always try to stick to the standards where practical.
- 12. Provide context and orientation information.** Use the `title` attribute with the `<frame>` tag and the `for` attribute in the `<label>` tag to help describe elements on the page.
- 13. Provide clear navigation mechanisms.** Always provide clearly labeled and consistent navigation on your Web pages (see "Navigation Dos and Don'ts" later in this chapter).
- 14. Ensure that documents are clear and simple.** Keep the style and language of content consistent across your entire Web site.

Understanding Hypertext, Navigation, and Controls

Many people confuse the terms *hypertext*, *navigation*, and *controls*. This is understandable, since on the Web, hypertext and navigation links are both created with the same HTML code: `<a> . . . `. But navigation is used to move from page to page within a Web site. Controls are used to let the visitor perform a specific action, such as submitting a form. Hypertext, on the other hand, may be any word, phrase, or image in a Web page that you click to get a more detailed explanation of that thing from anywhere on the Web. In theory, hypertext allows a single page to include a nearly infinite amount of information (**Figures 1.13 and 1.14**).

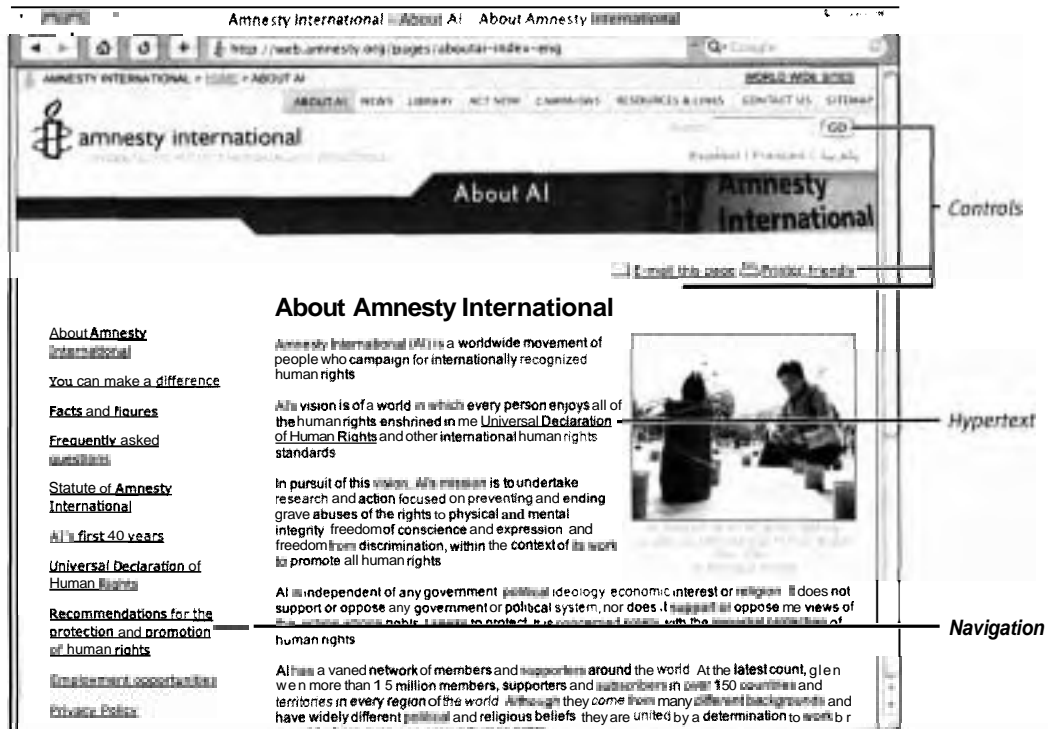


Figure 1.13 The Amnesty International site provides a good example of the difference between controls, navigation, and hypertext. The hypertext link in the main text offers access to a more detailed description of the term, and the navigation links move the visitor to another item on the site.



Figure 1.14 Clicking the hypertext link takes you to a page that defines the Universal Declaration of Human Rights.

Navigation or Controls?

It is vital not to confuse navigation with controls. Controls may move the visitor to another page (as with a form submission), but it first performs a specific action initiated by the visitor.

To illustrate this point, I was doing user testing on a Web site that included functionality to send ecards. The link in the menu to navigate to the page to send the card was labeled Send eCard.

Once on the page, the visitor filled out the information and then clicked a link labeled Send eCard to complete the transaction. However, the navigation link was still on the page in the menu. I found that visitors often confused the navigation link Send eCard with the control link Send eCard and kept resetting the page—leading to a frustrating experience,

The lesson here is to clearly distinguish links used for navigation from those being used to give commands. You should not only differentiate visually (in the case mentioned above, the navigation and control looked completely different) but also textually.

Web designers have three basic goals:

- ◆ To help visitors find the information they are looking for as quickly and efficiently as possible through clear navigation.
- ◆ To facilitate visitors' interactions (such as submitting form data) by providing unambiguous controls.
- ◆ To enable visitors to explore, define, and discuss information by providing further information using hypertext.

Believe it or not, that's pretty much what all interaction design comes down to.

- ◆ **Navigation:** These are links used to move the visitor between different general sections and pages within the site, such as Home, About Us, or Subscriber Services.
- ◆ **Controls:** These are links or buttons used to perform a specific action, such as submitting a form (which will also move the visitor to a new page), or opening and closing menus or other windows.
- ◆ **Hypertext:** These are links used to display additional information about the specific text being linked. For example, the link may be to the definition of the word/phrase or it may be a link to further information about a 50% off deal for a London vacation.

Navigation Dos and Don'ts

Before you can start thinking about making your Web pages more dynamic, it is important to first master the basic dynamic capability of the Web: navigation. Unfortunately, many designers are still addicted to simply pouring links into a page without giving much thought to how the visitor will interact with them. After years of my own experimentation, I have learned a few dos and don'ts that I can pass along.

Dos

Some of these points may seem obvious, but when you're in the middle of a project, it is easy to lose sight of even the most obvious considerations.

- ◆ **Keep navigation consistent.** This point is fairly straightforward: After you establish the general navigation structure of your site, don't move it around or radically alter it on each page. If the main navigation is on the left side of the screen, do not suddenly move it to the right on the next screen. If you placed the auxiliary navigation on the bottom of the screen, do not suddenly start placing banner ads in that area.

Too many sites play "Where's Waldo?" with their navigation. The best way to keep navigation consistent is to simply fix a menu bar on the site that stays in the same place regardless of which page the visitor is on or where they have scrolled to.

There are some exceptions, however. Home pages can have a different navigational layout from the rest of the site, just as the cover of a magazine has a different layout from the pages inside.

- ◆ **Let visitors know where they are and how they got there.** Remember Hansel and Gretel following their pebbles back home from the woods? Marking your path can keep you from getting lost and help you find your way back to where you started.

The most common way to do this is to list the pages (usually starting with the home page) that visitors traveled through to get where they are, as follows:

*Home » Forest » More forest »
Witch's house*

The path shows a logical progression, and each title is actually a link back to that page. Like Theseus following the thread back through the Minotaur's maze, visitors to your site can retrace their paths and go in other directions.

- ◆ **Show and tell if the navigation is vague.** It's nice to know where you are and how you got there. It's also nice to know where you are going. Visitors who are new to a site, however, may not understand some of the vague terms you use to describe the navigation.

One of the chief frustrations of many Web surfers is following the wrong path when they're looking for specific information. This frustration is compounded by having to wait to load pages that were not what they really wanted. Providing a sentence or two of explanation to tell visitors what they can expect to find behind the link helps lessen this problem. You may not have the screen space to provide a thorough explanation, but DHTML can come in handy for this, providing pop-ups, rollovers, and other methods for dynamically showing content.

◆ **Give the visitor control and flexibility.**

Not everyone works the same way. Most software these days allows users to move control palettes around on the screen, add and subtract tools, and generally tailor the interface to their needs.

The Web limits what you can do to give visitors control of the interface, but I like to offer a few things such as pull-out and collapsible menus in a panel.

+ **Give visitors somewhere to go next.**

Recently, I was reading an article in a popular Web magazine for information architects. When I reached the end of the article, I found that I could go backward and forward through the pages of the article, but there was no link back to the actual magazine or to any other part of the site.

There is nothing more irritating on a Web site than a dead end. This situation occurs with alarming regularity when you fill out forms online: You enter all your information and click Submit; a page comes up with a message thanking you for entering your information, and nothing more happens. Often, these pages don't even include the site's standard navigation.

You need to always provide link options on every page so that visitors don't hit a dead end and leave your Web site.

Don'ts

Now that you know what to do, here are some things you should avoid when designing your Web site navigation.

◆ **Don't waste space with navigation.**

Any link on a page that visitors are not interested in clicking is wasted space on the screen. Yet many Web designers fill the pages of their sites with links that visitors did not click on the home page and are unlikely to click at all.

Consider a banking site that offers services to individuals, small businesses, and corporations. The front page presents all three options as starting points that take visitors to the diverse services available in each category. But links to all three areas remain on subsequent pages. Why would someone who is interested in an individual banking account want to switch over to the corporate services? Even in this rare case, that person could always return to the home page, which should never be more than a click away.

One common way of giving visitors the maximum number of links without wasting space is to use drop-down or clamshell menus, which allow you to organize your links into main topics listed in a single menu. When one of the topics is selected, a submenu of related topics appears below the main topic, but the full menu of topics is still available.

◆ **Don't lose the navigation.** Imagine that you are typing a letter in your word processor. As you reach the bottom of the typing area, the menu bar scrolls off the top of the screen. Every time you need to access the menu bar, you have to scroll back up to the top of the page. This would make for a frustrating user interface, but it's exactly what most Web designers do.

continues on next page

One way around this problem is to include the ubiquitous "Return to top" links, but this solution gets clunky. I prefer to put my navigation in frames so that it is always available and in a consistent position on the screen. Visitors do not have to monkey around searching the page and wasting their valuable time. It is important, of course, to minimize the size of these frames to maximize the area available for the content.

◆ **Don't confuse navigation types.** Not all navigation is created equal. Web pages can contain many types of navigation, depending on a variety of factors. Visitors use the main navigation to travel among the most important areas of the site.

◆ **Don't rely on the browser's controls for navigation.** Many visitors depend on the built-in browser controls. They are comfortable with the way these controls work, and they know what to expect when they click the Back or Forward button. But at least as many visitors never think to touch the controls at the top of the browser window, and if they can't get where they want to go on your site, guess who gets the blame?

Make sure visitors can always get back to where they were in your Web site with a minimum of fuss. Never lead them down a dead end and then tell them to use the browser's Back button. Instead, give them options or at least include a link back to the page they just came from.

◆ **Don't put every link on every page.**

One of the most common mistakes that Web designers make is placing every possible link on every page of the site, in the mistaken belief that visitors may want to go anywhere at any time. Not only does this waste space, but, more importantly, it can make it harder for visitors to find what they're looking for (see the sidebar "Snowflake in a Blizzard").

Any link that visitors never click is wasted space. If visitors did not select the link on the home page of the site, the chance that they will click it on subsequent pages goes way down. This is not to say that visitors always follow a linear path through the site to their goal; you can expect visitors to skip around looking for what they want. But you should organize the site so that visitors can move among subjects quickly and then move to more-detailed subjects without having to see all the detailed links at the same time.

What Is the Web Good For?

In practice, including infinite information is not feasible, but you can include a great deal of information for the visitor to explore. Authors (the ones who provide the content) control what is and what is not referenced through a hypertext link and they have to consider every link they create.

Rather than being a medium for discourse—such as a book, through which one person speaks to many people—the best Web sites use the Web as a medium for intercourse (no, I'm not talking about the porn sites).

Using the Web, many people can learn from and speak to one another. Otherwise, what is the point of the Web? Conversation, video, audio, and text can all be provided on the Web but can be provided better via telephone, TV, radio, and magazines.

Dynamic by Design

People probably are not coming to your Web site to see cool special effects; they are coming to your site for information. Your design, dynamic or not, needs to support their information needs. A dynamic Web site, if properly designed, is always better than a static one as it can provide the visitor with greater versatility. However, a poorly designed Web site with dynamic elements can simply be painful to use. In the next section you will learn a four-step process for creating a dynamic Web site, but while you are working, keep in mind these points:

- ◆ **Keep it simple.** For many people, dynamic content means added complexity—more options, more functions, more to learn, and more to remember. Generally speaking, people do not want more options; they want what they want when they want it. You should use dynamic features to simplify the use of your site, not to add complexity.
- ◆ **Show only relevant information.** Information becomes knowledge when its relevance is understood. But information that is not immediately relevant can be distracting, and too much information can be just as confusing as too little. Use DHTML to show and hide relevant information as necessary.
- ◆ **Make changes clear.** We often do not notice changes in our environment, even when they occur right in front of our faces. Dynamic changes in the content of a Web page should be initiated by the visitor, should occur almost instantly, and should be easy to recognize.

- ◆ **Provide a sense of location and direction.** One common complaint about the Web is that it's easy to get lost. Compared with the real world, where we can turn to see where we came from and look ahead to see what is next, most Web pages seem to be very insular. Use DHTML to let visitors know where they came from and where they are going.
- ◆ **Direct, don't dictate.** The point of a Web site is to allow visitors to move freely within the content. As the author and designer, you want to direct visitors to the information you want them to see but simultaneously allow them to follow their own paths. In terms of what you are communicating to your audience, the links that you do or do not include in a Web page are as important as the words and graphics you put there. Think of yourself as a guide, not a tyrant.

The Four Ds of Web Design

All Web sites need a plan before they can be built. This plan works as a blueprint for the site, to help you define the structure around which all the fixtures of the site (graphics, text, and code) will be placed.

A large Web site built without the benefit of a blueprint will fail as surely as a large building built without the benefit of an architectural blueprint. But even if you are building only a garden-shed-size Web site, a good plan will save you time and aggravation in the long run.

Including dynamic content in this equation makes a good plan even more necessary. Because dynamic content adds more options, it adds more possibilities and more variables that can go wrong. A little forethought can go a long way.

In this section, I'll show you the steps to take to define, design, develop, and deploy your Web site (**Figure 1.15**).



Figure 1.15 To show you how to create a dynamic site, I'll be using the work I've done for the site that supports DHTML and CSS for the World Wide Web.

Snowflake in a Blizzard

The Web's distinguishing feature is its ability to link fragments of information so that visitors can understand the relationships among them. This feature is called *hypertext*, and it's what separates the Web from books, TV, radio, cinema, telephony, CDs, and even from CD-ROMs.

So, why are Web sites so hard to navigate? Some designers provide only vague navigation or fill their pages with graphics and banner ads that take forever to download. Even worse are the pages that include links to every other point in the site, so finding the one you want is like finding one snowflake in a blizzard. One reason so many Web sites present all this information is that the only way to change the content of a Web page used to be to load a new Web page with the additional content. While the new page loaded, visitors had to wait...and wait...and wait. Rather than forcing visitors to wait while they drilled down through several pages to find the information they wanted, designers saved time by placing all these links on the same page.

DHTML changes all that. Because DHTML allows you to show and change objects dynamically, you can tailor the content of the Web page for visitors based on their immediate needs. A variety of content can be loaded and then revealed without the long wait involved in loading new Web pages.



collect your content



know your audience



establish your goals



plan features and interactivity



consider the graphic design style

Figure 1.16 Define your Web site.

Step 1: Define

The Web is about information, and before you can begin your Web design, you have to collect, identify, and distill the information that will make up your site.

Designing a dynamic Web site requires a good deal more effort than simply stringing together a series of Web pages with links, but the rewards will be worth it.

Whether you are designing a Web site for your daughter's class play, a portfolio of photographs, or an intranet that allows insurance underwriters to process claims, you need to define what you are doing before you turn on your computer and begin coding (Figure 1.16).

Collect and review your content

This step may seem obvious, but you would be surprised how many sites begin their lives before the designers know what is going into them. While you're doing this, ask yourself several questions:

- ◆ **What is your site's message?** Your site's message will play into everything from the visual design you choose to the features and functionality you decide to include.
- ◆ **How can you group different types of information together?** How will visitors navigate all the information you present? How can similar chunks of information be grouped to create the different sections of the Web site? What should those sections be called, and how should they be represented? This decision will play an important role when you map out the site and determine its structure.

continues on next page

◆ **What kinds of content will you be presenting?** The answer to this question will determine a lot about the final design. This is an important consideration with a dynamic Web site, since you need to start thinking now about the best ways to present that content.

The content for your Web site includes not only the text, but also any graphic assets such as logos, photographs, or other images.

Know your audience

The type of content will determine the type of visitor who wants to view it. This may be the most important factor in determining a demographic for your site visitors. The more accurately you can define the content, the more accurately you can define the audience. In addition, the content will help determine the site's look and feel. The audience for a computer store's Web site, for example, probably will not expect to see a floral pattern in the background.

After you determine the general audience for your site based on content, try to answer the following questions, based on your experience or direct observation:

- ◆ **What type of equipment will visitors be using and where?** It is important to understand whether your audience will be on a slow or fast Internet connection and what type of computer they are using, since this can tell you a lot about the speed at which they'll experience your Web site.
- ◆ **What type of browser is the visitor using?** Although virtually all Web browsers today support DHTML and CSS, their actual capabilities differ. Identify and define the browsers you are designing for before you begin putting your site together.

◆ **What do you hope the visitor will achieve at this site?** It's not enough to simply provide information. Consider why people will be coming to your Web site and what will be attracting them there in the first place.

A single site might — and probably will — have many purposes. Set up different scenarios for several potential visitors who have different reasons for coming to your Web site. Map out how visitors might get from their entry point (probably the home page) to their goal, and use this information to optimize the navigation for your site.

Establish your goals

Now that you know who will be visiting your little corner of the Internet, it's time to solidify your goals. Don't confuse your goals for the Web site with the visitors' goals for coming to the Web site. Think about the following questions:

- ◆ **What do you expect to achieve by deploying the Web site?** Again, this is not what the visitor will be achieving, but what you want to achieve. You may want to increase sales, explain a new scientific theory, or just share your thoughts with the world. More than likely, though, you will have several goals you want to achieve.
- ◆ **What long-term goals do you hope to achieve by maintaining the Web site?** Beyond what you want to achieve with the immediate deployment of the Web site, think about what you hope to achieve during the life of the Web site. This can help point you in the direction of future enhancements.

- ◆ **Is what you are doing worth the time and money you are expending to achieve it?** This may take some analysis, but it's important to know at the start whether what you are doing will be cost effective for what you want to achieve.

For some Web sites, clients dictate the goals; for others, the goals may be more personal. Regardless, you must know what you are trying to accomplish with all your hard work

Plan the features and interactivity

Now it's time to identify the features you will be using to accomplish your goals. At this point, you should have already built up a strong impression of the content, audience, and goals for the site. This information should now come together when you consider the following questions:

- ◆ **How will the visitor be able to control the interface?** The "user interface" is how your visitor will interact with the Web site. With a dynamic Web site, the interface will change and adapt to the visitor as they work. Make a list of the different methods you'll use with your Web site: drop-down menus, floating menus, frames, and so on.
- ◆ **What media do you need on the Web site?** If you are going to be using audio or video on the Web site, you need to consider this in relation to what technologies your visitor will have available. For example, if you're going to show video clips, you need to make sure that your visitors will have the equipment and connection speed necessary to view them.

- ◆ **What technologies will you be using?** The interactivity that you can use often depends on, or is at least influenced by, the back-end technologies on your server. For example, sites built in ASP will work differently than sites built in straight HTML or even PHP. Even if you are a graphic designer, it's important to understand the strengths and weaknesses of the technology.

One important caveat to the above questions: Always try to find the simplest solution to get the job done.

Consider the graphic design style

Consider the most effective design for your Web site, based on the audience's expectations, your goals, and the site's features. Consider the colors, fonts, and other graphics you'll use.

- ◆ **What graphic design style will you use?** It's important to decide early on whether you want a simple minimalist style or something more ornate and rococo. There are trade-offs for both, but this decision needs to be made based on what you think will be most effective with your intended audience.
- ◆ **Which layout style will you use?** Choose one of the four layout types discussed earlier in this chapter.
- ◆ **What color scheme will you use?** Choose the primary and secondary colors for the site. This may already be dictated by the client, but you also need to decide which colors will be used where: in the background, foreground, and for links.
- ◆ **Which font families will you use?** Again, the client may dictate this, but you should consider which fonts to use for copy (paragraphs of text) and which to use for headlines, menus, and other elements. Generally, you should use no more than two different fonts.

Step 2: Design

Different people have different ideas when they think of the design of a site. For someone with a graphics background, *design* means the way the site will look, what colors will be used, and how the page will be laid out. For someone with a technology background, *design* implies how the various programming components will act and work together. The design of a Web site must address both of these factors equally.

During the design phase, you need to consider many aspects of your site to make sure all of them will work coherently.

Outline the site

An outline treats the Web site as a linear object, with each section of the site a top level in the outline, and each Web page fitting in a subcategory (**Figure 1.17**). A site outline probably will mirror the file structure of the site. One warning, though: Because of their linear nature, outlines can be highly misleading when they are applied to a Web site that has links among its sections.

Web Site: DHTML and CSS for the WWW

- Home
 - Introduction
 - Buy the Book
- Errata & FAQ
 - Book Errata
 - FAQ
- Downloads
 - Code
- Code Examples
 - Chapter 1-20

Figure 1.17 The outline helps you organize your site logically, placing the most important concepts at the top and the detailed ideas below them.

User or Visitor?

Web designers commonly refer to the audience of a Web site as *users*. I have never been terribly fond of this term, for several reasons. First, it often implies a negative and unhealthy relationship (as in drug user), and I really don't want to think of the people who come to my site in such a manner. Second, it sounds like something a marketer came up with to describe human beings, much like using the term *resources* to describe employees. To me, *user* dehumanizes the very real people for whom I am designing.

Instead, I refer to the audience of my sites as *visitors*. Yes, the difference is simply semantic, but language is a powerful tool that shapes our concepts of the people and things with which we interact. I would rather think of myself as a host to visitors than as a supplier to users.

Always remember that the visitor is the person for whom you are designing. Whether you are creating a personal site or a multimillion-dollar Web application, you should consider your visitors at all times, from conception and design to implementation and production.

Create a site map

A site map allows other people (clients, graphic designers, programmers, project managers, and so on) to see the site structure. It gives them a common blueprint to work with and reference (**Figure 1.18**).

Flowcharts show how various pages and functions within the site fit and work together. Generally, pages are represented by rectangles and links by lines, but you can add other symbols to the flowchart to indicate decision paths, external links, or other code (such as Java applets). The advantage of a flowchart over a simple outline is that it can show the relationships among pages more accurately.

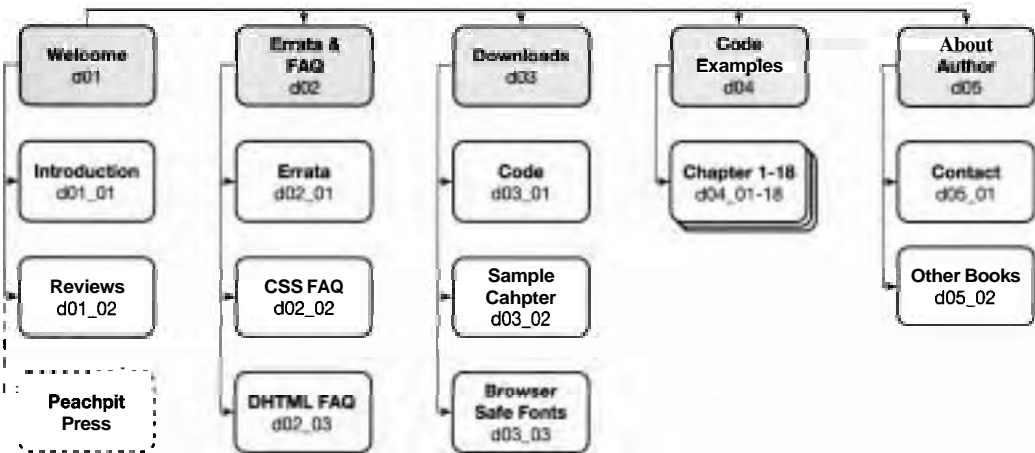


Figure 1.18 Site maps are a highly visual way to show the relationships among pages in a Web site. In this example, each box represents a Web page; lines represent links. Rounded boxes with dotted lines represent pages outside this Web site.

Sketch the layout grid

Select the Web layout style you want to use (see "Understanding Layout on the Web" earlier in this chapter). Then, on paper or in a drawing program, sketch the basic structure of a Web page in your site (**Figure 1.19**). You don't have to draw a detailed diagram of every page you plan to create. Instead, identify the general structure.

One important thing to also include in the layout grid is how CSS will be used to define different elements in the page. Unless you're

designing using tables (which I certainly hope you aren't!) then each block in the grid will be created in your HTML using either a `<div>` or `` tag, along with an ID or class. In each block, indicate the name of the ID or class, along with the dimensions of the block

In addition, you can use notes to describe the interactive components that will go in a particular area of a Web site.

Play with different layout concepts, and don't be afraid to experiment. It will be easier to try new things now than when you have the design in HTML.

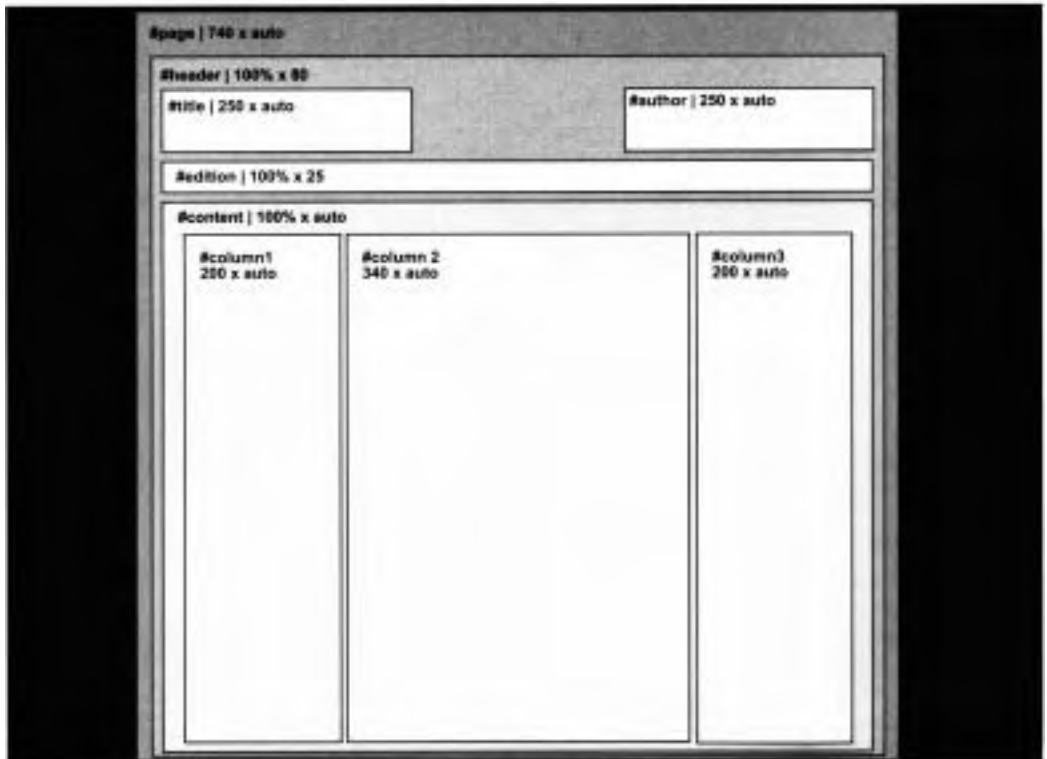


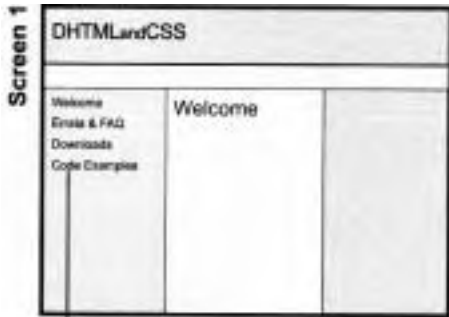
Figure 1.19 The layout-grid sketch allows you to plan where the various elements will go and how they will interact. In this example, the ID and the width and height are included to help developers while they create the CSS.

Create a storyboard

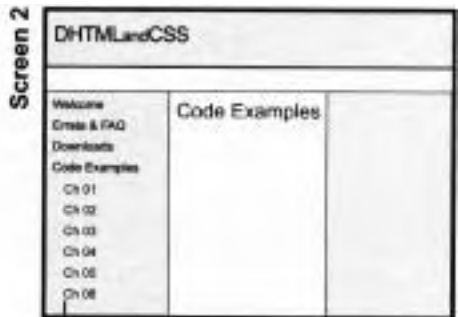
A storyboard works like a flowchart, but instead of just symbolic representations of pages, it provides a general visual representation of the page's content as the visitor will see it (**Figure 1.20**). The storyboard image is not meant to be an exact representation of each Web page. Instead, it indicates

the rough layout and shows how the site works—which can be invaluable for people who have a hard time imagining it without some visual representation. It is especially useful for describing interactive processes that occur on a single page.

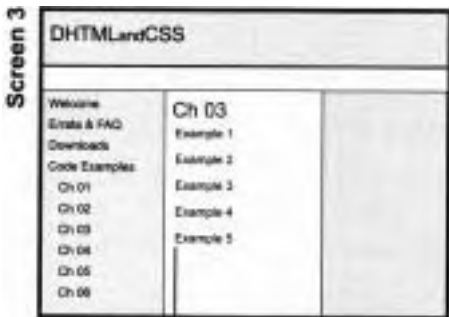
Storyboards generally take up a lot of space and need to be laid out on large paper to accommodate all their information.



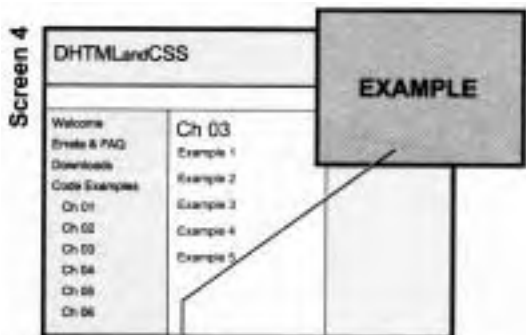
Visitor clicks link



Visitor chooses chapter



Visitor selects an example



Example Appears in new window

Figure 1.20 A storyboard is extremely effective for describing how a visitor will accomplish a particular task in a Web site. This storyboard shows what links a visitor would select to view a code example and what each screen would look like.

Define the look and feel

In a graphics program, create a mockup of your Web site, based on the layout grid from step 3 (**Figure 1.21**). Programs such as Adobe Photoshop and Macromedia Fireworks allow you to design interactive graphics and help you prepare them for use on the Web.

Again, you can play around with design ideas more easily at this stage of the game.

Step 3: Develop

Now the tire hits the pavement, as you create the Web site that you have been planning.

Before you start, though, make sure you review the information you gathered when defining your Web site (audience, goals, content, and features).

This is where all of your hard planning pays off. If you have properly prepared, creating the code and placing the content should be relatively simple.



Figure 1.21 The look and feel of the Web site are composed in Photoshop. Many elements will be created with HTML text and CSS in the final version, but it helps to plan what they will look like.

Understanding the Layout Grid

One tool at your disposal for ensuring consistency in the design of your Web site is the layout grid. Simply stated, the layout grid organizes the content on a Web page into specific rectangular areas. If your grid is well defined, it allows you to create a variety of layouts along a similar theme.

If used properly, a layout grid identifies where specific types of content will be placed on the page and provides continuity between different pages. This format works to visitors' advantage, because they'll always know where on the page to look for particular kinds of content (titles, illustrations, text, page numbers, and so on).

You can create a layout grid for any Web-layout category. Each format has its own strengths and weaknesses, depending on the size of the browser window. With an unrestrained layout, for example, you may find that the columns of text stretch uncomfortably wide for reading purposes. Restraining the width of the content area can create uncomfortably large areas of negative space in the design, and restraining the height of the content area discriminates against visitors who have large monitors.

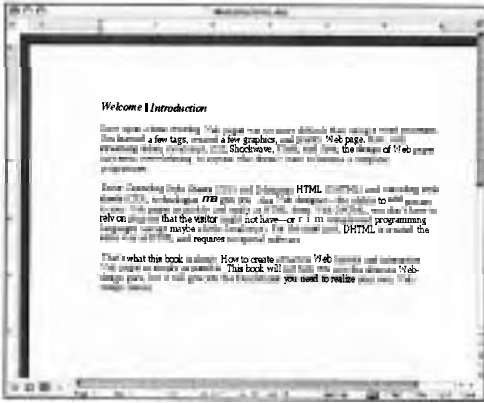


Figure 1.22 Check your content for spelling and grammar mistakes.



Figure 1.23 Before you create your Web site, test your code, and make any tweaks required for your Web site.

Prepare the content

Gather the content you will be using in the site—all the text, video, animations, and audio that will appear on the pages. Check all content thoroughly, and prepare it for use in the Web site. This may simply mean having the text available in a word processing file (Figure 1.22). Or you may want to go ahead and place the basic tags (`<p>`, `
`, `<h1>`) in the files.

Create the code

Even if you're copying JavaScript, DHTML, or CSS from this book or another source, you need to adapt it for your own design (Figure 1.23). How you create the content depends on your own preferences and the tools at your disposal. Many developers still prefer using a straight text editor, while more visually inclined Web designers prefer WYSIWYG applications like Adobe GoLive or Macromedia Dreamweaver.

Create a template

In an HTML editor (or whatever program you use to create HTML), set up a generic version of your Web page. Add the DHTML and CSS that you plan to use on every page as the basic navigation. Test this page in a variety of browsers, under a variety of conditions, and refine it until you are satisfied.

Now that you have a full working prototype of your Web page, strip out anything that is not needed on every page, and use what remains as the template. You can use this template to build the rest of the pages in a program such as Dreamweaver or GoLive.

Assemble the pages

Using your template as a starting point for each page, add the rest of the content (**Figure 1.24**). Make sure you keep a clean copy of the template for creating each new Web page.

Test your site

Always test your work before putting it online. Small mistakes can lead to big headaches if they aren't caught and corrected at this stage. You should test your pages in a wide variety of browsers and operating systems.

Step 4: Deploy

The Web site may be built and tested, but now it's time to take it live. After you've tested your work as thoroughly as possible, you're ready to upload it to your server. But wait, there are a few things to consider before you do this:

1. If the server you are deploying your work to is different than the one you were testing on during development, upload your pages to a subdirectory on your live server and test them there first before going live. This allows you to make sure that you won't run into any conflicts.
2. Upload a Site Being Updated page as the live index page (probably `index.html` or `index.htm`), rename the new index page as something like "`indexNew.html`" and upload the entire site into its live position. Then, test again (**Figure 1.25**). Obviously, links to `index.html` will link to the Site Being Updated page.
3. Once you're satisfied that everything is working, delete the Site Being Updated `index.html` page, and rename `indexNEW.html` to "`index.html`."

Your site is now live. Congratulations!



Figure 1.24 Assemble all the components.



Figure 1.25 The results of all my hard work are online at www.webbedenvironments.com/dhtml.

MASTERING CSS

The game Othello uses the slogan "A minute to learn, a lifetime to master." This is a good way to think about using CSS in Web site designs. Although it may take you more than a minute to learn, the basics of CSS are remarkably easy to learn. Understanding CSS is simply a matter of knowing the style definitions and applying them to HTML tags.

However, many Web designers seem to think that CSS has no more to offer them than the ability to style text, add some margins, and maybe control an object's position. There is so much more.

As with Othello, the complexity does not come from understanding the rules, but from being able to apply and mix techniques in a variety of circumstances.

Much of this book deals with specific techniques for applying CSS to achieve a desired effect. But before you delve into the techniques, it is important to first understand not just the *how* of using CSS but the *why* and *when*.

In this chapter, we will first consider some of the popular misconceptions about CSS and then reexamine some of the basics of using styles, hopefully giving you a new perspective on skills you may use daily.

The Myths of CSS

Even if you use CSS regularly, even if you're a great fan of how it simplifies Web site creation, even if you wouldn't think of designing a Web site without it, you probably encounter other designers, programmers, and managers who argue that using CSS is a bad idea. If you're one of those people who argue—then shame on you.

To a certain extent, I don't really blame the detractors. Although CSS has been around for years, browsers adapted the standard unevenly, and there have even been changes and additions to the standard. With styles in HTML, on the other hand, designers have a certain confidence that even if their designs aren't identical in all browsers, at least the basics will show through. However, the days of unreliable CSS are gone, and unless you're designing strictly for a legacy browser (such as Netscape 3) most of the arguments against CSS amount to a few myths that some still cling to.

CSS Myth # 1: CSS is not available in all browsers.

CSS has been available in Netscape since version 4 and Internet Explorer since version 3 (**Figure 2.1**). Although not all browsers (especially older browsers) have all of the CSS capabilities implemented, virtually everyone surfing the Web is using a browser that understands the most important CSS capabilities.

One thing to keep in mind while designing your pages, though, is that if the browser does not support CSS it should degrade gracefully into pure text. That is, the content of the page should still be intelligible.

CSS Myth # 2: CSS is not fully cross-browser compatible.

It is true that not all browsers, especially older browsers, have CSS implemented in



Figure 2.1 Virtually every visitor surfing the Web today is using a browser that supports CSS.



Figure 22
As with all things in the Web, browsers interpret the code differently, but with CSS overcoming these differences is relatively simple.

exactly the same way, but in many ways CSS is more cross-browser-compatible than HTML (**Figure 2.2**). Most Web designers are just used to accommodating the cross-browser HTML inconsistencies. While working it's important to keep in mind which CSS features are not implemented in a particular browser, just as you would for HTML.

CSS browser compatibility is listed in Appendix A. Sections later in this chapter will help you deal with two of the most frustrating inconsistencies, as well as provide a rundown on CSS features that are currently not implemented in Internet Explorer 6.

CSS Myth # 3: CSS cannot handle the layouts I need.

In the early days of Web layout, designers complained about the limitations of Web design (**Figure 2.3**). Over time, though, they learned to use its strengths and overcome its weaknesses. This has actually led to a very Web-specific design style, based on the use of tables to place content in the window. CSS has its own weaknesses in this regard, but designers are beginning to find its strengths and create much cleaner and more user-friendly designs than can be accomplished with HTML tables.

Chapter 5, "Layout," details how to create different layouts using CSS and how to turn some of its disadvantages into advantages.

CSS Myth # 4: CSS is harder to use than HTML.

Learning CSS takes additional effort, and for Web designers who have spent a lot of time learning to put up with HTML, it may seem like a waste of time, having to learn a whole new skill set while unlearning another skill set (**Figure 2.4**). Yet CSS will not only save you lots of time in the long run, it will give you much greater design flexibility.



Figure 23
Although the exact solutions may differ, CSS can handle any of the layout styles you see on the Web.



Figure 24
Although learning (and mastering) CSS takes some more time, using it will radically speed your development time.

Building Your Style Sheets

Putting together the styles for your Web site takes more than just spewing a few CSS rules onto a page. It's not like you have to create a detailed plan of your CSS structure before you start coding, but as you work, you need to consider where the definitions go. Sometimes this is a practical matter, since the exact order can determine the final appearance of your designs.

Beyond considering how the order affects the final appearance of your design, it's also important to categorize your styles systematically simply to help you locate them while you make changes. If you're designing a large, complex site, you may have dozens or even hundreds of different style rules. If they're simply on the page, you can waste a lot of time tracking down the one you need to alter.

Style order

The exact approach you take in ordering your styles is up to you. There is no "right" way, only the way that works best for you. However, if you're working as part of a team, you need to take your teammates into consideration as well. Someone else making changes to the styles in a Web site may not be able to see the method in your madness. Here are a few that not only help me keep my style sheets organized but make it easier for those who come after me to work with them (**Code2.1**).

Code 2.1 A well-organized style sheet will help you later when you're trying to change a particular style, track down problems, or explain to others how your styles are working.

```
/* Default Style sheet for melissaferrick.com */
/* HTML Selectors */
body {
    font-family: Arial, Helvetica, sans-serif;
    background-color: #2F2F2F;
    margin: 10px 0 20px;
}
h1 {
    color: white;
    font: 28px "arial black";
    margin-top: 0px;
    padding-top: 0px;
}
th {
    color: #bebebe;
    font-size: 10px;
    text-align: left;
}
td {
    color: #fff;
    font-size: 11px;
    padding: 2px 2px 5px;
}
/* Layout Logic */
#layoutAbc {
    display: block;
    margin: 0 auto;
    width: 700px;
}
#layoutaB {
    display: block;
    margin: 0 auto;
    width: 700px;
}
#layoutA {
    display: block;
    margin: 0 auto;
    width: 700px;
}
```

(code continues on next page)

```

Code
/* Page Components */
#header {
    text-align: left;
    display: block;
    width: auto;
}
#menu {
    display: block;
}
#content {
    color: #fff;
    font-size: 11px;
    width: auto;
    height: auto;
}
#footer {
    color: #fff;
    font-size: 10px;
}
font-weight: bold;
margin-top: 5px;
/* Classes */
.moduleContent {
    padding: 10px;
    scrolling: auto;
}
.modTitle {
    margin-top: 5px;
}
.emphasize {
    color: #903;
    font-size: 12px;
    font-family: Helvetica, Arial,
        SunSans-Regular, sans-serif;
    font-weight: bold;
    background-color: #979797;
    text-align: center;
    display: block;
    margin-top: 10px;
    padding-bottom: 3px;
}

```

(code continues on next page)

- **Place HTML selectors at the top of the list.** HTML selectors are often used to define styles at the highest level. For example, you may want to define the common appearance of all paragraphs in the page by defining the `<p>` tag. You can then create more specific classes to be used when the `<p>` tag appears in a headline news story or in the abstract for an article. Placing all of the HTML selectors at the top lets you quickly find the rules you need to make wide-ranging changes to your design. Start with the `<body>` tag first and then work your way through the different HTML selectors you'll be using in your Web page. Even if you don't use HTML selectors, though, I recommend placing blank rules in your style sheet for at least the most common, so that you *know* you haven't made any changes.
- **Place ID selectors together (especially those used for layout).** IDs are generally used to create the structure of your page and define the layout. At first, you may only need a few IDs, but the more you learn about using CSS, the more IDs you'll use to create your site design and get the exact layout you want. You not only want to keep all of these important rules together, you should place them in a hierarchy based on the order they appear in the HTML code. For example, place the IDs used to create a header first, then IDs for the content area next, and finally IDs for the footer. Ordering the IDs this way helps you better visualize the 2D structure of the final Web page despite the linear nature of looking at the code.

continues on next page

• **Place class and pseudo-class selectors last.** Class selectors are the workhorse of CSS and may be used simply to add text styles or to define the structure of various elements on the page. Generally you'll find that you define more classes than either HTML selectors or ID selectors. Since class selectors are so versatile, though, group them into categories. For example, place all of the classes used to style text (font-family, size, color, and so on) together.

Add notes

Notes are text within a style sheet that the browser ignores, even if it actually includes working code:

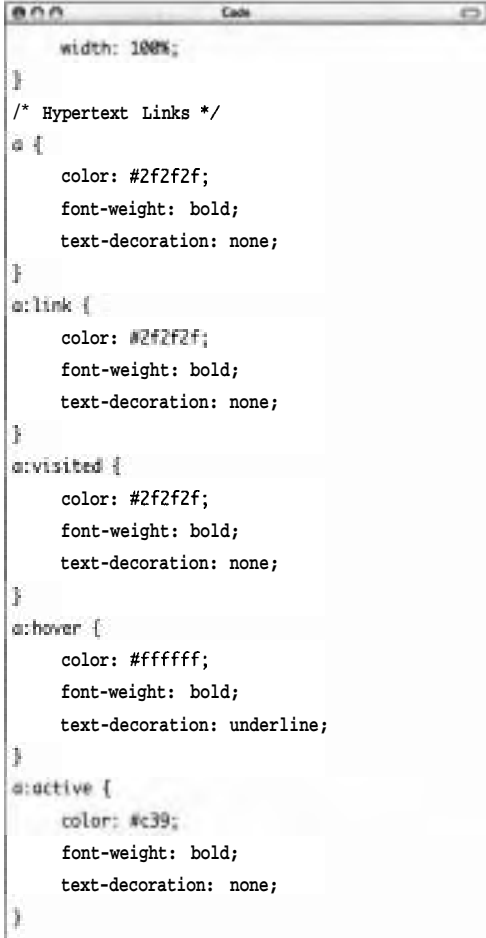
```
/* Do not include p { color: red } */
```

Notes are intended to help us mere mortals better understand what is going on in the code without having to spend hours puzzling through it.

As you diligently add, adjust, and tweak your styles, it's easy to forget to add notes indicating what particular styles or groups of styles are for. When you come back to your style sheets days, months, or years later, though, you may have absolutely no idea why you did what you did where you did it. Worse yet, someone else looking at your code may get lost in the maze of overlapping styles. Always add notes to your CSS while you work or plan to go back through your code at regular intervals to add notes.

Notes need not be verbose, just enough to jog the memory or point the reader in the right direction.

Code 21 *continued*



```
width: 100%;
}
/* Hypertext Links */
a {
    color: #2f2f2f;
    font-weight: bold;
    text-decoration: none;
}
a:link {
    color: #2f2f2f;
    font-weight: bold;
    text-decoration: none;
}
a:visited {
    color: #2f2f2f;
    font-weight: bold;
    text-decoration: none;
}
a:hover {
    color: #ffffff;
    font-weight: bold;
    text-decoration: underline;
}
a:active {
    color: #c39;
    font-weight: bold;
    text-decoration: none;
}
```

Web Typography: Beyond Times and Arial

Most Web designers still cling to the Web design myth that only two font choices guarantee that pages are displayed in the typeface in which they were created: Times or Arial. This leads to relatively monotonous typography on Web pages or an overreliance on text in graphics.

However, your typographic choices are far more extensive than you might believe.

Text in graphics

While putting text in graphics gives you the freedom to use whatever typeface you desire, it has several drawbacks.

- ◆ **Slower to download:** Graphics will invariably take longer to download than the same text in straight HTML.
- ◆ **Harder to change:** Although anyone with a text editor can change HTML text, graphic text requires image-editing software such as Photoshop. In addition, you generally have to re-create the entire graphic to change even a single letter. This means that if you create a graphic and someone else has to change it, they'll have to have the same font you were using on your computer.
- ◆ **No copy and paste:** Users will not be able to copy and paste text if they need to use it, and will instead have to retype it from scratch. For some purposes, this may actually be an advantage, but it's highly frustrating to most visitors.
- ◆ **Looks bad when printed:** Graphic text uses a relatively low resolution that looks fine for display on computer screens. However, when printed the text becomes noticeably jagged on the edges, especially if it's in transparent GIFs.

Still, there a few times when only text in graphics will do:

- ◆ **Logos:** Text in a logo will generally have to use a particular font or may be so heavily stylized that no font could come close to replacing it.
- ◆ **Corporate design standards:** Corporations often insist on certain fonts being used in their designs as part of a corporate standard. As much as possible, try to limit these fonts to top-level headers that are not changed often.
- ◆ **Special effects/styles:** CSS has a relatively limited set of effects and styles that can be added to text. Bold, underline, and color are standard but beveled, embossed, textured, and drop shadowed are not (although drop shadows are included in CSS, only a few browsers have added support for this style). If you really need effects in your design, text in graphics may be your only option.

Browser-safe fonts

There may not be thousands of typefaces at your disposal, but most people browsing the Web today are using computers with a dozen or more fonts that were preinstalled by the manufacturer (see Appendix D).

Using these fonts can dramatically open your typographic horizons, but there is one obvious drawback. Although there is some overlap between Mac and Windows fonts—especially if Microsoft Word or Internet Explorer is installed, which is likely—sticking to only the common fonts does limit you. Of course, the solution with CSS is simple. Just include a list of similar fonts in your definition list, pulling one from the Windows list and the other from the Mac list (**Figure 2.5**).

Of course, this means that you should test your designs in both fonts to ensure that one font doesn't do anything unexpected—but the changes should be minimal.

Choosing the right typeface

Before the visitor reads the first word on your Web site, they see the typeface. Thus, it is the typeface that makes the first (and lasting) impression about the message you are about to give them. Thus, it's important to think of the typeface you use as more than a simple aesthetic decision. Keep these points in mind when you choose the best font for your Web site.

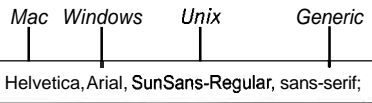


Figure 2.5 Include a list of fonts to cover Mac, Windows, and (if possible) Unix operating systems.

Font or Typeface?

Although the terms are often used interchangeably, font and typeface refer to different, yet symbiotic, ideas.

A *typeface* is a collection of characters (letters, numbers, symbols, punctuation marks, and so on) that have been designed to work congruously together. To put it succinctly, a typeface is an alphabet with a particular design.

A *font*, on the other hand, is a description of the typeface: a computer file, a metal die, or photographic film. A font, then, is the method used to create a typeface. Thus you would say that the font file `Arial.fff` is used to display text in the *Arial* typeface.

The distinction between typeface and font is rapidly disappearing, since most computer applications allow you to choose the typeface using the Font menu.

Readability

Some typefaces are simply easier to read than others, but there are no cut-and-dried rules for determining which typefaces your visitor will read easiest (**Figure 2.6**). Some fonts are easier to read at large sizes than others, but harder to read at smaller sizes. To complicate matters further, readability also depends on what the reader is used to seeing.

At smaller sizes, it has been generally accepted that sans-serif fonts are easier to read on the computer screen than serif fonts. However, it's easy to argue that this has been due to low screen resolutions. Now that increasing numbers of people are reading on screens that display typefaces using anti-aliasing (which effectively increases the resolution), more designers feel comfortable using serif typefaces.

So, how do you pick the most readable font? Unless your audience is radically different than yourself the best way to determine whether text is readable is to read it yourself. If you have a hard time scanning the lines, or feel yourself going back a lot, then try a different font.

Alice couldn't say honestly that he was. He had a tall red night-cap on, with a tassel, and he was lying crumpled up into a sort of untidy heap, and snoring loud-'fit to snore his head off!' as Tweedledum remarked.

ALICE COULDN'T SAY
HONESTLY THAT HE WAS. HE
HAD A TALL RED NIGHT-
CAP ON, WITH A TASSEL,
AND HE WAS LYING
CRUMPLED UP INTO A SORT
OF UNTIDY HEAP, AND
SNORING LOUD-'FIT TO
SNORE HIS HEAD OFF!' AS
TWEEDLEDUM REMARKED.

Figure 2.6 Which typeface is easier to read? On the left is Verdana and on the right is Herculanum.

Visual message

Although you want to ensure that the visitor can actually read the text on your Web page, no matter what font you choose, it is sending a visual message to the reader. Of course, the restricted number of fonts at your disposal in Web design (even when using the browser-safe fonts) will necessarily restrict your visual message. Still, there are enough available that you need to consider your choice carefully. **Table 2.1** presents a few visual messages that might be associated with particular browser-safe fonts.

Differentiate content types on the page

One common use of typefaces is to set apart different content on the page. For example, you might use one typeface for titles, one for sidebars, and another for callouts and figure captions and so on. In fact, looking at this book, you'll notice that my publishers have used multiple typefaces on the page.

It's important to choose fonts that clearly contrast each other on the page to help visually differentiate the content. Generally this means mixing the different font families (serif, sans serif, monospace, cursive, or fantasy) to make them distinctive enough to be useful.

However, it is important not to use too many typefaces on a single page or even within a single Web site. While some variation adds clarity and visual interest, using more than two or three distinct fonts generally leads to typographic noise, which actually works against readability.

Table 2.1

Typeface Visual Messages	
MESSAGE	TYPEFACE
Bold, dramatic, force	Arial Black, Impact, Charcoal, Chicago, Gadget, Marker Felt, Franklin Gothic Medium, Copperplate
Technological, futuristic, sci-fi	Andale Mono, Courier, Courier New, Lucid Console, Lucida Sans Unicode, Monaco
Handwritten, crude, childish	Comic Sans MS, Sand, Marker Felt
Happy, fun, whimsical	Arial Rounded, Brush Script, Textile
Professional, corporate, business	Times, Times New Roman, Trebuchet MS, Verdana, Microsoft Sans Serif, Tahoma
Historical, antiquity, ancient	American Typewriter, Brush Script MT, Herculanum, Papyrus, Zapfino, Tahoma
Sophisticated, elegant, refined	Adobe Minion Web, Cochin, Gill Sans, Palatino, Hoefler Text, Palatino Linotype
Old-fashioned, quaint, conservative	Georgia, Baskerville, Big Caslon, Palatino Linotype, Didot
Progressive, modern, liberal	Arial Narrow, Geneva, New York, Optima, Skia, Microsoft Sans Serif, Trebuchet MS

Creating Web Pages for Print

I have never seen a paperless office and would be quite surprised if I ever did. But the big promise that came along with the computer was the elimination of paper from our lives. No more filing cabinets, clutter, or dead trees, just an entropy-free utopia in which electrons were constantly recycled and reused, just like in *Star Trek*.

But something tells me that we'll have the technology to fly between the most distant stars before we eliminate paper from our lives.

With the advent of laser and inkjet printers, we seem to be buried under mounds of perfectly printed paper. Even the Web seems to increase the amount of paper we use. If a Web page is longer than a couple of scrolls, most people print it.

But the Web was created to display information on the screen, not on paper. Web graphics look blocky when printed, and straight HTML lacks much in the way of layout controls. That said, you can take steps to improve the appearance of printed Web pages. Looking good in print on the Web may take a little extra effort, but your audience will thank you in the long run.

Here are seven simple things you can do to improve the appearance of your Web page when it gets printed:

- ◆ **Use CSS.** Cascading style sheets are the future of Web design. CSS allows you to create documents that look as good printed as anything spit out of a word processor.
- ◆ **Define your media.** CSS allows you to define different style sheets to be used depending on the way the page is displayed—usually on a screen or on paper.

Type Size Onscreen: Relative or Absolute

There is some debate among Web designers over whether to use absolute or relative font sizes. Absolute font sizes (px, pt, cm, and so on) allow you more precise control over the layout of your design, ensuring that the size you set is the size at which the visitor sees the text. However, this discriminates against visitors who want to or have to view text at larger sizes. Setting a relative font size (em, %, ex, and so on) allows visitors to adjust the text size to suit their specific needs.

Which you use will depend on what you know about your audience's needs.

continues on next page

- ◆ **Use page breaks to keep headers with their text.** Although the page-break attribute is not widely supported at this time, it may be a universal standard before long.
- ◆ **Avoid using background colors or background graphics with light-colored text.** Although they can add flavor in the window, background colors and graphics turn into noise when printed. Some browsers allow you to print documents without backgrounds, but then light-colored text will not show up against the white background of the paper.
- ◆ **Avoid using transparent colors in graphics.** Especially avoid it if the graphic is on a graphic or a background color other than white. The transparent area of a GIF image usually prints as white, regardless of the color behind it in the window. This is not a problem if the graphic is on a white background to begin with, but the result is messy if the graphic is supposed to be on a dark background.
- ◆ **Avoid using text in graphics.** The irony of printing stuff off the Web is that text in graphics, which looks smooth onscreen, looks blocky when printed, but regular HTML text, which may look blocky onscreen, prints smoothly on any decent printer. Try to stick with HTML text as much as possible.
- ◆ **Provide a separate print-ready version of the Web site.** Rather than force visitors to follow every link on your site and print each page along the way, provide a single document of your Web site that visitors can download and print. Adobe Acrobat is a great way to provide this content in a more-or-less universal file format that retains most formatting, fonts, and graphics for delivery over the Web. Find out more about Acrobat at the Adobe Web site (www.adobe.com).

Building a Master Style Sheet

CSS allows you to place styles into an external style sheet (really just a text file holding all of the CSS rules) and then link or import that file into your HTML. This allows you not only to separate the styles from the code used to actually structure the page, but also to place multiple style sheets in a single Web page. Many Web designers do this—they link to various style sheets for various tasks by placing a `<link>` tag in the head of each document.

However, there is a disadvantage to using the `<include>` tag in the head of your document: What if you need to add or delete a linked style sheet from your pages globally? The solution is to link to a single style sheet and then use `@import` to aggregate the various style sheets (Figure 2.7). Then, to add or remove one, you can simply edit this single file.

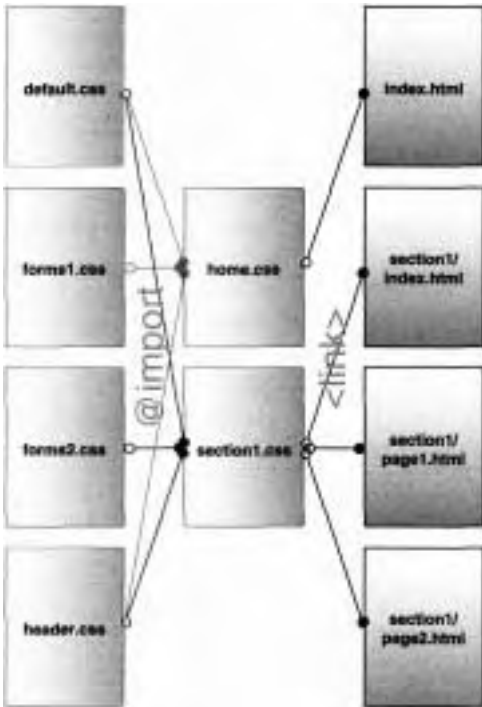


Figure 2.7 Create your external style sheets for a variety of purposes, then import them into one or more master style sheets which are in turn linked to your Web pages.

To create a master style sheet:

1. Create your external style sheets. The exact strategy you use to organize your various external style sheets will depend on your project, but try to place similar styles in a common external style sheet. For example, place the styles used to create the structure of the page in one style sheet and styles for forms in another. This is especially useful if you need to change the styles for different pages in the site, say to style forms in one way for one section and another way in a different section.
2. Create your master style sheet using `@import` to mix and match styles as needed (Code 2.2). You might even create several different style sheets for different uses within your site, for example, one master style sheet for the home page, then other style sheets for each section within the site. This way you can control exactly which of the style sheets you created in step 1 get used for particular page types.
3. Add any additional CSS you want included directly in the master style sheet. This is optional, since you could simply import all of your styles, but it is often useful and can cut down on the number of files you create.
4. Another optional addition is default CSS for Netscape 4 (see the sidebar "Degradating CSS Gracefully in Netscape 4").
5. Link to the master style sheet in your HTML pages (Code 2.3). You could also use `@import` here, but Netscape 4 would then not use any of the styles.

Code 2.2 To build your master style sheet, add the `@import` tag for each external style sheet you are including, and then include additional style definitions as needed.

```
Code
/* Master Style Sheet for Home Page */
/* Imported Style Sheets */
@import url("default.css");
@import url("forms1.css");
@import url("header.css");
/* Home Page Specific Styles */
body {
    background-color: #fff;
}
/* Netscape Styles */
p {
    margin-top: 5px;
    margin-bottom: 5px;
}
```

Code 2.3 Add the `<link>` tag to your HTML to bring the master style sheet home.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
 Transitional//EN" "http://www.w3.org/TR/
 xhtml1/070/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xml:lang="en">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I
      Building a Master Style Sheet</title>
    <link href="css/masterHome.css"
      rel="stylesheet" type="text/css"
      media="all"/>
  </head>
  <body>
  </body>
</html>
```

Table 2.2

SHORTHAND	EQUIVALENTS
border	border-width border-style border-color
border-top	border-top-width border-top-style border-top-color
border-right	border-right-width border-right-style border-right-color
border-bottom	border-bottom-width border-bottom-style border-bottom-color
border-left	border-left-width border-left-style border-left-color
font	font-style font-variant font-weight font-size line-height font-family
list-style	list-style-type list-style-position list-style-image
background	background-color background-image background-repeat background-attachment background-position
margin	margin-top margin-bottom margin-left margin-right
padding	padding-top padding-bottom padding-left padding-right

Understanding CSS Shorthand

Options are an inherent part of CSS. This is true of everything from the styles that define your page to the individual style properties that define the styles. While most CSS properties have only one form, there are several that can stand in for multiple other properties. These "shorthand properties" allow you to define a whole slew of styles in a single line of code (**Table 2.2**).

For example, you could describe the border around an element using the border-width, border-style, and border-color properties:

```
border-width: 1px;
border-style: dotted;
border-color: #ff0000;
```

However, it is more efficient simply to use the border attribute to create the same style:

```
border: 1px dotted #ff0000;
```

This assumes, though, that your border has the same format on all four sides. To describe each side separately you might use this:

```
border-top: 1px dotted #ff0000;
border-right: 2px dashed #660000;
border-bottom: 4px solid #00cc00;
border-left: 8px double #009900;
```

However, you could get the same effect simply using this:

```
border-width: 1px 2px 4px 8px;
border-style: dotted dashed solid double;
border-color: #ff0000 #006600 #00cc00
             #009900;
```

The values (separated by spaces) represent the top, right, bottom, and left sides in that order. These four values hold true for any property being applied to an element (margin, padding, or border).

Which properties you use to define your styles depend on your design needs and what is most convenient for you while coding. Some people like the quick shorthand of listing the top, right, bottom, and left values in a single sequence, while others struggle to remember the correct order of the values.

Shorthand and legacy browsers

Although all modern browsers recognize the various properties and their shorthand versions, some legacy browsers—most notably Netscape 4—do not. For example, while you can set all four borders of an element separately in Netscape 4 using the `border-style`, `border-width`, and `border-color` properties (using four values with each property as discussed above), the browser does not recognize the `border-top`, `border-right`, `border-left`, or `border-bottom` properties. If you're testing your design and it doesn't appear to be working, try an equivalent style.

Shorthand for browser-safe colors

Although there are many ways to define colors in CSS, most professional Web designers stick with hexadecimal notation, where the first two numbers represent the amount of red, the second two the amount of green, and the last two the amount of blue. Thus, 336699 is a dark blue color. Browser-safe colors are made up of couplets of 00, 33, 66, 99, cc, and ff.

One handy shorthand for designating colors is to use only a single digit for each couplet, like this: 369.

Although this method works universally, there is some debate in the Web community about its validity. Some Web designers have even gone so far as to call this "lazy coding." Whether you use this method or not is up to you (I must admit that I get "lazy" sometimes), but you should at least know about it in case you run into it in someone else's code.

Degrading CSS Gracefully in Netscape 4

One thing to keep in mind if you decide to create master style sheets using `@import` is that Netscape 4 does not recognize this code. However, you can actually make this work to your advantage.

Most Web designers do not need to concern themselves with Netscape 4 anymore since it only accounts for less than 1 percent of the browser market and is quickly disappearing altogether. But the browser is still out there.

If you do need to code for Netscape 4, one solution is to use JavaScript to detect the browser and version and then deliver separate style sheets—one for Netscape 4 and one for all other browsers. However, this also requires you to spend a lot of extra time coding for Netscape 4's quirky implementation of CSS.

An easier solution is to simply include the basic CSS that you know will work in the master style sheet. (Be sure to consult the CSS compatibility charts in Appendix A to make sure that this code will work properly in Netscape 4.) These styles will provide the basic style so that Netscape 4 will still look good, but not require you to recode your entire style sheet for a browser that is unlikely ever to be used with your Web site.

Using Grouping and Context

Although it is common to group selectors either to combine their definitions (separated by commas) or to set up contextual styles (separated by spaces), the full power of these CSS features often goes unnoticed.

Consider **Code 2.4**. This sets up styles to create two different columns (`colA` and `colB`) defined by ID and then sets up two different areas (`area1` and `area2`) as classes. In the body then, the two columns are set up using `<div>` tags with both areas repeated in columns 1 and 2. And yet when displayed in a browser (**Figure 2.8**) the areas have a different appearance depending on the column they are in.

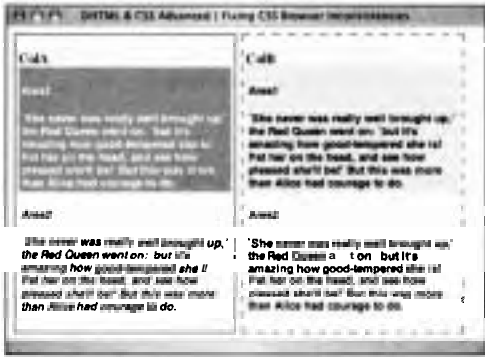


Figure 2.8 Although the same class is used to define `area1` and `area2`, because styles have been set up contextually, their appearance will depend on which column they are in.

Code 2.4 This code creates a page with styles to define two distinct columns and then two different areas as classes that can be used anywhere on the page. The areas are then both included in both columns, but will look different because of the contextual styles.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xml:lang="en" xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced : Using Grouping and Context</title>
    <style type="text/css" media="screen">
</style>
  </head>
  <body>
    <div id="colA" style="float: left; width: 250px; padding: 3px; margin-right: 5px; border: solid 3px #ff0000;">
      <div class="area1">
        The sailor was really well brought up,
        the Red Queen went on; but it's
        amazing how good-tempered she is!
        Put her on the boat, and see how
        pleased she'll be! But this was more
        than Alice had courage to do.
      </div>
      <div class="area2">
        The sailor was really well brought up,
        the Red Queen went on; but it's
        amazing how good-tempered she is!
        Put her on the boat, and see how
        pleased she'll be! But this was more
        than Alice had courage to do.
      </div>
    </div>
    <div id="colB" style="float: left; width: 250px; padding: 3px; margin-right: 5px; border: solid 3px #ff0000;">
      <div class="area1">
        The sailor was really well brought up,
        the Red Queen went on; but it's
        amazing how good-tempered she is!
        Put her on the boat, and see how
        pleased she'll be! But this was more
        than Alice had courage to do.
      </div>
      <div class="area2">
        The sailor was really well brought up,
        the Red Queen went on; but it's
        amazing how good-tempered she is!
        Put her on the boat, and see how
        pleased she'll be! But this was more
        than Alice had courage to do.
      </div>
    </div>
  </body>
</html>
```

(code continues on next page)

Understanding grouping

Using grouping in Code 2.4, we can define the common styles between the two columns, not only saving a little bit of space (less code is good), but also giving us one place to make changes to both columns' styles. For example, if we need to change the padding from 3px to 5px, we don't have to waste time hunting down both rules.

Understanding context

A more interesting effect in Code 2.4 is in the CSS code used to set up the areas. Here we have given them a common definition, but then also given them contextual definitions depending on the column (colA or colB) they have as their parent. So, if areal is in colA it will have a red background, but if it is in colB it will have a pink background.

Code 2.4 continued

```
border: dashed 3px #999999;
}
.areal, .area2 {
  font-size: 0.8em;
  font-family: Helvetica, Geneva, Arial,
    SunSans-Regular, sans-serif;
  padding: 3px;
}
#colA .areal {
  color: #ffffff;
  background-color: #ff0000;
}
#colA .area2 {
  font-style: italic;
}
#colB .areal {
  background-color: #ffc000;
}
#colB .area2 {
  font-weight: bold;
}
--></style>
</head>
<body>
  <div id="colA">
    <h3>Col A</h3>
    <div class="areal">
      <h4> Area 1</h4>
      'She never was really well
      brought up,' the Red Queen went
      on: 'but it's amazing how
      good-tempered she is! Pat her
      on the head, and see how
      pleased she'll be!' But this
      was more than Alice had courage
      to do.
    </div>
    <div class="area2">
      <h4> Area 2</h4>
```

(code continues on next page)

```

She never was really well
    brought up,' the Red Queen went
on: 'but it's amazing how
    good-tempered she is! Pat her
    on the head, and see how
    pleased she'll be!' But this
    was more than Alice had courage
    to do.
</div>
</div>
<div id="colB">
<h3>Col B</h3>
<div class="area1">
<h4> Area 1</h4>
'She never was really well
    brought up,' the Red Queen went
    on: 'but it's amazing how good-
    tempered she is! Pat her on the
    head, and see how pleased
    she'll be!' But this was more
    than Alice had courage to do.
</div>
<div class="area2">
<h4> Area 2</h4>
'She never was really well
    brought up,' the Red Queen went
    on: 'but it's amazing how good-
    tempered she is! Pat her on the
    ahead, and see how pleased
    she'll be!' But this was more
    than Alice had courage to do.
</div>
</div>
</body>
</html>

```

Fixing CSS Browser Inconsistencies

It is simply a fact of life on the Web: Not all browsers display CSS exactly the same way. This is because, like HTML and other Web standards, CSS is *interpreted* by the browsers. Rather than being strict rules for creating browsers, W3C recommendations are guidelines for how the styles should work. This leaves the burden for implementation on the browser manufacturers. Just as different people will interpret words with slight variation in the same language, browsers may interpret the same CSS code differently.

For the most part, these inconsistencies are so minor as to go unnoticed or are simply CSS features that have not been implemented in the browser (see "CSS Beyond Internet Explorer 6" later in this chapter). However, there are two variations between the major browsers that can really throw off your Web design: the page origin, and the box model. The good news is that there are ways to **fix** these problems and teach the browsers to play nicely together.

Setting the page origin

All things being equal, when you set the absolute position of an element in a browser window it should appear at that exact spot in the window. Say, for example, you specify the following position:

```

top: 10px;
left: 23px;

```

The object in question should lie 10 pixels from the top of the page and 23 pixels from the left. However, all browsers add a margin around the edge of the screen. Wouldn't it be great if they all used a standard margin? Well yes, but they don't (Figure 2.9). The difference is only a few pixels, but it's enough to be frustrating when you're trying to align objects precisely. The solution is simply to set both the margin and the padding for the `<body>` tag to 0 pixels:

```
body {
    margin: 0px;
    padding: 0px;
}
```

Setting this will ensure that all content on the page starts from the top-left corner of the screen as its origin. Keep in mind, though, that this affects all relatively positioned content, so your design will bump to the edge of the window.

The box model hack

One browser inconsistency that leads many Web developers to consider a different career is the way browsers handle borders and padding around a CSS element (the box). The total width of the box should be calculated by adding the border thickness on the left and right, the padding on the left and right, and the width of the content.

Consider the following code:

```
#object1{
    background: #ff9999;
    border: 5px solid #ff0000;
    padding: 15px;
    width: 100px;
}
```

According to the CSS standards, width refers to the width of the content. So this code should create a box with a 5-pixel border and 15 pixels of padding around the content, which has a 100-pixel width: a total width of 140 pixels (Figure 2.10).

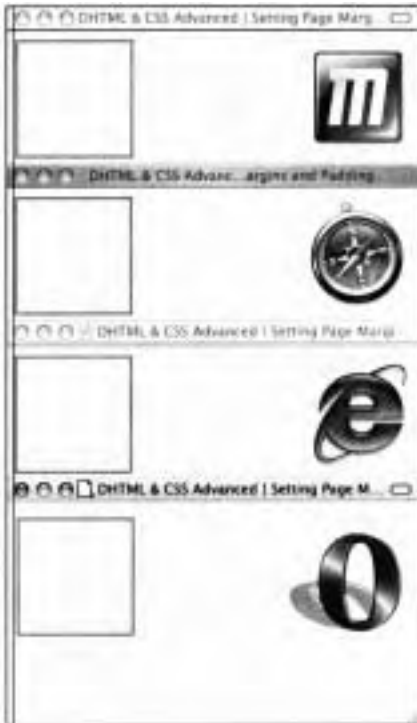


Figure 2.9 The same code displayed in four different browsers. Notice the slight difference in the position of each box.

140

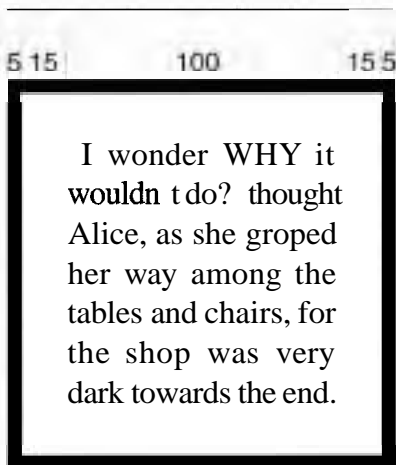


Figure 2.10 The box, properly displayed according to CSS standards, should be 140 pixels wide.

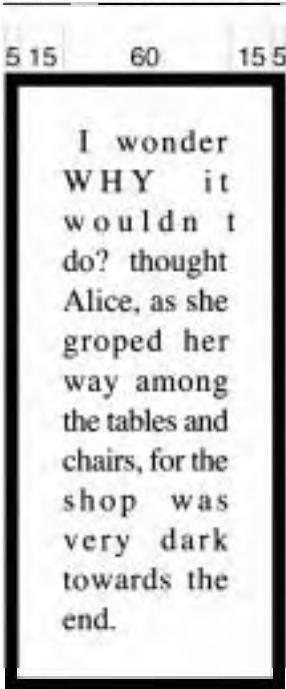


Figure 2.11 Some browsers, including Internet Explorer 5 for Windows, display the total width incorrectly.

✓ Tips

The box model hack was developed by Tantek Çelik, the lead developer of Internet Explorer for the Mac.

Although I generally recommend leaving it out anyway, do not include the XML prolog `<?xml version="1.0?"` if you need to use the box model hack, since it will produce quirky results in Internet Explorer 6.

Unfortunately, some common browsers—most notably Internet Explorer 5 for Windows—interpret this code incorrectly. Instead of applying the width to the content area, the width restrains the overall box size, so that the border and padding are subtracted from the overall width. So we're left with a box 100-pixels wide, but a content area only 60-pixels wide (**Figure 2.11**).

To fix this, we need to use a *hack*, that is, we have to intentionally introduce code that doesn't work in one browser so that both boxes have the same width. We will be taking advantage of a bug in Internet Explorer 5 that causes it to ignore certain CSS and initially set a width adjusted for padding and border being subtracted, and then set the true width:

```
#object1{
    background: #ff9999;
    border: 5px solid #ff0000;
    padding: 15px;
    width: 160px;
    voice-family: "\"}\" ";
    voice-family: inherit;
    width: 100px;
}
```

However, this will cause problems in Opera 5 unless you add the following definition immediately after:

```
html>body #object1 {
    width: 100px;
}
```

This allows browsers that understand the parent/child selector grouping to correctly interpret your code. Opera 5 does, Internet Explorer 5 does not.

CSS Beyond Internet Explorer 6

You may not encounter them often, at least not until Microsoft releases Internet Explorer 7, but there are several grouping and selector types that are included in CSS2, implemented in most modern browsers, but not included in Internet Explorer 6. The good news is that these will work in Netscape 6+, Safari 1+, and Opera 7+.

Child selectors

The child selector allows you to specify the style a selector receives if it is a direct child of the parent selector. The first selector is the parent and the second is the child selector that receives the definitions.

To set up child selectors:

1. `p.intro>em {...}`

Type the HTML for the parent tag (HTML, class, or ID), followed by the greater-than sign (>), then the child tag (**Code 2.5**). You can repeat this for as many different levels as you need, separating each selector with the greater-than sign.

Code 25 Child selectors are set up so that styles will only affect selectors nested directly within the parent selectors.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
= Transitional//EN" "http://www.w3.org/TR/
= xhtml1/DTD/xhtml1-transitional.dtd">
<html xml:lang="en" xmlns="http://www.w3.org/
= 1999/xhtml">
  <head>
    <meta http-equiv="content-type"
    = content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I Child
    > Selectors</title>
    <style type="text/css" media="screen"><!--
body {
  font-size: 12px;
}
p.intro>em {
  color: #ff0000;
  font-size: 28px;
}
--></style>
</head>
<body>
  <p class="intro">`Oh! PLEASE <em>don't
  > make such faces</em>, my dear!'
  she cried out, quite forgetting that
  > the King couldn't hear her. <b>`You
  > make me laugh so that <del>I can
  hardly hold you!</em></b>And don't
  keep your mouth so wide open! All
  the ashes will get into it--there,
  = now ■ think you're tidy enough!' she
  added, as she smoothed his hair, and
  set him upon the table near the
  > Queen. </p>
</body>
</html>
```

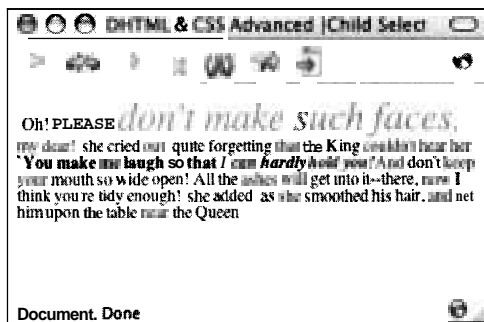


Figure 2.12 Notice that the text in the `` tag that is directly within the `<p>` tag with the `intro` class receives the style (larger and red), while the text in the `` tag that is also within the `` tag does not.

2. `<p class="intro">`Oh! PLEASE
> don't make such faces,
> +my dear!' she cried out...</p>`
Add your HTML code. Anywhere that the child selector is a direct descendent of the child tag, selector styles will be applied. But if the child selector is within another tag, the styles will be ignored (**Figure 2.12**).

Can IE7 Fix Browser Inconsistencies?

One of the great limitations on Web design with CSS is that Internet Explorer 6 contains several CSS bugs, as well as leaving out several important new CSS features. Unfortunately it will likely be 2006 before we see the release of Internet Explorer 7, which will hopefully catch up to the rest of the browsers.

In the mean time, Dean Edwards, the developer, is testing IE7 (not Internet Explorer 7 but "IE7"), which takes advantage of certain "features" of Internet Explorer 6 and some crafty DHTML to basically reprogram the browser to improve its CSS capabilities.

The code is still under development and is not ready for primetime, but shows a lot of promise. To learn more, visit Dean Edwards's Web site at <http://dean.edwards.name/IE7>.

Adjacent selectors

As its name implies, using the adjacent selector grouping allows you to apply styles to a sibling selector only if it is next to (but not nested in) its sibling in the HTML code.

To set up adjacent selectors:

1. `b+em {...}`

Type the first sibling selector (HTML, Class, or ID), a plus sign (+), then the second sibling selector (Code 2.6). Only the second sibling selector receives the styles.

2. `<p>The King immediately fell flat on his back, and lay perfectly still: and Alice was a little alarmed...</p>`

Set up your HTML code with the two sibling selectors. Remember, though, that this only works if the second selector follows the first (Figure 2.13).

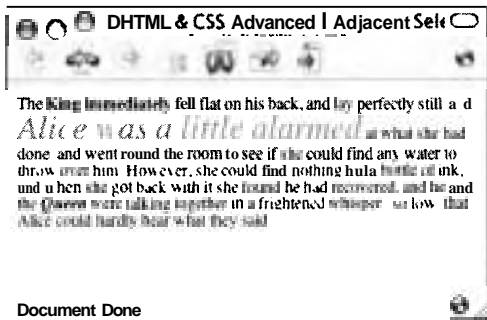


Figure 2.13 Notice that it is only the text in the `` tag that follows a `` tag that receives the styles (larger and red); the text nested in both `` and `` tags does not.

Code 2.6 Adjacent selectors are set up to allow you to define the style of a tag if it is preceded by another tag in the HTML code.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Transitional//EN" "http://www.w3.org/TR/
  xhtml1/DTD/xhtml1-transitional.dtd">
<html xml:lang="en" xmlns="http://www.w3.org/
  1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I
      Adjacent Selectors</title>
    <style type="text/css" media="screen">!--
body {
  font-size: 12px;
}
b+em {
  color: #ff0000;
  font-size: 24px }
--></style>
  </head>
  <body>
    <p>The <b>King immediately </b>fell
      flat on his back, and lay perfectly
      still: and <em>Alice was a little
      alarmed</em> at what she had, and
      went round the room to see if she
      could find any water to throw over
      him. However, she could find nothing
      but a bottle of ink, and when she got
      back with it, she found he had
      recovered, and he and the
      <b><em>Queen</em></b> were talking
      in a frightened whisper--so low, that
      Alice could hardly hear what they
      said.</p>
  </body>
</html>
```

Code 2.7 Pseudo elements allow you to set content that should appear immediately before or after the content of the selector it is applied to.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
- Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">
<html xml:lang="en" xmlns="http://www.w3.org/
1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I
      Pseudo Elements</title>
    <style type="text/css"
media="screen">!--
body {
  font-size: 16px;
}
p.inMiddle:before {
  color: #ff0000;
  display: block;
  content: "The Beginning";
}
p.inMiddle:after {
  color: #ff0000;
  display: block;
  content: "The End";
}
--></style>
</head>
<body>
  <p class="inMiddle">A little provoked,
    she drew back, and after looking
    everywhere for the queen (whom she
    spied out at last, a long way off),
    she thought she would try the plan,
    this time, of walking in the opposite
    direction.
  </p>
</body>
</html>
```

Pseudo elements

The `:firstline` and `:firstletter` pseudo elements are standard in all modern browsers, including Internet Explorer. However, there are two additional pseudo elements that allow you to insert content before and after an element.

To set up `:before` and `:after` pseudo elements:

1. `p.inMiddle:before {...}`

Type the selector in front of which you want your content to appear, a colon (:), then before (**Code 2.7**).

2. `content: "The Beginning";`

In the style definitions from step 1, include the `content` property, specifying the content you want to appear in front of the selector in quotes ("").

3. `p.inMiddle:after {...}`

To include content after the tag, repeat steps 1 and 2, but replace `before` with `after`.

4. `<p class="inMiddle">A little provoked, she drew back...</p>`

Set up your HTML code with the selector used in step 1. When viewed in a browser, the content you added in step 2 will appear above the element, and the content you added in step 3 below (**Figure 2.14**).

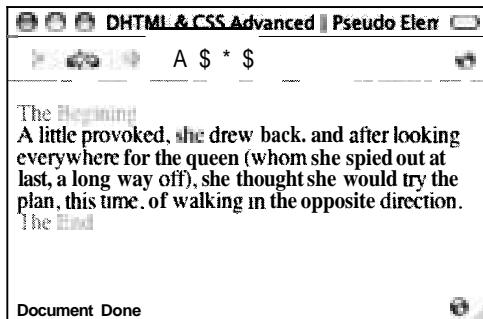


Figure 2.14 Notice that the content defined in the `:before` and `:after` pseudo elements is inserted above and below the `<p>` tag with the class `inMiddle`.

Dynamic pseudo classes

Although Internet Explorer recognizes the pseudo classes (:link, :visited, :hover, and :active), it limits their application to link tags. However, CSS allows :hover and :active to be applied to any element. In addition, the :focus pseudo class can be applied when the element is selected, which is especially useful for form fields.

To use dynamic pseudo classes:

1. p:hover {...}

Type the selector you want your content to appear before, a colon (:), then hover (Code 2.8). Add your style definitions for the hover state.

2. p:active {...}

Type the selector you want your content to appear before, a colon (:), then active. Add your style definitions for the active state.

3. input:focus {...}

Type the selector you want your content to appear before, a colon (:), then focus. Add your style definitions for the focus state.

4. <p>One thing was certain, that the · WHITE kitten had had nothing to do · with it...</p>

Add your HTML code. When displayed in a browser, the default styles are used at first, but when moused-over the hover state styles are used, when clicked the active styles are used, and when the form field is in focus the focus style is used (**Figures 2.15, 2.16, 2.17, and 2.18**).

Code 2.8 Dynamic pseudo classes can be applied to more than just link tags. They allow you to perform dynamic style changes to just about any element on the page.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
 Transitional//EN" "http://www.w3.org/TR/
 xhtml1/DTD/xhtml1-transitional.dtd">
<html xml:lang="en" xmlns="http://www.w3.org/
 1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I
      Dynamic Pseudo Classes</title>
    <style type="text/css"
      media="screen"><!--
  body {
    font-size: 12px;
  }
  p:hover {
    color: #ff0000;
    font-size: 16px;
    cursor: pointer;
  }
  p:active {
    color: #999999;
    font-size: 6px;
    cursor: move;
  }
  input:focus {
    background-color: #ffcccc;
  }
--></style>
  </head>
  <body>
```

(code continues on next page)

```

<p>One thing was certain, that the
- WHITE kitten had had nothing to
- do with it:--it was the black
kitten's fault entirely. For the
white kitten had been having its face
- washed by the old cat for the last
- quarter of an hour (and bearing it
- pretty well, considering); so you
- see that it COULDN'T have had any
- hand in the mischief. </p>

<form id="formName" action=""
- method="get" name="">
- <input type="text" name=""
- size="24"/><input type="submit"
- name="Submit Query"/>
</form>
</body>
</html>

```

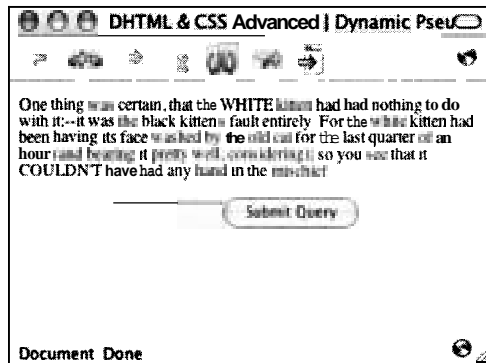


Figure 2.15 Before the visitor interacts with the page.

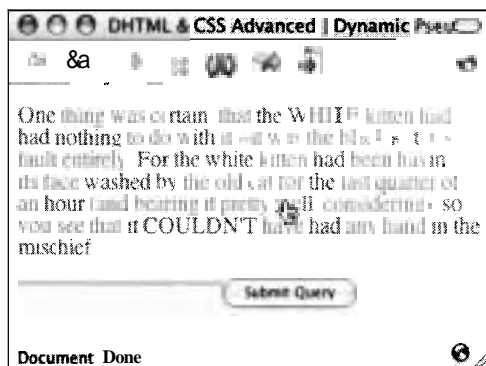


Figure 2.16 After the visitor places the mouse over the paragraph, the text grows larger and red.

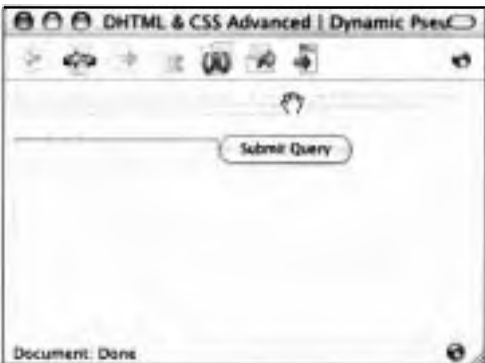


Figure 2.17 When the visitor clicks the paragraph the text shrinks and turns gray.

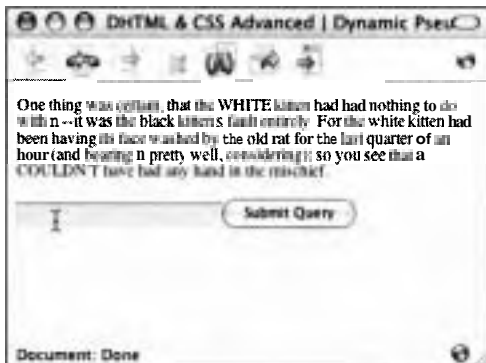


Figure 2.18 After the visitor clicks in the form field, the field's background turns pink.

Blank

ADVANCED

3

DHTML TECHNIQUES

Calling a Web page "dynamic" implies that it can change. But there must be something to change and those changes need to be remembered from Web page to Web page.

Content can, of course, be hard-coded onto Web pages, but that makes the content relatively static and difficult to change. However, if you place content and data into structures such as arrays and data objects, you gain dynamic control over the content. But what happens to the content after you load a new Web page? There are several ways to preserve your data so that Web pages can talk to each other, each with its own strengths and weaknesses.

In this chapter, you'll learn how to use arrays and data objects to store content and how to use frames, URLs, and cookies to transmit content between pages. I'll also show you how to add a delay to a function call and how to handle errors gracefully.

Setting Up and Accessing Arrays

A variable is used to hold a single value in a single signifier (the variable name). The value can be a number, a letter, a word, a sentence, or an entire book, but it can only hold one thing, one piece of datum, at a time. You can create as many variables as you want to hold as much data as you want, but this can get bulky and confusing if you have more than a few values to record.

Arrays are a type of variable that allow you to hold multiple values in a single signifier. This not only allows you to conveniently store your data, it also has some interesting implications in how you can manipulate that data.

All of the values in an array are accessed using the same array name, but are then differentiated by a numeric value, starting with 0. You can place as many different values in a single array as you need, but each one must be stored in a different position.

Four ways to set up arrays

As with all things in Web design, there is more than one way to set up an array. In this example, I'll set up four different arrays in an external JavaScript file, which can then be imported into an HTML file.

To set up an array:

1. Start a new empty text file and save the file as **array.js (Code 3.1)**. You can, of course, give it any name you want. The .js extension is not required. Steps 2-5 apply to any file.

Code 3.1 The arrays are placed in an external JavaScript file, nothing more than a text file saved with a .js extension. This example shows the four different ways to set up an array.



```
Code
var partyAttend = new Array();
partyAttend[0] = 'Tweedle Dee';
partyAttend[1] = 'Tweedle Dum';
partyAttend[2] = 'White Knight';
partyAttend[3] = 'Alice';
partyAttend[4] = 'Red Queen';
var someNumbers = new Array(95, 63, 86, 12, 32);
var imagesToShow = ['kingAndQueen.jpg',
  'whiteNight.jpg', 'aliceWithKitten.jpg',
  'jabberwocky.jpg'];
var tripPrices = [[100, 200, 300],
  [400, 500, 600], [700, 800, 900]];
```

2. The long way to set up an array is to list each value separately. First, the array is given a variable name and defined as a new array:

```
var partyList = new Array();
```

You would then add an entry for each value in the array, its position in the array listed in square brackets:

```
partyList[0] = 'Mad Hatter' ;
```

Strings need to be placed within single quotes ("), while numeric values do not.

You can add as many values as you want this way, but you can also limit the size of an array by adding a number to the `new Array()` when setting it up. For example, if you wanted to limit this array to a maximum of 100 entries you would define it this way:

```
var partyList = new Array (100);
```

All 100 values in the array will initially be set to null and if you tried to set a value to entry 101, it would be ignored.

3. A more compact method for creating an array is to simply list the array values when you set up the array.

```
var temperature = new Array (95, 63,  
    · 86, 12, 32);
```

This creates an array with five numeric entries (as opposed to strings). The disadvantage to this method is that it makes it more difficult to find values in the list if the list gets very long.

4. Another shorthand method for creating an array is to use square brackets:

```
var menu = ['tomato soup',  
    · 'sardines', 'bleu cheese',  
    · 'breath mints'];
```

This works identically to the method shown in step 3, but saves a bit more space.

5. To set up a multidimensional array, use shorthand similar to that shown in the previous steps, but adding multiple lists:

```
var tripPrices = [[100, 200, 300],  
    · [400,500,600], [700,800,900]];
```

In this example, I created a 2-dimensional (2D) array with 3 columns and 3 rows.

Accessing an array

Arrays are repositories for data, but until you make use of the data in your Web page, they really don't do much. Of course, what you do with the stored data depends on what type they are. If they're numeric data, you may want to perform computations. If the array records string data, then you may simply want to display them. Regardless of what you are doing with the data, you will need to access them.

Accessing a single value in an array is as straightforward as accessing a variable's value. However, it gets a bit more complex when you need to access the values in multidimensional arrays, or when you want to access all the values in an array.

To access the data in an array:

1. Add the array (or arrays) to your HTML. You can either add the arrays directly to your HTML code within a `<script>` tag or link to an external JavaScript file as shown in **Code 3.2**.

2. `someNumbers[0]`

To access a one-dimensional array, type in the array name, followed by square brackets containing the value for the slot in the array you want to access. This can be a numeric value or a variable.

3. `tripPrices[2][1]`

To access a multidimensional array, type the array name followed by square brackets for each dimension in the array, indicating the slots you want to access, either by numeric values or variables.



Tip

Arrays work just like variables in your code, and are especially useful for including dynamic content. In this example, I'm using an array to access a string with the name of an image to load onto the page (**Figure 3.1**): ``.

Code 3.2 The external JavaScript file with the arrays is linked to the main HTML page where it will be used. This page shows examples of how to access the data in the various arrays.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Transitional//EN" "http://www.w3.org/TR/
  xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xml:lang="en">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I
      Working with Arrays</title>
    <script language="javascript"
      type="text/javascript"
      src="arrays.js">
    </script>
  </head>
  <body>
    <script language="javascript"
      type="text/javascript">
      document.writeln(someNumbers[0]);
      document.writeln(tripPrices[2][1]);
      document.writeln('');
    </script>
  </body>
</html>
```

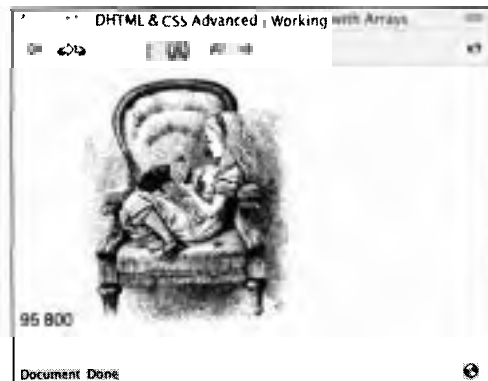


Figure 3.1 Values from the array are displayed on the page using JavaScript. The image is added by reading a URL stored in an array and then combining that (using JavaScript) with an `` tag.

Code 33 The external JavaScript file is linked to and then the function `displayArray()` uses a for loop to read all the values in an array and write them onto the page using the `innerHTML` method.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
 Transitional//EN" "http://www.w3.org/TR/
 xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
 xml:lang="en">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I
      Working with Arrays</title>
    <script language="javascript"
      type="text/javascript"
      src="arrays.js">
    </script>
    <script language="javascript"
      type="text/javascript">
      function displayArray(arrayName,
        areaName) {
        var object=document.getElementById
          (areaName);
        object.innerHTML='
        for (var i = 0; i <
          arrayName.length; i++) {
          if (arrayName [i] != null) {
            object.innerHTML +=
              arrayName [i];
            if (i < (arrayName.
              length-1)) object.
              innerHTML += ', '
          }
        }
      }
    </script>
  </head>
  <body onload="displayArray(partyAttend,
    'partyList')">
    <h1>Party List</h1>
    <div id="partyList">
      <!-- Dynamically Filled --></div>
  </body>
</html>
```

To access all the contents of an array:

1. Add the array (or arrays) to your HTML. You can either add the arrays directly to your HTML code within a `<script>` tag or link to an external JavaScript file as shown in **Code 3.3**.

2. function `displayArray(arrayName, areaName) { ... }`
Add the function `displayArray()` in your page. This function writes all of the elements in the specified array (`arrayName`) into the specified object (`areaName`) on the page. Although you don't have to use this function in order to access an array's values, it is a convenient way to display an array.

continues on next page

3. `onload="displayArray(partyAttend, 'partyList')"`

Add an event handler to trigger the `displayArray()` function created in step 2. In this example, we are using an `onload` event handler to write the array `partyAttend` into the area designated by the ID `partyList` (Figure 3.2).

4. `<div id="partyList">...</div>`

Add an object with an ID to define the area into which the array in step 3 will be written. You can place any initial content you want in this area, but it will be replaced once the array is written into it.

✓ Tips

- Remember that the first slot in an array is 0, not 1. So `someArray[0]` is a reference to the first value in the array. This throws a lot of people off since most people start counting at 1.
- You can also store multiple related data values using data objects, which are explained later in this chapter.

Although the example for reading an entire array in this section is intended for use in a one-dimensional array, you can use the same technique for multidimensional arrays. You will, however, need to nest a `for` loop for each dimension in the array.

- Not only can arrays hold numerical and string data, they can hold data objects and Boolean expressions as well.
- You can combine multiple data types (string, numeric, data object, and Boolean) in a single array, but this may lead to programming difficulties unless you're careful when constructing your functions.



Figure 3.2 The entire array is displayed, each value separated by a comma.

Although the examples in this chapter show arrays with hard-coded data, if you're also working with server-side technologies such as PHP, ASP, JSP, or ColdFusion, arrays are useful for loading data from a database into a Web page.

Code 3.4 The external JavaScript array being used with Code 3.5.

```
Code
var partyAttend = new Array();
partyAttend[0] = 'Tweedle Dee';
partyAttend[1] = 'Tweedle Dum';
partyAttend[2] = 'White Knight';
partyAttend[5] = 'Alice';
partyAttend[4] = 'Red Queen';
```

Changing an Array

Just as you can change the values in a variable after the Web page loads, you can change the values stored in an array or even add new values to them.

To make changes to an array:

1. Create your array (**Code 3.4**). You can either place the array directly in the HTML page or place it in an external JavaScript file and link to it.

2. `<script language="javascript" type="text/javascript" src="arrays.js"></script>`

If you are importing your array from an external JavaScript file, add a link in your HTML code (**Code 3.5**).

continues on next page

Code 3.5 The external JavaScript file (Code 3.4) containing the array is linked to. As with Code 3.3, the function `displayArray()` is used to read and write the array. The function `changeArray()` is used to add or remove a value from the array.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
<title>DHTML & CSS Advanced I Working with Arrays I Sorting an Array</title>
<script language="javascript" type="text/javascript" src="arrays.js">
</script>
<script language="javascript" type="text/javascript">
function changeArray(arrayName,areaName,newValue) {
    var arrayLength = arrayName.length;
    var valueReplaced = 0;
    for (var i = 0; i < arrayLength; i++) {
        if (arrayName[i] == newValue) {
            arrayName[i] = null;
            i = arrayLength + 1;
            valueReplaced = 1;
        }
    }
}
```

(code continues on next page)

Code 3.5 continued

```
    }
    if (valueReplaced == 0) arrayName[arrayLength+1] = newValue;
    displayArray(arrayName,areaName);
}

function displayArray(arrayName,areaName) {
    var object=document.getElementById(areaName);
    object.innerHTML=''
    for (var i = 0; i <= arrayName.length; i++) {
        if (arrayName [i] != null) {
            object.innerHTML += arrayName [i];
            if (i <= (arrayName.length-1)) object.innerHTML += ', ';
        }
    }
}
</script>
</head>
<body onload="displayArray(partyAttend,'partyList')">
    <h1>Party List</h1>
    <div id="partyList">
        <!-- Dynamically Filled --></div>
    <br />
    Add/Remove
    <a href="#" onclick="changeArray(partyAttend,'partyList','White Queen')">White Queen</a> |
    <a href="#" onclick="changeArray(partyAttend,'partyList','Red Knight')">Red Knight</a> |
    <a href="#" onclick="changeArray(partyAttend,'partyList','Humpty Dumpty')">Humpty Dumpty</a>
</body>
</html>
```

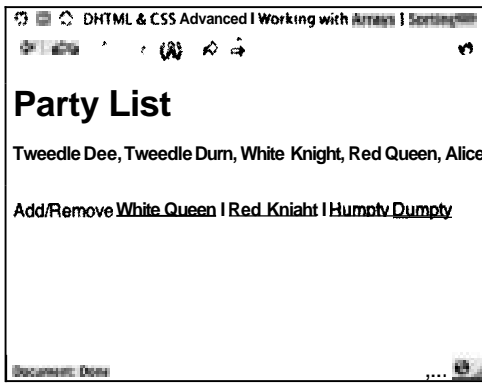


Figure 3.3 The original party list with options underneath to add new guests.

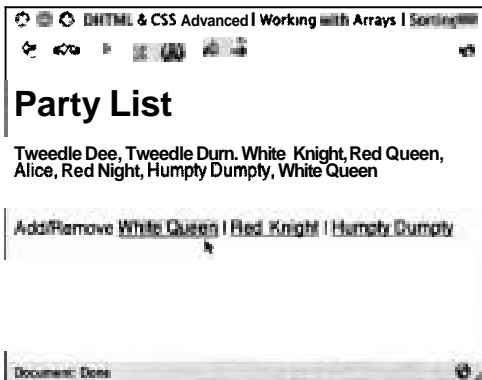


Figure 3.4 After one of the alternate party guests is selected, they are added to the list.

3. `function changeArray(arrayName, areaName, newValue) {...`

Add the function `changeArray()` to your page. This script compares the value being added to the values already in the array and removes it if it already exists or adds it to the end of the array if it does not. Finally, this function triggers the function `displayArray()` in order to display the newly adjusted array.

4. `function displayArray(arrayName, areaName) {...`

Add the function `displayArray()` to your page. This function writes all of the elements in the specific array (`arrayName`) into the specific object on the page (`areaName`). Although you don't have to use this function in order to access an array's values, it is a convenient way to display an array.

5. `onload="displayArray(partyAttend, 'partyList')"`

Add an event handler to trigger the `displayArray()` function created in step 3 to display the values initially in the array. In this example, we are using an `onload` event handler to write the array `partyAttend` into the area designated by the ID `partyList`.

6. `<div id="partyList">...</div>`

Add an object with an ID to define the area into which the array specified in step 4 will be written.

7. `White Queen`

Add an event handler to trigger the function `changeArray()` feeding it the name of the array you want to change, the area in which you want the revised array displayed, and the information you want to add to the array. In this example, I'm using an `onclick` event handler to add the White Queen to the array `partyAttend` and display the array in the area identified as `partyList` (Figures 3.3, 3.4).

Sorting an Array

Although you can manually sort the elements in an array when you create it, often you'll find it useful to sort the array alphabetically or numerically later. Although JavaScript provides the simple `sort()` method, its simplicity is deceptive. This method will only look at the ASCII value of the first character in a string. This means it's case-sensitive and uppercase letters will appear before lowercase.

To complicate matters further, the `sort()` method treats numeric values as strings as well, so that 20 would appear well before 7, because the method sees 20 as a string starting with 2.

For strings, the simplest answer is to make sure to keep the case of the first letter consistent throughout the array. For numbers, though, we will need to use a little JavaScript trick

Combining and Splitting Arrays

You can also quickly combine two arrays into one, or split a single array apart.

To combine two arrays, use this format:

```
var unitedArray =  
  - firstArray.concat(secondArray);
```

This creates a new array with the values of `firstArray[]` and then the values of `secondArray[]`. So, if the first array contained 4, 5, 6 and the second array contained 1, 2, 3, the new array would contain 4, 5, 6, 1, 2, 3.

To split an array, use this format:

```
var splitArray =  
  - firstArray.splice(2,4);
```

This creates a new array with the third, fourth, and fifth values in the array, but leaves the original array intact. So, for example, if the first array contains the values 4, 5, 6, 1, 2, 3, the split array will contain the values 6, 1, and 2.

Code 3.6 The arrays being used with Code 3.7.

```
var partyAttend = new Array();
partyAttend[0] = 'Tweedle Dee';
partyAttend[1] = 'Tweedle Dum';
partyAttend[2] = 'White Knight';
partyAttend[3] = 'Alice';
partyAttend[4] = 'Red Queen';
var temperature = new Array(95, 63, 86, 12, 32);
```

To sort an array:

1. Create your array (Code 3.6). You can either place the array directly in the HTML page, or place it in an external JavaScript file and link to it.
2. function `evaluateAscending(a,b) {...}`
In your HTML page, add the function `evaluateAscending()` to your page (Code 3.7). This function checks to see whether the array contains strings (which returns NaN for "Not a Number") or numbers and then reorders the elements in the array so that they are in ascending order.

continues on page 67

Code 3.7 Link to the external JavaScript file (Code 3.6). The function `displayArray()` (Code 3.3) is added along with the functions `sortArray()`, `evaluateAscending()`, and `evaluateDescending()` to change the array order.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
-xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced | Working with Arrays | Sorting an Array</title>
    <script language="javascript" type="text/javascript" src="arrays.js">
    </script>
    <script language="javascript" type="text/javascript">
      function evaluateAscending(a,b) {
        if (isNaN(a)) {
          var stringA = a.toLowerCase();
          var stringB = b.toLowerCase();
          if (stringA < stringB) return -1;
          if (stringA > stringB) return 1;
          return 0;
        }
        else return a - b;
      }
      function evaluateDescending(a,b) {
        if (isNaN(a)) {
          var stringA = a.toLowerCase();
          var stringB = b.toLowerCase();
```

(code continues on next page)


```

    if (stringA < stringB) return 1;
    if (stringA >= stringB) return -1;
    return 0;
}
else return b - a;
}
function sortArray(arrayName,areaName,direction) {
    if(direction == 'ascending') arrayName.sort(evaluateAscending);
    if(direction == 'descending') arrayName.sort(evaluateDescending);
    displayArray(arrayName,areaName);
}
function displayArray(arrayName,areaName) {
    var object=document.getElementById(areaName);
    object.innerHTML=''
    for(var i = 0; i <= arrayName.length; i++) {
        object.innerHTML += arrayName [i];
        if(i <= (arrayName.length-1)) object.innerHTML += ', ';
    }
}
</script>
</head>
<body onload="displayArray(temperature,'tempList');displayArray(partyAttend,'partyList')">
    <div id="tempList">
        <!-- Dynamically Filled --></div>
        Sort List <a href="#" onclick="sortArray(temperature,'tempList','ascending')">Ascending</a>
        | <a href="#" onclick="sortArray(temperature,'tempList','descending')">Descending</a>
    </div>
    <br />
    <div id="partyList">
        <!-- Dynamically Filled --></div>
        Sort List <a href="#" onclick="sortArray(partyAttend,'partyList','ascending')">Ascending</a>
        | <a href="#" onclick="sortArray(partyAttend,'partyList','descending')">Descending</a>
    </div>
</body>
</html>

```

3. function evaluateDescending(a,b) {...}

Add the function `evaluateDescending()` to your page. This function checks to see whether the array is a string (again returning the value NaN) or numbers and then reorders the elements in the array so that they are in descending order.

**4. function sortArray(arrayName,
 · areaName,direction) {...}**

Add the function `sortArray()` to your page. This function will execute either the `evaluateAscending()` or `evaluateDescending()` function depending on the value of the variable `direction`.

**5. function displayArray(arrayName,
 · areaName) {...}**

Add the function `displayArray()` to your page. This function writes all of the elements in a specific array (represented using the variable `arrayName`) into a specific object on the page (represented using the variable `areaName`). Although you don't have to use this function in order to access an array's values, it is a convenient way to display an array.

```
6. onload="displayArray(temperature,  
  'tempList');displayArray  
  (partyAttend,'partyList')"
```

Add an `onload` event handler to display the arrays when the page first loads.

```
7. <div id="tempList">...</div>
```

Add an object with an ID to define the area into which the arrays specified in step 6 will be written.

continues on next page

```
8. onclick="sortArray(temperature,
  - 'tempList', 'ascending')"
```

Add an event handler to trigger the function `sortArray()`, feeding it the name of the array, the area in which you want the revised array displayed, and the direction (ascending or descending) you want it sorted. In this example, I'm calling up the array `temperature`, writing it into the area identified as `tempList`, and sorting into `ascending` order (Figures 3.5, 3.6, and 3.7).

✓ **Tips**

If you run into trouble always keep in mind that when applied to a string, the sort function is looking at the ASCII value of the first character in the string, which is case sensitive.

Sorting a multidimensional array using the `sort()` method will only sort the first list of values.

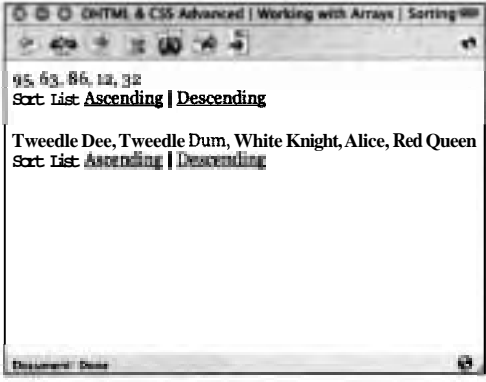


Figure 3.5 When the page first loads, the arrays are out of order.

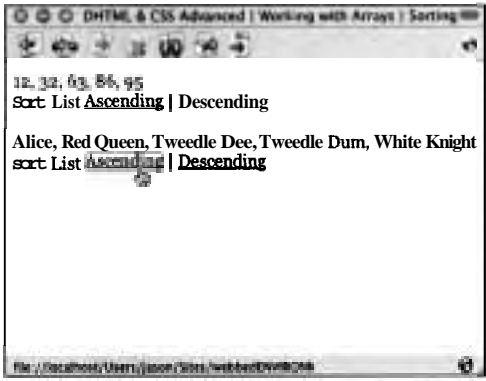


Figure 3.6 After the Ascending link is clicked.

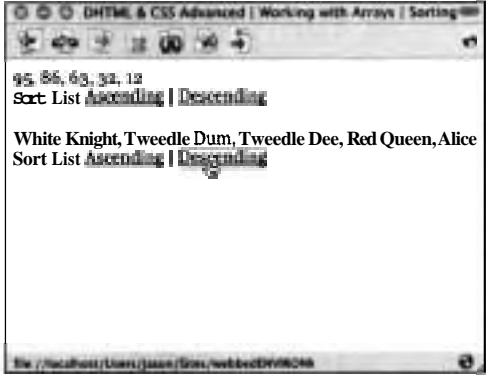


Figure 3.7 After the Descending link is clicked.

Code3.8 The data object is built in the `defineObject()` function, which also associates the function `displayImage()`, which writes an image onto the page using the parameters defined in the data object.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
 Transitional//EN" "http://www.w3.org/TR/
  xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xml:lang="en">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I
      *Working with Objects</title>
    <script type="text/javascript"
      language="javascript">
      function displayImage() {
        document.write ('<img src = " ' +
          this.imgName + " " width=" ' +
          this.imgWidth + " " height=" ' +
          this.imgHeight + "'>');
      }
      function defineObject(imgName,
        imgWidth, imgHeight) {
        this.imgName = imgName;
        this.imgWidth = imgWidth;
        this.imgHeight = imgHeight;
        this.showImage = displayImage;
      }
    </script>
  </head>
  <body>
    <script type="text/javascript"
      language="javascript">
      var image1 = new defineObject
        ('Alice-2-2. jpg', 200, 240);
      var image2 = new defineObject
        ('Alice-2-4. jpg', 300, 188);
      image1.showImage();
      image2.showImage();
    </script>
  </body>
</html>
```

Working with Data Objects

A third way to store variable information (variables and arrays being the first two) is to set up a data object. Data objects basically allow you to associate variables to a variable. Like an array, this lets you store multiple values in a single variable signifier, but unlike arrays, it allows you to give each of these values a variable name rather than a number.

For example, you might create a data object named `userProfile1` and then associate additional values with this object using the format

```
userProfile1.firstName,
userProfile1.lastName,
userProfile1.homePhone
```

and so on. Each of these records a different chunk of information, but are all associated with the data object `userProfile1`, so that if you pass `userProfile1` to a function, all of the values go along with it.

Furthermore, data objects can be directly associated with a function, allowing the function to be triggered with the data object automatically passed to it and then accessed using the `this` object.

To create a data object:

1. function `displayImage()` { ... }

Add the function `displayImage()` to your Web page. This function is an example of how to assign a function directly to the object, in this case allowing it to use its data to display an image (**Code3.8**).

continues on next page

```
2. function defineObject(imgName,
    .imgWidth,imgHeight) {...}
```

Add the function `defineObject()` to your Web page. This function is an example of how to assign values to an object, in this case a string for the image name, and then the image's width and height. In addition, the function `displayImage()` is associated with the object (see step 4).

```
3. var imagel = new defineObject
    ('Alice-2-2. jpg',200,240);
```

Next, you will need to define the object itself using the `defineObject()` function from step 2. In this example, we are creating a new object called `imagel`.

```
4. imagel.showImage();
```

In step 2, we associated the function `displayImage()` to the object. That allows you to execute the function directly, with all of the values in the object passed to it with no extra effort. Running this function causes the images to be displayed in the page (**Figure 3.8**).

✓ Tips

This is just one small example of the types of data you can assign to an object. For example, you could create an object to record data about an individual (first name, last name, address, and so forth) or use a single data object to record all of the values for an object on the screen (ID, width, height, visibility, and the like).

Although they can work together, do not confuse data objects with Web page objects, which are created using the ID tag. You can actually create Web page objects by using data objects to record the Web page objects' properties, making it easier to change them dynamically.



Figure 3.8 Information about the images of the Queen and Alice are stored in data objects.

What's all of this 'this'?

You'll notice in many functions used in this section and throughout the book that I use the keyword `this`. The `this` keyword allows you to directly access data for the object (data or page object) that invoked the function. Thus, it is invaluable for allowing functions to behave in context to the object being acted upon.

Code 39 First set up the frameset document `index.html` with the "invisible" `dataStore` frame to hold our data.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Frameset//EN" "http://www.w3.org/TR/xhtml1/
DTD/xhtml1-frameset.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xml:lang="en">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I
      Storing Data in Frames</title>
  </head>
  <frameset cols="*">
    <frameset cols="0,*" border="0"
      frameborder="no" framespacing="0">
      <frame name="dataStore"
        src="dataStore.html"
        noresize="noresize"
        scrolling="no"/>
      <frame name="content"
        src="page1.html"
        noresize="noresize"/>
    </frameset>
  </frameset>
</html>
```

Storing Data in Frames

Although the technology has been around since Netscape 2, frames have been a maligned and misunderstood technology. Many Web developers avoid using them like the plague, for a variety of reasons. However, one great use for frames is when they are heard but not seen.

The idea here is to create a hidden frame, one that is not visible to the visitor, where you can store data that will be available to Web pages in the main frame even when the content of that frame changes.

To do this, we create a frameset where one frame (let's call it `dataStore`) has a 0 width or height and no border. Into that page, we load an HTML page that contains all of the JavaScript variables we will need. In the main frame, we then load Web pages that will read and write to the invisible Web page.

To store data in frames:

1. `index.html`

Set up your frames document (**Code 3.9**) with at least two frames, one of which should have its height or width set to 0, scrolling set to no, name set to `dataStore`, and source set to `dataStore.html`. In the second frame, use `page1.html` as its initial source.

continues on next page

2. `dataStore.html`

Set up a new HTML document that is blank except for a `<script>` tag with variable declarations for all of the data you will be storing (Code 3.10). In this example, we are using simple JavaScript variables, but you could use arrays or objects as well.

3. `page1.html`

Set up a new HTML page and save it as `page1.html` (Code 3.11). Steps 5–7 apply to this page.

4. function `testFrame()` {...}

Add the function `testFrame()` to your page. This checks to make sure the page is in the proper frameset (or else this will not work) and opens that page (`index.html`) if it is not.

5. function `placeDataFrame(url)` {...}

Add the function `placeDataFrame()` to your page. This function will write data from the first form on the page into variables in the frame `dataStore` and then open the next page.

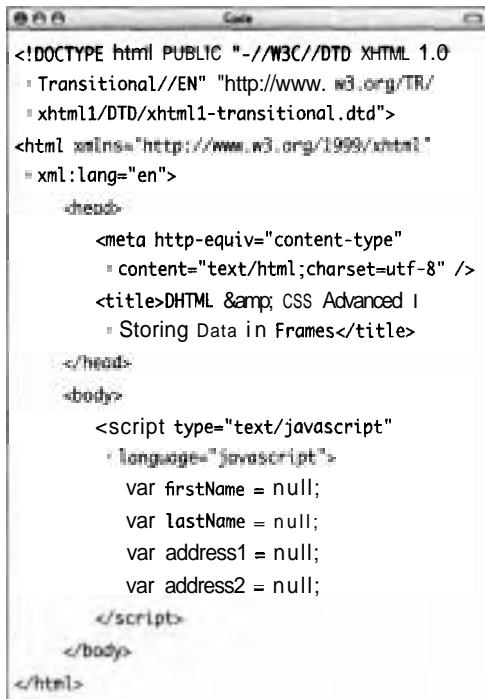
6. `onload="testFrame()"`

In the `<body>` tag, add an `onload` event handler to trigger the `testFrame()` function.

7. `action="javascript:placeDataFrame('page2.html')"`

Set up a form, with the action triggering the `placeDataFrame()` function. When the form Submit button is clicked (Figure 3.9), this will place the data into the `dataStore` frame before continuing to the next page (`page2.html`).

Code 3.10 Then set up the page used to store data in JavaScript variables, `dataStore.html`.



```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  = xml:lang="en">
  <head>
    <meta http-equiv="content-type"
      = content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I
      = Storing Data in Frames</title>
  </head>
  <body>
    <script type="text/javascript"
      = language="javascript">
      var firstName = null;
      var lastName = null;
      var address1 = null;
      var address2 = null;
    </script>
  </body>
</html>
```

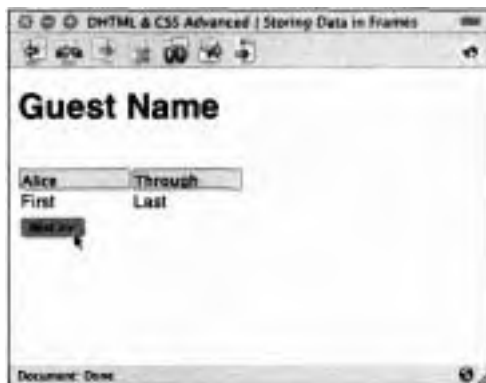


Figure 39 The initial page (`page1.html`). The visitor enters information and clicks Next.

continues on page 74

Code 3.11 The first page in the sequence (`page1.html`) allows the visitor to enter a first and last name, which are then stored in `dataStore.html`.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
  <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  <title>DHTML & CSS Advanced | Storing Data in Frames</title>
  <script type="text/javascript" language="javascript">
    function testFrame() {
      if(self.location == top.location) location.href = 'index.html';
    }
    function placeDataFrame(url) {
      top.dataStore.document.firstName = document.forms[0].firstName.value;
      top.dataStore.document.lastName = document.forms[0].lastName.value;
      location.href = url;
    }
  </script>
</head>
<body onload="testFrame()">
  <form id="FormName" action="javascript:placeDataFrame('page2.html');"
  method="get" name="FormName">
    <h1>Guest Name</h1>
    <br />
    <table>
      <tr>
        <td><input class="inputfield" type="text" name="firstName" value="" size="12"
        maxlength="15" tabindex="2"/><br />
        <span class="instructions">First</span>
        </td>
        <td><input class="inputfield" type="text" name="lastName" size="12" maxlength="15"
        tabindex="3"/><br />
        <span class="instructions">Last</span>
        </td>
      </tr>
    </table>
    <input type="submit" name="next" value="Next & & ">
  </form>
</body>
</html>
```


8. page2.html

Set up a new HTML page and save it as `page2.html` (**Code 3.12**). This page is almost identical to `page1.html`, but includes one additional function used to read the data from the frame. Steps 9–10 apply to this page.

9. function `getDataFrame(dataName) {...}`

Add the function `getDataFrame()` to this page. This function can be used to grab data from any variable in the `dataStore` frame.

10. `getDataFrame('firstName')`

Add a function call to `getDataFrame()`, passing it the name of the variable you want to access. In this example, I'm simply displaying the value for the variables (the first and last names) on the page (**Figure 3.10**).

continues on page 76

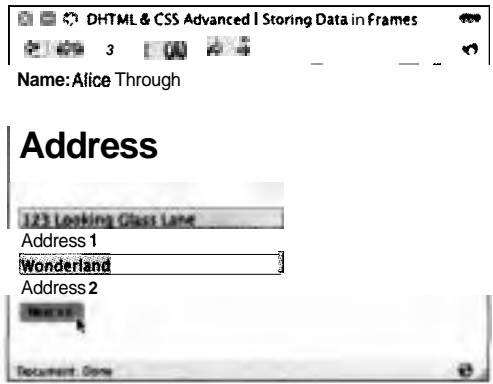


Figure 3.10 The second page in the sequence (`page2.html`) displays the information entered from the previous page and allows the visitor to enter an address.

Code 3.12 The second page in the sequence (`page2.html`) can not only write data to the `dataStore` frame, it also reads the data.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xml:lang="en">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced |
      Storing Data in Frames</title>
    <script type="text/javascript"
      language="javascript">
      function testFrame() {
        if (self.location ==
          top.location) location.href =
          'index.html';
      }
      function getDataFrame(dataName) {
        return top.dataStore.document
          [dataName];
      }
      function placeDataFrame(url) {
```

(code continues on next page)

```

top.dataStore.document.address1 = document.forms[0].address1.value;
top.dataStore.document.address2 = document.forms[0].address2.value;
location.href = url;
}
</script>
</head>
<body onload="testFrame()">
  <script type="text/javascript" language="javascript">
    if (top.dataStore) {
      document.write('Name: ' + getDataFrame('firstName') + ' ' + getDataFrame('lastName') );
    }
  </script>
  <br />
  <br />
  <form id="FormName" action="javascript:placeDataFrame('page3.html')"
  + method="get" name="FormName">
    <h1>Address</h1>
    <br />
    <table>
      <tr>
        <td><input class="inputField" type="text" name="address1" value="" size="30"
        + maxlength="38" tabindex="2"/><br/>
        <span class="instructions">Address 1</span>
      </td>
    </tr>
    <tr>
      <td><input class="inputField" type="text" name="address2" size="30" maxlength="38"
      + tabindex="3"/><br/>
        <span class="instructions">Address 2</span>
      </td>
    </tr>
  </table>
  <input type="submit" name="next" value="Next &gt;&gt;"/>
</form>
</body>
</html>

```

11. page3.html

Set up a new HTML page and save it as `page3.html` (**Code 3.13**). This page uses the function `getDataFrame()` to display all of the data recorded in the `dataStore` frame (**Figure 3.11**).

Code 3.13 The final page in the sequence (`page3.html`) writes the data from the previous two pages, which it reads from the `dataStore` frame.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
+ Transitional//EN" "http://www.w3.org/TR/
+ xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
+ xml:lang="en">
  <head>
    <meta http-equiv="content-type"
      + content="text/html; charset=utf-8" />
    <title>DHTML & amp; CSS Advanced !
      + Storing Data in Frames</title>
    <script type="text/javascript"
      + language="javascript">
      function testFrame() {
        if(self .location ==
          + top.location) location.href =
            + 'index.html';
      }

      function getDataFrame(dataName) {
        return top.dataStore.document
          [dataName];
      }
    </script>
  </head>
  <body onload="testFrame()">
    <script type="text/javascript"
      + language="javascript">
      if (top.dataStore) {
        document.write('Name: ' +
          + getDataFrame('firstName') + ' '
          + getDataFrame('lastName') );
        document.write('<br />Address 1: '
          + + getDataFrame('address1'));
        document.write('<br />Address 2: '
          + + getDataFrame('address2'));
      }
    </script>
    <br />
    <br />
  </body>
</html>
```



Figure 3.11 The final page (`page3.html`) displays all of the data from the previous two pages.

✓ Tips

Although I used simple JavaScript variables in this example, one of the benefits of using a hidden frame is that you can store any data type (variable, array, or data object) in the page.

You can also place your JavaScript functions in the hidden frame so that they do not have to load with each page, and then use the same format to reference them as you do variables.

One of the downsides of using frames for data storage is that if the visitor reloads the page, the information is lost.

In order for two Web pages to communicate with each other across frames, they must reside within the same domain. This is a security feature to prevent unscrupulous Web sites from stealing data.

One of the downsides generally associated with frames has to do with bookmarking. Many Web browsers will bookmark the frameset, so any time the visitor returns, they always start at the top page regardless of where they were when they set the bookmark. However, in situations where you are entering data, this will actually work to your advantage since it ensures that the visitor always starts on the first page.

Storing Data in URLs

URLs provides a unique "address" for all of the billions of Web pages. However, a URL can also contain strings of information, inserted after a question mark, which can then be interpreted by the page being loaded. These data are not literally a part of the URL and do not affect what page loads, although they can be used to affect the content on the page in a variety of ways.

In this example we will be using the URL method to pass form data from page to page. This allows us to create a multipage form rather than placing everything on one page.

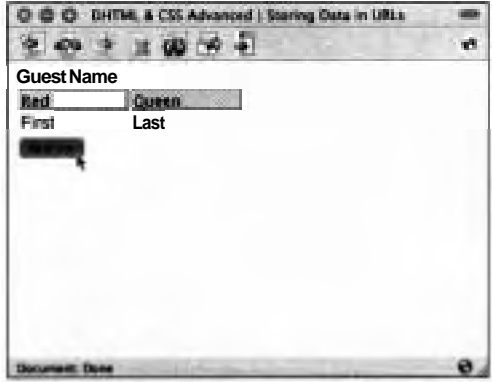


Figure 312 The initial page (page1.html), where the visitor can enter their first and last name.

To store data in a URL:

1. index.html
Set the first page in the sequence of pages that will have data placed between them (**Code 3.14**). Steps 2 and 3 apply to this page.
2. function placeDataURL(url) {...}
Add the function `placeDataURL()` to your page. This function will add data from the first form on the page into the URL for the next page, after a question mark (?). Each data string is in turn separated by a semicolon.
3. action= "javascript:placeDataURL
= ('page2.html')"
Set up a form, with the action triggering the `placeDataURL()` function. When the form Submit button is clicked (**Figure 3.12**), this will add the data to the URL before proceeding to the next page (`page2.html`). The URL that results will look something like this:
`page2.html?Alice;Glass;`

continues on page 80

Code 3.14 The first page in the sequence (`page1.html`) uses the function `placeDataURL()` to read data entered by the visitor from the form fields and then embed this information into the URL for the next page.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I Storing Data in URLs</title>
    <script type="text/javascript" language="javascript">
      function placeDataURL(url) {
        var firstName = document.forms[0].firstName.value;
        var lastName = document.forms[0].lastName.value;
        location.href = url + '?' + escape(firstName) + ';' + escape(lastName) + ';';
      }
    </script>
  </head>
  <body>
    <form id="FormName" action="javascript:placeDataURL('page2.html')"
    -method="get" name="FormName">
      <b>Guest Name</b><br/>
      <table>
        <tr>
          <td><input class="inputField" type="text" name="firstName" value="" size="12"
          -maxlength="15" tabindex="2"/><br/>
            <span class="instructions">First</span>
          </td>
          <td><input class="inputField" type="text" name="lastName" size="12" maxlength="15"
          -tabindex="3"/><br/>
            <span class="instructions">Last</span>
          </td>
        </tr>
      </table>
      <input type="submit" name="next" value="Next &gt;&gt;"/>
    </form>
  </body>
</html>
```

4. `page2.html`

Set up a new HTML page and save it as `page2.html` (**Code 3.15**). This page is almost identical to `page1.html`, but includes one additional function used to read the data from the URL. Steps 5–11 apply to this page (**Figure 3.13**).

5. `var stringBreak = new Array();`
Initialize a new array called `stringBreak[]` that will be used to record the number of characters in each data string.

6. `function getDataURL() {...}`
Add the function `getDataURL()` to your Web page. This function records the entire string of values after the question mark. If there are no values, the function then moves to the first Web page (`index.html`) so that the data can be entered.

7. `function breakString() {...}`
Add the function `breakString()` to your Web page. This function finds the position of each semicolon in the full string (beginning with 0). This information can then be used to extract each substring from the full string.

8. `var data = getDataURL();`
Set up a new variable called `data` that will be used to store the full string.

9. `breakString();`
Add a function call for `breakString()`. The array `stringBreak[]` will now have the locations to each of the semicolons in the string.

10. `var firstName = data.substring(0, stringBreak[0], stringBreak[1]);`
Set up a variable to record the first substring, which runs from `stringBreak[0]` to `stringBreak[1]`.

continues on page 82

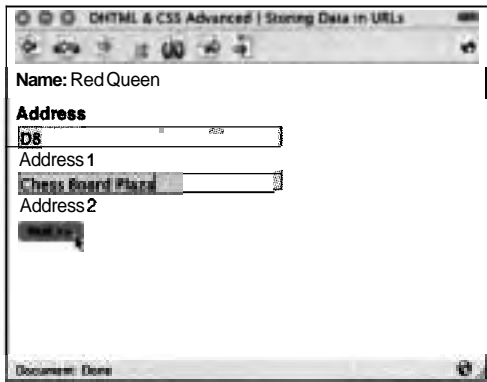


Figure 3.13 The second page (`page2.html`) displays the information from the first page and allows the visitor to enter an address.

Code 3.15 In `page2.html`, the function `getDataURL()` takes the data from the URL and displays it in the page.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  > xml:lang="en">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced |
      Storing Data in URLs</title>
    <script type="text/javascript"
      > language="javascript">
      var stringBreak = new Array();
      function getDataURL() {
        var searchString = unescape
          (location.search.substring
            (1, location.search.length));
        if (searchString.length > 0) {
          return searchString
        }
        else location.href = 'index.html';
      }
      function breakString() {
        stringBreak[0] = 0;
        i = 1;
```

(code continues on next page)

Code 3.15 continued

```

do {
    stringBreak[i] = data.indexOf(';', (stringBreak[i-1]+1));
    i += (stringBreak[i] != -1) ? 1 : 0;
} while (stringBreak[i]++ != -1)
}
function placeDataURL(url) {
    var address1 = document.forms[0].address1.value;
    var address2 = document.forms[0].address2.value;
    location.href = url + '?' + escape(firstName) + ';' + escape.lastName) + ';' +
        escape(address1) + ';' + escape(address2) + ';';
}
</script>
</head>
<body>
<script type="text/javascript" language="javascript">
    var data = getDataURL();
    breakString();
    var firstName = data.substring(stringBreak[0],stringBreak[1]);
    var lastName = data.substring((stringBreak[1]+1),stringBreak[2]);
    document.write('Name: ' + firstName + ' ' + lastName);
</script>
<br />
<br />
<form id="FormName" action="javascript:placeDataURL('page3.html')"
    method="get" name="FormName">
    <b>Address</b><br/>
    <table>
        <tr>
            <td><input class="inputField" type="text" name="address1" value="" size="38"
                maxlength="38" tabindex="2"/><br/>
                <span class="instructions">Address 1</span>
            </td>
        </tr>
        <tr>
            <td><input class="inputField" type="text" name="address2" size="38" maxlength="38"
                tabindex="3"/><br/>
                <span class="instructions">Address 2</span>
            </td>
        </tr>
    </table>
    <input type="submit" name="next" value="Next &gt;&gt;"/>
</form>
</body>
</html>

```



```
11. var lastName = data.substring  
    › ((stringBreak[1]+1),stringBreak  
    › [2]);
```

Set up a variable to record the second substring. This uses the same basic method as shown in step 10, but the starting position has to be offset by one to jump over the semicolon.

12. page3.html

Set up a new HTML page and save it as `page3.html` (**Code 3.16**). This page uses the function `getDataURL()` to display all of the data recorded in the URL (**Figure 3.14**).

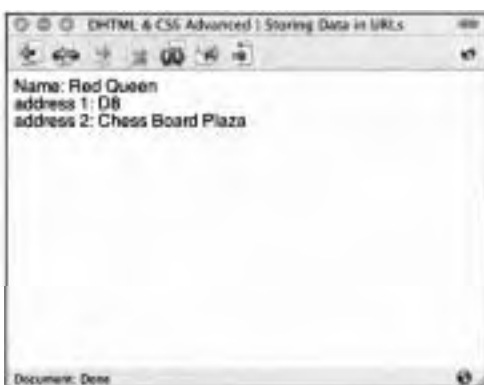


Figure 3.14 The final page (`page3.html`) in the sequence displays the data from the previous two pages, read from its URL.

✓ Tip

- The advantage of using multiple pages for forms is that you can make them contextual, so that the answers given on one page affect the questions on the next.

Escaping and Unescaping Values

Some special characters, such as the ampersand (&), cannot be directly placed in a URL, because they either are not allowed or are interpreted for use rather than as a character. In order to make sure that the string being added to a URL is safe for transmission, you have to first use the `escape()` method to translate unsafe characters into a more conducive format. On the flip side, when you're reading the data from a URL, you will need to use the `unescape()` method to translate the string back into a usable format.

Code 3.16 In page3.html, the data from the previous two pages is displayed using the `getDataURL()` function.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
= xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <title>DHTML &amp; CSS Advanced | Storing Data in URLs</title>
    <script type="text/javascript" language="javascript">
      var stringBreak = new Array();
      function getDataURL() {
        var searchString = unescape(location.search.substring(1, location.search.length));
        if (searchString.length >= 0) {
          return searchString;
        }
        else location.href = 'index.html';
      }
    ]
    function breakString() {
      stringBreak[0] = 0;
      i = 1;
      do {
        stringBreak[i] = data.indexOf(';', (stringBreak[i-1]+1));
        i += (stringBreak[i] != -1) ? 1 : 0;
      } while (stringBreak[i]++ != -1)
    }
  </script>
</head>
<body>
  <script type="text/javascript" language="javascript">
    var data = getDataURL();
    breakString();
    firstName = data.substring(stringBreak[0], stringBreak[1]);
    lastName = data.substring((stringBreak[1]+1), stringBreak[2]);
    address1 = data.substring((stringBreak[2]+1), stringBreak[3]);
    address2 = data.substring((stringBreak[3]+1), stringBreak[4]);
    document.write('Name: ' + firstName + ' ' + lastName);
    document.write('<br /> address 1: ' + address1);
    document.write('<br /> address 2: ' + address2);
  </script>
  <br />
  <br />
</body>
</html>
```

Storing Data in Cookies

Unlike the previous two methods for storing data, where once the browser window closes the data is lost for good, using cookies allows you to place a copy of the data in a small text file on the visitor's computer. This is a powerful feature, but browsers provide powerful safety features to make sure it is not abused. For the designer's part, we can use cookies not only to pass data between Web pages, but to store data even after the visitor has quit the browser altogether.

Before you get to the actual process of using cookies to store data, though, you first need to set up a library of commonly used functions not only to read, write, and delete cookies, but also to create dates based on how many days, hours, and minutes in the future you want your cookie to expire.

To store data in a cookie:

1. cookieLibrary.js

Set up an external JavaScript file called `cookieLibrary.js` (Code 3.17). This file will be used to collect several files needed to read and write cookies. Steps 2–6 apply to this file.

2. function getWholeCookie(offset) {...}

Add the function `getWholeCookie()` to your JavaScript file. This function is used to capture the value for a cookie from a string starting at the offset through to the end of the value.

3. function getCookie(name) {...}

Add the function `getCookie()` to your JavaScript. This function breaks up the cookie string, then passes the offset values to the function `getWholeCookie()` for the final value.

Code 3.17 The external JavaScript file `cookieLibrary.js` contains several functions needed to work with cookies.

```
function getWholeCookie(offset) {
    var endstr = document.cookie.indexOf
        (';', offset);
    if(endstr ==-1) {
        endstr = document.cookie.length;
    }
    return unescape(document.cookie.substring
        (offset,endstr));
}

function getCookie(name) {
    var arg = name + '=';
    var argLength = arg.length;
    var cookieLength = document.cookie.length;
    var i = 0;
    while(i <= cookieLength) {
        var j = i + argLength;
        if (document.cookie.substring(i,j)
            == arg) {
            return getWholeCookie(j);
        }
        i = document.cookie.indexOf(' ',i) + 1;
        if(i ==0) break;
    }
    return "";
}

function setCookie(name,value) {
    document.cookie = name + '=' +
        escape(value);
}

function tossCookie(name) {
    if (getCookie(name)) {
        document.cookie = name + ' = null;
            expires=Thu, 01-Jan-2001
            00:00:01 GMT';
    }
}
```

Code 3.18 The HTML file `page1.html` uses the function `placeDataCookie()`, which uses functions from `cookieLibrary.js` to create a cookie with the data from the form fields on the page.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Transitional//EN" "http://www.w3.org/TR/
  xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xml:lang="en">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I
      Storing Data in Cookies</title>
    <script type="text/javascript"
      language="javascript"
      src="cookieLibrary.js"></script>
    <script type="text/javascript"
      language="javascript">
      function placeDataCookie(url){
        var fName = document.forms[0].
          firstName.value;
        var lName = document.forms[0].
          lastName.value;
        if (navigator.cookieEnabled) {
          var filteredDate =
            filterDate(365,0,0);
          setCookie('firstName',
            fName);
          setCookie('lastName',
            lName);
          location.href = url;
        }
        else alert('You need a
          cookie-enabled browser to
          use this site. Please
          either turn on cookies or
          update your browser.');
```

(code continues on next page)

4. function `setCookie(name,value) {...}`

Add the function `setCookie()` to your JavaScript. This function is passed several values, which are used to add a cookie:

name holds the name for the cookie being created.

value is the string or numeric value being assigned to the cookie.

5. function `tossCookie(name,path,domain) {...}`

Add the function `tossCookies()` to your JavaScript. This function can be used to instantly delete a cookie by setting its expiration date to a past date (in this example January 1st 2001) so that the browser removes it as expired.

6. index.html

Set the first page in the sequence of pages that will have data placed between them (**Code 3.18**). Steps 8 and 10 apply to this page.

continues on next page

```
7. <script type="text/javascript"
  · language="javascript"
  · src="cookieLibrary.js"></script>
```

First, add `cookieLibrary.js` to the file so that the cookie functions are available to the page.

8. function placeDataCookie(url) {...}

Add the function `placeDataCookie()` into the Web page. This function uses the `getCookie()` function from the cookie library to add cookies to record the first and last name entered into the form fields.

9. action= "javascript:placeDataCookie('page2.html')"

Set up a form, with the action triggering the `placeDataCookie()` function. When the form Submit button is clicked (**Figure 3.15**), this will create two cookies before proceeding to the next page (`page2.html`).

Code 8.18 continued

```
Code
<form id="FormName" action="javascript:
· placeDataCookie('page2.html')"
· method="get" name="FormName">
  <b>Guest Name</b><br/>
  <table>
    <tr>
      <td><input class="inputField"
        · type="text" name="firstName"
        · value="" size="12"
        · maxlength="15"
        · tabindex="2"/><br/>
        <span class="instructions"
          · >First</span>
      </td>
      <td><input class="inputField"
        · type="text" name="lastName"
        · size="12" maxlength="15"
        · tabindex="3"/><br/>
        <span class="instructions"
          · >Last</span>
      </td>
    </tr>
  </table>
  <input type="submit" name="next"
    · value="Next &gt;&gt;"/>
</form>
</body>
</html>
```

Figure 3.15 The first page (`page1.html`) provides form fields for the visitor's name.

10. page2.html

Set up a new HTML page and save it as `page2.html` (**Code 3.19**). This page is almost identical to `page1.html`, but includes one additional function used to read the data from the URL. Steps 12–14 apply to this page.

continues on next page

Code 3.19 The HTML file `page2.html` uses the function `getDataCookie()` to read and display the data from the previous page that is stored in the cookies `firstName` and `lastName`.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <title>DHTML &amp; CSS Advanced ! Storing Data in Cookies</title>
    <script type="text/javascript" language="javascript" src="cookieLibrary.js"></script>
    <script type="text/javascript" language="javascript">
      function getDataCookie(cookieName) {
        if (navigator.cookieEnabled) {
          data = getCookie(cookieName);
          if (data != null) {
            return data;
          }
          else location.href = 'index.html';
        }
        else alert('You need a cookie-enabled browser to use this site. Please either turn on
          = cookies or update your browser.');
```

(code continues on next page)

```
11. <script type="text/javascript"
    › language="javascript"
    › src="cookieLibrary.js"></script>
```

Add cookieLibrary.js to the file so that the cookie functions are available to the page.

```
12. function getDataCookie(cookieName)
    = {...}
```

Add the function getDataCookie() to your Web page. This function will retrieve the data for the cookie name passed to it.

Code 3.19 continued



```
<body>
  <script type="text/javascript"
    › language="javascript">
    var firstName=getDataCookie
      =('firstName');
    var lastName=getDataCookie
      =('lastName');
    document.writeln('Name: ' +
      › firstName + ' ' + lastName);
  </script>
  <br />
  <br />
  <form id="FormName" action="javascript:
    › placeDataCookie('page3.html')"
    › method="get" name="FormName">
    <b>Address</b><br/>
    <table>
      <tr>
        <td><input class="inputField"
          › type="text" name="address1"
          › value="" size="30"
          › maxLength="30"
          › tabindex="2"/><br/>
          <span class="instructions">
            › Address 1</span>
        </td>
      </tr>
      <tr>
        <td><input class="inputField"
          › type="text" name="address2"
          › size="30" maxLength="30"
          › tabindex="3"/><br/>
          <span class="instructions">
            Address 2</span>
        </td>
      </tr>
    </table>
    <input type="submit" name="next"
      › value="Next &gt;&gt;"/>
  </form>
</body>
</html>
```

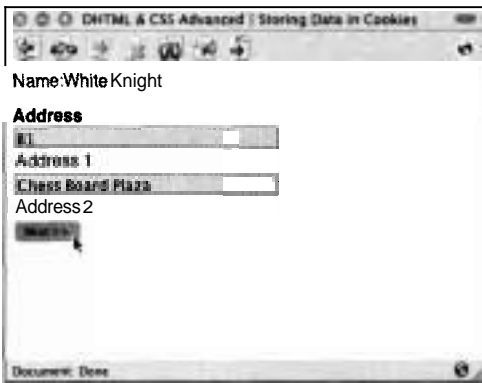


Figure 3.16 The second page in the sequence (page2.html) displays the data stored in the cookie from the first page and has form fields for the visitor's address.

Code 3.20 The HTML file page3.html displays the data stored in the previous two pages.

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
 Transitional//EN" "http://www.w3.org/TR/
 xhtml1/0TD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
 xml:lang="en">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced |
      Storing Data in Cookies</title>
    <script type="text/javascript"
      language="javascript"
      src="cookieLibrary.js"></script>
    <script type="text/javascript"
      language="javascript">
      function getDataCookie(cookieName) {
        if (navigator.cookieEnabled) {
          data = getCookie(cookieName);
          if (data != null) {
            return data;
          }
        }
        else location.href =
          'index.html';
      }

```

(code continues on next page)

```

13. var firstName=getDataCookie
    = ('firstName');

```

Add a variable to your Web page to use the function `getDataCookie()`, passing it the name of the cookie you want to find the value for (**Figure 3.16**).

14. page3.html

Set up a new HTML page and save it as `page3.html` (**Code 3.20**). This page uses the function `getDataCookie()` to display the data recorded in the cookies (**Figure 3.17**).

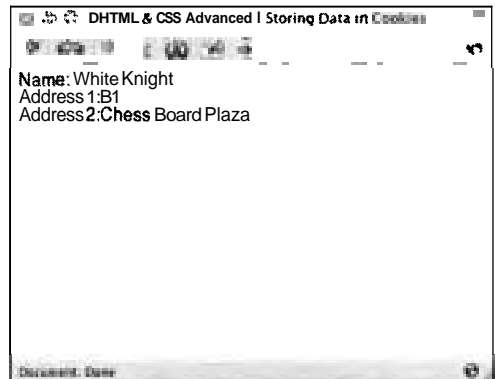


Figure 3.17 The final page (page3.html) shows the data recorded from the previous two pages.

✓ Tips

These scripts will work in Internet Explorer 4.

All cookie values are stored as strings. Thus, even when you store a number it is treated as a string (see the sidebar "Converting Strings to Numbers").

Most browsers will limit you to 20 cookies per server and the cookie can have up to 4,000 characters, but they are impractical for bulk storage.

Some browsers will keep cookie data in memory until the browser is closed and then save the data to the cookie file(s). This could cause problems if the computer or browser crashes.

- Internet Explorer saves each cookie as an individual text file, while Mozilla browsers (including Netscape) save all cookies in a single file.

Cookies are a great way to save user settings. In Chapter 7, "Controls," I'll show you how to use cookies to save a user's font-size preferences.

Code 3.20 continued

```
Code
else alert('You need a
    = cookie-enabled browser to use
    › this site. Please either turn
    o n cookies or update your
    = browser. ');
}
</script>
</head>
<body>
<script type="text/javascript"
    = language="javascript">
    firstName=getDataCookie('firstName');
    lastName=getDataCookie('lastName');
    address1=getDataCookie('address1');
    address2=getDataCookie('address2');
    document.writeln('Name: ' +
        › firstName + ' ' + lastName);
    document.writeln('<br />Address 1:
        = ' + address1);
    document.writeln('<br />Address 2:
        = ' + address2);
</script>
<br />
<br />
</body>
</html>
```

Converting Strings to Numbers

To convert the number string to a true number value, use this format:

```
var myNumValue =
    - parseFloat(myNumString);
```

Code 3.21 The delayed action uses the `setTimeout()` method to pause before triggering a function and the `clearTimeout()` method to stop the delay.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Transitional//EN" "http://www.w3.org/TR/
  xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xml:lang="en">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I
      Delaying an Action</title>
    <script>
      var justWait;
      function showAlert(theDelay) {
        alert('Has it been ' +
          (theDelay/1000) +
          ' seconds already?')
      }
      function startAlert() {
        var theDelay = document.forms[0].
          seconds.value * 1000;
        justWait = setTimeout
          ('showAlert(' + theDelay +
            ')', theDelay)
      }
      function stopAlert() {
        clearTimeout(justWait);
        alert('The timer has been
          stopped; ');
      }
    </script>
  </head>
  <body>
    <form id="EntryField" action="none"
      name="EntryField">
      <input class="inputField"
        type="text" name="seconds"
        id="theDelay" size="5"
        tabindex="0"/><br/>
      <span class="instructions">Enter
        number of seconds to pause</
        span><br />
  </body>
</html>
```

(code continues on next page)

Delaying or Stopping an Action

Generally when we start a dynamic action on a page by triggering a JavaScript function, we expect the action to take place immediately after the event that triggered it. Click the button and see the message. However, actions often need to be delayed by a few seconds or longer. For example, after clicking a drop-down menu, you may want it to pause briefly before automatically disappearing.

One way to force a pause would be to use a `for` loop to count a certain number of milliseconds. While this works, a more effective and versatile technique is to use the `setTimeout()` method. This is not only more compact, it also allows you to immediately trigger a function after the pause.

Some other added benefits of the `setTimeout()` method over `for` loops are that other JavaScripts will continue to run during the pause, and, if we so desire, we can stop the delay at any point, preventing the function from being triggered.

To set up a delayed function:

1. `var justWait;`
Add the variable `justWait` to your JavaScript outside of any function so that it is globally accessible (**Code 3.21**).
2. `function showAlert(theDelay) {...}`
Add the function `showAlert()` to your JavaScript. This function will be triggered by the function `startAlert()` after a slight delay.

continues on next page

Code 3.21 continued

```
<br />
<p>1. Enter the number of seconds to
  = delay. </p>
<p>2. Click <a href="javascript:
  = startAlert()">here</a> to start
  the delayed function.</p>
<p>3. Wait, or click <a href=
  = "javascript:stopAlert()">here</a>
  = to stop the delayed function.</p>
</form>
</body>
</html>
```

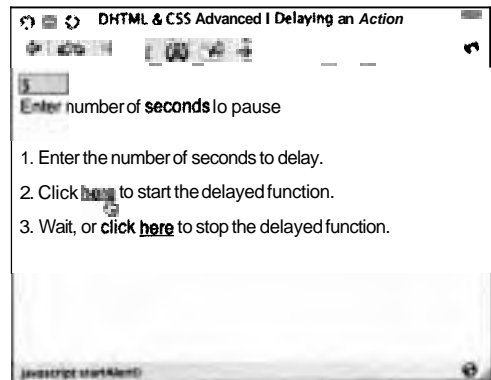


Figure 3.18 The visitor enters the number of seconds they want the function to pause before triggering the next function.

3. function startAlert() {...}

Add the function `startAlert()` to your JavaScript. This function will take the value from a specified form field (**Figure 3.18**) and multiply it by 1,000 to get the number of milliseconds to wait before triggering the `showAlert()` function using the `setTimeout()` method.

The first value is a string with the name of the function to be executed and the second value is the number of milliseconds to delay before taking the action. Notice that we are also passing the variable as a parameter.

4. function stopAlert() {...}

Add the function `stopAlert()` to your JavaScript. If the function `startAlert()` is still in pause mode and this function runs, it will stop it using the `clearTimeout()` method.



Figure 3.19 After pausing the specified number of seconds, an alert appears.



Figure 3.20 If during the pause the visitor clicks the other link, a different alert appears.

5. `here`

Add a trigger for the `startAlert()` function. In this example, I'm using a simple link tag with the function call in the href. When clicked, the link will start the delay based on the number of seconds entered in the form field (Figure 3.19).

6. `here`

Add a trigger for the `stopAlert()` function. When clicked, this function will stop the timer (Figure 3.20).

✓ Tips

If you wanted to continuously repeat the function call (rather than just triggering it once) you could use the `setInterval()` method, which works exactly like `setTimeout()` but can be canceled using the `clearInterval()` method.

- You can create a series of functions strung together, each triggering the next in the sequence. You can even create a loop that way, so that the last function triggers the first function. But be careful: infinite loops can cause browsers to crash.

Handling Errors on the Fly

No one likes them, but errors are a fact of life. Whenever you are creating anything with JavaScript, errors will occur, and this is not always due to poor programming. Often it is caused by a glitch in the browser or a fluke in your code's logic. Errors occur either while the page is loading (compile-time errors) or after the page has loaded (runtime errors). Compile-time errors are generally due to poor scripting, and thus require you to rethink your code, you can control runtime errors so that the visitor is unaware that anything has gone wrong.

To handle errors (gracefully):

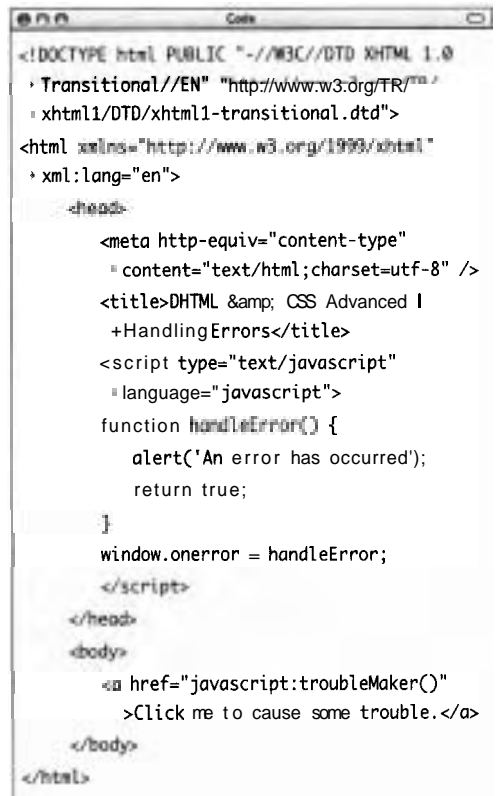
1. function handleError() {...}

Add the function `handleError()` to your JavaScript (**Code 3.22**). This function can contain anything you want, but for this example, I have it simply alerting us that an error has occurred.

2. window.onerror = handleError;

Add a global event handler for the window to trigger the function `handleError()` when there is an error.

Code 3.22 The `handleError()` function is triggered whenever an error event is generated in the window.



```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Transitional//EN" "http://www.w3.org/TR/
  xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xml:lang="en">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I
    + Handling Errors</title>
    <script type="text/javascript"
      language="javascript">
      function handleError() {
        alert('An error has occurred');
        return true;
      }
      window.onerror = handleError;
    </script>
  </head>
  <body>
    <a href="javascript:troubleMaker()"
      >Click me to cause some trouble.</a>
  </body>
</html>
```

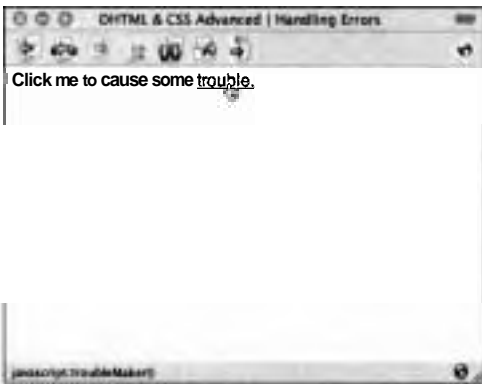


Figure 3.21 An unassuming link on the page, which triggers a JavaScript function...

3. `...`

Whenever a runtime error occurs—for example you try to trigger a function that does not exist (**Figure 3.21**)—the function `handleError()` goes into action (**Figure 3.22**).



Figure 3.22 ...but the function does not exist so an alert appears.

Blank

CONTENT

There is a saying among Web designers and developers: "Content is king." The Web is all about presenting the content that the visitor needs and expects, at the right time and in the right place. Generating the actual content (text, images, forms, tables, sounds, animations, video, and so forth) is up to you. But in this chapter I want to explore several ways to place content into an HTML file.

Imagine you are designing a large Web site with the same header on every page. Every time you need to change the content of the header, you must change every page. Not only is this process time-consuming, but the possibility of making mistakes is high. Wouldn't it be nice to have that menu in one file and import it into each page as the visitor uses the Web site? Then you can correct one file and have the changes reflected throughout the site.

To do this, you need a way to import external content into HTML files. You can use any of several methods, each of which has strengths and weaknesses.

In addition, we will look at ways to use JavaScript to generate new content dynamically after the page has been loaded.

CONTENT

There is a saying among Web designers and developers: "Content is king." The Web is all about presenting the content that the visitor needs and expects, at the right time and in the right place. Generating the actual content (text, images, forms, tables, sounds, animations, video, and so forth) is up to you. But in this chapter I want to explore several ways to place content into an HTML file.

Imagine you are designing a large Web site with the same header on every page. Every time you need to change the content of the header, you must change every page. Not only is this process time-consuming, but the possibility of making mistakes is high. Wouldn't it be nice to have that menu in one file and import it into each page as the visitor uses the Web site? Then you can correct one file and have the changes reflected throughout the site.

To do this, you need a way to import external content into HTML files. You can use any of several methods, each of which has strengths and weaknesses.

In addition, we will look at ways to use JavaScript to generate new content dynamically after the page has been loaded.

Preloading Images

Although the Web is speeding up due to the increasing use of high-speed Internet access, having a Web page that loads quickly is still the first step in creating a positive user experience. One of the chief lags in download time is always going to be the graphic content. Although Web graphics are compressed to minimize their file size, the more graphics you include in a page, the slower it will load.

To speed things along, though, most browsers will store image files in a *cache* locally on the visitor's computer and then use the local version of the file if the image is used on another page in the Web site. This can radically speed the appearance of subsequent pages in the site.

Using a simple CSS trick, we can take advantage of the cache to load images invisibly on the home page before they are ever used in the Web site, so that when the visitor travels to a new page, the new images seem to appear almost instantly.

To preload images using CSS:

1. `.hideImage {...}`

Set up the class `hideImage` with only definitions to set the width and height to `0px` (**Code 4.1**). When this class is applied to any element (including an image) it will remove the element completely from the page.

Code 4.1 The `.hideImage` class simply reduces the size of the image to nothing, so the image loads but does not appear in the page.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
· Transitional//EN" "http://www.w3.org/TR/
· xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
· xml:lang="en">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I
      Preloading Images</title>
    <style type="text/css">
      .hideImage {
        width: 0px;
        height: 0px;
      }
    </style>
  </head>
  <body>
    <p>'I should
      see the garden far better...'  
</p>
  </body>
</html>
```

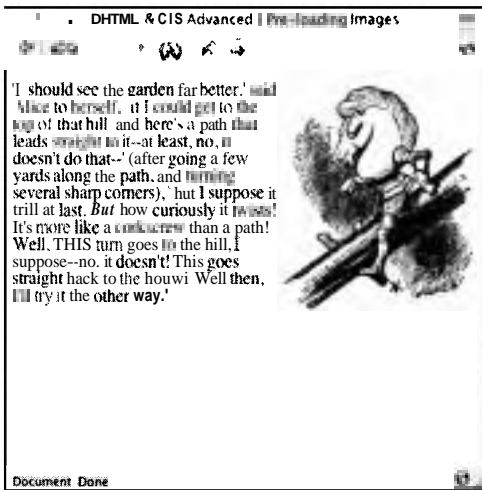


Figure 4.1 The second image does not appear because its dimensions have been set to 0 pixels, effectively removing it from the page.



Figure 4.2 How the page would appear if the `.hideImage` calls were not applied.

2. `class="hideImage"`

Add any images that should appear in the page, and then, after all of the other content on the page, add image tags for the graphics you want to preload, giving them the `hideImage` class (**Figures 4.1 and 4.2**).

✓ Tips

Although preloading images will speed the download of future pages, it will slow the display of the initial page. However, since these images will download last, the pages should appear to be loaded even though the images are still downloading.

Another way to hide images for preloading is to use `display: none`. However, the CSS standard does not require browsers to load images that have `display: none` set, so this may not work in all browsers.

One trick I will show you in Chapter 6, "Navigation," uses another trick to load multiple rollover states for graphic buttons.

Adding External Content with iframes

Adding an iframe to a Web page is a standard and effective method for embedding one HTML file into another. In addition, the content of an iframe can be changed without affecting the content of the rest of the page.

However, one drawback to using iframes to embed content is that, unless you have the width and height of the iframe set to the exact width and height of the page being embedded, you will either get a scrollbar for the iframe or excess space.

To overcome this limitation, we need to add a little JavaScript to dynamically resize the iframe so that it exactly fits the width and height of the content it contains.

To import external content with iframes:

1. `index.html`

Create a new HTML file (we'll call it `index.html`), which will contain the iframe (**Code 4.2**). You can add any other content you want to this file, but it will only appear in this page. Steps 2–4 apply to this page.

2. `function iframeResize(iframeWindow) { ... }`

Add the function `iframeResize()` to the page. This identifies the iframe element to be resized (based on the name passed to it in the function call) and then resizes the iframe based on the width and height of the content it contains.

3. `<iframe>...</iframe>`

Add your `<iframe>` tag to the page with its initial source and a name and id. Other properties can be added as desired.

Code 4.2 This HTML file (`index.html`) uses an iframe to import content and then the function.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Transitional//EN" "http://www.w3.org/TR/
  xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I
      Using Iframes</title>
    <script language="javascript"
      type="text/javascript">
      function iframeResize(iframeWindow) {
        var iframeElement = parent.document.
          getElementById(iframeWindow.name);
        if (iframeWindow.document.height) {
          iframeElement.style.height =
            iframeWindow.document.height +
            35 + 'px';
          iframeElement.style.width =
            window.document.width + 5 +
            'px';
        }
        else if (iframeWindow.document.
          documentElement.scrollHeight) {
          iframeElement.style.height =
            iframeWindow.document.
              documentElement.
                scrollHeight + 35 + 'px';
          iframeElement.style.width =
            iframeWindow.document.
              documentElement.scrollWidth
              + 5 + 'px';
        }
      }
    </script>
  </head>
  <body>
    <iframe
      id="chapterContent"
      name="chapterContent"
      src="external.html"
      frameborder="1"
      </iframe>
  </body>
</html>
```

(code continues on next page)

Code 42 continued

```
Code
marginwidth="0" marginheight="0"
scrolling="yes"
align="middle"
height="200px" width="100%">
<a href="external.html">
  External Content</a>
</iframe>
</body>
</html>
```

Code 43 The external content (external.html) being imported into index.html. Notice the onload event handler in the <body> tag used to trigger the function in code 4.2 that resizes the frame.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/
  xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xml:lang="en">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I
      External iFrame</title>
  </head>
  <body onload="parent.iframeResize(window)">
    <div style="text-align:center">
      <h1>Alice Through the Looking
        Glass</h1>
      <h3>Chapter 5</h3>
      
    </div>
  </body>
</html>
```

4. External Content

Inside the <iframe> tag, add a link to external.html for browsers that do not support iframes.

5. external.html

Create a new HTML file, and save it as external.html (Code 4.3). This file contains whatever HTML and content you want to use. Step 6 also applies to this page.

6. onload="parent.iframeResize(window)"

In the <body> tag of any pages you will be loading into the iframe, you'll need to add an onload event handler to trigger the iframeResize() function in the surrounding (parent) frame (Figure 4.3).

✔ Tip

Because the iframe acts like an independent frame, you have to make sure to target links to the right place.

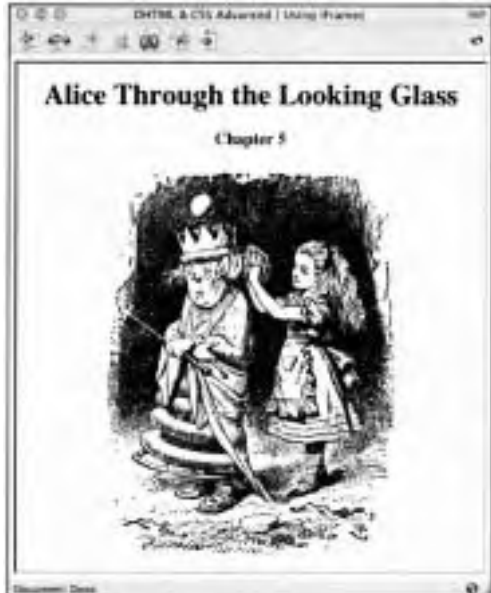


Figure 43 The content from external.html is loaded into the iframe, which automatically snaps to fit the content of the document. In this example, the iframe border is still visible, but these can be turned off, leaving the visitor no idea that the content was not an integral part of the page.

Adding External Content Using Server-Side Includes

One way to import external content directly into a Web page is to use a *server-side include*. This is not an HTML tag, but a command that tells the computer serving your Web pages to the Internet (aptly named the *server*) to add the referenced file to the page.

Since this is all done before the page is actually sent to the visitor's browser, you can break up the page into components as separate files and reassemble them. This allows you to place elements like headers and footers into files that can be reused on multiple Web pages, so changes you make to one file will affect multiple pages. The resulting page will look like a normal Web page when delivered.

Although this method's success depends on whether your server understands this command, almost all servers do these days.

To add a server-side include:

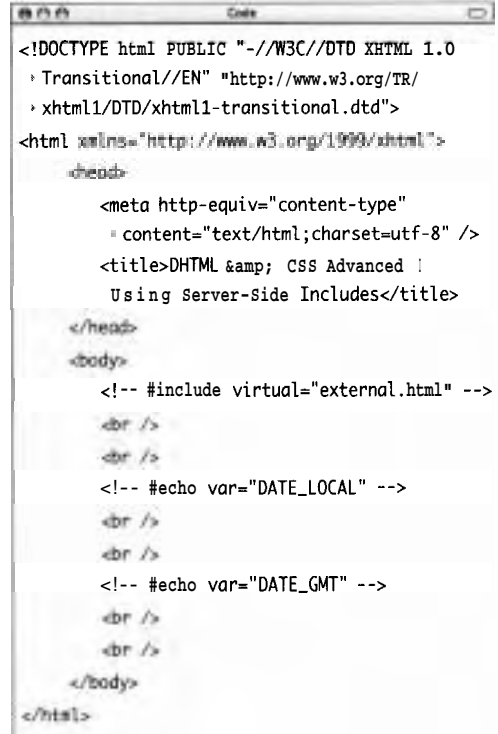
1. index.html

Create a new HTML file into which you'll add the server-side includes into and save it (**Code 4.4**). Steps 2 and 3 apply to this page.

2. `<!-- #include virtual="external.html" -->`

Add the include anywhere in the body of your HTML document, and set it to import `external.html`. The path for this file must correspond to the relative location of the file. Absolute URLs cannot be used.

Code 4.4 The server-side include commands in `index.html` allow you to import external content, but only if the page is being delivered from a Web server. If you view the source code of this page in your browser, you won't see the include tags.



```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      = content="text/html; charset=utf-8" />
    <title>DHHTML &amp; CSS Advanced |
      Using Server-Side Includes</title>
  </head>
  <body>
    <!-- #include virtual="external.html" -->
    <br />
    <br />
    <!-- #echo var="DATE_LOCAL" -->
    <br />
    <br />
    <!-- #echo var="DATE_GMT" -->
    <br />
    <br />
  </body>
</html>
```

Code 4.5 The external content from `external.html` is imported into `index.html`. This content can be any standard HTML code; just don't use `<html>`, `<head>`, or `<body>` tags.

```
<div style="text-align:center">
  <h1>Alice Through the Looking Glass</h1>
  <h3>Chapter 6</h3>
  
</div>
```

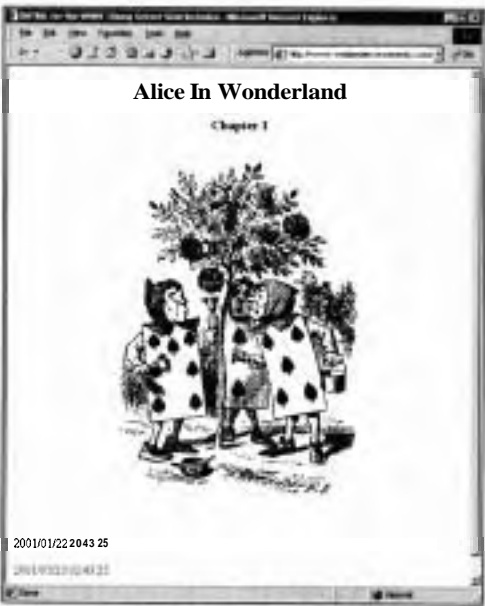


Figure 4.4 Using the server-side include, content is placed into the HTML document even before the page is sent to the browser.

3. `<!-- #echo var="DATE_LOCAL" -->`
`<!-- #echo var="DATE_GMT" -->`

Another useful server-side command is `#echo`, which tells the server to include either its local time and date or Greenwich Mean Time (GMT) on the page.

4. `external.html`

Create a new HTML file, and save it as `external.html` (Code 4.5). This file does not contain the regular open and close `<html>` tags, because those are supplied by the main document—only any HTML that could be included with the `<body>` tag in a regular HTML document. The results are shown in Figure 4.4.

✓ Tips

The disadvantage of this method is that you can't see the external content unless it's coming from a server. If you try to view this file on your local hard disk, you'll see a whole lot of nothing or, worse, the commands as text.

You can also use JavaScript to add a working clock that shows visitors their local time, as demonstrated later in this chapter.

Adding External Content with JavaScript

Although JavaScript is typically used to add interactivity to a Web page, the `<script>` tag can also import an external file. That file can contain HTML tags and content, which are in turn placed into the HTML document using the `document.write` method.

The advantage of this method is that you can use the JavaScript to tailor the content as needed, dynamically including titles, images, or anything you can think of.

To add an external JavaScript file:

1. index.html

Create an HTML file into which the external JavaScript file will be embedded and save it (**Code 4.6**). Steps 2 and 3 apply to this file.

2. cNum=5;

Add any JavaScript variables that will be used by `external.js` to customize the page.

3. `<script src="javascript/external.js"></script>`

Importing an external JavaScript file is as simple as adding the `src` attribute to the `<script>` tag. Don't place anything between the open and close `<script>` tags. This method places the external JavaScript in the HTML file at this exact location. If the JavaScript will add HTML tags to the page, those tags will be added to the page in this location.

Code 4.6 You can place the `<script>` tag anywhere in your HTML document, but to add visible content, you need to place it in the body of the document. This allows you to write HTML into the page with JavaScript.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I Using an External JavaScript File</title>
    <script language="JavaScript"><!--
      cNum=5; //Chapter Number
      cTitle='Wool and Water';
      //Chapter Title
    -->
  </script>
</head>
<body>
  <script src="javascript/external.js"></script>
  <p>She caught the shawl as she spoke...</p>
</body>
</html>
```

Escaping Characters for JavaScript

Remember, content to be written using JavaScript has to be inside single quotes (`'`). If you need to include a single quote in the content you're writing with JavaScript, it has to be preceded by a backslash (`\`).

So it won't work to use `document.write ('How's it going?');`

The browser will interpret the apostrophe between `w` and `s` as a close quote.

Instead use `document.write ('How\'s it going?');`

Code 4.7 The external JavaScript file (`external.js`) can include any JavaScript code, but can also be used to write HTML code and content into the page.

```

// External JavaScript used to create chapter
headers
var writePage='';
writePage+="Table of
Contents";
writePage+="  
<br/>";
if(cNum != 0) {
    writePage+="

## 


```

Figure 4.5 Although the JavaScript imports the content, it also relies on variables in the main HTML document to fill in the blanks (chapter number and chapter title).

4. `external.js`

Create an external JavaScript file, and save it as `external.js` (**Code 4.7**), preferably in a folder called `javascript`, since you may need to add multiple external JavaScript files. This file can contain any standard JavaScript, but to deliver content, you need to have the code write the HTML tags and other content. Steps 5–9 apply to this file.

5. `var writePage='';`

Set up a variable that will record all of the HTML code that will eventually be written to the page.

6. `writePage+="Table of Contents";`

Add each line of HTML to the variable `writePage`.

7. `if (cNum != 0) {...}`

You can use other JavaScript—such as conditionals—to dynamically add content according to variables on the page into which the JavaScript file is imported.

8. `writePage+="`

You can also combine JavaScript variables to create the code dynamically. In this example, the JavaScript will read variables in `index.html` to specify the volume number, article number, titles, and date.

9. `document.write(writePage);`

Finally, use a `document.write()` to add all of the code to the page (**Figure 4.5**).

✓ Tips

The drawback of this method is that you have to place every line of HTML code in JavaScript. This can be labor-intensive, and it makes the file harder to debug and fix in most WYSIWYG programs.

If you do place anything between the `<script>` tags, that content is ignored.

Viewing Someone Else's External JavaScript or CSS

The best way to learn about DHTML and CSS (other than reading this book) is to look at other Web sites and dissect their code. Unfortunately, Web designers sometimes hide their code by placing it in external files that don't show up when you attempt to view the source in the usual way.

Never fear—you can view the code if you're willing to dig a bit.

To view hidden code:

1. Open the Web page for which you want to find the code (**Figure 4.6**).
2. View the source code.

Different browsers have different ways of providing access to the source code, but you'll typically find a View Source or Source command in the View menu (**Figures 4.7 and 4.8**).

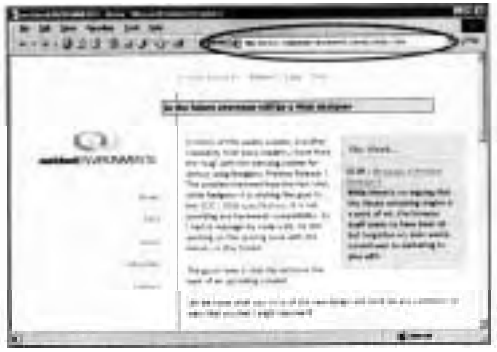


Figure 4.6 The page with the external code you want to see. The absolute path is at the top.

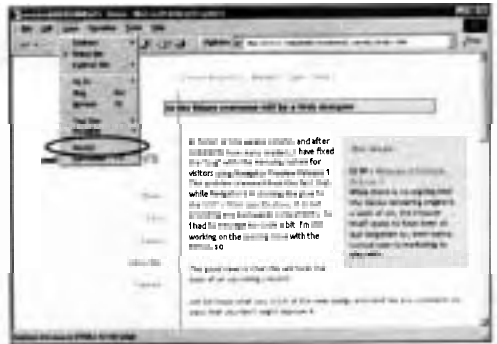


Figure 4.7 The View menu in Internet Explorer.

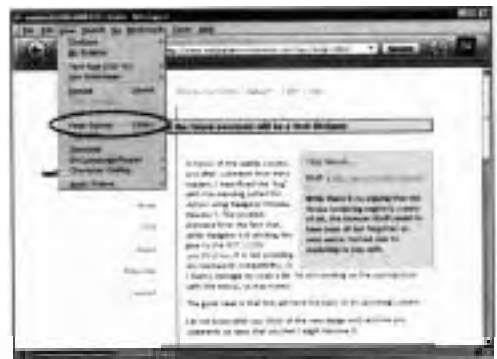


Figure 4.8 The View menu in Netscape.



Figure 4.9 The source code for the Web page. Many of the links to external files use a relative path (meaning the URL is relative to the current page's URL).

3. In the source code, find the reference to the external code you want to view (Figure 4.9).

You may have to do some hunting, but the `<script>` (for JavaScript) or `<link>` (for CSS) tag will be in the head of the document.

4. Piece together the URL to the external content: `<script>` tags have a `src` attribute for the URL and `<link>` tags use `href`.

If the URL is an absolute path (begins with `http://`), use that exact URL in step 5.

If it is a relative path, start with the current URL (the one in the browser's location/address bar), less the filename.

For every `../` in the relative URL, remove an additional level from the end of the current page's URL, and add the relative path to it. Then add the remainder of the page's URL to the relative path without `../` (Figure 4.10).

continues on next page

<http://www.webbedenvironments.com/aux/index.html>

+ [../javascript/article.js](http://www.webbedenvironments.com/aux/index.html)

<http://www.webbedenvironments.com/javascript/article.js>

Figure 4.10 A simple equation that finds the absolute URL for the external content.

5. Open the URL for the external content that you found in step 4 (Figure 4.11). Again, different browsers have different methods, but the File menu typically contains an Open Location command. A shortcut is to type the URL in the browser's address/location bar. Some browsers open the code directly in the browser window; others require you to download the code file first (Figure 4.12) and then open it in a program such as Notepad (Windows) or TextEdit (Mac) (Figure 4.13).



Figure 4.11 The external code's URL typed in the browser's location bar.



Figure 4.12 Some browsers require you to download the file rather than view it in your browser.



Figure 4.13 The external source code you seek.

Code 4.8 Adding the `.php` extension for `index.php` alerts the Web server that this page requires special processing before it is sent out to the visitor's browser.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
 Transitional//EN" "http://www.w3.org/TR/
 xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
 xml:lang="en">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced |
      Using PHP to Add External Content</
      title>
  </head>
  <body>
    <?php
      require("external.php");
    ?>
  </body>
</html>
```

Adding External Content with PHP

Like server-side includes, PHP allows you to tell the computer serving the Web page how to process it before it's sent to the visitor's browser. This allows you not only to add content, but as with JavaScript, to dynamically create the page. However, since it is server-side, your server must support PHP, which is not a given. If you aren't sure, check with your Web site host.

One advantage of PHP is that the lion's share of the dynamic work is performed on the server (which is most likely faster than the visitor's computer), but it is also scriptable like JavaScript. We'll only be dipping our toes into the sea of PHP's possibilities here (Chapter 10 goes into a bit more detail), but it's important to see PHP as an alternative to iframes, server-side includes, and JavaScript as a method for embedding external content files.

To add external content using PHP:

1. `index.php`

Create a new file and save it using the `.php` rather than `.html` extension (**Code 4.8**).

The `.php` extension alerts the server that it should look on this page for PHP code, but you can also include any HTML you want on this page. Steps 2–4 apply to this page.

2. `<?php`

To insert PHP code on the HTML page, use the `<?php` open tag. Notice, though, that unlike HTML tags, there is no closing chevron (`>`). That comes later.

continues on next page

3. `require("external.php");`

Add the `require` command with the path of the file you want to insert at this point in your HTML. You can include as many `require` statements as you desire here to mix files from multiple locations.

4. `?>`

Close your PHP tag. You can repeat steps 2–4 anywhere on the page, as many times as you want.

5. `external.php`

Create a new file, also saving it with the `.php` extension (**Code 4.9**). This page contains all of the HTML and content you want to import—but not the `<html>`, `<head>`, and `<body>` tags—and can even include additional PHP code if needed (**Figure 4.14**).

✓ Tips

Unlike the other methods described in this chapter for inserting external content, PHP will not degrade gracefully if the external file being referenced does not exist.

Another distinct advantage of PHP is that since the action takes place on the server, you don't have to worry about browser inconsistencies or incompatibilities.

Code 4.9 The external PHP file can contain any content, HTML, JavaScript, or PHP code you need.

```
Code
<div style="text-align:center">
  <h1>Through the Looking Glass</h1>
  <h3>Chapter 7</h3>
  
</div>
```

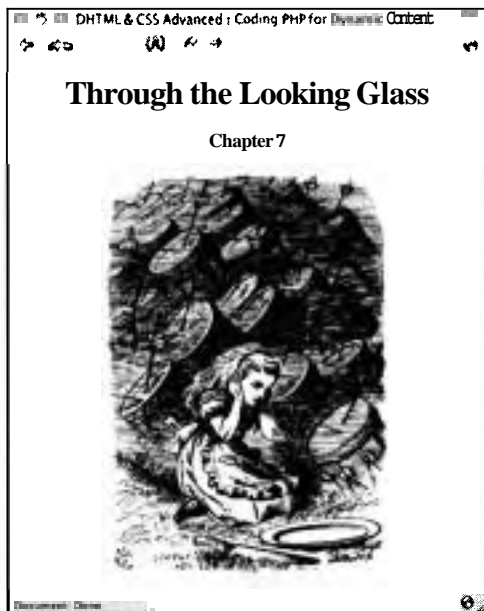


Figure 4.14 Like the server-side include, PHP actually embeds content before the HTML document is sent to the browser so that it's an integral part of the page. The only way to tell that this is not a standard HTML document is the `.php` extension on the filename.

Inserting a New Element

To create dynamic Web pages, you often need to make changes to the page in reaction to the visitor's actions. Often, you will make changes by showing and hiding content that is already on the page. However, you can't always plan for every contingency.

In this section (and the next two) I'll show you how to add new elements to the Web page. These dynamically created elements can be inserted in a variety of ways. In this example, we'll look at how to add to an image tag, inserting it at the end of the document using the `appendChild()` method.

In the next three sections, you can see examples of how to replace elements before a particular object on the page, replace a particular object on the page, and remove an object from the screen entirely (**Table 4.1**).

Table 4.1

Content Replacement Methods	
METHOD	WHERE IT WORKS
<code>appendChild(newElement)</code>	Places new element as the last child in the invoking child's parent.
<code>insertBefore(newElement, referenceObject)</code>	Places the new element immediately before the referenced object.
<code>replaceChild(newElement, oldObject)</code>	Replaces the old element with the new.
<code>removeChild(referenceObject)</code>	Removes the referenced object.

To insert a new element:

```
1. function addImageElement(srcVal,
    › widthVal,heightVal,altVal) {...}
```

Add the function `addImageElement()` to your JavaScript (**Code 4.10**). This function will be used to add an `` tag to the page, but you can change this function to add any HTML tag you desire, using the values passed to it for the various attributes. Steps 2–4 apply to this function.

Code 4.10 The function `addImageElement()` is used to add a new `` tag to the HTML document without having to reload the page.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
. xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
  <head>
    <meta http-equiv="content-type"
      › content="text/html; charset=utf-8" />
    <title>DHTML &amp; CSS Advanced ! Inserting a New Element</title>
    <script language="javascript" type="text/javascript">
      function addImageElement(srcVal,widthVal,heightVal,altVal) {
        var newImageElement = document.createElement('img');
        newImageElement.setAttribute('src',srcVal);
        newImageElement.setAttribute('width',widthVal);
        newImageElement.setAttribute('height',heightVal);
        newImageElement.setAttribute('alt',altVal);
        document.body.appendChild(newImageElement);
      }
    </script>
  </head>
  <body>
    <h2>CHAPTER VII</h2>
    <h3>The Lion and the Unicorn</h3>
    <a href="javascript:void('')" onclick="addImageElement('Alice_7_4.jpg','300','230','the Lion
    › and the Unicorn')">View Image</a>
    <p>The next moment soldiers came running through the wood...</p><br />
  </body>
</html>
```

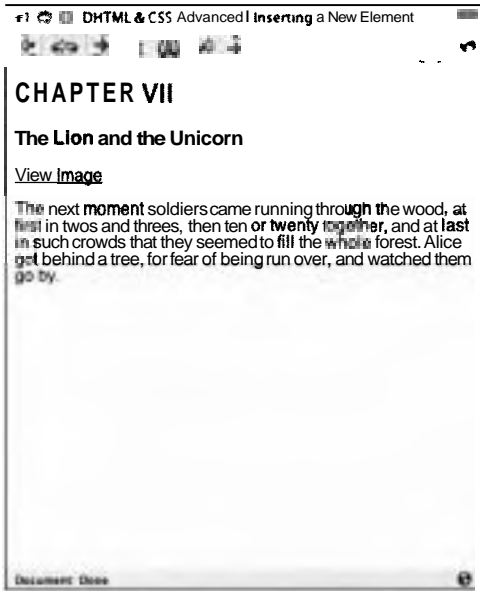



Figure 4.15 Before the link is clicked.

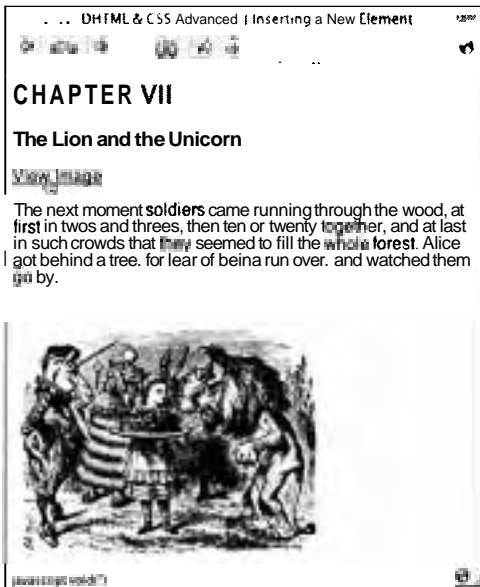


Figure 4.16 After the link is clicked the image is inserted into the page at the bottom. Clicking the link again will insert another copy of the image.

2. `var newImageElement = document.createElement('img');`
Add a new variable called `newImageElement` that will represent the code for the new image, and use `createElement()`, passing it the selector of the HTML tag you'll be creating, in this case `img`.
3. `newImageElement.setAttribute('src', srcVal);`
Use `setAttribute()` to add attributes to the tag, passing it the attribute name and then the value to be assigned to it. For the image tag there are several attributes you can add, but you must specify the source (`src`).
4. `document.body.appendChild(newImageElement)`
The tag is then actually added to the document using `appendChild()`, which places the new tag at the bottom of the document.
5. `onclick="addImageElement('Alice_7_4.jpg', '300', '230', 'the Lion and the Unicorn')`
Add an event handler to trigger the function `addImageElement()`. In this example, I'm using a simple link, but the event handler could be anywhere. When clicked, the image is added at the end of the document (**Figures 4.15 and 4.16**). Every time the link is clicked, a new instance of the image is created.

✓ Tips

You can't stick the function call for `addImageElement()` directly to the `href` for the link since it wouldn't generate an actual event as required for the function to work.

If the image file referenced by the new tag isn't in the right place, the broken image tag will appear, which is not generally considered a good thing.

Inserting a New Text Element

In the previous section, I showed you how to add a new element to a Web page after the page has loaded. However, the technique shown would only allow you to add tags without any text. In this section, I'll show you how to place text between the tags.

Also, the previous example demonstrated how to insert the new element at the end of the document. This example shows you how to add the new element before a specific object on the page by using `insertBefore()`.

To insert a new tag with text:

1. `function addTextElement`
 `(beforeObjectID, textVal) {...}`
Add the function `addTextElement()` to your JavaScript (Code 4.11). This function is passed a text value (`textVal`), which it then places before a specific object (`beforeObjectID`). Steps 2–6 apply to this function.

2. `var newTextElement =`
 `document.createElement('p');`
Define a variable to store the new element, using `createElement()` to pass the selector for the tag used to create the new element. In this example, we are setting up a paragraph (`p`), but you could use any tag that can contain text (`span`, `i`, `b`, and so on).

3. `var newText = document.`
 `createTextNode(textVal);`
Define a variable to create a new text block (node), using `createTextNode()` to pass it the string of text (`textVal`) to be used.

Code 4.11 The function `addTextElement()` not only adds a new HTML tag to the document, it also adds text to the tag.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Transitional//EN" "http://www.w3.org/TR/
  xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xml:lang="en">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I
      Inserting a New Text Element</title>
    <script language="javascript"
      type="text/javascript">
      function addTextElement
        (beforeObjectID, textVal) {
          var newTextElement =
            document.createElement('p');
          var newText = document.
            createTextNode(textVal);
          newTextElement.appendChild
            (newText);
          var beforeObject = document.
            getElementById(beforeObjectID);
          document.body.insertBefore
            (newTextElement, beforeObject);
        }
    </script>
  </head>
  <body>
    <h2>CHAPTER VIII</h2>
    <h3>The Lion and the Unicorn</h3>
    <div id="chapterText">
    <a href="javascript:void(')' onclick=
      "addTextElement('chapterText','The
      next moment soldters came running
      +through the wood, at first in twos and
      threes, then ten or twenty together,
      +and at last in such crowds that they
      seemed to fill the whole forest. Alice
      +got behind a tree, for fear of being
      +run over, and watched them go
      +by.')">Read Text</a><br />
  </div>
  </body>
</html>
```

(code continues on next page)

```


</div>
</body>
</html>

```

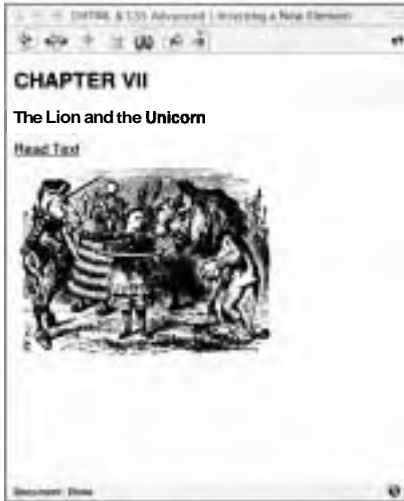


Figure 4.17 Before the link is clicked.

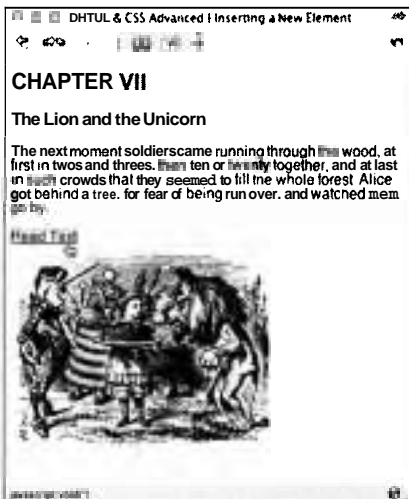


Figure 4.18 After the link is clicked the text is inserted before the link. Clicking the link again will insert another copy of the text.

4. `newTextElement.appendChild(newText);`
Use `appendChild()` to add the text node from step 3 into the element created in step 2.
5. `var beforeObject = document.
 · getElementById(beforeObjectID);`
Define a new variable to record the object before which the new text element will be inserted, using `getElementById()`.
6. `document.body.insertBefore
 · (newTextElement, beforeObject);`
Use the `insertBefore()` method to add the new text element before the specified object.
7. `onclick="addTextElement
 · ('chapterText', 'The next moment
 · soldiers...')">Read Text`
Add an event handler in your HTML to trigger the `addTextElement()` function, passing the function the ID for the object before which you want the text inserted and then (in single quotes) the text you want inserted (**Figures 4.17** and **4.18**).

✓ Tip

Although this example shows how to add text that has been hard-coded, you could also adapt it to read text that has been typed into a form field and then write that text to the page.

Inserting a New iframe Element

Although similar to inserting any other element on a page, inserting an `iframe` provides an additional challenge because Internet Explorer for Windows won't allow you to dynamically assign the `src` attribute for an `iframe`. So to load the source into the frame, we'll instead need to change the `href` value using JavaScript.

The previous two sections demonstrated how to add elements at the end of the document and before a specific object. In this example, you'll learn how to replace a specific object with the new element.

To insert a new iframe element:

1. `addIframeElement(replaceObjectID, srcVal, widthVal, heightVal, idVal, frameborderVal, scrollingVal, alignVal) {...}`

Add the function `addIframeElement()` to your JavaScript (Code 4.12). This function will add a new `<iframe>` tag to your Web page based on the attributes passed to it, and then load the indicated source. Steps 1-6 apply to this function.

2. `var newIframeElement = document.createElement('iframe');`
Define a variable to record the `iframe` element using `createElement()`.
3. `newIframeElement.setAttribute('width', widthVal);`
Use `setAttribute()` to set the various attributes associated with this `iframe`.
4. `var replaceObject = document.getElementById(replaceObjectID);`
Define a variable to locate the object to be replaced by the new `iframe` using `getElementById()`.

Code 4.12 The function `addIframeElement()` places a new `iframe` tag in the HTML document and then loads the initial HTML source file into the `iframe`.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Transitional//EN" "http://w.w3.org/TR/
  xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I
      Inserting a New Iframe Element</title>
    <script language="javascript"
      type="text/javascript">
      function addIframeElement
        (replaceObjectID, srcVal, widthVal,
         heightVal, idVal, frameborderVal,
         scrollingVal, alignVal) {
          var newIframeElement = document.
            createElement('iframe');
          newIframeElement.setAttribute
            ('width', widthVal);
          newIframeElement.setAttribute
            ('height', heightVal);
          newIframeElement.setAttribute
            ('frameborder', frameborderVal);
          newIframeElement.setAttribute
            ('scrolling', scrollingVal);
          newIframeElement.setAttribute
            ('align', alignVal);
          newIframeElement.setAttribute
            ('id', idVal);
          newIframeElement.setAttribute
            ('name', idVal);
          var replaceObject = document.
            getElementById(replaceObjectID);
          document.body.replaceChild
            (newIframeElement, replaceObject);
          top[idVal].location.href=srcVal;
        }
      </script>
    </head>
    <body>
      <h2>CHAPTER VII</h2>
```

(code continues on next page)

Code 4.12 continued

```
<div id="chapterText">
  <a href="javascript:void('')
    onclick="addIframeElement
      ('chapterText','external.html',
      '100%', '350', 'newIframe', 'yes',
      'yes', 'left')>View Page</a>
</div>
</body>
</html>
```

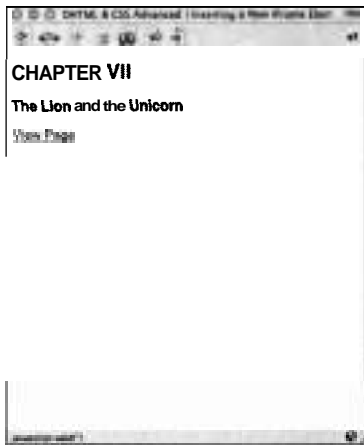


Figure 4.19 Before the link is clicked.



Figure 4.20 After the link is clicked the iframe is inserted, replacing the layer containing the link.

5. `document.body.replaceChild`
 `>(newIframeElement,replaceObject);`
Use `replaceChild()` to replace the specified object with the new iframe.
6. `top[idVal].location.href=srcVal;`
Set the href value for the newly created iframe to the desired source, based on the `srcVal` variable passed to the function.
7. `onclick="addIframeElement`
 `>('chapterText','external.html',`
 `>'100%', '350', 'newIframe', 'yes',`
 `'yes', 'left')"`

Add an event handler in your Web page to trigger the `addIframeElement()` function. In this example, I placed the event handler into a link, using the `onclick` event handler, telling it to replace an area defined by the id `chapterText`, which also happens to contain the originating link. When the visitor clicks the link the new iframe appears and the link disappears (Figures 4.19 and 4.20).

✓ Tips

Notice that I included both the `id` and `name` attributes in the `iframe`. Although `name` is generally disappearing in favor of `id`, `id` is needed to manipulate the `iframe` with DHTML, while `name` is still needed to target content. However, they can both have the same value.

You can set the `frameborder` value to `no` to get rid of the border for a more seamless design.

Removing an Element

There are several ways to remove or hide content, but the code that created the content will still remain on the page. The previous three sections showed you how to place new code on the page using JavaScript. Conversely we can remove the code from the HTML using `removeChild()`. However, since we're using JavaScript, the element in question will need to have an ID associated with it (turning it into an object) in order to give the DOM something to address.

To remove an object:

- 1. function objectBeGone(objectID) {...}**
Add the function `objectBeGone()` to your JavaScript (**Code 4.13**). This function uses the object ID passed to it to remove HTML code and text from the Web page being displayed. Steps 2 and 3 apply to this function.
- 2. var removeObject = document.
getById(objectID);**
Use `getElementById()` to define a variable storing the location of the object to be removed.
- 3. removeObject.parentNode.
removeChild(removeObject);**
Use `removeChild()` to remove the object. Notice that you actually have to address this through the parent node (in this case the body), to access the node for removal.
- 4. <div id="story1">...</div>**
Set up the object that will be removed. It can be any element on the screen, but will need to have the `id` attribute added to it with a unique value.

Code 4.13 The function `objectBeGone()` removes the indicated content from the page.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/
 xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
 xml:lang="en">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I
      *Removing an Element</title>
    <script type="text/javascript"
      language="javascript">
      function objectBeGone(objectID) {
        var removeObject = document.
          getElementById(objectID);
        removeObject.parentNode.
          removeChild(removeObject);
      }
    </script>
  </head>
  <body>
    <h2>CHAPTER VII</h2>
    <h3>The Lion and the Unicorn</h3>
    <div id="story1">
      <a href="javascript:void('')"
        onclick="objectBeGone('story1')">
        Shred the Story
      </a>
      <p>The next moment
        soldiers came running through the
        wood...</p><br />
    </div>
  </body>
</html>
```



Figure 4.21 Before the link is clicked.

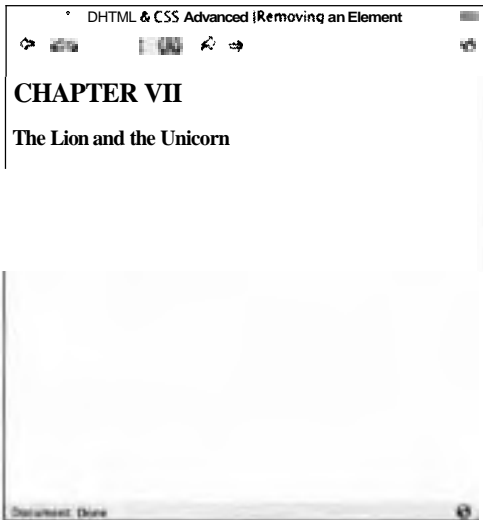


Figure 4.22 After the link is clicked, the code used to create the content is removed and the content disappears.

5. `onclick="objectBeGone('story1')"`
Add an event handler to trigger the `objectBeGone()` function. In this example, I used a simple link with an `onclick` event handler. Notice that I placed the link inside the object being removed so that the link will also disappear when the object is removed (Figures 4.21 and 4.22).

✓ Tips

Keep in mind that although this removes the code from the currently displayed version of the page, it has no effect on the version of the HTML document residing on the server. When the visitor reloads this page, all of the code is restored.

Although the result of this technique looks a lot like simply setting the `display` property to `none`, you can always set `display` back to something visible, but once you use `removeChild()` the code is actually gone from the currently displayed version of the page.

Including Random Content

Variety is the spice of life, and being able to add some random content allows you to add some variety to your Web pages. To do this, you first need to generate random numbers, and then use them to load random files or images.

In this example, we'll have five different image files (Alice-4-1.jpg, Alice-4-2.jpg, Alice-4-3.jpg, Alice_4_4.jpg, and Alice-4-5jpg). We'll generate random numbers and then use these numbers to fill in the last digit in the image's file name.

To add random content:

1. function randomRange(lowVal,highVal) = {...}

Add the function randomRange() to your JavaScript (Code 4.14). This is handled using the Math.random (generates a random decimal number) and Math.floor (rounds a decimal number to an integer) methods:

```
Math.floor(Math.random() *  
    (highVal - lowVal + 1)) + lowVal;
```

This will generate a random number between lowVal (the lowest value in the range) and highVal (the highest value in the range).

Code 4.14 The function randomRange() generates a (pseudo) random number within the range specified. This can then be used to add random content to the page using JavaScript.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0  
  Transitional//EN" "http://www.w3.org/TR/  
  xhtml1/DTD/xhtml1-transitional.dtd">  
<html xmlns="http://www.w3.org/1999/xhtml"  
  xml:lang="en">  
  <head>  
    <meta http-equiv="content-type"  
      content="text/html; charset=utf-8" />  
    <title>DHTML & CSS Advanced I  
      Including Random Content</title>  
    <script type="text/javascript"  
      language="javascript">  
      function randomRange(lowVal,highVal) {  
        return Math.floor(Math.random() *  
          (highVal - lowVal + 1)) + lowVal;  
      }  
    </script>  
  </head>  
  <body>  
    <script type="text/javascript"  
      language="javascript">  
      document.write ("<img src=  
        "Alice_4_" + randomRange(1,5) +  
        ".jpg"/>");  
    </script>  
  </body>  
</html>
```




Figure 4.23 The first time the page is loaded, a random image is selected from the list of five images.



Figure 4.24 The second time the page loads, another image is randomly selected.



Figure 4.25 The third time the page loads, another image is randomly selected.

2. `document.write ('');`
To load an image randomly from the list, add a JavaScript `document.write()` that adds the image tag, replacing the numeral value in the filename with a random number between 1 and 5. When the page loads, a random image shows up (**Figures 4.23, 4.24, 4.25, and 4.26**).

✓ Tips

- Actually JavaScript only allows you to generate pseudo-random numbers. These numbers are random, but if you know what the first number is, you can predict the subsequent numbers. This may not be good enough if you are trying to create an online casino, but will serve our purposes just fine.

This example is relatively straightforward, but can be adapted for a wide variety of purposes. For example, you could combine it with the techniques discussed earlier in this chapter to randomly include an external JavaScript file or randomly load different files into iframes.



Figure 4.26 The fourth time the page loads, one of the random images is repeated.

Including Multiple Pages in a Single Page

One bottleneck for Web surfing is simply going from HTML document to HTML document. Every time you load a new Web page, you have to wait as a signal is sent from your computer, travels over the Internet to the server, and then another signal is sent back to your machine. This can take less than a second, but it often takes longer depending on the Web page being loaded.

One way to speed things up is to actually include more than one page's worth of content in a single HTML document, but hide that content until it is needed for display. To do this, you set up different layers with content, but set their display attributes to none. Then when the visitor clicks a link, the new layer is displayed and the previous layer is hidden. To the visitor this looks as if the page loaded instantly, making for a more satisfying Web experience.

To create a single Web page with multiple pages of content:

1. `var objPage = null;`

Define a global variable called `objPage` to record the current page layer being displayed (**Code 4.15**). This variable should initially be set to `null` since no page is displayed.

Code 4.15 The function `showPage()` is used to hide the previous page and then show the selected page.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
- Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  > xml:lang="en">
  <head>
    <meta http-equiv="content-type"
      > content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I
      > Including Multiple Pages in a Single
      > Page</title>
    <script language="javascript"
      > type="text/javascript">
      var objPage = null;
      function showPage(pageName) {
        if (objPage) objPage.style.
          > display = 'none';
        objPage=document.getElementById
          > (pageName);
        objPage.style.display = 'block';
      }
    </script>
    <style type="text/css">
      .page { display:none;}
    </style>
  </head>
  <body>
    <h1>THROUGH THE LOOKING-GLASS</h1>
    <h2>by LEWIS CARROLL</h2>
    <a href="javascript:showPage('page1')">
      > target="_self">Chapter 1</a> |
    <a href="javascript:showPage('page2')">
      > target="_self">Chapter 2</a> |
    <a href="javascript:showPage('page3')">
      > target="_self">Chapter 3</a>
    <br /><br />
    <div id="page1" class="page">
      <h2>CHAPTER I</h2>
      <h3>Looking-Glass House</h3>
```

(code continues on nextpage)

```

<p>One thing
was certain, that the WHITE kitten
had had nothing to do with it...</p>
</div>
<div id="page2" class="page">
  <h2>CHAPTER II</h2>
  <h3>The Garden of Live Flowers</h3>
  <p>'I should see the
garden far better,' said Alice to
herself...</p>
</div>
<div id="page3" class="page">
  <h2>CHAPTER III</h2>
  <h3>Looking-Glass Insects</h3>
  <p>Of course the first
thing to do was to make a grand
survey of the country...</p>
</div>
</body>
</html>

```

Tip

Of course, the more content you place in a single HTML document the longer it will take to download. For text, this is usually not a huge problem, but for graphics it may mean a significant delay. However, some browsers won't load the images until the layer they are on is made visible.

2. function showPage(pageName) {...}

Add the function `showPage()` to your JavaScript. This function hides the currently showing page (if there is one) and then shows the page indicated by the `pageName` variable passed to it. Steps 3–6 apply to this function.

3. if (objPage) objPage.style.display = 'none';

Check to see if there is currently a page being displayed and, if there is, hide it by setting its `display` state to `none`.

4. objPage = document.getElementById (pageName);

Add the variable `objPage` to record the ID for the page object you'll be showing.

5. objPage.style.display = 'block'

Change the `display` property for the page layer object to `block` so that it becomes visible.

6. .page { display:none;}

Create a class called `page` with the `display` state set to `none` that will be assigned to all of the page layers so that they're initially not displayed on the page.

7. Chapter 1

In your HTML, add a link to trigger the `showPage()` function, passing it the name of the page layer to be displayed.

8. <div id="page1" class="page">... </div>

Set up a layer for each page of content. When the link from step 7 is clicked, the indicated page will be shown (Figures 4.27, 4.28, 4.29, and 4.30 on next page).

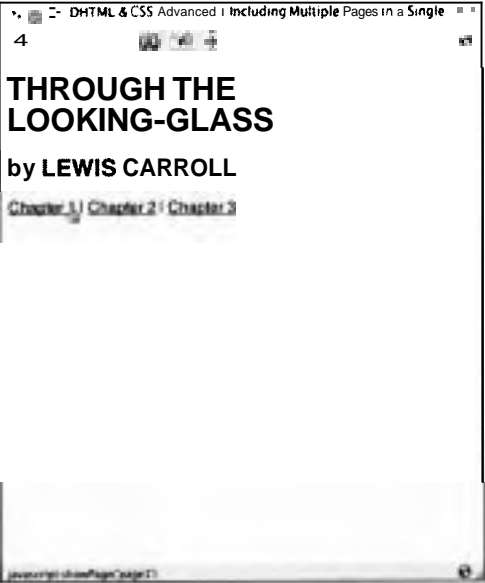


Figure 4.27 Before a chapter link is clicked.

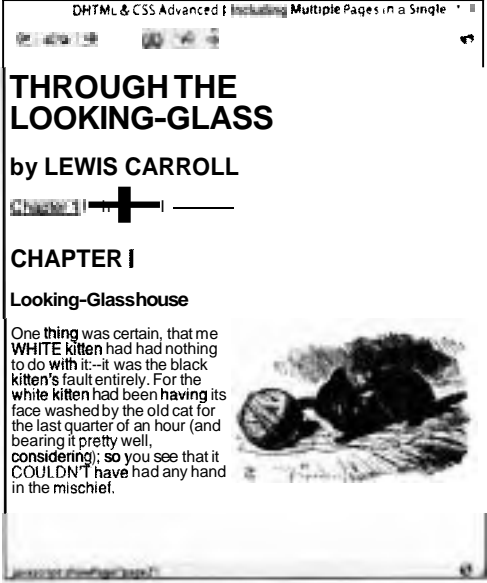


Figure 4.28 After the first link is clicked, the first chapter shows.

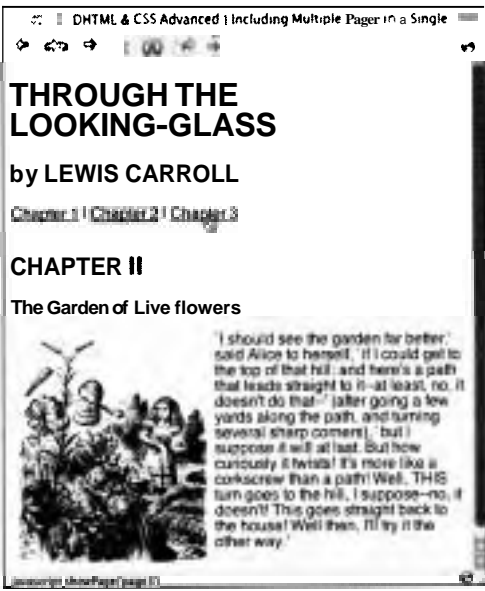


Figure 4.29 After the second link is clicked, the first chapter disappears and the second chapter appears.

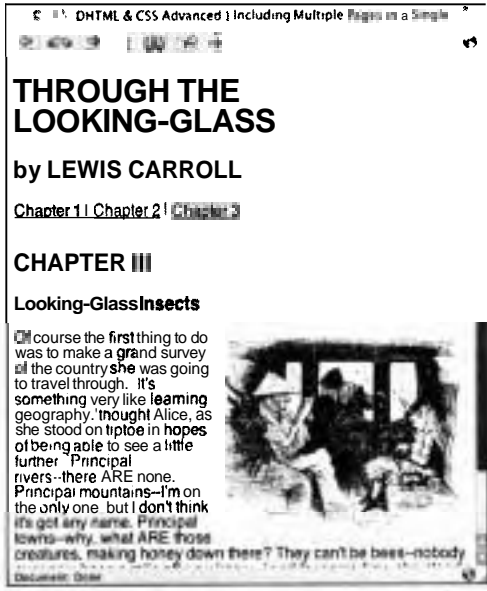


Figure 4.30 After the third link is clicked, the second page disappears and the third page appears. Keep in mind that visitors can jump between any of the pages; they don't have to go linearly.

Code 4.16 The code writes itself into a layer recursively running the function `countThis()`, updating the time every second.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
 Transitional//EN" "http://www.w3.org/TR/
 xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced :
      Adding a Clock</title>
    <script>
function countThis(){
  var digitalClock = new Date();
  var hours = digitalClock.getHours();
  var minutes = digitalClock.getMinutes();
  var seconds = digitalClock.getSeconds();
  var dn = "am"
  if (hours>12){
    dn="pm";
    hours=hours-12;
  }
  if (hours==0) hours=12;
  if (minutes<=9) minutes="0"+minutes;
  if (seconds<=9) seconds="0"+seconds;
  var clockObject = document.
    getElementById('clockFace');
  var countDown = '<span class=
    = "clockStyle">'+hours+':'+minutes+ ' :
    = '+seconds+ ' '+dn+'</span>';
  clockObject.innerHTML=countDown;
  setTimeout("countThis()",1000);
}
</script>
```

(code continues on next page)

Including a Clock

As Groucho Marx said, "Time flies like an arrow, and fruit flies like a banana." If you're designing a site where users may enter time-sensitive information, or if you just want to remind visitors of how much time they're spending on the Web, you may want to include a clock on your Web page.

This clock displays the local time reported by the visitor's computer.

To add a clock to your Web page:

1. function countThis(){...}

Add the function `countThis()` to the JavaScript (**Code 4.16**). This function checks the hours, minutes, and seconds reported by the computer's clock, translates this data from military format to a.m./p.m. format, and writes that information in the counter CSS layer. The function runs recursively until the page is unloaded. Steps 2–10 apply to this function.

2. var digitalClock=new Date();

Define a variable called `digitalClock` that records the current `Date` object.

3. var hours=digitalClock.getHours();

Use `digitalClock` to define variables to record the current hour, minute, and second values from the `Date` object.

4. var dn="am"

Define a variable called `dn` to record whether the time is a.m. or p.m. Test to see whether it is currently p.m. (hours is greater than 12), then translate the hours value from military time.

5. if (hours==0) hours=12;

If it is midnight, translate the time from 0 to 12.

continues on next page

6. `if (minutes<=9) minutes="0"+minutes;`
 If the minutes or seconds value is 9 or less, add a leading 0.

7. `var clockObject = document.
 .getElementById('clockFace');`
 Define a variable called `clockObject` to record the object for the layer where the clock is displayed on the page.

8. `var countdown='<span class=
 "clockStyle">'+hours+'':
 '+minutes+'': '+seconds+'
 '+dn+'';`

Define a variable called `countDown` to record the HTML used to display the current time.

9. `clockObject.innerHTML=countDown;`
 Use `innerHTML` to insert the new time on the page, replacing what was currently in that area. The upshot of this is that it will look as if the clock is ticking away every second.

10. `setTimeout("countThis()",1000);`
 Use `setTimeout()` to run the function `countThis()` again after a 1-second delay.

11. `#clockFace {..}`

Create an ID called `clockFace`, which defines the clock's layer. Give this layer a position, width, and height, depending on the font size you define in `clockStyle` (step 12).

```

Code
<style type="text/css"
media="screen">!--
#clockFace {
    position:absolute;
    z-index:201;
    top:10px;
    left:10px;
    width:200px;
    height:20px;
}
.clockStyle {
    font:bold 25px helvetica, arial;
}
--</style>
</head>
<body onload="countThis()">
    <div id="clockFace">
        <!-- Dynamically Filled -->
    </div>
</body>
</html>

```

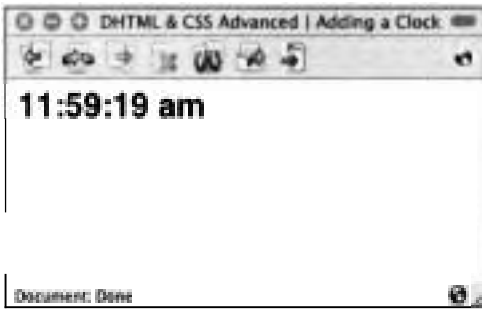


Figure 4.31 The digital clock, displaying the visitor's local time.

12. .clockStyle {...}

Create a class called `clockStyle`, which defines the appearance of the clock. The clock can be small or large, depending on the font size.

13. onload="countThis()"

Add the `onload` event handler to the `<body>` tag, and have it trigger the `countThis()` function. This starts the clock.

14. <div id="clockFace"></div>

Create the clock layer: a `<div>` tag with `id` set to `clockFace`. Initially, this area is empty when the page first loads, but is quickly filled with the ticking clock (**Figure 4.31**).

✓ Tips

This function does not work in Netscape 6.

This time is very different from the server time and GMT code I showed you in "Adding External Content Using Server-Side Includes" earlier in this chapter. This clock shows (and updates every second) the local time as set on the visitor's computer. Thus, if a visitor has their time set incorrectly, this clock will display the wrong time.



LAYOUT

There are as many ways to use CSS and DHTML as there are Web designers to use them. Yet these design tools are still relatively new, and designers are still discovering their capabilities and limitations. In addition, some designers who were initially captivated by the "gee-whiz" abilities of CSS to create dynamic HTML neglected its many layout strengths.

In the rush to experiment with the dynamic aspects of CSS, many designers overlooked some of the nuts-and-bolts problems that CSS solves: It facilitates solid, compelling page layout on the Web.

This chapter explores some of the valuable solutions CSS offers for everyday design issues and the best ways to integrate DHTML into the layout.

Creating Simple Columns

Although there are a variety of ways to lay out a Web page, one of the most common is to use columns of content immediately next to each other with a small space in between called a *gutter*. Using columns for layout has several advantages over simply stacking content blocks on top of each other:

- ◆ Columns are indispensable for creating a regular layout grid as described in Chapter 1.
- ◆ Columns allow you to maximize the use of horizontal space to present content, allowing viewers to scroll down for additional material.
- ◆ Shorter columns can make text easier to read or scan by preventing it from stretching across the screen.
- ◆ Columns allow related content to be presented side by side.

For CSS, creating columns simply requires you to float two or more layers next to each other.

To create simple columns:

1. #layoutLogic {...}

Add the ID `layoutLogic` to your CSS (Code 5.1). For now this is simply used to allow the content to fill the width of the area it is within. When you learn how to create contextual layouts later in this chapter, you will use this to control the exact layout.

Code 5.1 Columns are created in CSS by floating two layers next to each other.

```

Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Transitional//EN" "http://www.w3.org/TR/
  xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xml:lang="en">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I
      Creating Simple Columns</title>
    <style type="text/css"><!--
#layoutLogic {
  width: 100%; }
.section {
  display: block;
  margin: 0 auto;
  width: 465px; }
.col-A, .col-B {
  float: left; }
.col-A {
  text-align: right;
  display: block;
  margin-right: -1px;
  padding-right: 5px;
  width: 150px;
  border-right: 1px solid black; }
.col-B {
  display: block;
  padding-left: 5px;
  width: 300px;
  border-left: 1px solid black; }
.clear {
  clear: both; }
--></style>
</head>
<body>
  <div id="layoutLogic">
    <div class="section">
      <div class="col-A">
        <h1>CHAPTER I</h1>

```

(code continues on next page)

```

<h2>Looking-Glass House</h2>
</div>
<div class="col-B">
  <p>She said afterwards that
    she had never seen in all
    her life such a face as the
    King made...</p>
</div>
</div>
<br class="clear" />
<div class="section">
  <div class="col-A">
    <h1>CHAPTER II</h1>
    <h2>The Garden of Live
      Flowers</h2>
  </div>
  <div class="col-B">
    <p>'I should see the garden
      far better...</p>
  </div>
</div>
</div>
</div>
</body>
</html>

```

2. `.section {...}`

Add the class `section` to your CSS. This class is used to control the exact width that the two columns will occupy, as well as to center the content within the layout logic.

3. `.col-A, .col-B {...}`

Add the classes `col-A` and `col-B` to your CSS. To begin with, define the common definitions between the two classes. For this example, that would be `float: left`. Then add the unique definitions for each column, including their widths and any borders. If you want the border between two columns to extend to the height of the tallest column, set the right margin for `col-A` to `-1px` so that the borders will overlap.

4. `.clear {...}`

Add the class `clear` to your CSS. We'll use this class with `
` tags between different sections in the content to keep them from floating next to each other.

5. `<div id="layoutLogic">...</div>`

In the body of your document, set up layers using the classes and styles you added above using this basic structure:

```

layoutLogic
  section
  col-A
  col-B

```

continues on next page

6. `<br class="clear"/>`

After the section `<div>`, but still in the `layoutLogic`, add a `
` tag with the `clear` class.

You can then repeat steps 5 and 6 for as many different sections as you need (**Figure 5.1**).

✓ Tips

In this example, I showed you how to set up two columns, but you can float as many columns as you want (or can fit) next to each other. However, if you want the vertical rule to stretch the entire height of more than two columns, you will need to use the techniques for balancing columns described later in this chapter.

One common problem you might run into is the right column slipping underneath the left column(s) if the browser window is not wide enough. Generally, this is caused by leaving out the section class, which serves as an envelope holding the pieces together. Without it, the columns will float next to each other only if there is enough room.

If the gutters between columns are too wide, then the columns may look unrelated.

If the gutters between columns are too narrow, then text may become unreadable.



Figure 5.1 A simple two-column layout example, placing the chapter name to the left of the chapter text with a thin rule between.

Creating Balanced Columns

Take a look at any newspaper, and you will notice that the news is presented in columns, usually three or more on a page. Web designers often follow this basic layout style, placing content into distinct columns to help organize it. Originally, this layout was easily achieved using HTML tables, and then CSS came along, and (for a long time) absolutely nothing changed.

This was not because you couldn't easily create columns using CSS. CSS columns are created by floating layers next to each other, as shown in the previous section.

However, unlike tables, CSS columns will not balance visually. Using tables, if you place three columns next to each other, regardless of the amount of content in any particular one, you can place a background color or vertical rule to visually stretch each column to the height of the tallest one.

With CSS columns, though, visual balance proved more challenging. The problem is that since the CSS columns are not directly connected, columns with less content will stop abruptly, leaving empty space next to columns with more content. Although this is not a fatal flaw, it is generally less attractive than balanced columns.

To overcome this limitation, we need to place our columns inside an enclosing layer and then simply set the left and right borders of the enclosing layer to be the widths of the left and right columns. These borders will then stretch the full height of the layout, placing a background color behind the columns, and creating the illusion that the columns are balanced.

To create balanced columns:

1. #layoutLogic {...}

Add the ID `layoutLogic` to your CSS (Code 5.2). This ID envelopes the entire structure of the layout and can be used to add a defining border (as in this case) or not.

You can also set a minimum width to prevent the center column from collapsing completely, and forcing a horizontal scrollbar in the browser window to accommodate the content. Keep in mind, however, that this will not work in Internet Explorer for Windows.

2. #header, #footer {...}

Add the IDs `header` and `footer` to your CSS. These will be used to create areas at the top and bottom of the page that span all three columns. Although I've set them up together in this example with the same definitions, you can separate them if they need different styles.

3. .page {...}

Add the page class your CSS. This will be used to envelope the three columns and is also where we add the solid background color behind the left and right columns (col-A and col-C). To do that, set the `border-left` and `border-right` attributes to the same widths you will be using for col-A and col-C. The color you set for the borders will then show up as the background color for these columns, stretching the height of the design. To set the background color for col-B, simply set the `background-color` attribute in this class.

Code 5.2 In order to create balanced columns using CSS, the trick is to use borders in an enveloping layer to add background colors to the left and right columns.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Frameset//EN" "http://www.w3.org/TR/xhtml1/
  DTD/xhtml1-frameset.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xml:lang="en">
  <head>
    <meta http-equiv="content-type"
      = content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I
      'Creating Balanced Columns</title>
    <style type="text/css"><!--
  body {
    color: #000;
    background-color: #fff; }
  p, h1, h2 {
    margin: 0;
    padding: 0.5em; }
  #layoutLogic {
    background-color: #000;
    margin: 0 -1px;
    width: 100%;
    min-width: 740px;
    border: solid 1px black; }
  #header, #footer {
    color: white;
    position: relative;
    z-index: 13;
    width: 100%; }
  .page {
    background-color: #fff;
    width: auto;
    border-right: 150px solid #333;
    border-left: 200px solid #999; }
  .section {
    margin: 0;
    width: 100%; }
  .col-A, .col-B, .col-C {
    position: relative;
    float: left; }
```

(code continues on next page)

```

.col-A {
    margin-right: 1px;
    margin-left: -200px;
    z-index: 10;
    width: 200px; }
.col-B {
    margin: 0 -3px 0 -2px;
    z-index: 20;
    width: 100%; }
.col-C {
    color: white;
    margin-right: -150px;
    margin-left: 1px;
    z-index: 10;
    width: 150px; }
.clear {
    clear: both; }
.page > .section {
    border-bottom: 1px solid transparent; }
</head>
<body>
<div id="LayoutLogic">
  <div id="header">
    <h1>Header</h1>
  </div>
  <div class="page">
    <div class="section">
      <div class="col-A">
        <h2>Column A</h2>
        </div>
      <div class="col-B">
        <h2>Column B</h2>
        <p>There was a book lying
          near Alice on the
          table...</p>
        <p>It was like this.</p>
        <p>YK COWREBB AJ</p>
      </div>
    </div>
  </div>
</body>

```

(code continues on next page)

4. .section {...}

Add the **section** class to your CSS. In this example we will only be including a single section.

5. .col-A, .col-B, .col-C {...}

Add **col-A**, **col-B**, and **col-C** to your CSS and define the common attributes. You will need all three to float left with relative positioning so that you can set the **z-index** for the left and right columns later.

6. .col-A {...}

Customize the definitions for the left column, setting its width (which should match the width set for the left border in the **page** class), and then using that value as a negative left margin. Also set a 1px right margin and a **z-index** of 10 so that this column will not overlap the center column.

7. .col-B {...}

Customize the definitions for the center column, setting its width to **100%** for a fluid layout or a specific width for a fixed layout.

8. .col-C {...}

Customize the definitions for the right column, setting its width (which should match the width set for the right border in the **page** class), and then use that value as a negative left margin. Also set a 1px right margin and a **z-index** of 11 so that this column will not overlap the center column.

9. .clear {...}

Add the class **clear** to your CSS, which is used to separate horizontal areas on the page so that they do not float next to each other.

continues on next page

10. `.page > .section {...}`

In order for this trick to work in Mozilla browsers, you will also need to add an invisible border to the bottom of the section layer. To keep this from interfering with Internet Explorer for Windows, though, we will use the previous-parent selector (discussed in Chapter 2), which Internet Explorer for Windows does not support and so will ignore.

11. `<div id="layoutLogic">...</div>`

Finally, set up your layers using the IDs and classes created above, with this basic structure:

```
layoutLogic
  header
  page
  section
    col-a
    col-b
    col-c
  footer
```

This will create a three-column background with solid-colored backgrounds in the left and right columns and a header and footer across the top and bottom (**Figure 5.2**).

✓ Tip

For `col-C`, I mentioned that you could set a specific width for the column to restrict the overall width of your layout. Another way to do this is to simply set the width of the `layoutLogic` ID. This allows you to control the overall width of the design more quickly.

Code 5.2 *continued*

```
<div id="layoutLogic">
  <div id="header">
    <h1>Header</h1>
  </div>
  <div id="page">
    <div id="section">
      <div id="col-a">
        <p>This is column A</p>
      </div>
      <div id="col-C">
        <h2>Column C</h2>
        <p>This was the poem that
          <br>= Alice read, u^p^=
        <p>JABBERWOCKY</p>
        <p>'Twas brillig, and the
          <br>= slithy toves...</p>
        <p>Beware the
          <br>= Jabberwock...</p>
      </div>
      <div id="col-b">
        <p>This is column B</p>
      </div>
    </div>
  </div>
  <div id="clear"></div>
</div>
<div id="footer">
  <h1>Footer</h1>
</div>
</body>
</html>
```



Figure 5.2 A three-column balanced layout where the middle column stretches so that the design fills the width of the browser window.



Figure 5.3 The background image design.



Figure 5.4 The three slices that will be used on the Web page.



Figure 5.5 The `bg_top` slice is used as the top cap to the columns.



Figure 5.6 The `bg_center` slice is tiled vertically behind the content.



Figure 5.7 The `bg_bottom` slice forms the bottom of the columns.

Creating Graphic Background Columns

You now know how to create balanced columns using solid-color borders as backgrounds. Another way to create balanced columns is by using a background graphic. This has a lot of practical applications, as it allows you to create simple line dividers between columns or more complex graphic effects, while also visually balancing the columns.

To add a graphic background, we will start by creating a single image, then slicing it into three pieces to use behind the content and create the illusion of columns. I'll use Adobe Photoshop, but you *can* use any program that *can* create images and save them in GIF, JPEG, or PNG format

To create columns with graphic backgrounds:

1. `background.psd`

Create your background graphic using a program such as Adobe Photoshop, Adobe ImageReady, or Macromedia Fireworks (**Figure 5.3**). The exact design is up to you; for this example, I've created a 740-pixel-wide, three-column graphic with a gradient background in the left and center columns and an aluminum texture in the third. Save this file as `background.psd`. Steps 2–3 apply to this file.

2. Slice your image lengthwise into three different full-width pieces of the smallest height that still preserves any textures used in the graphic (**Figure 5.4**). Generally a height of 10–20 pixels should be fine.

The first piece, `bg_top`, is going to serve as the top cap to the background (**Figure 5.5**).

The second piece, `bg_center`, is the middle, which will tile vertically down the design (**Figure 5.6**).

The third, `bg_bottom`, will serve to cap the bottom of the design (**Figure 5.7**).

continues on next page

3. Save each piece as a separate image file. In this example, I'm saving each as a JPEG in a folder called images.

4. #layoutLogic {...}

In your HTML document, add the `layoutLogic` ID to your CSS (**Code 5.3**). For this example, we'll create a layout with a specific width of 740 pixels, which is generally considered best to accommodate Web surfers with a display resolution of 800×600.

5. #background-top, #background-bottom {...}

Add the `background-top` and `background-bottom` classes to your CSS. These will be used to create layers that sit above and below the columns, creating graphic caps that provide the illusion of rounded corners.

6. #background-top {...}

Customize the `background-top` and `background-bottom` classes to add the background images created in steps 1–3 and saved in the images folder.

7. .page {...}

Add the `page` class your CSS. This will be used to envelope the three columns and is also where the main background image is added, set to repeat vertically only, so it will show for the full height of the content, regardless of how much content there is.

Code 5.3 To add a graphic treatment to your designs, add a top and bottom graphic cap and then tile a graphic vertically in the center. The columns will then float over top of this.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Transitional//EN" "http://www.w3.org/TR/
  xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xml:lang="en">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I
      +Creating Balanced Columns</title>
    <style type="text/css"><!--
body {
  color: #000;
  background-color: #fff; }
p, h1, h2 {
  margin: 0;
  padding: 0.5em; }
#layoutLogic {
  background-color: #fff;
  margin: 0 -1px;
  width: 740px; }
#background-top, #background-bottom {
  color: white;
  background-repeat: no-repeat;
  position: relative;
  z-index: 13;
  width: 100%;
  height: 20px; }
#background-top {
  background-image: url(images/bg_top.jpg); }
#background-bottom {
  background-image: url(images/bg_bottom.jpg); }
.page {
  background-color: #fff;
  background-image: url(images/bg_center.jpg);
  background-repeat: repeat-y;
  width: auto; }
.section {
  margin: 0;
```

(code continues on next page)

```

width: 100%; }
.col-A, .col-B, .col-C {
margin: 0;
padding: 0;
position: relative;
float: left; }
.col-A {
color: #fff;
margin-left: 10px;
z-index: 10;
width: 140px; }
.col-B {
margin-right: 35px;
margin-left: 15px;
z-index: 20;
width: 350px; }
.col-C {
z-index: 10;
width: 180px; }
.clear {
clear: both; }
.page > .section {
border-bottom: 1px solid transparent; }
--></style>
</head>
<body>
<div id="layoutLogic">
<div id="background_top">
&nbsp;</div>
<div class="page">
<div class="section">
<div class="col-A">
<h2>Column A</h2>
</div>
<div class="col-B">
<h2>Column B</h2>
<p>If all the strange
things that Alice saw in
her journey Through The
Looking-Glass...</p>
</div>

```

(code continues on next page)

8. `.section {...}`

Add the section class to your CSS. In this example, we will only include a single section.

9. `.col-A, .col-B, .col-C {...}`

Add `col-A`, `col-B`, and `col-C` to your CSS, and define the common attributes. You will then need to customize the definitions for each column to the specific widths you want to use for your columns, including a margin to allow for any graphic dividers.

10. `.clear {...}`

Add the class `clear` to your CSS; this is used to separate horizontal areas on the page so that they do not float next to each other.

11. `.page > .section {...}`

In order for this trick to work in Mozilla browsers, you also need to add an invisible border to the bottom of the section layer. To keep this from interfering with Internet Explorer for Windows, though, use the previous parent selector discussed in Chapter 2, which Internet Explorer for Windows does not support and so will ignore.

continues on next page

12. `<div id="layoutLogic">...</div>`

Finally, set up your layers using the IDs and classes created above, with this basic structure:

```

layoutLogic
  background-top
  page
  section
  col-a
  col-b
  col-c
  background-bottom

```

This will create three columns with the main background graphic repeating behind them and the cap graphics at the top and bottom (**Figure 5.8**).

✓ **Tips**

If you simply want to add a line between each column, you can still use this technique, creating a background image with lines horizontally spaced at intervals that will then stretch vertically to form the lines.

The obvious disadvantage of this technique is that in order to change the background, you have to make changes to the image.

For this example, I used the JPEG image format because I was using some complex visual effects that would not reproduce as well using GIF.

Code 5.3 continued

```

Code


A screenshot of a web browser displaying a three-column layout. The left column is dark with white text. The middle column is light with dark text. The right column is light with dark text. The background features a repeating graphic of a landscape with a large, stylized figure. The browser's address bar shows a URL starting with 'http://www.kitfox.com/'.



Figure 5.8 The three-column layout with a graphic background.


```

Code 5.4 layout.css. The layoutLogic ID comes in three flavors and the page will be laid out differently depending on which one you use.

```
body {
  color: #000;
  background-color: #fff; }
p, h1, h2 {
  margin: 0;
  padding: 0.5em; }
#layoutLogic-ABC, #layoutLogic-AB,
#layoutLogic-A {
  background-color: #000;
  margin: 0 -1px;
  width: 100%;
  border: solid 1px black; }
#header, #footer {
  color: white;
  position: relative;
  z-index: 13;
  width: 100%; }
#layoutLogic-ABC .page {
  background-color: #fff;
  width: auto;
  border-right: 150px solid #333;
  border-left: 200px solid #999; }
#layoutLogic-AB .page {
  background-color: #fff;
  width: auto;
  border-left: 200px solid #999; }
#layoutLogic-A .page {
  background-color: #fff;
  width: auto; }
.section {
  margin: 0;
  width: 100%; }
.col-A, .col-B, .col-C {
  position: relative;
  float: left; }
.col-A {
  margin-right: 1px; }
.col-B {
  margin: 0 -3px 0 -2px; }
```

(code continues on next *page*)

Creating Contextual Layouts

In Chapter 2, I discussed using contextual selectors to change how content is displayed according to where it is placed in the page. By using the contextual selector method, you can define multiple definitions for the same rule. Then, depending on the parent element you place the selector in, the appearance can be radically different.

In this example, I want to take that technique one step further and show you how to control the layout of the entire page, using the same HTML code to create pages with one, two, or three columns controlled using a single ID.

To create contextual layouts:

1. `columnLayout.css`
Create an external style sheet and save it as `columnLayout.css` (**Code 5.4**). Steps 2–12 apply to this file.
2. `#layoutLogic-ABC, #layoutLogic-AB, #layoutLogic-A {...}`
Set up three different `layoutLogic` IDs for the three layout styles: three columns (ABC), two columns (AB), and one column (A).
3. `#layoutLogic-ABC .page {...}`
Add a contextual definition for the page class to use if it is a child element of `layoutLogic-ABC`. This version includes borders that add background colors in the left and right columns.

continues on next page

4. #layoutLogic-AB .page {...}

Add a contextual definition for the page class to use if it is a child element of layoutLogic-AB. This version includes a border that adds a background color in the left column.

5. #layoutLogic-A .page {...}

Add a contextual definition for the page class to use if it is a child element of layoutLogic-A. This version does not include any borders since there is only one column.

6. .section {...}

Add the section class to your CSS. In this example, we will only include a single section.

7. .col-A, .col-B, .col-C {...}

Add col-A, col-B, and col-C to your CSS and define the common attributes.

8. #layoutLogic-ABC .col-A, #layoutLogic-AB .col-A {...}

Add contextual definitions for how the left column class should be treated when in the three- and two-column layouts (ABC and AB). In this example, the column will be 200 pixels wide.

9. #layoutLogic-ABC .col-B, #layoutLogic-AB .col-B {...}

Add contextual definitions for how the center column class should be treated when in the three- and two-column layouts (it will not appear in the one-column layout). In this example, the column will be 100% of the available width.

Code 5.4 continued

```
Code
.col-C {
    margin-left: 1px; }
#layoutLogic-ABC .col-A, #layoutLogic-AB .col-A
{
    margin-left: -200px;
    z-index: 10;
    width: 200px; }
#layoutLogic-ABC .col-B, #layoutLogic-AB .col-B
{
    z-index: 20;
    width: 100%; }
#layoutLogic-ABC .col-C {
    color: #fff;
    margin-right: -150px;
    z-index: 10;
    width: 150px; }
#layoutLogic-A .col-B, #layoutLogic-AB .col-C,
#layoutLogic-A .col-C {
    display: none; }
.clear {
    clear: both; }
.page > .section {
    border-bottom: 1px solid transparent; }
```

Code 5.5 index.html. You can control the number of columns being displayed on a particular page using a single line of code. By changing the layoutLogic ID, you can have the same HTML code display as one (A), two- (AB), or three-column (ABC) layouts.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Frameset//EN" "http://www.w3.org/TR/xhtml1/
  DTD/xhtml1-frameset.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xml:lang="en">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML &amp; CSS Advanced I
      & Creating Contextual Layouts</title>
```

(code continues on next page)

```

<link href="columnLayout.css"
      rel="stylesheet" type="text/css"
      media="all"/>
</head>
<body>
  <div id="layoutLogic-ABC">
    <div id="header">
      <h1>Header</h1>
    </div>
    <div class="page">
      <div class="section">
        <div class="col-A">
          <h2>Column A</h2>
          <p>'Nearly there!' the
            Queen repeated...</p>
        </div>
        <div class="col-B">
          <h2>Column B</h2>
          <p>'Now! Now!' cried the
            Queen. 'Faster! Faster!</p>
          <p>The Queen propped her
            up against a tree...</p>
          <p>Alice looked round her
            in great surprise...</p>
          <p/>
        </div>
        <div class="col-C">
          <h2>Column C</h2>
          <p>'Of course it is,' said
            the Queen, 'what would
            you have it?'</p>
        </div>
        <div class="clear"></div>
      </div>
    </div>
    <div id="footer">
      <h1>Footer</h1>
    </div>
  </div>
</body>
</html>

```

10. #layoutLogic-ABC .col-C {...}

Add contextual definitions for how the right column class should be treated when in the three-column layout (it will not appear in the one- and two-column layouts). In this example, the column will be 150 pixels wide.

11. #layoutLogic-A .col-B,
 #layoutLogic-AB .col-C,
 #layoutLogic-A .col-C {...}

Add contextual definitions to specify in which layouts the left and right columns should not be displayed.

12. .page > .section {...}

In order for this trick to work in Mozilla browsers, you also need to add an invisible border to the bottom of the section layer. To keep this from interfering with Internet Explorer for Windows, though, use the previous parent selector discussed in Chapter 2, which Internet Explorer for Windows does not support and so will ignore.

13. index.html

Create a new HTML page (**Code 5.5**). Steps 14 and 15 apply to this file.

14. <link href="columnLayout.css"
 rel="stylesheet" type="text/css"
 media="all"/>

Add a link to the columnLayout.css style sheet.

continues on next page

```
<div id="layoutLogic-ABC">
```

Set up your layers using the IDs and classes created in `columnLayout.css`, with this basic structure:

```
layoutLogic-ABC
```

```
header
```

```
page
```

```
section
```

```
col-A
```

```
col-B
```

```
col-C
```

```
footer
```

This will create a three-column background with solid-colored backgrounds in the left and right columns and a header and footer across the top and bottom (**Figure 5.9**). Simply by changing the `layoutLogic` ID, though, we can transform this into a two-column layout with `layoutLogic-AB` (**Figure 5.10**) or a one-column layout with `layoutLogic-A` (**Figure 5.11**).

✓ Tips

This example shows a relatively straightforward example of using contextual layout to create different column layouts, but the possibilities are limitless, especially when combined with dynamic interactions.

If all of your pages are hard-coded, this technique will not be particularly effective, but if you're creating Web pages from a template using some server-side technology such as PHP, JSP, or ASP, then this allows you to simply change the layout logic to deliver the exact layout you want without having to maintain different code bases for each version.

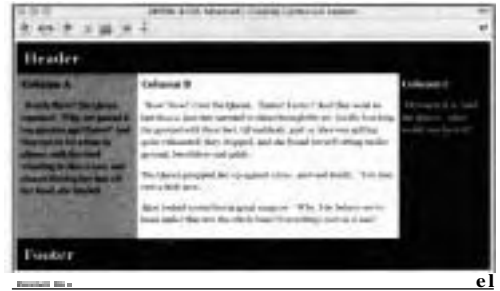


Figure 5.9 The three-column layout (ABC).

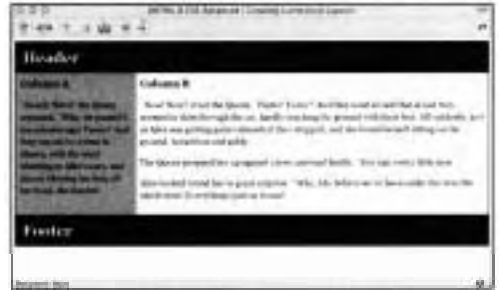


Figure 5.10 The two-column layout (AB).



Figure 5.11 The one-column layout (A).

Code 5.6 The horizontal and center IDs work together to place an element in the center of the browser window. However, this will only work if you know the width and height of the element.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Transitional//EN" "http://www.w3.org/TR/
  >xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  >xml:lang="en">
  <head>
    <meta http-equiv="content-type"
      > content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I
      > Centering Content Horizontally and
      > Vertically</title>
    <style type="text/css"><!--
body {
  background-color: #999;
  margin: 0;
  padding: 0; }
#horizontal {
  background-color: transparent;
  visibility: visible;
  display: block;
  position: absolute;
  top: 50%;
  left: 0;
  width: 100%;
  height: 1px;
  overflow: visible; }
#center {
  background-color: #fff;
  visibility: visible;
  margin-left: -150px;
  position: absolute;
  top: -105px;
  left: 50%;
  width: 300px;
  height: 210px;
  border: solid 1px black; }
.content {
  font-size: 14px;
  padding: 5px; }
```

(code continues on next page)

Centering Layouts Horizontally and Vertically

Earlier in this chapter, when I was showing you how to create a simple column layout, I used an easy technique to horizontally center the columns in the middle of the page by setting the margins for the content to auto for the left and right: `margin: 0px auto;`

Centering the content vertically, though, is a bit more challenging. It would be great if we could simply use auto for that as well, but neither life nor Web design always works out the way we would like it to, at least not in all browsers. Instead, we will need to combine two layers together, adjusting the margin and absolute positioning.

One caveat with this technique, though: It only works if you know the exact width and height of the element being centered in the page.

To center elements horizontally and vertically in the page:

1. `body {...}`

To overcome the positioning differences among the various browsers, set the margin and padding for the body to 0 (**Code 5.6**).

2. `#horizontal {...}`

Add the horizontal ID to your CSS. This ID is used to position the element horizontally on the page 50% from the top.

3. `#center {...}`

Add the center ID to your CSS. This ID is the workhorse for centering the element. Begin by setting the width and height of the element, then set the left margin to a negative value half the width of the element and the top margin to a negative value half the height of the element.

continues on next page

4. .content {...}

Add the content class to your CSS. This class is used to add padding to the element without having to worry about the box model hack described in Chapter 2.

5. <div id="horizontal">...</div>

In your HTML, set up your layers using the IDs and classes created above, following this structure:

```
horizontal
  center
    content
```

This will place the element in the exact center of the Web page (**Figure 5.12**) even after the Web page is resized (**Figure 5.13**).

Code 5.6 continued

```
Code
--></style>
</head>
<body>
  <div id="horizontal">
    <div id="center">
      <div class="content">
        
        <p>`Well, &quot;
          >OUTGRABING&quot; is
          > something between bellowing
          and whistling...</p>
      </div>
    </div>
  </div>
</body>
</html>
```



Figure 5.12 The element is centered smack-dab in the middle of the browser window...



Figure 5.13 ...and stays in the center even if the window is resized.

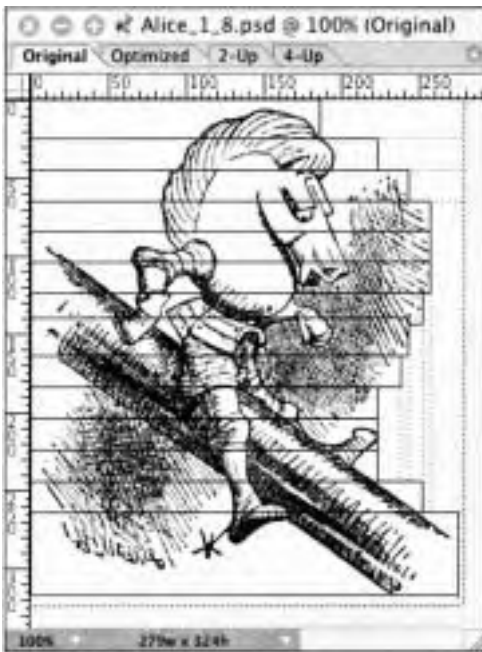


Figure 5.14 The image sliced into 14 horizontal pieces ready for layout.

Creating Curved Text Wrapping

With HTML, you can wrap text around a graphic using the `float` property. However, since all images are rectangular, the text wrapping around the image will follow the edge of the image, creating a distinctly rectangular pattern. Most print layout programs, though, allow you to control the exact shape of the wrap around the image, creating curved shapes. There is no easy way around rectangular text wrapping using CSS.

However, if you have a bit of patience, you can create a faux-curved text wrap around what appears to be a single image. The trick is to slice the image into horizontal strips and then float those next to the text.

To create curved text wrapping:

1. Alice1_8.psd

Open the graphic file you want to wrap your text around in an image editing application such as Adobe Photoshop, Adobe ImageReady, or Macromedia Fireworks. Cut the image into horizontal strips with a height of 25 pixels (**Figure 5.14**).

Save each of the strips as a separate image file. In this example, I used the original filename (`alice1_8`) and then added another sequential number for each slice.

continues on next page

2. `img.curveText {...}`

In your HTML document, add the class `curveText` to your CSS (**Code 5.7**). This class will be used to float the image strips to the left.

3. ``

Add each image in order to the page. At first the images will look like a jumbled jigsaw puzzle (**Figure 5.15**) until you add the `curveText` class.

4. `<p>But oh!' thought Alice...</p>`

Add your text within a `<p>` tag immediately after the images. The images will all float to the left, and the text will wrap to each strip, creating the illusion that the text is curving around a solid image (**Figure 5.16**).

✓ Tips

You may have to play around with the slicing of the image before you get the right fit.

The exact height into which you cut the strips will depend on the size of the font you use for your text. A good rule of thumb is to add about 10 pixels to the font size for the height of each strip. The width of each strip depends on the contours of the image.

This technique does not exactly work with Internet Explorer for Mac. The image will look fine, but the text will still follow the rectangular edge of the widest strip rather than curving.



Figure 5.15 The images in the window before the `curveText` class is added.

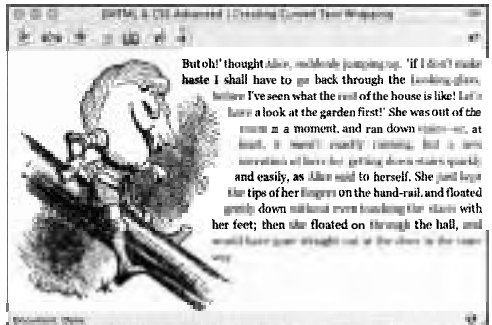


Figure 5.16 After the `curveText` class is applied, the text appears to curve around the image. Really, it's several images stacked up, but the viewer need never know.

Code 5.7 The class `curveText` is added to the image strips to float them to the left. The text will then wrap to the edge of each individual strip, creating the illusion of a curve.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <title>DHTML &amp; CSS Advanced | Creating Curved Text Wrapping</title>
    <style type="text/css"><!--
img.curveText {
  margin-right: 25px;
  float: left;
  clear: left; }
p {
  font-size: 14px;
  text-align: justify;
  margin-top: 0; }
--></style>
  </head>
  <body>
    
    
    
    
    
    
    
    
    
    
    
    
    
    
    <p>But oh! thought Alice, suddenly jumping up...</p>
  </body>
</html>
```

Creating Curved Borders

Due to the limitations of HTML and CSS, most designs are rather rectangular in nature. There is simply no easy way to create rounded rectangles. The only consistent way to get rounded corners is using graphics. The good news is that CSS makes it relatively easy to incorporate not only round-edged graphics into your design, but other effects as well.

To do this, you'll first need to create a large graphic with the design you want to use, and then use CSS to piece it together in the HTML.

To create curved borders around an element:

1. CurvedBorders.psd

Using image editing software such as Adobe Photoshop, Adobe ImageReady, or Macromedia Fireworks, create a rectangular shape with rounded corners and add any other special effects you desire, as long as they can tile (**Figure 5.17**).

In this example, I created a simple rounded rectangle 750×750 pixels, with a slight emboss and drop shadow to create a glossy tile effect.

2. Once you're satisfied with your design, slice it into six pieces (**Figure 5.18**):



Figure 5.17 The original image (750×750 pixels) used to create the curved borders.



Figure 5.18 The design has been sliced into six individual pieces.




Figure 5.19 The top left will be formed by `bg_topLeft`.




Figure 5.20 The top right will be formed by `bg_topRight`.




Figure 5.21 The left edge will be formed by tiling `bg_centerLeft` vertically.




Figure 5.22 The center and right edge will be formed by tiling `bg_centerRight` vertically.




Figure 5.23 The lower-left corner will be formed by `bg_bottomLeft`.




Figure 5.24 The lower-right corner will be formed by `bg_bottomRight`.

`bg_topLeft` stretches from the top-left side of the design across to the edge of the curve on the opposite side. This piece will slide to allow a variable width to the background (Figure 5.19).

`bg_topRight` stretches from the edge of `bg_topLeft` to the top-right edge of the design (Figure 5.20).

`bg_centerLeft` stretches from the center left (no curve) of the design to just within the edge of the left border. This piece will tile vertically to create the left edge (Figure 5.21).

`bg_centerRight` stretches from `bg_centerLeft` to the center-right edge of the design. This piece will tile vertically to create the center and right edge (Figure 5.22).

`bg_bottomLeft` stretches from the bottom-left side of the design across to the edge of the curve on the opposite side. This piece will slide to allow a variable width to the background (Figure 5.23).

`bg_bottomRight` stretches from the edge of `bg_bottomLeft` to the bottom-right edge of the design (Figure 5.24).

3. Save each piece as a separate image file in the folder images.

continues on next page

4. `.curvedArea {...}`

Add the class `curvedArea` to your CSS (**Code 5.8**). This class is used to control the width of the area.

5. `.topLeft {...}`

Add the class `topLeft` to your CSS. This class uses the `bg_topLeft` image, floating it to the left and stretching across the top.

6. `.topRight {...}`

Add the class `topRight` to your CSS. This class uses the `bg_topRight` image, floating it to the right, adding the top-left corner.

7. `.centerLeft {...}`

Add the class `centerLeft` to your CSS. This class uses the `bg_centerLeft` image, floating it to the left, adding the left edge of the design.

8. `.centerRight {...}`

Add the class `centerRight` to your CSS. This class uses the `bg_centerLeft` image, floating it to the right and creating the center and right edge.

9. `.bottomLeft {...}`

Add the class `bottomLeft` to your CSS. This class uses the `bg_bottomLeft` image, floating it to the left and stretching across the bottom.

Code 5.8 The background images are used to add a scalable graphic background behind the `curvedArea` layer.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
 Transitional//EN" "http://www.w3.org/TR/
 xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
 xml:lang="en">
  <head>
    <meta http-equiv="content-type"
      content="text/html;charset=utf-8" />
    <title>DHTML & CSS Advanced I
      *Creating Curved Borders</title>
    <style type="text/css"><!--
.curvedArea {
  display: block;
  width: 250px; }
.topLeft {
  background-image:
    url(images/bg_topLeft.jpg);
  width: 100%;
  height: 45px;
  float: left; }
.topRight {
  background-color: #fff;
  background-image:
    url(images/bg_topRight.jpg);
  background-repeat: no-repeat;
  background-position: 100% 0;
  width: 50px;
  height: 45px;
  float: right; }
.centerLeft {
  background-image:
    url(images/bg_centerLeft.jpg);
  background-repeat: repeat-y;
  width: 100%;
  float: left; }
.centerRight {
  background-image:
    url(images/bg_centerRight.jpg);
  background-repeat: repeat-y;
  background-position: 100% 0;
  width: 100%;
  height: 100%;
  float: right; }
```

(code continues on nextpage)

Code 5.8 continued

```
.bottomLeft {
  background-image:
    url(images/bg_bottomLeft.jpg);
  background-repeat: no-repeat;
  width: 100%;
  float: left; }

.bottomRight {
  background-color: #fff;
  background-image:
    url(images/bg_bottomRight.jpg);
  background-repeat: no-repeat;
  background-position: 100% 0;
  width: 50px;
  height: 52px;
  float: right; }

.content {
  margin-right: 25px;
  margin-left: 25px; }

.clear {
  clear: both; }
--></style>
</head>
<body>
  <div class="curvedArea">
    <div class="topLeft">
      <div class="topRight">
        &nbsp;</div>
      </div>
    <div class="centerRight">
      <div class="centerLeft">
        <div class="content">
          <p>But this did not seem likely to
            happen. She went on and on, a
            long way. But whenever the road
            divided there were sure to be
            two finger posts pointing the
            same way, one marked "TO
            TWEEZLER'S HOUSE" and
            the other "TO THE HOUSE OF
            TWEEZERS".
          </p>
        </div>
      </div>
    </div>
    <div class="bottomLeft">
      <div class="bottomRight">
        &nbsp;</div>
      </div>
    </div>
  </div>
</body>
</html>
```

10. .bottomRight {...}

Add the class `bottomRight` to your CSS. This class uses the `bg_bottomRight` image, floating it to the right, creating the bottom-right corner.

11. .content {...}

Add the class `content` to your CSS. This will be used to add margins to the content that goes within the curved area so that the content does not bump up against the edge of the design.

12. <div class="curvedArea">...</div>

In your HTML, set up your layers using the IDs and classes created above, following this structure:

```
curvedArea
  topLeft
    topRight
  centerRight
    centerLeft
  bottomLeft
    bottomRight
```

No, it is not a misprint: `centerLeft` goes within `centerRight` for this to work (Figure 5.25).



Figure 5.25 The final layout with all of the pieces in place. This technique can stretch vertically indefinitely and horizontally 750 pixels.

Creating a Drop Shadow Around an Element

Drop shadows are a time-honored way to make elements pop off a page. With Web designs, though, drop shadows are generally only available if they are included as part of a larger graphic.

In this example, I want to show you how to place drop shadows behind any rectangular element on the screen, and even control the offset of the drop shadow in the CSS.

To create a drop shadow:

1. dropshadow.psd

Using image editing software such as Adobe Photoshop, Adobe ImageReady, or Macromedia Fireworks, create a rectangular shape and use a Gaussian blur filter to blur the edges (**Figure 5.26**). This will form the shadow—make it any color you want.

In this example, I created a simple black rectangle 550x550 pixels with a Gaussian blur of 3 pixels. I also set the opacity to 75% to lighten the shadow a bit.

2. Once you're satisfied with your shadow, slice it into three pieces (**Figure 5.27**):



Figure 5.26 The original drop-shadow graphic 550 by 550 pixels.

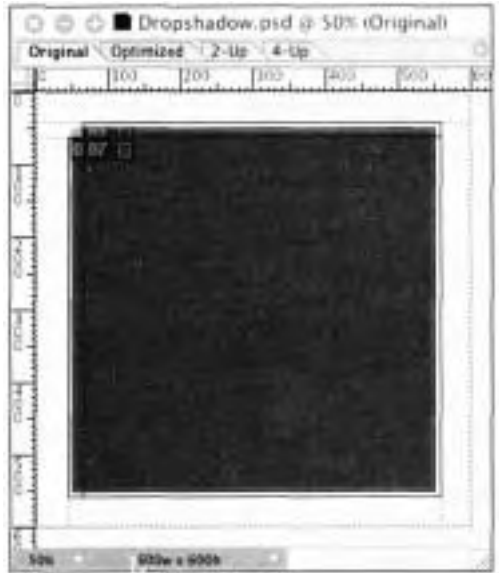


Figure 5.27 The drop-shadow graphic sliced into three pieces.

Figure 5.28 The shadow-top piece will form the top.



Figure 5.29 The shadow-main piece will form the center.

shadow-top stretches from the left of the shadow to the right edge of the shadow. This piece will slide horizontally to create the top-right corner of the shadow (**Figure 5.28**).

shadow-main stretches horizontally and vertically to the bottom-right corner of the shadow (**Figure 5.29**).

shadow-left stretches from the top of the shadow to the bottom edge of the shadow. This piece will slide vertically to create the bottom-left corner of the shadow (**Figure 5.30**).

3. Save each piece as a separate image file in the folder images.

continues on next page

Figure 5.30 The shadow-left piece will form the left.

4. `.shadowMain`, `.shadowBLCorner`,
`.shadowTRCorner`

Add the classes `shadowMain`, `shadowBLCorner`, and `shadowTRCorner` to your CSS (Code 5.9). For their common definitions, set the display state to `inline-table` (for Internet Explorer) and then again as `block`, surrounding the definition with comment marks in this pattern:

```
/*\*/ definition /**/
```

This hides the CSS definition from Internet Explorer. The second display definition is vital, or else the code will crash some versions of the Opera browser.

5. `.shadowMain {...}`

```
.shadowBLCorner {...}
```

```
.shadowTRCorner {...}
```

Customize the shadow classes to display the background. To control the offset for the drop shadow, change the bottom and right padding for `shadowTRCorner`.

Code 5.9 The drop shadow is pieced together from three different graphics, which allows the shadow to scale simply by changing the padding in the CSS.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Transitional//EN" "http://www.w3.org/TR/
  xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xml:lang="en">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I
      Creating a Drop Shadow</title>
    <style type="text/css"><!--
. shadowMain, .shadowBLCorner, .shadowTRCorner {
  display: block; }
. shadowMain {
  background: url(images/shadow_main.gif)
  no-repeat right bottom;
  float: left; }
. shadowBLCorner {
  background: url(images/shadow_left.gif)
  no-repeat left bottom; }
. shadowTRCorner {
  background: url(images/shadow_top.gif)
  no-repeat right top;
  padding: 0 16px 16px 0; }
.content {
  background-color: white;
  padding: 5px;
  width: 400px;
  border: solid 2px black; }
--></style>
</head>
<body>
  <div class="shadowMain">
    <div class="shadowBLCorner">
      <div class="shadowTRCorner">
        <div class="content">
          

```

(code continues on next page)

```

<p>Of all the strange
  things that Alice saw in
  her journey.. .</p>
</div>
</div>
</div>
</div>
</body>
</html>

```

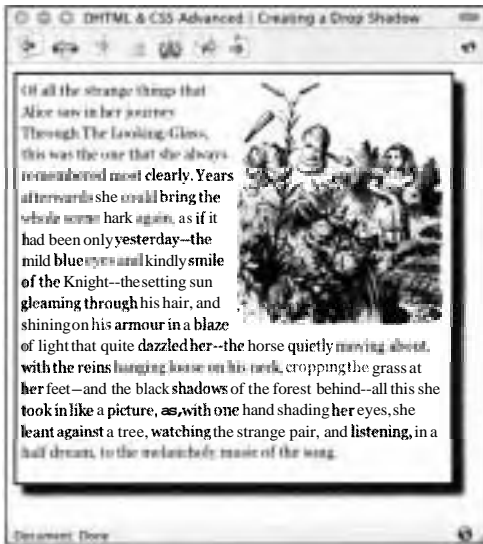


Figure 5.31 The final layout with a drop shadow offset by 16 pixels. You can change this offset just by changing the padding in the `shadowTRCorner` class.

6. `.content {...}`

Add the class `content` to your CSS. This is used to control the appearance of the area behind which the drop shadow will appear.

7. `<div class="shadowMain">...</div>`

In your HTML, set up your layers using the IDs and classes created above, following this structure:

```

shadowMain
  shadowBLCorner
  shadowTRCorner
  content

```

This will create an area on the screen with a drop shadow offset behind it (**Figure 5.31**).

✓ Tip

This technique assumes that you know the background color for the Web page.

Creating a Frame Drop Shadow

Drop shadows are often—arguably, a little too often—used to add visual interest to graphics while adding emphasis to important elements on the screen.

However, when used with frames, drop shadows get a little trickier. What we have to accomplish here is an integrated effect around different frames, since the shadow for one frame might be in several different frames. I'll show you how to set up a drop shadow on any side of a frame.

To do this we'll create a single graphic that looks like our Web page in miniature. We then split this graphic up and set each piece into a frameset made up of nine different frames: one central content frame surrounded by eight other frames containing the drop shadow.

To add a drop shadow to a frame:

1. `frame_dropshadow.psd`

Use an image editing application such as Adobe Photoshop, Adobe ImageReady, or Macromedia Fireworks to create the backgrounds that will be used for the shadows in each frame (**Figure 5.32**)

Start with a large square graphic (I used one 300×300 pixels), filled with the background texture or graphic of your choice. I created mine using the noise and emboss filters to make a rough, stonelike surface.

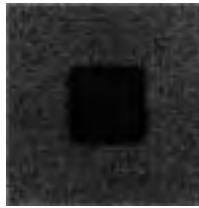


Figure 5.32 The original image used to create the drop shadows for the frame.

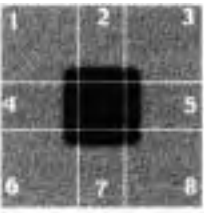


Figure 5.33 The lines show where to slice your image.

Code 5.10 The style sheet `dropShadow.css` is applied to `ds1-8` to add the drop shadows. Placing the images in one central file means you can make any changes in just one place.

```
Code
.ds_1, .ds_2, .ds_3, .ds_4, .ds_5, .ds_6, .ds_7,
.ds_8 {
    background-color: #666; }
.ds_1 {
    background-image: url(../images/ds_1.gif);
    background-repeat: no-repeat; }
.ds_2 {
    background-image: url(../images/ds_2.gif);
    background-repeat: repeat-x; }
.ds_3 {
    background-image: url(../images/ds_3.gif);
    background-repeat: no-repeat; }
.ds_4 {
    background-image: url(../images/ds_4.gif);
    background-repeat: repeat-y; }
.ds_5 {
    background-image: url(../images/ds_5.gif);
    background-repeat: repeat-y; }
.ds_6 {
    background-image: url(../images/ds_6.gif);
    background-repeat: no-repeat; }
.ds_7 {
    background-image: url(../images/ds_7.gif);
    background-repeat: repeat-x; }
.ds_8 {
    background-image: url(../images/ds_8.gif);
    background-repeat: no-repeat; }
```

- Next, create another square in the middle of the image and fill it with the color of your central content frame. Duplicate this square underneath the first one, fill it with black, and then apply the Gaussian blur filter to create the shadow.

The more space you place between the inner square and the outer boundary of the graphic, the wider or taller your frames will need to be to accommodate the background graphics. So, if you want a 100-pixel border around the frame with the drop shadow, leave 100 pixels between the inner and outer squares.

- Slice the image into eight different pieces: four sides and four corners (**Figure 5.33**). Don't cut right on the edge of the shadow, but about 10 pixels inside the inner square. This gives a better fit for the drop shadows and allows a border around scrollbars in the center frame.
- Save each individual piece as a separate file. I called mine `ds_1.gif`, `ds_2.gif`, `ds_3.gif`, and so forth.

5. `dropShadow.css`

Create a new external CSS file with classes `.ds_1-8` that define the background image for each quadrant in the frameset and how that background should tile (**Code 5.10**).

continues on next page

6. ds1.html

Create a separate HTML file to house each background graphic. These files should be named `ds_1.html`, `ds_2.html`, `ds_3.html`, and so on (Code 5.11). They will need a link to the `dropShadow.css` file and the relevant class applied to the `<body>` tag.

7. content.html

Create a new HTML document to hold the content in the center frame.

8. index.html

Create a new HTML frame document to put all the pieces together (Code 5.12). You'll need to take into account the size of your background graphics when defining the frame size.

Notice that I eliminated the frame borders and gave all of the outer frames the `noresize` attribute.

The background graphics in the side frames (`ds_2`, `ds_4`, `ds_5`, and `ds_7`) will tile to fill either the horizontal or vertical space of that frame no matter what the dimensions of the window are, while the corner backgrounds (`ds_1`, `ds_3`, `ds_6`, and `ds_8`) remain a constant size to bridge the gaps (Figure 5.34).

Tip

To prevent unwanted repeating of the background graphics, I recommend making the frames 5 pixels shorter than the background graphic.

Code 5.11 These HTML files will all look pretty much the same, except for the class applied to the `<body>` tag.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
 Transitional//EN" "http://www.w3.org/TR/
 xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=
      iso-8859-1" />
    <title>DHTML & CSS Advanced I
      Adding a Drop Shadow to a Frame</title>
    <link href="css/dropShadow.css"
      rel="stylesheet" type="text/css"
      media="all"/>
  </head>
  <body class="ds_1"></body>
</html>
```



Figure 5.34 The final layout with the frames around the center frame forming a drop shadow.

Code 5.12 The frameset index.html creates a grid three frames by three frames. Each of the outer frames holds a piece of the drop shadow.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-frameset.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
    <title>DHTML &amp; CSS Advanced | Adding a Drop Shadow to a Frame</title>
  </head>
  <frameset rows="115,* ,115" border="0" frameborder="0" framespacing="0">
    <frameset cols="115,* ,115" border="0" frameborder="0" framespacing="0">
      <iframe src="ds_1.html" noresize="noresize" scrolling="no" />
      <frame src="ds_2.html" noresize="noresize" scrolling="no" />
      <iframe src="ds_3.html" noresize="noresize" scrolling="no" />
    </frameset>
    <frameset cols="115,* ,115" border="0" frameborder="0" framespacing="0">
      <iframe src="ds_4.html" noresize="noresize" scrolling="no" />
      <frame src="content.html" noresize="noresize" />
      <iframe src="ds_5.html" noresize="noresize" scrolling="no" />
    </frameset>
    <frameset cols="115,* ,115" border="0" frameborder="0" framespacing="0">
      <frame src="ds_6.html" noresize="noresize" scrolling="no" />
      <iframe src="ds_7.html" noresize="noresize" scrolling="no" />
      <frame src="ds_8.html" noresize="noresize" scrolling="no" />
    </frameset>
  </frameset>
</frameset>
<noframes>
<body bgcolor="#FFFFFF" text="#FF0063">
  <p><b><font size="7">Sorry</font></b></p>
  <p><font size="5">But you'll need a frames capable<br />
  browser to see this one with. </font></p>
</body>
</noframes>
</html>
```


Keeping Pages Framed

Imagine that you're reading a book. You finish for the night, place a bookmark on the last page you read, and put down the book. The next night, you pick up the book to resume reading where you left off. Through some strange shift in reality, however, the bookmark returns you to the cover, and you have to flip through all the intervening pages again to get back to where you were. If you're creating a Web site with frames, that's how the site may seem to your visitors: They want to bookmark an interior page but end up bookmarking the cover.

This isn't the only problem with referencing framed Web pages. What if you want to refer visitors to a specific page within the site from an email or from another Web site? Sure, you can give them the URL for that single page, but if you send them directly to that page without the frames, it's like giving them the book without the rest of the pages or the spine. You could create a different frameset document for every page, but this is problematic and unwieldy.

There is an easier method. You can't change the way that frames work, and for some reason browser manufacturers have ignored these major usability problems. But you can implement a workaround in your Web site to help visitors overcome these problems.

In this example, I'll show you how to detect whether a page is framed and then deliver it to the correct frameset if it is not.

To place an HTML page in a frameset automatically:

1. framed.js

Create an external JavaScript file, and save it as `framed.js` (**Code 5.13**). Add script that first determines whether the document is loaded into a frameset by

Code 5.13 When any Web page that has this framed.js code is loaded in a browser, it will immediately load the file `index.html`, passing that file its URL so that it can be placed in a frame.

```
Code
myPage = self.location;
thisPage = top.location;
if (thisPage == myPage)
{
    contentSRC = escape(myPage);
    frameURL = 'index.html?' + contentSRC;
    top.location.href = frameURL;
}
```

Code 5.14 The frameset `index.html` includes JavaScript that will check the full URL to see if there is another URL it should use for the content frame.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Frameset//EN" "http://www.w3.org/TR/xhtml1/
DTD/xhtml1-frameset.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      content="text/html;
      charset=iso-8859-1" />
    <title>DHTML &amp; CSS Advanced I
      Keeping Pages Framed</title>
  </head>
  <script language="JavaScript">
    contentSRC = (location.search.
      substring(1) ? location.search.
      substring(1) : 'content.html');
    contentSRC = unescape(contentSRC);
    var writeFrame = ''
    writeFrame += '<FRAMESET COLS="*,575,*"
      BORDER="0" FRAMESPACING="0"
      FRAMEBORDER="NO">';
    writeFrame += '<FRAME
      SRC="filler.html">';
    writeFrame += '<FRAMESET
      ROWS="50,*,50">';
    writeFrame += '<FRAME
      SRC="filler.html">';
    writeFrame += '<FRAME SRC="' +
      ( contentSRC ) + '" NAME="content"
      NORESIZE>';
    writeFrame += '<FRAME
      SRC="filler.html">';
    writeFrame += '</FRAMESET>';
    writeFrame += '<FRAME
      SRC="filler.html">';
    writeFrame += '</FRAMESET>';
    document.write(writeFrame);
  </script>
</html>
```

comparing the document's URL with the URL of the entire window. If the document is in a frameset, the URLs are different, and nothing happens. If the URLs are the same (no frameset), the script combines the URL for the frameset the document should be in (in this case, `index.html`) with its own URL (`myPage`), separating these two URLs with a question mark (?). The function finishes by resetting the location of the window to this new `frameURL`.

2. `index.html`

Set up a frameset document, using JavaScript to write the HTML code into the page, and save it as `index.html` (**Code 5.14**). This JavaScript extracts the URL for the page to be loaded into the content frame from the URL of the page currently loaded. If a question mark (?) appears in the main URL for the page, the script uses whatever is after that question mark for the URL of the content frame saved in the variable `contentSRC`. Otherwise, the script uses a default URL—in this case, `defaultContent.html`—which you can set to be any file that you want.

continues on next page

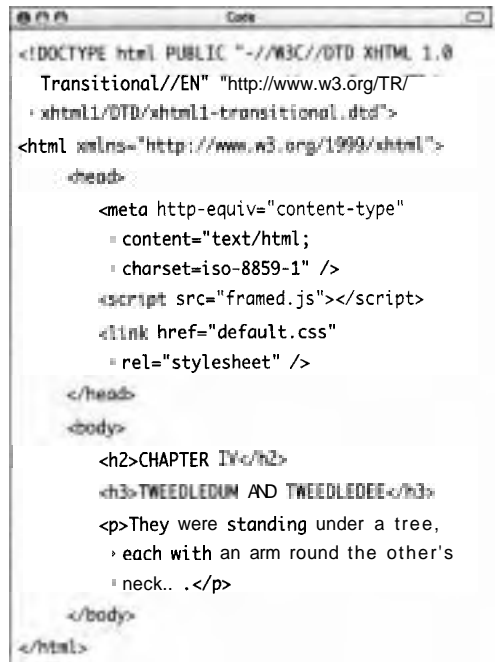
3. content.html

Create the Web pages for your site (**Code 5.15**). In the head of all the documents that you want to place dynamically in the content frame of the frameset, link to the external JavaScript file you created in step 1:

```
<script src="framed.js"></script>
```

When this page first loads in a browser window (**Figure 5.35**) it will sense whether it is framed. If it is not, it will reload itself in the frameset index.html (**Figure 5.36**).

Code 5.15 This page is loaded if no question mark (?) is used in the URL. It includes the framed.js link to force the page into a frameset, if necessary.



```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
 Transitional//EN" "http://www.w3.org/TR/
 xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      content="text/html;
      charset=iso-8859-1" />
    <script src="framed.js"></script>
    <link href="default.css"
      rel="stylesheet" />
  </head>
  <body>
    <h2>CHAPTER IV</h2>
    <h3>TWEEDLEDUM AND TWEEDLEDEE</h3>
    <p>They were standing under a tree,
      each with an arm round the other's
      neck.. .</p>
  </body>
</html>
```



Figure 5.35 The Web page initially loads into a naked screen, but almost instantly...



Figure 5.36 ...the screen blinks, and the page you were viewing is now in the appropriate frameset.

✓ Tips

If you want to point someone directly to a page within your site, just send the URL for the document created in step 3 (content.html). Any HTML page will place itself in the frameset automatically as long as you import the file framed.js into that page, as shown in step 3.

- When visitors follow a link to one of these pages and then bookmark it, they return to that page with the frameset intact.
- Netscape for Windows and Internet Explorer for both Mac and Windows allow you to bookmark a page in a frame by clicking and holding (Mac) or right-clicking (Windows) and then choosing the appropriate option from the contextual menu. If a visitor bookmarks a page in a frameset that was created with the technique described in this section, when that page is loaded from the bookmark, it will frame itself automatically.
- This technique does not solve all the problems involved with bookmarking frames, but until the browser makers get their acts together, it will help make your framed Web sites much more usable.

Switching Layouts on the Fly

One of the greatest strengths of CSS is its versatility in presenting the same content using different layouts, being able to simply swap a style sheet to create a completely different look. This control over the layout need not merely be limited to the creator of the Web site, but can equally be shared with visitors, giving them the freedom to choose between multiple style sheets or even to mix and match styles from a variety of style sheets.

In this example, I'll show you how to allow the visitor to choose among three completely different styles, all applied to the same content.

To switch layouts on the fly:

1. default.css

Create your default CSS file (**Code 5.16**). This is what visitors will see before they choose an alternative style sheet.

2. style1.css

Create as many alternative CSS designs as you want, placing each in its own external file with its own unique name (**Code 5.17**). Place all of these files with the default CSS file into a folder called css.

3. index.html

Create and save a new HTML file (**Code 5.18**). For this example, I'm saving it as `index.html`. Steps 4–10 apply to this file.

Code 5.16 The style sheet `default.css` is used when the page first loads, before the visitor chooses an alternative style.

```
body {
    background-color: #fff;
    color: black;
    font-family: Arial;
}
```

Code 5.17 The style sheet `style1.css` is one of the alternative style sheets available to the visitor.

```
body {
    background-color: #000;
    color: white;
    font-family: Times;
}
```

Code 5.18 As the page `index.html` loads, it checks a cookie to find the style sheet it should use to display its content. If the cookie has not been set, it uses the default.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
 Transitional//EN" "http://www.w3.org/TR/
 xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
 xml:lang="en">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced |
      Switching Layouts on the Fly</title>
    <script type="text/javascript"
      src="cookieLibrary.js"></script>
    <script type="text/javascript"
      language="javascript">
      function getDataCookie(cookieName) {
        var data = '';
        if (navigator.cookieEnabled)
          data = getCookie(cookieName);
        if (data == '') data = 'default';
      }
    </script>
  </head>
  <body>
  </body>
</html>
```

(code continues on next page)

Code 5.18 continued

```

return data;
}
function changeStyle(newStyle) {
    if (navigator.cookieEnabled) {
        setCookie('styleName', newStyle);
        location.href = self.location;
    }
    else alert('You need a
        = cookie-enabled browser to use
        this site. Please either turn
        on cookies or update your
        browser.');
```

```

}
</script>
<script type="text/javascript"
    = language="javascript">
    var currentStyle = getDataCookie
        = ('styleName');
    document.write ('<link href="css/'
        = + currentStyle + '.css"
        = rel="stylesheet" type="text/css"
        = media="all"/>');
```

```

</script>
</head>
<body>
<a href="javascript:changeStyle
    = ('default');">Default</a> | <a
    = href="javascript:changeStyle
    = ('style1');">Style 1</a> |
    = <a href="javascript:changeStyle
    = ('style2');">Style 2</a>
<p>But this did not seem likely to
    = happen...</p>
</body>
</html>
```

```

4. <script type="text/javascript"
    = src="cookieLibrary.js"></script>
```

Add a `<script>` tag using `cookieLibrary.js` as its source. This is the file we created in "Storing Data in Cookies" in Chapter 3. Make sure to copy that file into the same folder where you saved `index.html`.

```

5. function getDataCookie(cookieName)
    = {...}
```

Add the function `getDataCookie()` to your JavaScript. This function makes use of the `getCookie()` function from `cookieLibrary.js`, to read the data in a particular cookie stored on the visitor's computer. It will return the value stored in the cookie if the cookie exists, or "default" if there is no cookie.

```

6. function changeStyle(newStyle) {...}
```

Add the function `changeStyle()` to your JavaScript. This function makes use of the `setCookie()` function from `cookieLibrary.js` to either add or change the cookie `styleName` and then reload the page.

```

7. var currentStyle = getDataCookie
    = ('styleName');
```

Start a new JavaScript container and use the function `getDataCookie()` to find the filename for the style sheet to be used, stored in the cookie called `styleName`.

continues on next page

```
8. document.write ('<link href="css/'
+ currentStyle + '.css"
+ rel="stylesheet" type="text/css"
+ media="all"/>');
```

Add a document.write() to your JavaScript that will add a link tag to your HTML using the variable currentStyle to specify the style sheet filename. The first time the Web page is loaded, currentStyle will be default, and the default.css style sheet is used (Figure 5.37).

```
9. <a href="javascript:changeStyle
('style1');">Default</a>
```

In the body of your Web page, add links for each style option to trigger the changeStyle() function, feeding it the name of the new style sheet to be used. Clicking one of these links will change the cookie styleName and reload the page with the alternative style sheet (Figure 5.38). Clicking another link will switch the style again (Figure 5.39).

✓ Tips

- Although I've only shown you how to change a single style sheet, this technique could easily be adapted to allow visitors to choose among multiple style sheets for a single page. For example, one style sheet could control background colors, another fonts, and a third foreground colors. Adding a different link and currentStyle variable for each would allow visitors to mix-and-match these three groups to get the look they desire.

This example makes use of JavaScript and cookies to control the current style sheet being used, but you can also use PHP, JSP, or ASP to control it on the server, which should increase page load-speed.

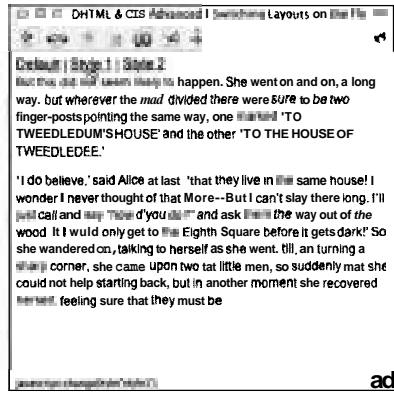


Figure 5.37 When the page first loads, it is using the default style sheet.



Figure 5.38 Clicking the Style 1 link reloads the page using style1.css.

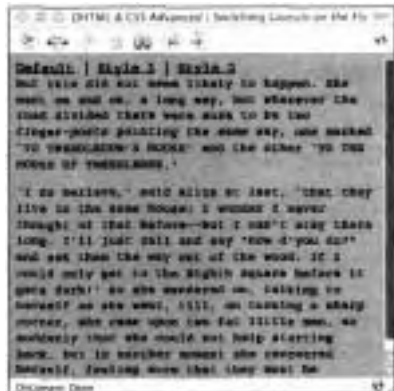


Figure 5.39 Clicking the Style 2 link reloads the page using style2.css.

Code 5.19 The function `highlightRow()` is used to set the zebra strips in the table rows as well as add event handlers to highlight rows when visitors place their mouse over it.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Transitional//EN" "http://www.w3.org/TR/
  xhtml1/DTD/xhtml1-transitional.dtd">
<html xml:lang="en" xmlns="http://www.w3.org/
  1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I
      Dynamic Table Design</title>
    <style type="text/css"><!--
table {
  font: lem helvetica, arial, sans-serif; }
td, th {
  padding: 5px;
  border-bottom: 1px solid #000; }
th {
  color: #fff;
  background-color: #000;
  text-align: left; }
#dataTable01 {
  width: 400px; }
tr.highlightOffOdd {
  color: #000;
  background: #ccc; }
tr.highlightOffEven {
  color: #000;
  background: #999; }
tr.highlightOn {
  color: #000;
  background: #f99;
  border-left: 1px solid transparent; }
--></style>
<script language="javascript"
  type="text/javascript">
  function highlightRow() {
    if(document.getElementById) {
      var tables=document.get
        ElementsByTagName('table');
```

(code continues on next page)

Highlighting Table Rows

Although CSS is increasingly replacing tables for the basic layout of Web pages, the `<table>` tag is still the best way to present the content it was originally intended for: tabular data. In fact, tables can greatly benefit from a close association with CSS. Not only can CSS be used for the obvious—defining headers, borders, and row colors—but with a bit of DHTML, data tables can be made more useful with rollover effects, so visitors can highlight data along a row.

In this example, we'll set up a simple table with alternating row colors and a highlight color whenever the visitor rolls over one of the rows.

To create highlighted table rows:

1. #dataTable01 {...}

Add the ID `dataTable01` to your CSS (**Code 5.19**). This ID is used to distinguish different tables on the page. In this example, we will only set up a single data table, but you can apply the technique to multiple tables on the same page.

2. tr.highlightOffOdd {...}

`tr.highlightOffEven {...}`
Add the classes `highlightOffOdd` and `highlightOffEven` to your CSS. These will control the appearance of odd and even table rows when the mouse cursor is not over them, alternating background colors. They are dynamically assigned to the table rows in the `highlightRow()` function.

3. tr.highlightOn {...}

Add the class `highlightOn` to your CSS. This is used to control the appearance of table rows when the mouse is over them. This class is assigned to table rows in the `highlightRow()` function.

continues on page 171

Code 5.19 *continued*

```
for (var i=0;i<tables.length;i++) {  
    if(tables[i].className=='highlightTable') {  
        var trs=tables[i].getElementsByTagName('tr');  
        for(var j=0;j<trs.length;j++) {  
            if(trs[j].parentNode.nodeName=='TBODY') {  
                if(j % 2 == 1) var highlightClass='highlightOffOdd';  
                else var highlightClass = 'highlightOffEven';  
                trs[j].className=highlightClass;  
                trs[j].onmouseover=function(){  
                    this.className='highlightOn';  
                    return false  
                }  
                if (highlightClass=='highlightOffOdd') {  
                    trs[j].onmouseout=function(){  
                        this.className='highlightOffOdd';  
                        return false  
                    }  
                }  
                if (highlightClass=='highlightOffEven') {  
                    trs[j].onmouseout=function(){  
                        this.className='highlightOffEven';  
                        return false  
                    }  
                }  
            }  
        }  
    }  
}
```

```
</script>  
</head>  
<body onload="highlightRow()">  
    <table class="highlightTable" id="dataTable01" summary="Scores from the game" border="0"  
        cellspacing="0" cellpadding="0">  
        <thead>  
            <tr>  
                <th scope="col">#</th>  
                <th scope="col">Player</th>  
                <th scope="col">Game 1</th>  
                <th scope="col">Game 2</th>  
                <th scope="col">Game 3</th>
```

(code continues on next page)

```

</tr>
</thead>
<tbody>
  <tr>
    <td>1</td>
    <td>Alice</td>
    <td>2</td>
    <td>3</td>
    <td>4</td>
  </tr>
  <tr>
    <td>2</td>
    <td>Red Queen</td>
    <td>4</td>
    <td>6</td>
    <td>8</td>
  </tr>
  <tr>
    <td>3</td>
    <td>White Knight</td>
    <td>6</td>
    <td>9</td>
    <td>12</td>
  </tr>
  <tr>
    <td>4</td>
    <td>Tweedle Dum</td>
    <td>8</td>
    <td>12</td>
    <td>16</td>
  </tr>
  <tr>
    <td>5</td>
    <td>Tweedle Dee</td>
    <td>18</td>
    <td>15</td>
    <td>28</td>
  </tr>
</tbody>
</table>
</body>
</html>

```

4. `function highlightRow() {...}`

Add the function `highlightRow()` to your JavaScript. This is used to dynamically assign styles to alternate rows in the table and add `onmouseover` and `onmouseout` event handlers to control the highlight. Steps 5–8 apply to this function.

5. `var tables=document.`

`getElementsByTagName('table');`
Initialize the variable `tables` that will use the `getElementByTagName()` function to identify tables on the page. The function then cycles through tables in the page to assign row colors and `onmouseover` and `onmouseout` event handlers.

6. `if (j % 2 == 1) var highlightClass= 'highlightOffOdd';`

Add an `if` statement to determine whether this is an even or odd row and then assign the appropriate class name to the row.

7. `trs[j].onmouseover=function(){...}`

Add an `onmouseover` event handler to the row that will switch its class to `highlightOn`.

8. `trs[j].onmouseout=function(){...}`

Add an `onmouseout` event handler to the row that will switch its class back to `highlightOffEven` or `highlightOffOdd`.

9. `onload="highlightRow()"`

In your `<body>` tag, add an `onload` event handler to initialize the table's appearance by triggering the `highlightRow()` function.

continues on next page

```
10. <table class="highlightTable"
    id="dataTable01" summary="Scores
    from the game">...</table>
```

Set up your table, making sure to add the `highlightTable` class, giving the table a unique ID, and specifying a table head (`thead`) and table body (`tbody`).

When the table first loads, the rows will alternate shades of gray (Figure 5.40).

When the visitor passes their mouse over a table row, the row will be highlighted in light red (Figure 5.41).



Figure 5.40 When the page first loads, the table uses alternating shades of gray.

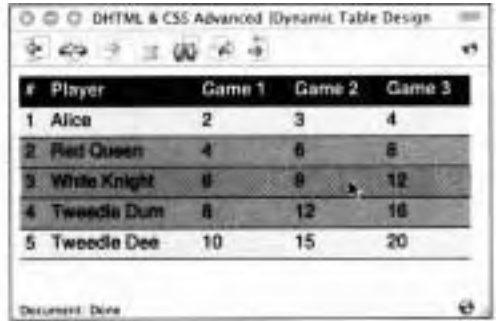


Figure 5.41 When the visitor places their mouse over a row, the row's background color turns light red.

NAVIGATION

When visitors come to your Web site, it's because they are trying to find something. Whether that's technical specs for the latest MP3 player, the scores from this weekend's college football games, or the date of the Battle of Hastings, the quality of your navigation will determine the visitor's success at finding that information. The ability to navigate through a Web site to find information is the most important criterion for a successful Web site. However, the point of navigation is not to make visitors think, "Wow! What great navigation!" but to create an environment in which they don't have to think about the navigation at all: It just works.

What makes great navigation? Here are a few things to keep in mind:

Clear but succinct: Navigation links should provide clear expectations about what the visitor will see after clicking, but not be overly verbose.

Comprehensive but concise: Navigation should provide all of the links that the visitor is likely to need on the page to move around within the Web site, but not simply be a laundry list of links for every page on the site.

Convenient but unobtrusive: Navigation should always be available in a consistent location, but not overwhelm the rest of the page.

In this chapter, we'll look at ways to use CSS and DHTML to achieve the goals presented above. I'll show you navigation schemes and layouts you can use to create great navigation and a great Web site.

Working with Link Styles

One constant of the Web is that hypertext links are underlined. While you can use CSS to turn underlining off for links, many Web designers avoid doing this because they know that this might confuse their visitors. Yet lots of underlined links on a page can look messy and add a lot of visual noise. Part of the problem is that the line under a link is automatically the exact same color as the linked text. But it doesn't have to be this way. Using CSS borders and backgrounds, you have a variety of ways to create link styles that will enhance your designs rather than detracting from them.

Custom underline colors

By default, all hypertext links include the `underline` style. While you can turn this off using CSS, you can also use the `CSS border` property not only to specify the underline color, but also to change the line pattern (dotted, dashed, double, and so on), and even to have different colors or patterns for the different link states (link, visited, hover, and active).

To specify a custom underline color:

1. `a {...}`

In your CSS, set up a link definition for the anchor tag and set the `text-decoration` property to `none` (**Code 6.1**).

2. `a:link {...}`

Add the `a:link` pseudo class and set the `border-bottom` property defining the thickness, style, and color you want to use for your default underline.

In this example, I set up a solid pink 3-pixel-thick line.

Code 6.1 Use the `border-bottom` property for the different link states for greater control over link underlines.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Transitional//EN" "http://www.w3.org/TR/
  xhtml1/DTD/xhtml1-transitional.dtd">
<html xml:lang="en" xmlns="http://www.w3.org/
  1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      = content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I
      Working with Link Styles I
      Custom Underline Colors</title>
    <link href="../../default.css"
      rel="stylesheet" type="text/css"
      media="all"/>
    <style type="text/css" media="screen">!--
p {
  line-height: 150%;
}
a {
  text-decoration: none;
  padding-bottom: -2px;
}
a:link {
  color: #f00;
  border-bottom: 3px solid #fcc;
}
a:visited {
  color: #f00;
  border-bottom: 3px dotted #fcc;
}
a:hover {
  color: #f00;
  background-color: #fcc;
  border-bottom: 3px solid #fff;
}
a:active {
  color: #fff;
  background-color: #f00;
  border-bottom: 3px solid #fff;
}
--></style>
</head>
<body>
  <p>Alice had no more breath for talking,
    so they <a href="#">trotted on in
    silence</a></p>
</body>
</html>
```

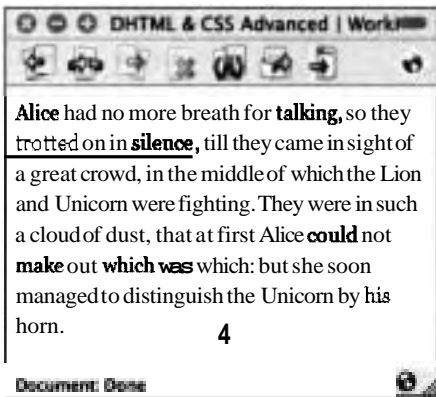


Figure 6.1 Link state: Notice that the line underneath the red link is pink and much thicker than a normal link underline.

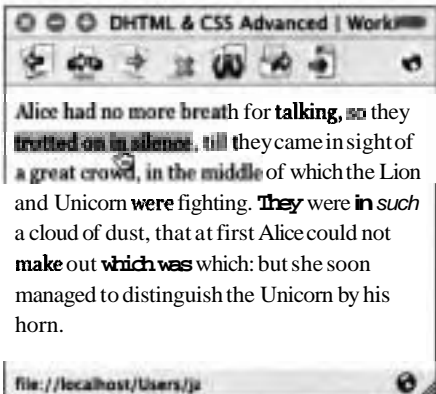


Figure 6.2 Hover state: The background color turns pink giving definite feedback.

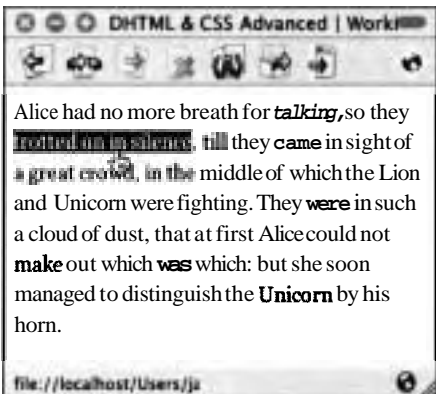


Figure 6.3 Active state: The background turns red with the foreground color turning white.

3. `a:visited {...}`

Add the *visited*, *hover*, and *active* pseudo classes, setting the *border-bottom* attribute to the style you want to use for each of these link states.

In this example, I set up a dotted pink line for visited links, and a solid red line for both the hover and active states. In addition to the border, each of the states includes different background and foreground colors to help the links stand out (Figures 6.1, 6.2, 6.3, and 6.4).

✓ Tip

You may also need to adjust the *padding-bottom* property for links using this technique so that the border doesn't overlap text underneath.

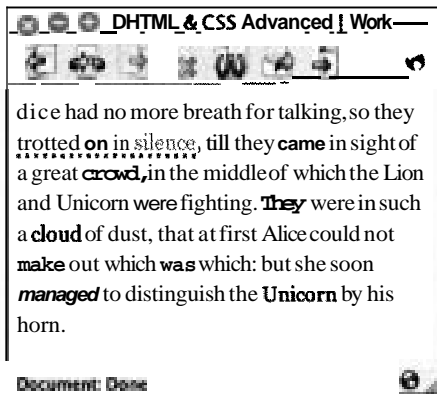


Figure 6.4 Visited state: The underline is now dotted to indicate that this link has already been followed.

Graphic link styles

Although setting the border property to replace the underline in links is effective, it is still limited to setting only lines. However, you can also use the `background-image` property to apply custom-designed graphic link styles to any hypertext link.

A few things to keep in mind while creating your underline graphic:

- ◆ Try to keep it as thin as possible. The thicker the image, the more space it will take up underneath the text.
- If you want the graphic to form a continuous line under the hypertext link, make sure that the image will tile horizontally. In other words, when placed side-by-side, the image should appear seamless.
- ◆ If you will only be using the image once under the link, make sure that it is no wider than your narrowest link. In other words, don't place an image 60 pixels wide under a link on a three-letter word. If you do, the graphic will be clipped at the right edge of the link, which may look strange.

To specify a custom graphic link style:

1. greek.png

airbrush.png

triangle.png

Using an image-editing software such as Adobe Photoshop, Adobe ImageReady, or Macromedia, create your custom underline graphic.

In this example, I have created three basic graphics:

- A **Greek:** A simple Greek pattern, saved as `greek.png` (**Figure 6.5**). This pattern will tile horizontally under the link

Figure 6.5 The Greek pattern that will be tiled horizontally (6x5 pixels).



Figure 6.6 The default airbrush-stroke graphic (97x8 pixels) will appear once under a link.



Figure 6.7 The hover airbrush stroke graphic (97x8 pixels) will appear once.

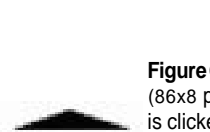


Figure 6.8 The triangle graphic (86x8 pixels) will appear as the link is clicked.

Code 6.2 Add background images to the link tag, aligned to the bottom of the element, to create graphic underlines for your hypertext links.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xml:lang="en" xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I
      Working with Link Styles I
      Graphic Link Styles</title>
    <link href="../../default.css"
      rel="stylesheet" type="text/css"
      media="all"/>
    <style type="text/css"
      media="screen"><!--
a.greek {
  color: #f00;
  text-decoration: none;
  background-repeat: repeat-x;
  background-position: bottom;
  background-image: url(images/greek.png);
  padding-bottom: 5px;
}
a.airbrush {
  border-bottom: none;
  color: #f00;
```

(code continues on next page)

```

    text-decoration: none;
    padding-bottom: 6px;
}
a.airbrush:link {
    background-image: url(images/airbrush.png);
    background-repeat: no-repeat;
    background-position: left bottom;
}
a.airbrush:hover {
    background-image: url(images/airbrush_
hover.png);
    background-repeat: no-repeat;
    background-position: left bottom;
}
a.triangle {
    color: #f00;
    text-decoration: none;
}
a.triangle:link {
    border-bottom: 1px solid #fcc;
}
a.triangle:visited {
    color: #f00;
    text-decoration: none;
    border-bottom: 1px dotted #fcc;
}
a.triangle:hover {
    background-image: url(images/triangle.png);
    background-repeat: no-repeat;
    background-position: center bottom;
    margin-bottom: -5px;
    padding-bottom: 5px;
    border-bottom: 1px solid #f00;
}
a.triangle:active {
    border-bottom: 1px solid #fff;
}
--></style>
</head>
<body>
    This is what a link looks like with a
    href="#" class="greek">a greek flair
    </a><br />
    This is what a link looks like with
    href="#" class="airbrush"
    >an airbrush stroke</a><br /><br />
    This is what a link looks like with
    ><a href="#" class="triangle"
    >a triangle pointer</a>
</body>
</html>

```

▲ **Airbrush:** A simulated airbrush stroke about 5 pixels tall and 100 pixels wide. I created two versions, one for the link state, saved as `airbrush.png` (**Figure 6.6**) and one for the hover state, saved as `airbrush-hover.png` (**Figure 6.7**). This image will be repeated once under the link

A **Triangle:** This is an upward-pointing triangle 46 pixels wide and 8 pixels tall, saved as `triangle.png` (**Figure 6.8**). The triangle will be centered under the link, creating an interesting pointer effect.

2. a.greek {...}

To add the graphic backgrounds as link underlines, you will need to first define anchor tags associated with a class name (**Code 6.2**). For the first link style, we'll create the `greek` class, and add the `greek.png` image as the background, telling it to repeat horizontally at the bottom of the element (**Figure 6.9**).

```
a.airbrush {...}
```

The `airbrush` class uses the lighter version of the airbrush graphic in the default link state with no repeat and justified to the bottom left of the link. The full-strength version is then used for the hover state (**Figure 6.10**).

```
a.triangle {...}
```

The `triangle` class uses a border and graphic together to go from a light underline to a strong hover state where the underline is now pointing at the center of the link (**Figure 6.11**).

continues on next page

3. class="greek"

Add the class for the link styles you want to use.

✓ Tip

Remember that you can create different versions of the graphics for use with each of the link states (link, visited, hover, and active).

Differentiating link types

All links on a Web page are not created equal. Some links point to other pages within the Web site. Some links point to other sections within the Web site. Some links point to pages outside the current Web site. Some links open a new window to display content. There is no inherent way to indicate where a link is pointing. They all look pretty much the same, regardless of whether they are internal or external links. However, the destination of the link may affect how visitors interact with it. When they're reading text, click on a link, and find themselves suddenly transported to a new Web site, it can be confusing. Wouldn't it be nice if you could quickly indicate to visitors what they can expect when clicking the link?

Using a few small graphics and a bit of CSS, we can place distinct icons directly after each text link to indicate its nature.

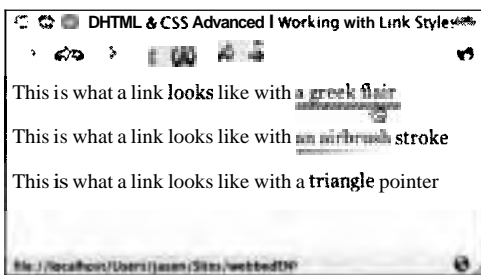


Figure 6.9 The Greek pattern tiles horizontally under the link.

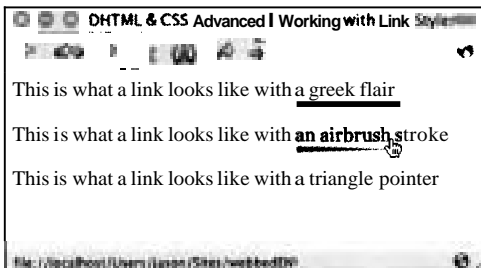


Figure 6.10 The airbrush stroke goes from pink to red in the hover state.

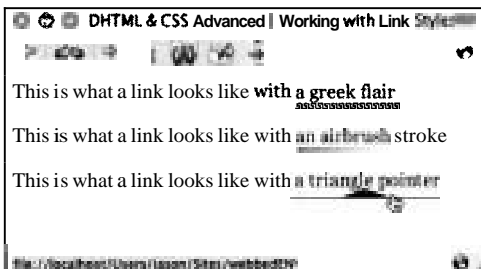


Figure 6.11 The triangle appears pointing at the link.



Figure 6.12 These are the default states for the internal (left) and external (right) linktypes.



Figure 6.13 The active states for the internal (left) and external (right) link types are red.

Code 6.3 You can add an icon to any link using the `background-image` property. Align it to the left or right, don't repeat it, and then add padding to the same side to offset the text.

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
 Transitional//EN" "http://www.w3.org/TR/
 xhtml1/DTD/xhtml1-transitional.dtd">
<html xml:lang="en" xmlns="http://www.w3.org/
 1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced !
      Working with Link Styles !
      > Differentiating Link Styles</title>
    <link href="../../default.css"
      rel="stylesheet" type="text/css"
      media="all"/>
    <style type="text/css" media="screen"><!--
a.internal, a.external {
  font-size: 16px;
  background-repeat: no-repeat;
  background-position: right center;
  text-decoration: none;
  margin-right: 2px;
  padding-right: 12px;
}
a.internal:link {
  color: #f00;
  background-image:
    url(images/internal_link.png);
}
a.internal:visited {
  color: #f00;
  background-image:
    url(images/internal_link.png);
}
a.internal:hover {

```

(code continues on next page)

To add differentiating link icons:

1. external-link.png

internal_link.png

Using a graphic program such as Adobe Photoshop, Adobe ImageReady, or Macromedia Fireworks, create your icons. You'll create different icons for internal and external links (Figure 6.12), and perhaps different versions of both for the hover state (Figure 6.13). You might also want to create versions for the active and visited states. The exact design is up to you, but you should make it visually clear which icon is for internal links and which is for external.

Save these images in separate files with logical names. In this example I saved the icons as `external-link.png`, `external_hover.png`, `internal_link.png`, and `external_link.png`.

2. a.internal, a.external {...}

In your HTML document, add a CSS rule for the `internal` and `external` classes, associating them with the link tag (Code 6.3). Collect all of the common definitions here so that the background will repeat only once and be centered to the right of the link text.

You should also add padding on the right to offset the text enough so that it will not overlap the icon, generally the icon width plus 2 or 3 pixels.

continues on next page

3. `a.internal:link {...}`

`a.external:link {...}`

For each of the link states (link, visited, hover, and active) for the `internal` and `external` classes, set up rules to define the background image.

Keep in mind that the rule you set up in step 2 positions this background image. You can also set other attributes for the link states; in this example I'm using a background color change to emphasize visited links.

Code 6.3 continued

```
background-image:
  url(images/internal_hover.png);
}
a.internal:active {
  color: #fff;
  background-image:
    url(images/internal_hover.png);
}
a.external:link {
  color: #f00;
  background-image:
    url(images/external_link.png);
}
a.external:visited {
  color: #f00;
  background-image:
    url(images/external_link.png);
}
a.external:hover {
  background-image:
    url(images/external_hover.png);
}
a.external:active {
  color: #fff;
  background-image:
    url(images/external_hover.png);
}
--></style>
</head>
<body>
  Alice had no more class="internal"
  href="#">breath for talking</a>,
  so they trotted on in silence,
  till they came in sight of a
  great crowd, in the middle of which
  the class="external" href="#">Lion
  and Unicorn </a>were fighting. They
  were in such a cloud of dust, that at
  first Alice could not make out which
  was which: but she soon managed to
  distinguish the Unicorn by his horn.
</body>
</html>
```

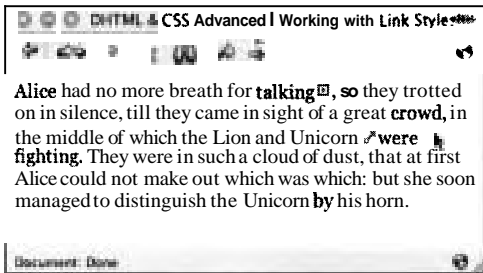


Figure 6.14 When the page first loads, external and internal links are differentiated with icons to the right of the link.

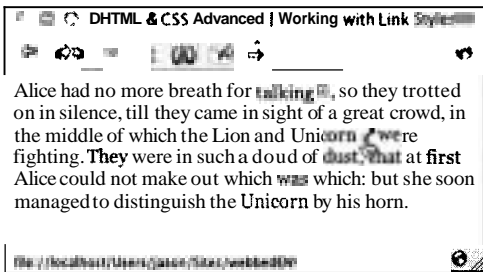


Figure 6.15 When the visitor rolls over the link, the icon changes color.

4. `class="internal"`

Finally, add the class names to the relevant link tags in your HTML code. When the page first loads, internal and external links are clearly marked (**Figure 6.14**). When the visitor mouses over one of these links, it will change color to indicate that it's ready to be clicked (**Figure 6.15**).

✓ Tips

If you want the icon to the left of the text, change the background position, and add padding on the left instead of right.

Generally, I don't recommend setting up background images as part of the default link style, since this will affect all links on the page and may not always look pleasant. Imagine your icon next to a linked graphic!

The exact size of your icons will depend on the size of the text you use them with: You want them to be large enough to be noticeable without distracting or disrupting visitors as they read your text.

Creating an HTML Text Graphic Button

The distinct disadvantage of placing text into a graphic element to create buttons is that it then becomes very difficult to change the text without returning to the original graphic editing software. Conversely, there is only so much that you can do with the design of HTML text on the screen. The solution? Combine graphics with HTML text to create seamless buttons that are as easy to edit as any text.

In this example, we'll make an Aqua-style button with a glossy look and rounded ends.

To create a graphic button with HTML text:

1. `Aqua-left.gif`

`Aqua-middle.gif`

`Aqua-right.gif`

Using your favorite image editing software, create the basic shape of the button (**Figure 6.16**). Make sure that the center of the button can stretch horizontally. Slice the button into three pieces: the left and right ends and then a thin sliver (1 pixel is wide enough) for the middle of the button (**Figure 6.17**). Save each piece as a GIF or PNG file.

2. `#menu {...}`

In the CSS of your HTML document, add a `menu` ID and set its width to the maximum width you want for the button (**Code 6.4**). You can then use as many different menu IDs as needed to control multiple menus around your Web page, but each should have its own unique ID name.

continues on page 185



Figure 6.16 The full image that will be used for the button.

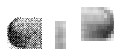


Figure 6.17 The three slices used to create the button: `Aqua-left.gif`, `Aqua-middle.gif`, and `Aqua-right.gif`.

Code 6.4 Place HTML text on a tiling background graphic with capping graphics at either end to create a snazzy button that you can change and add to without ever opening Photoshop.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
  xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I Creating an HTML Text Graphic Button</title>
    <style type="text/css" media="screen"><!--
#menu {
  width: 175px;
}
.button {
  display: block;
  position: relative;
  width: 100%;
  height: 36px;
  clear: both;
  background: url(images/Aqua_middle.gif);
  background-repeat: repeat-x;
  overflow: hidden;
}
.buttonCapLeft {
  background-image: url(images/Aqua_left.gif);
  background-repeat: no-repeat;
  display: inline;
  position: relative;
  width: 30px;
  height: 36px;
  float: left;
}
.buttonCapRight {
  background-image: url(images/Aqua_right.gif);
  background-repeat: no-repeat;
  display: inline;
  position: relative;
  width: 30px;
  height: 36px;
  float: right;
}
.buttonText {
  font-size: 12px;
  width: auto;
  font-family: Arial, Helvetica, Geneva, Swiss, SunSans-Regular;
  font-weight: bold;
  padding-top: 5px;

```

(code continues on next page)

```
height: 36px;
float: left;
}
a:link {
color: black;
text-decoration: none;
}
a:hover {
color: blue;
text-decoration: none;
text-shadow: #ccf 0px 0px 3px;
}

--></style>
</head>
<body>
  <div id="menu">
    <a class="button" href="#">
      <span class="buttonCapLeft">&nbsp;</span>
      <span class="buttonText">Chapter 1</span>
      <span class="buttonCapRight">&nbsp;</span>
    </a>
    <a class="button" href="#">
      <span class="buttonCapLeft">&nbsp;</span>
      <span class="buttonText">Chapter 2</span>
      <span class="buttonCapRight">&nbsp;</span>
    </a>
    <a class="button" href="#">
      <span class="buttonCapLeft">&nbsp;</span>
      <span class="buttonText">Chapter 3</span>
      <span class="buttonCapRight">&nbsp;</span>
    </a>
    <a class="button" href="#">
      <span class="buttonCapLeft">&nbsp;</span>
      <span class="buttonText">Chapter 4</span>
      <span class="buttonCapRight">&nbsp;</span>
    </a>
    <a class="button" href="#">
      <span class="buttonCapLeft">&nbsp;</span>
      <span class="buttonText">Chapter 5</span>
      <span class="buttonCapRight">&nbsp;</span>
    </a>
  </div>
</body>
</html>
```

3. `.button {...}`

Add the button class to your CSS with the same height as the button graphics and using the middle graphic repeating horizontally for the background image. This creates the basic background for the button.

4. `.buttonCapLeft {...}` `.buttonCapRight {...}`

Add classes for the left and right caps, making sure to float them to the left and right respectively and giving them the same width and height as the graphic.

5. `.buttonText {...}`

Define how the HTML text within the graphic button should look. The most important thing here is to add a little padding to the top so that the text doesn't sit right at the top of the graphic.

6. `a:link {...}`

Add any styles you want for the actual link. In this example, I added a color change for the hover state.

7. `id="menu"`

Add a layer using a div tag with the menu ID.

8. `class="button"`

Within the menu layer, add the link using the button class. All of the areas within this will be a part of the link so that the entire button graphic will be clickable.

9. `class="buttonCapLeft"`

Within the link tag, add the `` tag with the `buttonCapLeft` class with a non-breaking space as its only content. This will force the cap graphic to appear on the left of the button.

10. `class="buttonText"`

Within the link tag, add a span tag using the `buttonText` class. In this tag, type the text you want to appear in the button. This area will stretch as much as necessary to fill the space between the two caps, creating the integrated button effect.

11. `class="buttonCapRight"`

Within the link tag, add a span tag using the `buttonCapRight` class and a non-breaking space as its only content. This is the mirror side of the button that finishes the integrated illusion.

Repeat Steps 8-11 for as many buttons as you need in your menu (**Figure 6.18**).

✓ Tips

Keep in mind that if the text in the button grows wider than the menu-layer width (minus the width of the two end caps) the text may break unattractively.

If you want the buttons to stretch the width of their container (in this example the menu layer) set the width for the button class to 100%. If you want to have the button only stretch as wide as is needed to display the text in the button, set the button class width to auto.



Figure 6.18 The middle slice tiles under the text and the caps float left and right to make the button.

Creating Tabbed Navigation

For better or worse, tabs have taken over the Web. You see them everywhere these days, but more often than not their design is based either on a solid graphic (hard to edit) or on HTML text with a flat, square background (not very attractive).

In the previous section I showed you how to combine HTML text with graphics to create attractive buttons. To create tabs now, we'll use a similar technique with a few twists that allow the tabs to degrade gracefully if CSS is not available in the browser displaying the code.

To create tabbed navigation:

1. tab.psd

Using the image editing software of your choice, create your tab graphic as one solid piece (Figure 6.19). The tab can be as narrow as need be in the final HTML, but it's important to make the graphic the maximum width you will need.

In addition to an unselected version of the graphic, create a version to be used for the tab of the currently selected page (Figure 6.20)

2. tab-left.gif

Once you have created your graphics, it's time to carve them up into pieces and save them as individual files:

- ▲ Slice the left edge of the tabs (Figure 6.21) and save as `tab-left.gif` and `tab-left-current.gif`. You want to make this graphic as narrow as possible, but be sure to capture any curved corners.

- ▲ Slice from where `tab-left` finished to the extreme right of the images (Figure 6.22) and save as `tab-right.gif` and `tab-right-current.gif`. This piece will extend or contract to the needed width to contain the HTML text in the tab.



Figure 6.19 The full tab graphic before it's sliced.



Figure 6.20 Create a second version for the selected tab.



Figure 6.21 Slice the left of the tab graphic thin.



Figure 6.22 Slice the top and right of the tab graphic thin.

Code 6.5 This tabbed navigation relies on unordered list and list tags to merge the `tab-right.gif` and `tab-left.gif` graphics to create a seamless tab with HTML text on top.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
 Transitional//EN" "http://www.w3.org/TR/
 xhtml1/DTD/xhtml1-transitional.dtd">
<html xml:lang="en" xmlns="http://www.w3.org/
 1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML &amp; CSS Advanced I
      Creating Tabbed Navigation</title>
    <style type="text/css"><!--
#menuBar {
  font: 0.8em "arial black";
  background: #fff;
  width: 100%;
  float: left;
}
#menuBar ul {
  list-style: none;
  margin: 0;
  padding: 0;
}
#menuBar li {
  background: url(images/tab_right.gif)
    no-repeat right top;
  margin: 0;
  padding: 0;
  float: left;
border-bottom: 1px solid black;
}
#menuBar a {
text-decoration: none;
  background: url(images/tab_left.gif)
    no-repeat left top;
  display: block;
  padding: 5px 15px;
  color: #c00;

```

(code continues on next page)

3. #menuBar {...}

In the CSS of your Web page, add the `menuBar` ID (**Code 6.5**). This defines the containing layer around the tabs in the menu bar, and is where you want to define the font to be used for the HTML text in the tabs.

4. #menuBar ul {...}

For the tabs trick to work, we must use the `` tag with list elements inside. Set the unordered list tag's *list* style to *none* (so that no bullets are inserted) and *margin* and *padding* to 0.

5. #menuBar li {...}

The right side of the tab graphic goes into the background of the `` tag. The list tag is then placed around the HTML text, stretching the graphic to the right as far as needed. However, if the text extends wider than the `tab-right.gif` graphics, the background color will fill the extra space. In addition, we'll add a thin border under the tabs here to help separate them from the underlying page.

6. #menuBar a {...}

The left side of the tab graphic goes into the link tag, which will be placed within the list tag and floated to the left. This completes the illusion of the tab.

7. #menuBar .current {...}

#menuBar .current a {...}

Set up the *current* class to be used in the menu bar, and a second CSS rule for how links are treated within the *current* class. This is where the alternate versions of the tab graphics for the currently selected page go. We'll apply this class only to the `` tag for the currently selected tab. To offset the bottom border we set in step 5, set a top margin of 1px for the current tab.

continues on next page

8. /*IE5-Mac Hac*/

```
#menuBar a {...}
```

There is a bug in Internet Explorer for Macintosh that prevents the above code from being displayed properly. However, by exploiting another bug in how Internet Explorer for Mac interprets notes, we can add a line of code to the link tag to correct the deficiency.

9. <div id="menuBar">...</div>

Set up your menu bar in <div> tags to create an independently controllable layer.

10.

Start your unordered list.

11. Cover

Add a separate list tag for each tab, including a link inside with the appropriate text.

12. <li class="current">

```
<a href="#">Chapter 1</a></li>
```

For the tab for the currently displayed page, add the **current** class to the list tag (**Figure 6.23**).

Code 6.5 continued

```
float: left;
}
#menuBar .current {
    background: url(images/tab_right_
    * current.gif) no-repeat right top;
    margin-top: 1px;
    border-bottom: medium none;
}
3
#menuBar .current a {
    color: black;
    text-decoration: none;
    background: url(images/tab_left_
    * current.gif) no-repeat left top;
}
3
/*IE5-Mac Hac*/
#menuBar a {
    float: none;
}
3
--></style>
</head>
<body>
    <div id="menuBar">
        <ul>
            <li><a href="#">Cover</a></li>
            <li class="current">
                <a href="#">Chapter 1</a></li>
            <li><a href="#">Chapter 2</a></li>
            <li><a href="#">Chapter 3</a></li>
        </ul>
    </div>
    <br />
    <br />
    <h1>CHAPTER I</h1>
    <p><b>Looking-Glass house</b></p>
    <p>One thing was certain,
    that the WHITE kitten had had nothing
    to do with it...<br />
    </body>
</html>
```

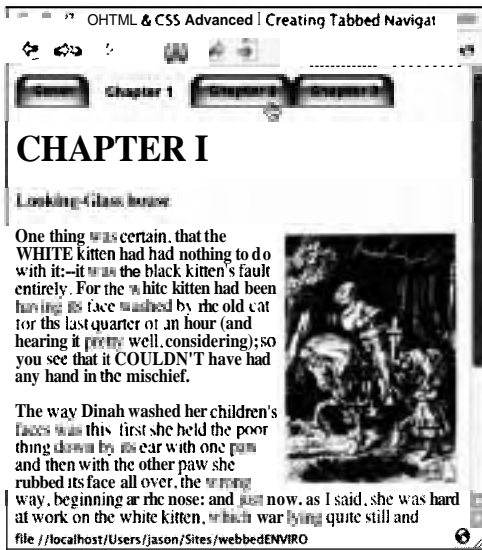


Figure 6.23 The tabs are placed at the top of the page. Although they are using a graphic background, the text is HTML, so they are easy to change.

Internal or External JavaScript?

Most of the JavaScript functions I present in the examples in this book can be included directly in the HTML file (internal) or in a JavaScript file (external) so that the functions can be shared from file to file. For the sake of brevity, in this book I've tended to use internal JavaScript to keep the examples shorter. However, there may come a time when you need to copy and paste these scripts into an external JavaScript file, rather than typing them into every Web page where you use them.

Adding a Simple Menu

More often than not, simplicity is the best solution to a design problem. Multi-option menus are one of the most common ways to present navigation on a Web page, but they can often prove confusing.

In this section, I want to show you how to create a simple but attractive menu that provides definite feedback to the user.

For all of the link states we want to control the following attributes:

- ◆ **color:** The color of the link text in the menu provides important feedback to the visitor. Since this is a menu, I will choose one color for the link and visited states and another for the hover and active states.
- ◆ **background-color:** Once we set the padding and width of the menu option, the background color becomes dominant in the design, so pick one that will not overwhelm other items.
- ◆ **border-right:** For this example I'll make changes to the right border in the various states to add extra feedback for the user. You could alter another of the border sides or even leave this out altogether if it doesn't fit your design needs.
- ◆ **text-shadow:** This is an optional effect I'm including just because I like it. Currently, it only works on a few browsers, including Safari, but it will not interfere with other browsers.

To add a simple menu:

1. `a.menu {...}`

Begin by setting the general appearance of the menu (**Code 6.6**). These styles, applied directly to the menu class without any of the pseudo classes, will be used to display the menu options, unless superseded by definitions in the pseudo classes.

Beyond the font attributes, we also set the overall width of the menu option, padding, and put a small margin at the bottom to add a dividing line between each option. Most importantly, however, we have to set the menu option to be block display style. This will force each menu option onto a new line without having to resort to adding a `<p>` or `
` between, providing maximum control. For example, if you decide to change the space between each menu option or, even more radically, turn the vertical menu into a horizontal menu, you can do that simply through the CSS without having to touch the HTML code.

2. `a.menu:link {...}`

After setting the general menu appearance, set the default link appearance. This is how the menu option looks when the page first loads if the URL associated with the link has not already been visited. For this state, I chose neutral grays and a slight dropshadow.

3. `a.menu:visited {...}`

For the visited state, we duplicate the link state. Generally, menus should not show a visited state, since visitors will jump around and don't need to know whether they have been to a link or not.

Code 6.6 The simplicity of this menu comes not only from how easy it is to use, but also how easy it is to set up.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Transitional//EN" "http://www.w3.org/TR/
  xhtml1/DTD/xhtml1-transitional.dtd">
<html xml:lang="en" xmlns="http://www.w3.org/
  1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I
      Adding a Simple Menu</title>
    <style type="text/css"
      media="screen"><!--
body {
  color: black;
  background-color: #fff;
}
a.menu {
  font-size: 1em;
  font-family: Verdana, Arial, Helvetica,
    sans-serif;
  text-decoration: none;
  display: block;
  margin-bottom: 1px;
  padding: 3px;
  width: 200px;
  text-align: left;
}
a.menu:link {
  color: #2f2f2f;
  background-color: #d3d3d3;
  border-right: 20px solid #6a6a6a;
  text-shadow: #999 0px 2px 2px;
}
a.menu:visited {
  color: #2f2f2f;
  background-color: #d3d3d3;
  border-right: 20px solid #6a6a6a;
  text-shadow: #999 0px 2px 2px;
}
a.menu:hover {
```

(code continues on next page)

Code 6.6 continued

```
    text-align: right;
    color: #d6dbbf;
    background-color: #2f2f2f;
    border-right: 20px solid #f00;
    text-shadow: none;
}
a.menu:active {
    text-align: right;
    color: #d6dbbf;
    background-color: #f00;
    border-right: 20px solid #2f2f2f;
    text-shadow: none;
}
--></style>
</head>
<body>
  <a class="menu" href="#">Chapter 1</a>
  <a class="menu" href="#">Chapter 2</a>
  <a class="menu" href="#">Chapter 3</a>
  <a class="menu" href="#">Chapter 4</a>
  <a class="menu" href="#">Chapter 5</a>
  <a class="menu" href="#">Chapter 6</a>
  <a class="menu" href="#">Chapter 7</a>
</body>
</html>
```

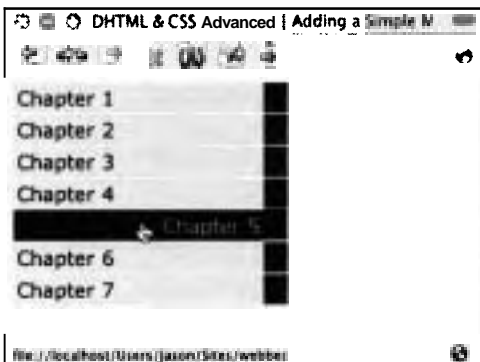


Figure 6.24 The simple menu presents the main navigation links you need to get around in the Web site, with a strong hover state providing feedback to visitors on which option they are choosing.

4. a.menu:hover {...}

The hover state is the most important for providing dynamic feedback to the visitor. We can change a variety of attributes, but in this example we will invert the color and background color, change the right border to a bright red, and justify the text to the right rather than left.

5. a.menu:active {...}

The active state reverses the background and border colors, so visitors get a clear signal when they click a menu option.

6. class="menu"

Add the menu class to each link in the menu (**Figure 6.24**). You can then place the menu into any design you decide, such as the three-column or two-column layouts shown in Chapter 5.

✓ Tips

It is important that visitors be able to quickly find your menu when they navigate from page to page, so always keep the menu in a consistent location on every page in the Web site.

In addition to the background color, you could also use the background image for the various menu options to create some interesting effects.

Adding a Fixed Drop-down Menu

One constant in all operating systems is the fixed menu bar. Whether it's across the top of the entire screen (Mac) or at the top of each individual window (Windows) this provides a consistent point of interaction for the user.

Strangely, Web designers seem to all but ignore this convention of user interface design, generally adding a menu bar at the top of the page that scrolls up and off the page. Imagine if you were working in Word, reached the bottom of the page while typing, and your menu bar disappeared at the top! That is, in effect, what Web designers are doing.

There are several possible solutions to this problem, but the most universally applicable one is to use two horizontal frames. The thin top frame contains the menu bar, while the content frame underneath also contains drop-down menus triggered from the top frame. The upshot is that no matter where the user scrolls, the menu bar will always be available at the top of the screen.

To add a fixed drop-down menu:

1. menus.js

Create and save a new external JavaScript file called `menus.js` as described in Chapter 4 (Code 6.7). This file will be used to create the actual drop-down part of the drop-down menu, which is then imported into `content.html` (step 22). Steps 2-4 apply to this file.

2 var writeMenu = ''

Set up the variable `writeMenu`. This variable will be used to record the code for the menus, and then write them out on the page in step 4.

Code 6.7 The external JavaScript file `menus.js` holds the HTML code that creates the drop-down part of the drop-down menu, so it can be placed in multiple files.

```
Code
var writeMenu = ''
writeMenu += '<div id="dropMenu1"
  class="menuDrop">'
writeMenu += '<a class="menuLink"
  href="#">Link 1</a>'
writeMenu += '<a class="menuLink"
  href="#">Link 2</a>'
writeMenu += '<a class="menuLink"
  href="#">Link 3</a>'
writeMenu += '<a class="menuLink"
  href="#">Link 4</a>'
writeMenu += '<a class="menuLink"
  href="#">Link 5</a>'
writeMenu += '</div>'
writeMenu += '<div id="dropMenu2"
  class="menuDrop">'
writeMenu += '<a class="menuLink"
  href="#">Link 1</a>'
writeMenu += '<a class="menuLink"
  href="#">Link 2</a>'
writeMenu += '<a class="menuLink"
  href="#">Link 3</a>'
writeMenu += '<a class="menuLink"
  href="#">Link 4</a>'
writeMenu += '<a class="menuLink"
  href="#">Link 5</a>'
writeMenu += '</div>'
writeMenu += '<div id="dropMenu3"
  class="menuDrop">'
writeMenu += '<a class="menuLink"
  href="#">Link 1</a>'
writeMenu += '<a class="menuLink"
  href="#">Link 2</a>'
writeMenu += '<a class="menuLink"
  href="#">Link 3</a>'
writeMenu += '<a class="menuLink"
  href="#">Link 4</a>'
writeMenu += '<a class="menuLink"
  href="#">Link 5</a>'
writeMenu += '</div>'
document.writeln(writeMenu);
```

Code 6.8 The external style sheet `menu.css` contains the styles for both the header and drop-down portions of the menu.

```
Code W
a.menuLink {
    display: block;
    padding: 2px 5px;
    border-top: 1px solid #cccccc;
}
a.menuLink:link {
    color: #000000;
    text-decoration: none;
}
a.menuLink:visited {
    color: #000000;
    text-decoration: none;
}
a.menuLink:hover {
    color: #ffffff;
    background-color: #000000;
    text-decoration: none;
}
a.menuLink:active {
    color: #ffffff;
    text-decoration: none;
    background-color: #cc0000;
}
.menuDrop {
    position: absolute;
    z-index: 1000;
    visibility: hidden;
    color: #999999;
    font-size: 10px;
    font-family: arial, Helvetica, sans-serif;
    background-color: #ffffff;
    background-repeat: repeat;
    margin: 0;
    padding: 0;
    top: 60px;
    left: 0;
    width: 175px;
    height: auto;
    border-style: solid;
    border-width: 0 1px 1px;
    border-color: #003365;
}
```

3. `writeMenu += '<div id="dropMenu1" > class="menuDrop">'`

Add each line of the HTML code used to create the menu to the `writeMenu` variable. Each menu will require an open and close `<div>` tag with a unique ID and the `menuDrop` class (the CSS definitions will be in the `menu.css` file created in step 5). Then, each menu option will have a link containing the `menuLink` class.

4. `document.writeln(writeMenu);`

After adding all of the HTML to the `writeMenu` variable, you can now use a single `document.writeln` to add all of the code to the content pages without having to add it directly.

5. `menu.css`

Create and save a new external style sheet called `menu.css` (**Code 6.8**). Steps 6 and 7 apply to this file.

6. `a.menuLink {...}`

Add the `menuLink` class for the anchor tag for all link states (link, visited, hover, active). This class will be used to define the appearance of options in the drop-down menu.

7. `.menuDrop {...}`

Add the `menuDrop` class to your CSS. This defines the general appearance of the entire drop-down part of the menu. The exact design is up to you, but make sure to include definitions to make the menu positioned absolutely, a z-index higher than anything else on the page, and the visibility hidden.

continues on next page

8. menu.html

Create and save the file `menu.html` (**Code 6.9**). This file will not only hold the HTML code to show the fixed menu displayed in the menu frame, but will also contain all of the JavaScript used to control the drop-down menu in the content frame. Steps 9–20 apply to this file.

9. `var objNavMenu = null;`

The drop-down menus use several variables to control their interaction. All of these variables have to be initialized before use:

- ▲ `objNavMenu`: Records the object reference for the currently selected menu option.
- ▲ `prevObjNavMenu`: Records the object reference for the previously selected menu option.
- A `prevObjDropMenu`: Records the object reference for the drop-down of the previously selected menu.
- A `numDropMenu`: Sets the number of available drop-down menus (you set this)
- ▲ `bgLinkColor`, `bgLinkHover`, `bgLinkActive`: Set the background color for the menu options (you set these).
- ▲ `linkColor`, `linkHover`, `linkActive`: Set the foreground link colors for the text in the menus (you set these).
- A `frameName`: Sets the name of the frame where the drop-down menus are located. In this example, the frame is named `content`.

Code 6.9 The file `menu.html` will go into the menu frame; it contains the menu header links as well as all of the JavaScript to control the drop-downs in the content frame.

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Menu</title>
    <meta http-equiv="Content-Type"
      content="text/html; charset=utf-8" />
    <script><!--
var objNavMenu = null;
var prevObjNavMenu = null;
var prevObjDropMenu = null;
var numDropMenu = 3;
///// link styles
var bgLinkColor = '#cccccc';
var bgLinkHover = '#ffffff';
var bgLinkActive = '#000000';
var linkColor = '#000000';
var linkHover = '#000000';
var linkActive = '#ffffff';
var frameName = 'content';
var isIE = null;
if (navigator.appName.indexOf("Microsoft
  > Internet Explorer") != -1) isIE=1;
function initDropMenu () {
  document.onclick = hideDropMenu;
  for (i=1; i<=numDropMenu; i++) {
    menuName = 'dropMenu' + i;
    navName = 'navMenu' + i;
    objDropMenu = top[frameName].document.
      getElementById(menuName);
    objNavMenu = document.getElementById
      (navName);
    objMenu = document.getElementById
      ('menuBar');
    objNavMenu.onmouseout = menuOut;
    objNavMenu.onclick = menuOut;
    objNavMenu.onclick = hideDropMenu;
    objNavMenu.onmouseover = menuHover;
    objNavMenu.onmouseover = showDropMenu;
    parent[frameName].document.onclick =
      hideDropMenu;
```

(code continues on next page)

```

parent[frameName].document.onscroll =
    hideDropMenu;

objNavMenu = null;
return;
3
function menuHover(e) {
    hoverObjNavMenu = document.getElementById
        (this.id);
    if (hoverObjNavMenu != objNavMenu) {
        hoverObjNavMenu.style.color = linkHover;
        hoverObjNavMenu.style.backgroundColor =
            bgLinkHover;
    }
3
function menuOut(e) {
    outObjNavMenu = document.getElementById
        (this.id);
    if (outObjNavMenu != objNavMenu) {
        outObjNavMenu.onclick = showDropMenu;
        outObjNavMenu.style.color = linkColor;
        outObjNavMenu.style.backgroundColor =
            bgLinkColor;
    }
3
function showDropMenu(e) {
    menuName = 'drop' + this.id.substring
        (3, this.id.length);
    objDropMenu = parent[frameName].document.
        getElementById(menuName);
    if (prevObjDropMenu == objDropMenu) {
        hideDropMenu();
        return;
    }
3
    if (prevObjDropMenu != null) hideDropMenu();
    objNavMenu = document.getElementById
        (this.id);
    if ((prevObjNavMenu != objNavMenu) ||
        (prevObjDropMenu == null)) {
        objNavMenu.style.color = linkActive;
        objNavMenu.style.backgroundColor =
            bgLinkActive;
    }

```

(code continues on next page)

10. var isIE = null;

Add a simple browser sniff to detect for Internet Explorer so that we can add some special code to offset problems with that browser in step 14.

11. function initDropMenu () {...}

Add the function `initDropMenu()` to your JavaScript. This function is invoked when the page is loaded to initialize the events applied to each of the menu headers (see step 20) used on the page to show and hide menu drop-downs:

▲ **Menu shows** when the menu option is hovered over.

▲ **Menu hides** when the menu option is no longer being hovered over, the menu header is clicked, or the visitor clicks or scrolls in the content frame.

12. function menuHover(e) {...}

Add the function `menuHover()` to your JavaScript. This function has the event object passed to it and is invoked when the menu option is hovered over by the visitor to set the menu option's foreground and background colors as well as to set the click event for the menu option to hide the drop-down; otherwise clicking the menu header has no effect.

13. function menuOut(e) {...}

Add the function `menuOut()` to your JavaScript. This function is invoked when visitors move their mouse off the menu header to return its colors to normal.

continues on next page

14. function showDropMenu(e) {...}

Add the function `showDropMenu()` to your JavaScript. This function is not only responsible for simply showing the menu (a simple matter of changing the **visibility** property to **visible**) but also for finding where the drop-down menu should be positioned after the visitor has scrolled.

The function begins by creating the ID for the drop menu by pulling the number of the menu header (it's the third character in the triggering menu header's ID) and adding it to the string **'drop'**. It then uses that to locate the drop menu in the **content** frame.

The function will then hide any currently displayed drop menus and change the menu header's background and foreground colors. Finally, the function calculates where the drop-downs should appear based on the position of the triggering menu header and where the content page has scrolled to, with an additional adjustment for Internet Explorer, before changing the drop-down's visibility to **visible**.

Code 6.9 *continued*

```
Code
}
if (objDropMenu) {
  xPos = objNavMenu.offsetParent.
    .offsetLeft + objNavMenu.offsetLeft;
  yPos = parent[frameName].document.body.
    .scrollTop;
  if (isIE) {
    yPos -= 1;
    xPos -= 6;
  }
  objDropMenu.style.left = xPos + 'px';
  objDropMenu.style.top = yPos + 'px';
  objDropMenu.style.visibility = 'visible';
  prevObjDropMenu = objDropMenu;
  prevObjNavMenu = objNavMenu;
}
}
function hideDropMenu() {
  if (prevObjDropMenu) {
    prevObjDropMenu.style.visibility =
      'hidden';
    prevObjDropMenu = null;
    prevObjNavMenu.style.color = linkColor;
    prevObjNavMenu.style.backgroundColor =
      bgLinkColor;
  }
  document.onclick = null;
  objNavMenu = null;
}
</script>
<style type="text/css"><!--
body {
  margin: 0px;
  padding: 0px;
}
#menuBar {
  color: #999999;
  font-size: 12px;
  font-family: arial, Helvetico, sans-serif;
  font-weight: bold;
  text-align: left;

```

(code continues on next page)

```

text-transform: capitalize;
display: block;
margin-bottom: 5px;
position: relative;
height: 100%;
width: 99%;
overflow: hidden;
vertical-align: middle;
border: solid 1px #000000;
background-color: #cccccc;
z-index: -10;
}
.menuHeader {
color: #000000;
text-decoration: none;
white-space: nowrap;
cursor: pointer;
padding: 5px;
margin: 0px;
padding-right: 15px;
display: inline;
position: relative;
border-right: 1px solid #000000;
}
--></style>
</head>
<body onload="initDropMenu()">
  <div id="menuBar">
    <div id="navMenu1"
class="menuHeader">
      Menu 1</div>
    <div id="navMenu2"
class="menuHeader">
      Menu 2</div>
    <div id="navMenu3"
class="menuHeader">
      Menu 3</div>
  </div>
</body>
</html>

```

15. `function hideDropMenu() {...}`

Add the function `hideDropMenu()` to your JavaScript. This function hides the previously displayed menu option before setting the menu header styles back to normal.

16. `#menuBar {...}`

Add the `menuBar` ID to your CSS. This is used to contain and display the menu headers.

17. `.menuHeader {...}`

Add the `menuHeader` class to your CSS. This class defines the appearance for each of the menu header options.

18. `onload="initDropMenu()"`

Add an event handler to the `<body>` tag to trigger the `initDropMenu()` (step 11) function when the page loads.

19. `<div id="menuBar">...</div>`

Add a layer using the `menuBar` ID. This layer will contain all of the menu headers.

20. `<div id="navMenu1" class="menuHeader">...</div>`

Add each of the menu headers in its own `<div>`. Notice that they do not contain any of the event handlers themselves. These were all set up using the `initDropMenu()` function (step 11), and so do not show directly in the code.

continues on next page

21. content.html

Create and save the file content.html (Code 6.10). Of course, you might have several content pages, so look at the instructions for this one as the way to set all of the pages displayed in the content frame.

Steps 22 and 23 apply to this file.

22. href="menu.css"

Add a link to the style sheet file menu.css created in step 5.

23. src="menus.js"

Add a <script> tag with the source pointing to the JavaScript file menus.js created in step 1.

Code 6.10 The file content.html will go in the content frame; all content pages in the site need a link to menu.css and menus.js.

```

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Content</title>
    <meta http-equiv="Content-Type"
      + content="text/html;
      + charset=iso-8859-1" />
    <link href="menu.css"
      + type="text/css"
      + rel="stylesheet" media="all" />
  </head>
  <body>
    <script src="menus.js"></script>
    <div id="pageContent">
      <h1>CHAPTER V</h1>
      <p>Wool and Water<br />
    </body>
</html>

```

Code 6.11 The frameset file index.html is made up of a thin frame at the top to contain menu.html and a large frame beneath for content.

```

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>DHTML & CSS Advanced I
      Adding a Fixed Drop-down Menu</title>
    <meta http-equiv="Content-Type"
      + content="text/html;
      + charset=iso-8859-1" />
  </head>
  <frameset rows="20,*" cols="*" border="0"
    + frameborder="NO" framespacing="0">
    <frame name="menu" src="menu.html"
      + noresize="noresize" scrolling="NO" />
    <frame name="content"
      + src="content.html" />
  </frameset>
</html>

```



Figure 6.25 The frameset with the thin menu frame above the content frame, which fills the rest of the browser window.

24. `index.html`

Create and save the file `index.html` (Code 6.11). This is your frameset file used to set up the thin menu frame over top of the content frame (Figure 6.25). When the user places their mouse over a menu header in the top (menu) frame the drop-down menu appears in the content frame underneath (Figure 6.26), regardless of how the page scrolls (Figure 6.27).

✓ Tip

Notice that we do not include the XHTML document type definition (DTD) in the content documents (content.html). Including the XHTML DTD interferes with determining the scroll height in the frame.



Figure 6.26 The drop-down menu appears over other content when the visitor mouses over the menu header link.

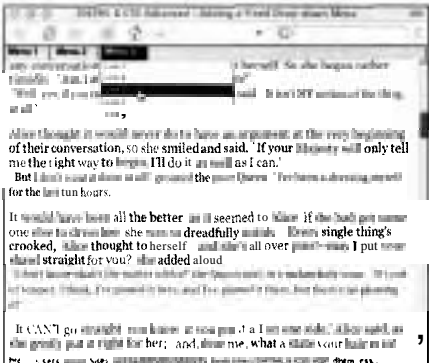


Figure 6.27 Even after the visitor scrolls down, the menu head is still at the top of the page and the menus will appear underneath.

Adding a Floating Menu Bar

As with the frames-based menu bar presented in the previous section, the floating menu bar allows you to maintain a menu bar at the top of the screen, but offers a couple of advantages:

- ◆ You don't have to worry about the issues inherent in using frames, such as book-marking problems.
- ◆ You can have the menu bar initially positioned under a header element, but then at the very top of the screen as the user scrolls down.

To add a floating menu:

1. function findScrollTop() {...}

Add the function `findScrollTop()` to your JavaScript (**Code 6.12**). This function will use whichever method works in the visitor's browser to find the current scroll position of the page and returns that value.

Code 6.12 The floating menu bar works by detecting when the visitor has scrolled, and then repositioning the absolutely positioned menu bar so that it is always at the top of the browser window.

```
Code
<html xml:lang="en" xmlns="http://www.w3.org/
  = 1999/xhtml">
  <!--
    <meta http-equiv="content-type"
      = content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced |
      Adding a Floating Menu</title>
    <script language="javascript"
      = type="text/javascript">
      function findScrollTop() {
        if (window.pageYOffset != null)
          return window.pageYOffset;
        if (document.body.scrollHeight != null)
          return document.body.scrollTop;
        return (null);
      }
      function moveMenuBar() {
        var object=document.getElementById
          ('menuBar');
        var y = findScrollTop();
        if (y <= 75) y = 75;
        object.style.display = 'none';
        object.style.top = y + 'px';
        object.style.display = 'block';
      }
    </script>
    <style type="text/css"><!--
body {
  margin: 0;
  padding: 0;
}
h1 {
  font-size: 1.5em;
  background-color: #dedede;
  margin: 0;
  padding: 25px;
  width: 590px;
}
#Content {
  margin-top: 50px;
  padding: 25px;
  width: 640px;
}

```

(code continues on next page)

Code 6.12 continued

```
Code
}
#menuBar {
  font-size: 0.9em;
  background-color: #d3d3d3;
  background-image: url(bg_menu.png);
  background-repeat: repeat-x;
  position: absolute;
  top: 75px;
  left: 0;
  width: 640px;
  height: 25px;
  border: solid 1px #000;
}
.menuInside {
  display: block;
  padding: 5px;
}
--></style>
</head>
<body onscroll="moveMenuBar()">
  <div id="menuBar">
    <span class="menuInside">
      <a href="#">&lt; Previous
      </a>
      <a href="#"> Table of Contents</a> |
      <a href="#">Next Chapter&gt;</a>
    </span>
  </div>
  <h1>Through the Looking Glass</h1>
  <div id="content">
    <h2>CHAPTER VI<br />
    Humpty Dumpty</h2>
    However, the egg
    only got larger and larger, and
    more and more human...
  </div>
</body>
</html>
```

2. function moveMenuBar() {...)

Add the function `moveMenuBar()` to your JavaScript. This function uses the results of the `findScrollTop()` function to move the object with the ID `menuBar` to the top of the browser window, allowing for a 75-pixel header if the page is scrolled to the top.

To help alleviate some of the flashing often seen with this technique as the menu bar moves up and down, the menu is hidden before moving and then redisplayed after movement.

3. body {...)

Since the floating menu bar relies on exact positioning, you need to set the padding and margin for the page to 0.

continues on next page

4. #menuBar {...}

Add the ID `menuBar` to your CSS. This sets the styles for the menu bar. In this example, I created a simple gradient graphic (**Figure 6.28**) that is placed as the background for the menu bar. Also, offset the menu bar 75 pixels from the top so it doesn't overlap the header created using the `<h1>` tag.



Figure 6.28 The gradient background helps set the menu off from the rest of the page.

5. .menuInside {...}

Add the class `menuInside` to your CSS. This class is used inside the `menuBar` ID to define the padding for the content. In theory, we should be able to simply set padding in the `menuBar` ID, but remember that Internet Explorer has problems with padding (see Chapter 2). This is an alternative solution to using the box model hack.



Figure 6.29 When the page first loads, the menu bar is placed below the header.

6. onscroll="moveMenuBar()"

Add an event handler to the `<body>` tag so that every time the page is scrolled, the `moveMenuBar()` function is triggered.

7. <div id="menuBar">...</div>

Set up the menu bar in your HTML page using the `menuBar` ID. When the page first loads, the menu bar appears 75 pixels from the top (**Figure 6.29**). After the visitor scrolls down, the menu bar will stick like glue to the top of the browser window (**Figure 6.30**).



Figure 6.30 After the visitor scrolls, the menu bar will be pushed to the top of the screen.

✓ Tip

The menu bar may behave differently in different browsers. Some browsers will smoothly scroll the menu bar, but others will scroll it in a choppy motion or may even make it disappear during scrolling and then reappear when scrolling has stopped. In fact, Internet Explorer for Mac has problems redrawing the menu and it may look smeared across the page.

Code 6.13 All of the functions that make the clipping menu function are in the `clipMenu.js` file so they can be shared from page to page. For these functions to work, though, you'll also need to have the `cookieLibrary.js` file (described in Chapter 3) in the same directory.

```
var articleRef = new Array ();
var articleTitle = new Array ();
var articleRead = new Array ();
var articleNum = -1;
var emptyMessage = 'To add articles, click
the <span class="clipLinkStyle"
style="color:#00f">+</span> next to the
article title<br />';
function getClipMenu() {
    articleNum = getCookie('articleNum');
    if (!articleNum) articleNum = -1;
    var object=document.getElementById
    ('clipMenu');
    if (articleNum >= 0) {
        for (var i = 0; i <=
        articleNum; i++) {
            var getArticleTitle =
            'articleTitle' + i;
            var getArticleRef =
            'articleRef' + i;
            var getArticleRead =
            'articleRead' + i;
            articleTitle[i] = getCookie
            (getArticleTitle);
            articleRef[i] = getCookie
            (getArticleRef);
            articleRead[i] = getCookie
            (getArticleRead);
            var objectArtRef = document.
            getElementById(articleRef[i]);
            if (objectArtRef) {
                objectArtRef.innerHTML =
                '<span class=
                "clipLinkStyle">+</span>';
                articleRead[i] = 1;
            }
        }
    }
    linkClass = "article"
    if (articleRead[i] == 1)
        linkClass = "articleRead";
```

(code continues on nextpage)

Adding a Clipping Menu

Although all Web browsers worth their salt provide some way to bookmark Web pages (sometimes called favorites) these approaches can be cumbersome—especially if you just want to bookmark a page until you've had a chance to read it. Inevitably, the bookmark folder fills with links that you looked at once and may never want to see again.

Consider a newspaper or magazine site with a home page full of links to dozens of articles. Most such Web sites will require you to click the link for the article you want to read, and once you're done reading the article, click the Back button or Home button to return to the table of contents.

One way to make life easier for weary readers is to allow them to create a list of article clippings, which are stored in a menu in your interface. These clippings allow users to create a customized list of Web pages within your site they want to visit, and then use a convenient menu to visit each one.

To add a clipping menu:

1. `clipMenu.js`

Create and save a new external JavaScript file called `clipMenu.js` (**Code 6.13**).

This file will hold the functions needed to retrieve, gather, and store our clipping menu using arrays and cookies. The functions in this file will rely on the external JavaScript file `cookieLibrary.js` (Chapter 3), which has to be loaded in the same page as this file (see step 15).

Steps 2–9 apply to this file.

continues on page 206

```

        object.innerHTML += '<a href="' + articleRef[i] + '.html" class="' + linkClass + '">' +
            articleTitle[i] + '</a>';

```

```
    }

```

```
    }

```

```
    else object.innerHTML = emptyfmessage;

```

```
    }

```

```
function clipLink(newArticleRef) {

```

```
    articleNum++;

```

```
    var objectArtTitle = document.getElementById(newArticleRef + 'Title');

```

```
    var newArticleTitle = objectArtTitle.innerHTML;

```

```
    var object=document.getElementById('clipMenu');

```

```
    if (articleNum == 0) object.innerHTML = '';

```

```
    object.innerHTML+= '<a href="' + newArticleRef + '.html" class="article">' + newArticleTitle +
        '</a><br />' + '<br />';

```

```
    articleTitle[articleNum] = newArticleTitle;

```

```
    articleRef[articleNum] = newArticleRef;

```

```
    articleRead[articleNum] = 0;

```

```
    var objectArtRef=document.getElementById(newArticleRef);

```

```
    if (objectArtRef) objectArtRef.innerHTML = '<span class="clipLinkStyle">*</span>';

```

```
    }

```

```
function clearClipMenu() {

```

```
    for (var i = 0; i <= (articleRef.length-1); i++) {

```

```
        if (articleRef[i] != null) {

```

```
            var objectArtRef=document.getElementById(articleRef[i]);

```

```
            if (objectArtRef) objectArtRef.innerHTML = '<a class="clipLinkStyle" href="javascript:
                clipLink(\"' + articleRef [i] + '\")>+</a>';

```

```
        }

```

```
    }

```

```
    var object=document.getElementById('clipMenu');

```

```
    object.innerHTML = emptyfmessage;

```

```
    articleLink = new Array ();

```

```
    articleRef = new Array ();

```

```
    articleRead = new Array ();

```

```
    articleNum = -1;

```

```
    }

```

```
function clearClipMenuRead() {

```

```
    var object=document.getElementById('clipMenu');

```

```
    object.innerHTML = '';

```

```
    for (var i = 0; i <= (articleRef.length-1); i++) {

```

```
        if (articleRead[i] == 0) {

```

(code continues on next page)

Code 6.13 continued

```
object.innerHTML += '<a href="" + articleRef[i] + ".html" class="article">' + articleTitle[i]
    + '</a><br />';
}
else {
    if (articleRef[i] != 0) {
        var objectArtRef = document.getElementById(articleRef[i]);
        if (objectArtRef.innerHTML = '<a class="clipLinkStyle" href="javascript:
            + clipLink('\'' + articleRef [i] + '\''></a>');
            articleRef[i] = -1;
            articleNum = articleNum - 1;
        }
    }
}
if (articleNum <= 0) object.innerHTML = emptyMessage;
}

function saveClipMenu() {
    var articleNum=-1;
    for (var i = 0; i <= (articleRef.length-1); i++) {
        if (articleRef[i] != -1) {
            articleNum++;
            saveArticleTitle = 'articleTitle' + articleNum;
            saveArticleRef = 'articleRef' + articleNum;
            saveArticleRead = 'articleRead' + articleNum;
            setCookie(saveArticleTitle,articleTitle[i]);
            setCookie(saveArticleRef,articleRef[i]);
            setCookie(saveArticleRead,articleRead[i]);
        }
    }
    setCookie('articleNum',articleNum);
}
```

```
2. var articleRef = new Array ();  
   var articleTitle = new Array ();  
   var articleRead = new Array ();
```

Initialize three separate arrays:

`articleRef[]` stores the reference for the article, which is used to create the URL.

`articleTitle[]` stores the title of the article as a text string.

`articleRead[]` stores a value of `1` if the article has been read or `0` if the article has not been read.

```
3. var articleNum = -1;
```

Initialize the variable `articleNum`, which is used by the arrays in step 2 to keep track of the number of articles in the clipping menu. Since we are dealing with arrays, the initial value for `articleNum` is `-1`, meaning that the array is empty

```
4. var emptyMessage='...';
```

Initialize a variable with a text string to use if there are no articles in the clipping menu.

```
5. function getClipMenu() {...}
```

Add the function `getClipMenu()` to your JavaScript. This function is used to retrieve and display the clipping menu from cookies. The function begins by querying the cookie named `articleNum` to find the number of articles recorded (if any) and then loops through to record each article reference, article title, and whether the article has already been read, recording this information in the three arrays from step 2.

The function then checks whether the currently displayed article is in the clipping menu, and disables the clip icon next to the article title if it is.

Finally, the function adds a link to the clipping menu using different classes to style the article depending on whether it has been read already or not. If there are no articles in the clipping menu, then the empty message is displayed in the menu.

```
6. function clipLink(newArticleRef) {...}
```

Add the function `clipLink()` to your JavaScript. This function will add a selected link to the clipping menu. The function first increases the `articleNum` variable by one before finding the article title and reference and adding them to the appropriate arrays, sets the article's read state to unread (`0`), adds the new link to the clipping menu, and then disables the clip control next to the article title.

```
7. function clearClipMenu() {...}
```



Add the function `clearClipMenu()` to your JavaScript. This function resets the clipping menu to its empty state. First it resets the clip control next to article titles and replaces any links in the menu with the empty message, then it clears all of the arrays and sets the number of articles back to `-1`.

```
8. function clearClipMenuRead() {...}
```

Add the function `clearClipMenuRead()` to your JavaScript. This function works much like the `clearClipMenu()` function, but only removes article links that have already been read from the list, leaving other links in place. The function first clears the clipping menu, then runs back through the list. If an article's `articleRead` value is `1` (read) then it is skipped and its `articleRef` is set to `-1`. Otherwise, the link is placed back in the array and written back into the clipping menu. If all articles are cleared, then the empty message is written into the clipping menu.

Code 6.14 The external CSS file `clipMenu.css` describes how the clip menu and controls should be displayed.

```
Code
.menu {
    background-color: #ccc;
    width: 200px;
    padding: 10px;
    float: left;
}
.content {
    background-color: #ecec;
    padding: 10px;
    width: 340px;
    float: left;
}
a.article {
    color: #f00;
    display: block;
    padding-bottom: 10px;
}
a.articleRead {
    color: #900000;
    display: block;
    padding-bottom: 10px;
}
.clipLinkStyle {
    color: #c3c3ae;
    font-size: 24px;
    font-family: webdings;
}
a.clipLinkStyle {
    color: blue;
    text-decoration: none;
    position: relative;
    width: 15px;
}
```

9. `function saveClipMenu() {...}`
Add the function `saveClipMenu()` to your JavaScript. This function loops through all of the values in the arrays, writing them to a sequence of cookies, from which it can then be retrieved by the function `getClipMenu()`. If an article reference has been set to `-1` (cleared) then it is skipped. Finally, the function will place the number of articles in the clipping menu in the cookie called `articleNum`.
10. `clipMenu.css`
Create and save the external CSS file `clipMenu.css` (**Code 6.14**). This file will hold all of the styles used to create the clipping menu and clip controls. Steps 11–13 apply to this file.
11. `a.article {...}`
`a.articleRead {...}`
Add the classes `article` and `articleRead` to be used with the link tag. These classes are used to display the links in the clipping menu.
12. `.clipLinkStyle {...}`
Add the class `clipLinkStyle` to your CSS. This is the general definition for the clip controls. The most important thing to notice here is that I've set the font as `Webdings`. This will allow us to use specific icons without having to resort to graphics. The `†` sign will look like  for articles that can be clipped, and the `*` will look like  for articles that have already been clipped. I'm using `Webdings` because it's a browser-safefont.

continues on next page

13. `a.clipLinkStyle {...}`

For links, you'll want to change the `clipLinkStyle` slightly to use a blue color and no underlining.

14. `index.html`

Create and save the file `index.html` (**Code 6.15**). This file will act as a table of contents for the articles in our little example site. You can apply this to other pages within the site to carry the clipping menu from page to page.

Steps 15–23 apply to this page.

15. `src="cookieLibrary.js"`

Add a `<script>` tag with `cookieLibrary.js` as its source. You'll need to copy the file (shown in Chapter 3) to the current directory. This file contains functions needed to retrieve and set cookies that will be used by functions in `clipMenu.js`.

Code 6.15 The clipping menu in `index.html` begins life as an empty layer, but it is dynamically filled and can then be added to using the controls next to each article title.

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Transitional//EN" "http://www.w3.org/TR/
  xhtml1/DTD/xhtml1-transitional.dtd">
<html xml:lang="en" xmlns="http://www.w3.org/
  1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced !
      Creating a Clipping Menu</title>
    <script type="text/javascript"
      language="javascript"
      src="cookieLibrary.js"></script>
    <script type="text/javascript"
      language="javascript"
      src="clipMenu.js"></script>
    <link href=" ../default.css"
      rel="stylesheet" type="text/css"
      media="all"/>
    <link href="clipMenu.css"
      rel="stylesheet" type="text/css"
      media="all"/>
  </head>
  <body onload="getClipMenu()"
    onunload="saveClipMenu()">
    <div class="menu">
      <h3>Clip Menu</h3>
      <div id="clipMenu">
<!-- Dynamically Filled -->
      </div>
      <br />
      <a href="javascript:
        clearClipMenu()">Clear
        Menu</a><br />
      <a href="javascript:
        clearClipMenuRead()">Clear
        Articles You've Read</a>
    </div>
    <div class="content">
    <h1>THROUGH THE LOOKING-GLASS</h1>

```

(code continues on next page)

```

<h2>by LEWIS CARROLL</h2>
<h3>Table of Contents</h3>
<span id="a1"><a class=
  = "clipLinkStyle" href="javascript:
  clipLink('a1')"></a></span>
<a href="a1.html" id="a1Title"
  class="articleTitle">Chapter 1
  = - Looking-Glass house</a>
<br />
<span id="a2"><a class=
  = "clipLinkStyle" href="javascript:
  clipLink('a2')"></a></span>
<a href="a2.html" id="a2Title"
  class="articleTitle">Chapter 2
  = - The Garden of Live Flowers</a>
<br />
<span id="a3"><a class=
  = "clipLinkStyle" href="javascript:
  clipLink('a3')"></a></span>
<a href="a3.html" id="a3Title"
  class="articleTitle">Chapter 3
  = - Looking-Glass Insects</a>
<br />
</div>
</body>
</html>

```

16. `src="clipMenu.js"`

Add another `<script>` tag with `clipMenu.js` as its source. This will import the JavaScript from steps 1–9.

17. `href="clipMenu.css"`

Add a link tag to import `clipMenu.css`, the external style sheet created in step 10.

18. `onload="getClipMenu()" onunload="saveClipMenu()"`

Add event handlers to the body to run the `getClipMenu()` function when the page loads and the `saveClipMenu()` function when the page is unloaded.

19. `<div id="clipMenu">...</div>`

Add an empty layer with the ID `clipMenu`. This layer will have its content filled in by the `getClipMenu()` function either with clipped links or with the empty message (**Figure 6.31**).

continues on next page



Figure 6.31 When the page first loads, the clipping menu is empty, ready to be filled by clicking the icon next to an article title.

20. `<a href="javascript:`

`clearClipMenu()">`

``
`
`

Add a link for the `clearClipMenu()` function. Clicking this link will remove all links from the clipping menu and display the empty message (Figures 6.32 and 6.33).

21. `<a href="javascript:`

`clearClipMenuRead()">`

``
`Clear Articles You've Read`

Add a link to the `clearClipMenuRead()` function. When clicked, this will remove all articles from the menu that have already been read or at least been loaded (Figures 6.34 and 6.35).



Figure 6.32 The clipping menu has a couple articles listed.



Figure 6.33 Clicking the Clear Menu link resets the menu to empty.



Figure 6.34 The first and third clipped articles have already been visited.



Figure 6.35 Clicking the Clear Articles You've Read link will clear all but the second clipping option since it's the only one that hasn't been read.

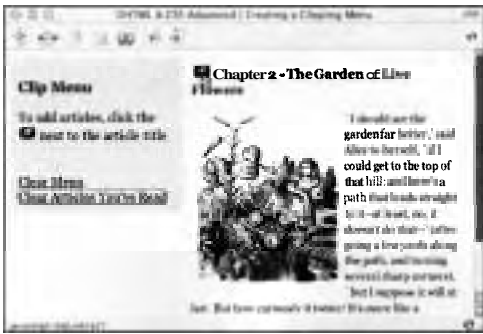


Figure 6.36 The icon next to the article title can be clicked...



Figure 6.37 ...to add the article to the clipping menu.

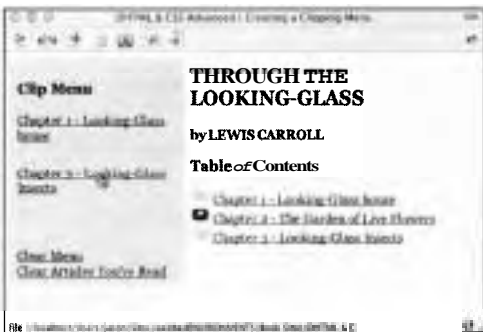


Figure 6.38 The visitor can now choose options from the clipping menu to navigate the site.

22. ``
`<a class="clipLinkStyle" href=`
`· "javascript:clipLink('a1')">+`
``

Add a link in front of each article title to trigger the `clipLink()` function, passing it the reference for the article to be clipped, wrapped inside a `` tag with a unique ID for the clipped article. Clicking this link will add the article to the clipped menu (Figures 6.36 and 6.37).

23. `<a href="a1.html" id="a1Title"`
`· class="articleTitle">...`

Add a text link to the article, including an ID combining the clip ID with `Title`. When this link is clicked, the new page is loaded, but the clipping menu follows along (Figures 6.38 and 6.39).

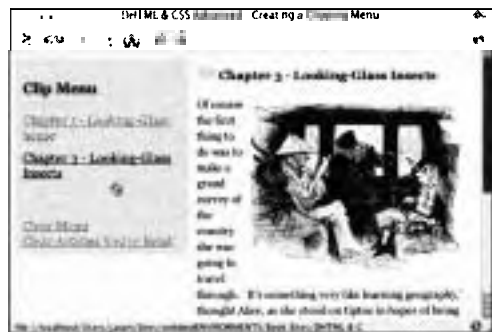


Figure 6.39 Regardless of which page they navigate to, though, every visitor will have their customized menu.

Adding a JumpMenu

"Quick and dirty" is the name of the game with jump menus, allowing you to quickly set up a drop-down menu using the form's select menu. To some Web designers this is a hack—using form code for purposes for which it was never intended. Nevertheless these types of menus work and allow you to put multiple links into a small area.

To add a jump menu:

1. jumpLibrary.js

Create and save the file `jumpLibrary.js` (Code 6.16). Step 2 applies to this file.

2. function jumpTo(evt) {...}

Add the function `jumpTo()` to the file. This function looks at the value associated with the form element that triggered the function (which should be either a relative or absolute URL) and uses that to refresh the browser to the new Web page.

3. jumpMenu.js

Create and save the file `jumpMenu.js` (Code 6.17). This file is used to write the code used for the jump menu into each page that needs it (see "Adding External Content with JavaScript" in Chapter 4). Steps 4-7 apply to this file.

4. var writeMenu = '';

Add the variable `writeMenu` to the file. This variable will be used to store all of the HTML code used to write the menu.

Code 6.16 The external JavaScript file `jumpLibrary.js` provides the code that makes the jump menu work.

```
Code
function jumpTo(evt) {
    evt = (evt) ? evt : ((event ? event : null));
    if (evt) {
        var object = (evt.target) ? evt.target :
            ((evt.srcElement) ? evt.srcElement :
            null);
        if (object && object.tagName.toLowerCase()
            == 'select' && object.value) {
            location.href = object.value;
        }
    }
}
```

Code 6.17 The HTML code that creates the menu is in the external JavaScript file `jumpMenu.js` so that it can be shared by different files easily.

```
Code
var writeMenu = '';
writeMenu += '<form id="jumpMenu"
    name="jumpMenu">';
writeMenu += '<select name="selectName"
    size="1" onchange="jumpTo(event)">';
writeMenu += '<option selected="selected">
    Table of Contents</option>';
writeMenu += '<option value="index.html">
    Chapter 1</option>';
writeMenu += '<option value="chapter2.html">
    Chapter 2</option>';
writeMenu += '<option value="chapter3.html">
    Chapter 3</option>';
writeMenu += '</select>';
writeMenu += '</form>';
document.write(writeMenu);
```

Code 6.18 Link to the `jumpLibrary.js` and `jumpMenu.js` files to put the jump menu into any Web page, such as `index.html`, shown here.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Transitional//EN" "http://www.w3.org/TR/
  xhtml1/DTD/xhtml1-transitional.dtd">
<html xml:lang="en" xmlns="http://www.w3.org/
  1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced |
      Adding a Jump Menu | Page 1</title>
    <script src="jumpLibrary.js" language=
      "javascript" type="text/
      javascript"></script>
  </head>
  <body>
    <script src="jumpMenu.js" language=
      "javascript" type="text/
      javascript"></script>
    <br />
    <br />
    <h1>CHAPTER I</h1>
    <p><b>Looking-Glass house</b></p>
    One thing was certain,
      that the WHITE kitten had had nothing
      to do with it...</p>
  </body>
</html>
```

5. `writeMenu += '<form id="jumpMenu" name="jumpMenu" onchange="jumpTo(event)">'`

Add the `<form>` tag to the `writeMenu` variable using the name/ID `jumpMenu` and an `onchange` event handler to trigger the `jumpTo()` function.

6. `writeMenu += '<option value="index.html">Chapter 1</option>'`

Add form options for each of the menu options you want in the jump menu, including the URL as the value for each option. The URL can be either relative or absolute.

7. `document.write(writeMenu);`

Add a `document.write()` to add the code collected in the `writeMenu` function to the page. This will write the HTML so that the browser treats it as a part of the Web page.

8. `index.html`

Create and save a file called `index.html` (**Code 6.18**). You can place the jump menu into any HTML page you desire, so consider this a template for other pages that need the jump menu. Steps 9 and 10 apply to this file.

9. `src="jumpLibrary.js"`

In the head of your document, add a `<script>` tag with the source pointing to `jumpLibrary.js`.

continues on next page

10. src="jumpMenu.js"

In the body of your Web page, at the position where you want your menu to appear, add a `<script>` tag with the source pointing to `jumpMenu.js`. The jump menu will be placed in that position when the page is loaded into a browser (**Figure 6.40**). When the visitor selects an option from the menu (**Figure 6.41**) the browser will load the selected page (**Figure 6.42**).

✓ Tip

One obvious disadvantage to this method is that if you use the browser Back button, the menu will still show the last page selected. Choosing that page again does not trigger the change event, however, so the page will not be changed.



Figure 6.40 The jump menu is a compact way to provide quick navigation for the visitor.



Figure 6.41 The visitor clicks the menu to see and choose their options.

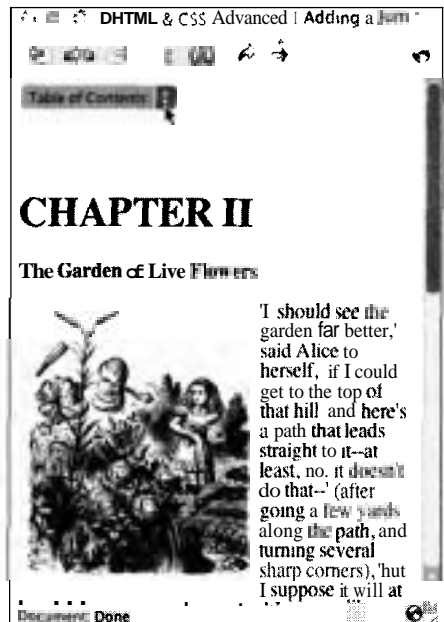


Figure 6.42 After the visitor selects the menu option, the new page loads.

Adding Pop-up Menus

Sometimes called contextual menus, pop-up menus allow you to place a list of links anywhere on the page, associated with a link. Rather than simply being placed under a menu, pop-up menus will appear around a link. Generally, they'll appear below and slightly to the right of the link, but if there isn't enough room, the pop-up will position itself so that it fits in the window without forcing the user to scroll to view it.

To add a pop-up menu:

1. `var objPopUp = null;`

In your JavaScript in the head of the page, initialize the variable `objPopUp` to null (**Code 6.19**). This variable will be used to record which object is currently popping up.

continues on page 218

Code 6.19 The pop-up menu works by showing a hidden layer with links on it and positioning that layer next to where the mouse clicked. It will then disappear the next time the visitor clicks in the browser window.

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>DHTML & CSS Advanced ! Adding a Pop-up Menu</title>
    <script type="text/JavaScript" language="javascript"><!--
      var objPopUp = null;
      var popMenuOn = 0;
      function popUp(evt,objectID) {
        if (popMenuOn == 1) popHide();
        document.onclick = popHide;
        var evt = (evt) ? evt : ((window.event) ? event : null);
        objPopUp = document.getElementById(objectID);
        xPos = evt.clientX;
        yPos = evt.clientY;
        if (xPos + objPopUp.offsetWidth >= document.body.clientWidth) xPos =
          xPos - objPopUp.offsetWidth;
```

(code continues on next page)

```

if (yPos + objPopUp.offsetHeight > document.body.clientHeight) yPos =
    yPos - objPopUp.offsetHeight;
objPopUp.style.left = xPos + 'px';
objPopUp.style.top = yPos + 'px';
objPopUp.style.visibility = 'visible';
}
function popHide() {
    if (popMenuOn == 0) {
        popMenuOn = 1;
        return
    }
    else {
        objPopUp.style.visibility = 'hidden';
        objPopUp = null;
        popMenuOn = 0;
        document.onclick = null;
    }
}
//--></script>
<style type="text/css" media="screen"><!--
body {
    margin: 0px;
    padding: 0px;
}
p {
    padding: 10px;
}
a:link {
    white-space: nowrap;
}
.popUp {
    font-size: 10px;
    font-family: Verdana, Arial, Helvetica, sans-serif;
    background-color: #fff;
    visibility: hidden;
    position: absolute;
    width: 125px;
    border: solid 1px black;
}
.popUpLink {
    color: #f00;

```

Code 6.19 *continued*

```

    cursor: nw-resize;
}
a.menuLink {
    color: #f00;
    display: block;
    text-decoration: none;
    padding: 3px;
    border-bottom: 1px solid #ccc;
}
a.menuLink:hover {
    background-color: #ccc;
}
--></style>
</head>
<body>
<p>'But it certainly WAS funny,' (Alice said afterwards, when she was telling her sister the history of
all this,) 'to find myself singing "<span id="pop1" class="popUpLink" onclick="popUp(event,'popUp1')">
HERE WE GO ROUND THE MULBERRY BUSH."</span> I don't know when I began it, but somehow I felt as if I'd
been singing it a long long time!'</p>
<p>The other two dancers were fat, and very soon out of breath. 'Four times round is enough for one
dance,' <span id="pop2" class="popUpLink" onclick="popUp(event,'popUp2')">Tweedledum panted out,
</span> and they left off dancing as suddenly as they had begun: the music stopped at the same moment.
</p>
<div id="popUp1" class="popUp">
  <a href="#" class="menuLink" onclick="popHide()">Link 1</a>
  <a href="#" class="menuLink" onclick="popHide()">Link 2</a>
  <a href="#" class="menuLink" onclick="popHide()">Link 3</a>
</div>
<div id="popUp2" class="popUp">
  <a href="#" class="menuLink" onclick="popHide()">Link 4</a>
  <a href="#" class="menuLink" onclick="popHide()">Link 5</a>
  <a href="#" class="menuLink" onclick="popHide()">Link 6</a>
</div>
</body>
</html>

```


2. `var popMenuOn = 0;`

Initialize the variable `popMenuOn` to 0. This variable is used to record whether the pop-up menu is currently showing or not.

3. `function popUp(evt,objectID) {...}`

Add the function `popup()` to your JavaScript. This function shows the pop-up menu close to where the visitor clicked a link. The function hides any other pop-up menus currently showing, sets a global event handler to hide the menu if the visitor clicks anywhere in the browser window, calculates the position where the click event took place, and positions and shows the menu, offsetting it if it won't fully fit in the browser window without scrolling.

4. `function popHide() {...}`

Add the function `popHide()` to your JavaScript. This function hides the currently displayed pop-up menu and sets the global click event to null so that the user can again click in the screen.

5. `body {...}`

Add definitions in the `<body>` tag to set the padding and margin for the document to 0. Remember, we're positioning the pop-up menus using absolute positioning, but not all browsers agree on exactly where the edge of the screen is. Setting the margin and padding to 0 helps level the playing field.

6. `a:link {...}`

Add a definition for the link tag to your CSS and set the `white-space` property so that text in links will not wrap. This is optional, but Internet Explorer has difficulty with a pop-up that break across two lines of text. Setting links not to wrap will prevent this.

7. `.popup {...}`

Add the class `popup` to your CSS. This class is used to control the general appearance of the pop-up menu. The exact design is up to you, but you'll need to set `visibility` to `hidden` and `position` to `absolute`.

8. `.popupLink {...}`

Add the class `popupLink` to your CSS. This class sets the appearance of the link used to trigger the pop-up menu.

9. `onclick="popUp(event, 'popUp1')"`

Using `` tags, add `onclick` event handlers to your content that will trigger the `popUp()` function, passing it the event object and the unique ID for the pop-up menu you want to show.

10. `<div id="popUp1" class="popup">...</div>`

Use `<div>` tags to set up layers using the `popup` class and a unique ID. These layers will contain the menu options.

11. `...`

Add links within the layer created in the previous step that will be displayed as part of the menu. Whenever the visitor clicks a link in the content, the pop-up menu appears near the mouse pointer, allowing the visitor to choose from the list of links (**Figure 6.43**). If the pop-up would be below the bottom edge of the browser window or off the screen to the right, the pop-up will move itself to appear above and/or to the left of the triggering link (**Figure 6.44**).

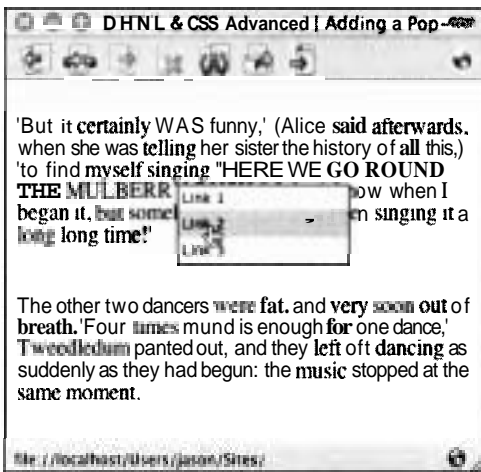


Figure 6.43 The pop-up menu appears wherever the visitor needs it on the screen, allowing you to add multiple navigation links to a single link.

Tip

One drawback to this technique, or any technique that shows and hides layers, is that in many browsers, layers will always appear below form fields regardless of what z-index has been set. The only workaround for this (other than just never allowing your pop-up layers to appear over form elements) is to have form elements hidden whenever a pop-up layer appears.

Figure 6.44 If there isn't enough room to display the menu below and to the right of the mouseclick, the menu adjusts itself above and to the left.

Educating the Browser

When you get right down to it, computers and browsers are stupid. All they know is what we tell them. A Web browser has no idea what a Web page is about, what came before it, and what comes next.

Meta data can help fill in some of the blanks, providing information such as copyright data, keywords, and descriptions. The `<meta>` tag goes in the head of your document and is used to describe the nature of the Web page. Some of these tags will be used directly by the browser if the visitor views information about the HTML file. However, metadata is most frequently used by search engines to help categorize Web pages.

Link relationships tell the browser about the current page's relationship to other pages within a Web site. These link relationships are the same `<link>` tags you use to import a style sheet. The browser, in turn, can use this information to set up site navigation outside of what appears on the actual Web page.

Unfortunately, only iCab, a relatively obscure Mac browser (www.icab.de), supports this use of the `<link>` tag (Figure 6.45). But there is always the future.

To set up a `<link>` tag:

- ◆ `<link rel="home" href="index.html">`

In the head of your document, include the `<link>` tag. Include a `rel` (relationship) from the list in Table 6.1 in quotes; then list the `href` (URL) to which this link should point (Code 6.20).

continues on page 222



Figure 6.45 iCab (Mac) has link controls that take their directions right from your Web page, but only if you set up those relationships. The links include home, start of site, previous page, next page, end of site, index, help, copyright, and contact information.

Table 6.1

Important Link Relationships	
REL	WHERE IT SHOULD LINK
home	Home page
start	First page of the Web site
next	Following page of the Web site
prev	Preceding page of the Web site
contents	Table of contents or site map
index	Table of contents or site map
glossary	Glossary of terms
help	Help page
chapter	Beginning of a chapter
section	Beginning of a section
subsection	Beginning of a subsection
appendix	Appendices
copyright	Copyright information
made	About page or <code>mailto:</code> link for the author

Code 6.20 The <meta> and <link> tags tell the browser what this page is about and describe its relationship to other pages on the Web site.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
  xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
    <title>DHML for the WWW | Using Meta & Link Tags</title>
    <meta name="Author" content="Jason Cranford Teague" />
    <meta name="copyright" content="1998-2000 Jason Cranford Teague" />
    <meta name="keywords" content="Jason Cranford Teague, Webbed Environments, DHML, JavaScript,
      HTML, Search Engines, Information Architecture, World Wide Web, Web Art, GIF, JPEG, PNG,
      Macromedia Flash" />
    <meta name="description" content="In the Future Everyone will be a Web Designer." />
    <link href="index.html" rel="home" />
    <link href="index.html" rel="index" />
    <link href="index.html" rel="first" />
    <link href="copyright.html" rel="last" />
    <link href="siteMap.html" rel="next" />
    <link href="index.html" rel="prev" />
    <link href="glossary.html" rel="glossary" />
    <link href="index.html" rel="chapter" />
    <link href="index.html" rel="section" />
    <link href="index.html" rel="subsection" />
    <link href="appendix.html" rel="appendix" />
    <link href="help.html" rel="help" />
    <link href="copyright.html" rel="copyright" />
    <link href="mailto:question@webbedenvironments.com" rel="made" />
    <style type="text/css" media="screen">!--
h1 {
  color: silver;
  font-size: 36px;
  font-family: Arial, Helvetica, Geneva, Sans-Serif;
}
--></style>
  </head>
  <body>
    <h1>Home</h1>
    <p>Et quid erat, quod me delectabat..
  </body>
</html>
```

To set up a <meta> tag:

```
<meta name="Author" content="Jason  
Cranford Teague">
```

In the head of the document, include the <meta> tag with the name of the content you are providing, as listed in **Table 6.2**, followed by that content in quotes. The name defines the type of content being provided.

✓ Tips

The major browsers may never embrace <link> tags, but I still think they're a neat idea. Are they worth adding to all your old Web pages? No. But if you have a template for your Web site, you may want to include these tags in the future.

Keep in mind that these tags can link to anything. The Web police will not come knocking down your door if you link the *home* relationship to *help.html*, but you may end up with very confused visitors.

Despite popular belief, <meta> tags are not the magic solution to search-engine woes. In fact, not all search engines use these tags, and those that do generally consider them to be a small part of the relevancy equation. This is not to say that you should not include them or that they don't help, but don't rely on them to propel you to the top of the search-engine charts.

That said, you should always include two <meta> tags in the <head> of a document to catch a search engine's eye. The *description* is a sentence or two that tells the search engine what's on the page; the search engine may use its content to describe the page in the results. More important still is the *keywords* tag. This is a list of words, provided by the author, that are relevant to the page.

Table 6.2

Important Meta Names	
NAME	WHAT IT DESCRIBES
author	Who created these pages
copyright	Copyright information
keywords	List of words that are relevant to the subject of this page
description	Brief description of the page



CONTROLS

With straight HTML, your visitors' ability to control what happens on the screen is fairly limited. They have scrollbars, but no means of changing the content on the screen to meet their needs.

A truly dynamic Web site, however, allows visitors to interact with the pages, and affect the content. You must provide controls that permit that interaction.

In this chapter, I'll show you how to add interactive functions that give visitors greater control over the way the Web page is presented to them.

Creating Customized Browser Controls

Whenever you use JavaScript to open a new browser window, you have the option of removing the browser's controls, including the Back, Forward, Reload, and Print buttons. However, doing so limits what visitors can do, or at least what they think they can do. Even when removed, most of the controls are still available using the keyboard shortcuts, but many visitors may not know that.

Using JavaScript, though, you can add controls directly into the Web page that mimic those in the browser control bar. This gives you jurisdiction over these controls, allowing you to show or hide them as needed.

To add browser control buttons:

1. `browserControlLibrary.js`

Create and save an external JavaScript file called `browserControlLibrary.js` (**Code 7.1**). To keep your files neat, save it in a folder called `javascript` in the same directory as your other files.

Steps 2–5 apply to this file.

2. `function goBack() {...}`

Add the function `goBack()`. This function uses the `history.go()` method with a value of `-1` to go back one step in the browser's history. The exact page will depend on the previous page in the visitor's browser history.

Code 7.1 To create your own Back, Forward, Reload, and Print buttons, use the JavaScript in the external file `browserControlLibrary.js`.

```
function goBack()
{
    history.go(-1);
}
function goForward()
{
    history.go(1);
}
function reloadPage() {
    location.reload()
}
function printPage () {
    window.print()
}
```

Code 7.2 Set up styles in the external style sheet `browserButton.css` to make the controls look more button-like.

```
Code
a.browserButton {
    color: #f00;
    font-size: 12px;
    font-family: "Courier New", Courier,
Monaco, monospace;
    font-weight: bold;
    text-decoration: none;
    background-color: #c3c3c3;
    margin-right: 5px;
    padding: 2px 5px;
    border: solid 1px #333;
}
a.browserButton:hover {
    color: #c3c3c3;
    background-color: #f00;
}
```

Removing the Browser Controls

There is a lot of debate about whether you should ever remove the browser controls from a visitor's window. Some studies conclude that visitors do not like having control taken away from them. But your decision to remove browser controls should be balanced against whether the controls may cause catastrophic usability problems.

I have visited Web sites that require visitors to fill out a multiple-page application. If you click the Back button to return to a previous page in the application to make a correction, the entire process fails. You not only have to start the process over, you also have to restart your browser. This is one example where it makes sense to remove the browser controls.

3. `function goForward() {...}`
Add the function `goForward()`. This function uses the `history.go()` method with a value of 1 to go forward one step in the browser's history. If there are no URLs after this one in the visitor's browser history, this link will not work
4. `function reloadPage() {...}`
Add the function `reloadPage()`. This function uses the `reload` method to reload the current location, refreshing it to its default state. Form fields that have already been filled by the visitor, though, will maintain the last value entered.
5. `function printPage() {...}`
Add the function `printPage()`. This function uses the `print` method to open the browser's Print dialog box to print the current page in the browser window.
6. `browserButton.css`
Create and save the external CSS file `browserButton.css` that will contain styles to be used with the customized browser buttons (**Code 7.2**). To keep your files organized, save this file in a folder called `CSS`.
Step 7 applies to this style sheet.
7. `a.browserButton {...}`
Add the `browserButton` class, associated with the anchor tag, to your CSS to control the appearance of browser buttons on the Web page. You can also add styles for the other rollover states as desired.

continues on next page

8. `index.html`
Create and save the pages to which you want to add your customized browser controls. In this example, we'll create the page `index.html` (**Code 7.3**) with `page2.html`, and `page3.html` to navigate to (code not shown, but it's pretty much the same as `index.html`).

Steps 9–11 apply to this page.

9. `src="javascript/
browserControlLibrary.js`

Add a `<script>` tag that calls `browserControlLibrary.js` as its source. This will load the external JavaScript file from step 1.

10. `href="css/browserButton.css"`

Add a `<link>` tag to call the external CSS file `browserButton.css` from step 6.

Code 7.3 The controls are simply links within the page `index.html` using JavaScript function calls as the source.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
 Transitional//EN" "http://www.w3.org/TR/
 xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=
      iso-8859-1" />
    <title>DHTML & CSS Advanced I
      Creating Customized Browser
      Controls</title>
    <script type="text/javascript"
      src="javascript/browserControl
      Library.js"></script>
    <link href="css/browserButton.css"
      rel="stylesheet" type="text/css"
      media="all" />
  </head>
  <body>
    <p>
      <a class="browserButton" href=
        "javascript:goBack()">Back</a>
      <a class="browserButton" href=
        "javascript:goForward()">Next</a>
      <a class="browserButton" href=
        "javascript:reloadPage()">Reload</a>
      <a class="browserButton" href=
        "javascript:printPage()">Print</a>
    </p>
    Chapters: <a href="index.html">1</a> |
      <a href="page2.html">2</a> |
      <a href="page3.html">3</a>
    <h1>CHAPTER I</h1>
    <h2>Looking-Glass house</h2>
    One thing was certain,
      that the WHITE kitten had had nothing
      to do with it...</p>
  </body>
</html>
```

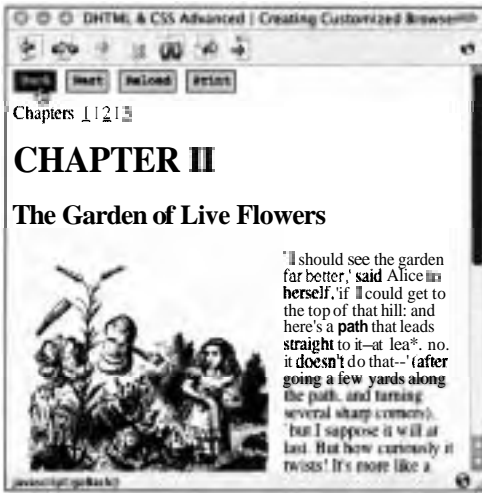


Figure 7.1 Click the Back button on page 2...

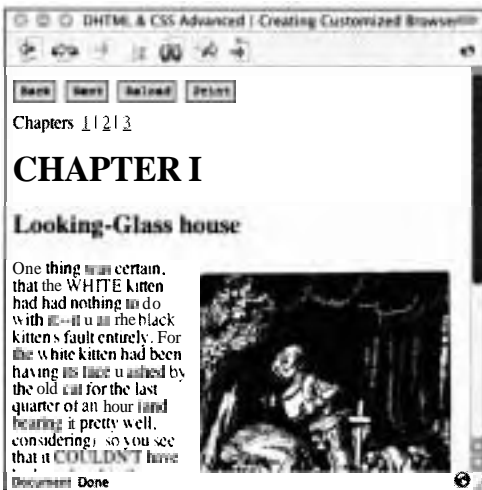


Figure 7.2 ...to return to page 1 (or wherever you came from).

11. `Back`

Add links to each HTML page on which you want to use your customized controls. The controls will appear on the page, returning control to the user (Figures 7.1 and 7.2).

✓ Tips

Although you can use the history element to move up and down in the history tree, you cannot access the actual URLs or even detect whether the tree contains URLs before or after the current one. This situation severely limits your ability to display or hide the Back and Forward links dynamically.

You could also include an external style sheet specifically for printing. To do this, link to another external style sheet as in step 10, but set the media to `print`. When printing, the browser will use this style sheet instead. One advantage of doing this is that you can set the `browserButton` class to `display: none`, preventing the control from showing up when the page is printed.

Creating a Sortable Table

Data is not knowledge, but one way people can turn information into knowledge is by sorting it various ways for comparison. Tables are one of the primary methods of presenting different kinds of data, but once loaded in a Web page, tables are static. One common way to overcome this is to allow the visitor to re-sort table data by clicking a column header.

In this example, we'll set up a three-column table using data stored in an array. When first loaded, the table will display the data as in the array (**Figure 7.3**). We will then provide controls that sort these arrays when any column's header is clicked (**Figure 7.4**).



Figure 7.3 When the page first loads, the table displays the data in the order entered.

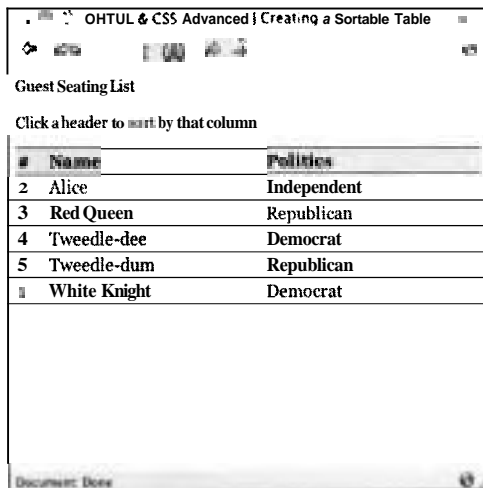


Figure 7.4 After you click the Name column header, the table is sorted in ascending alphabetical order by name.

Code 7.4 The external JavaScript file `array.js` holds the array used to record each guest's information: seating order, name, and politics.

```
var guestList = new Array();
guestList[0] = {orderNum:3,guestName:'
  'Red Queen',politics:'Republican'};
guestList[1] = {orderNum:5,guestName:
  'Tweedle-dum',politics:'Republican'};
guestList[2] = {orderNum:1,guestName:
  'White Knight',politics:'Democrat'};
guestList[3] = {orderNum:2,guestName:
  'Alice',politics:'Independent'};
guestList[4] = {orderNum:4,guestName:
  'Tweedle-dee',politics:'Democrat'};
```

To create a sortable table:

1. `array.js`

Create and save the external JavaScript file `array.js` in the folder `javascript` (**Code 7.4**). This file will contain the data for the table, stored in an array.

Steps 2 and 3 apply to this file.

2. `var guestList = new Array();`

Add the array `guestList[]`. This array will be used to record the data presented in the table.

3. `guestList[0] = {orderNum:3, 'guestName':'Red Queen',politics: 'Republican'};`

Add an array definition for each row you need in your table, defining the value for each column. In this example, we're setting up 5 rows with 3 columns (`orderNum`, `guestName`, and `politics`).

continues on next page

4. tableSortLibrary.js

Create and save the external JavaScript file `tableSortLibrary.js` (**Code 7.5**). This file will contain functions used to sort and display the data table. Each column will have its own function to sort by that column.

Steps 5–9 apply to this file.

5. var sortBy = null;

Add the variable `sortBy` to your JavaScript and set its initial value to `null`. This variable is used to store which column the table is currently sorted by.

6. function sortByOrderNum(a,b) {...}

Add the function `sortByOrderNum()` to your JavaScript. This function will be used to sort the table by the values in the first column. This function can only sort numeric values.

If you're adapting this function for your own tables, use it only if the column contains only numeric values.

7. function sortByGuestName(a,b) {...}

Add the functions `sortByGuestName()` and `sortByPolitics()`. Both of these functions are used to sort columns that contain alpha/numeric data.

If you're adapting this function for your own sortable table, use this function if your table contains letters, numbers, and other characters.

8. function sortArray(arrayName, sortBy, areaName) {...}

Add the function `sortArray()`. This function works as the switchboard to determine which of the sort functions from steps 6 and 7 to use based on the value passed to it in the `sortBy` variable. It then runs the `displayTable()` function to refresh the data table on the screen with the new order.

Code 7.5 The external JavaScript file `tableSortLibrary.js` will be used to sort the data table.

```
var sortBy = null;
function sortByOrderNum(a,b) {
    return a.orderNum - b.orderNum;
}
function sortByGuestName(a,b) {
    var stringA = a.guestName.toLowerCase();
    var stringB = b.guestName.toLowerCase();
    if (stringA >= stringB) return 1;
    if (stringA <= stringB) return -1;
    return 0;
}
function sortByPolitics(a,b) {
    var stringA = a.politics.toLowerCase();
    var stringB = b.politics.toLowerCase();
    if (stringA >= stringB) return 1;
    if (stringA <= stringB) return -1;
    return 0;
}
function sortArray(arrayName, sortBy, areaName) {
    if (sortBy == 'orderNum')
        arrayName.sort(sortByOrderNum);
    if (sortBy == 'guestName')
        arrayName.sort(sortByGuestName);
    if (sortBy == 'politics')
        arrayName.sort(sortByPolitics);
    displayTable(arrayName, areaName);
}
function displayTable(arrayName, areaName) {
    var tr;
    var td;
    object=document.getElementById(areaName);
    while (object.rows.length > 0)
        object.deleteRow(0)
    for (var i = 0; i <= arrayName.length; i++) {
        tr = object.insertRow(object.rows.length);
        td = tr.insertCell(tr.cells.length);
        td.setAttribute('class', 'col1');
        td.innerHTML = arrayName[i].orderNum;
        td = tr.insertCell(tr.cells.length);
        td.setAttribute('class', 'col2');
        td.innerHTML = arrayName[i].guestName
        td = tr.insertCell(tr.cells.length);
        td.setAttribute('class', 'col3');
        td.innerHTML = arrayName[i].politics;
    }
}
```

Code 7.6 Although the table and the header are in the HTML, the table body is written into the page `index.html` using JavaScript from Code 7.5.

```
var sortBy = null;
function sortByOrderNum(a,b) {
    return a.orderNum - b.orderNum;
}
function sortByGuestName(a,b) {
    var stringA = a.guestName.toLowerCase();
    var stringB = b.guestName.toLowerCase();
    if (stringA > stringB) return 1;
    if (stringA < stringB) return -1;
    return 0;
}
function sortByPolitics(a,b) {
    var stringA = a.politics.toLowerCase();
    var stringB = b.politics.toLowerCase();
    if (stringA > stringB) return 1;
    if (stringA < stringB) return -1;
    return 0;
}
function sortArray(arrayName,sortBy,areaName) {
    if (sortBy == 'orderNum')
        arrayName.sort(sortByOrderNum);
    if (sortBy == 'guestName')
        arrayName.sort(sortByGuestName);
    if (sortBy == 'politics')
        arrayName.sort(sortByPolitics);
    displayTable(arrayName,areaName);
}
function displayTable(arrayName,areaName) {
    var tr;
    var td;
    object=document.getElementById(areaName);
    while (object.rows.length >0)
        object.deleteRow(0)
    for(var i = 0; i < arrayName.length; i++) {
        tr = object.insertRow(object.rows.length);
        td = tr.insertCell(tr.cells.length);
        td.setAttribute('class','col1');
        td.innerHTML = arrayName[i].orderNum;
        td = tr.insertCell(tr.cells.length);
        td.setAttribute('class','col2');
        td.innerHTML = arrayName[i].guestName
        td = tr.insertCell(tr.cells.length);
        td.setAttribute('class','col3');
        td.innerHTML = arrayName[i].politics;
    }
}
```

9. `function displayTable(arrayName, areaName) {...}`

Add the function `displayTable()`. This function will write the data from the array passed to it in the variable `arrayName` into the specified table body in the HTML (`areaName`). Each `<td>` tag in a row will include a class attribute for `col1`, `col2`, or `col3`.

10. `index.html`

Create and save the file `index.html` (Code 7.6). This file will hold the actual data table.

Steps 11–20 apply to this file.

11. `src="JavaScript/arrays.js"`

Add a `<script>` tag with `array.js` as its source to load the JavaScript file from step 1.

12. `src="JavaScript/tableSortLibrary.js"`

Add another `<script>` tag with the source `tableSortLibrary.js` to load the JavaScript file from step 4.

13. `th {...}`

In your CSS, add a definition for the table header tag. One important rule to add here is the `cursor` attribute, which you should set to display as the pointer.

14. `.col1, .col2, .col3 {...}`

Add definitions for the `col1`, `col2`, and `col3` classes that will be used to display the content of the table data tags in the corresponding columns.

continues on next page

15. `onload="displayTable(guestList, 'table1');"`

In the `<body>` tag, add an `onload` event handler that will trigger the `displayTable()` function from step 9 in order to display the initial table in the Web page. Designate the name of the array you will be using (`guestList`) and the name of the table to write the data into (`table1`).

16. `<table border="0" cellspacing="0" cellpadding="0">...</table>`

In your HTML, add the `<table>` tag, setting all of its attributes to 0.

17. `<thead>...</thead>`

In the table from step 16, add your table head container tag.

18. `<th class="col1">...</th>`

In the table head tag from step 17, add a table head tag for each of the columns, giving each column its relevant class (`col1`, `col2`, `col3`).

19. `#`

In each table header tag from step 18, add a `` tag with an `onclick` event handler to trigger the function `sortBy()`, passing it the name of the array (`guestList`), the name of the column to sort by (`orderNum`, `guestName`, or `politics`), and the name of the table to put the sorted data into (`table1`).

20. `<tbody id="table1"></tbody>`

In the table from step 16, add the table body tag, giving it a unique ID. In this example, we are using the ID `table1`. Leave the contents of this tag empty since it will be dynamically filled by the function `displayTable()`.

Code 7.7 The default styles in `default.css` are applied to the page if no custom font size is specified by the user.

```
body {
    font-size: small;
}
a.fontSizeControl {
    color: #666666;
    font-family: Verdana, Arial, Helvetica,
sans-serif;
    font-weight: bold;
    text-decoration: none;
    margin-right: 5px;
    margin-left: 5px;
}
a.fontSizeControl:hover{
    color: #000;
}
```

Adding Font Size Controls

Every visitor coming to your Web site has different needs. One of those needs is to read text on your page as comfortably as possible. Unfortunately, a comfortable font size for one visitor may be too small or too large for another. You can overcome this by allowing visitors to set the font size that is best for them.

In this example, we will provide four font-size buttons: small, default, medium, and large. The visitor's font-size preference will be recorded in a cookie, then used when the visitor goes to any page in the Web site or returns later. The default size (the second button) will be used on the first visit to the Web site and if they don't specify a custom font size. To implement this, we will set up four different external style files. The first (`default.css`) will contain all of the styles used for the site plus the default font size. The other three CSS files (`small.css`, `medium.css`, and `large.css`) will contain only font-size definitions.

To add font size controls:

1. `default.css`

Create and save the file `default.css` (**Code 7.7**). To keep things organized, create a folder called `css` to save this and other style files in. This file will hold the styles for your Web site, including the default font size for text.

Steps 2 and 3 apply to this file.

2. `body {...}`

Add a rule for the `body` tag that defines the font size for the Web page as `small`. This relative size will be applied to all text on the page.

continues on next page

3. a.fontSizeControl {...}

Add the class `fontSizeControl`, associated with the anchor tag. This class will be used to set the general appearance of the links used to control font sizes.

4. large.css

Create and save three external CSS files in the `css` folder, each of which will set the relative font size for the `<body>` tag:

- A **large.css**: Set the `font-size` for the `<body>` tag to **large** (Code 7.8).
- A **medium.css**: Set the `font-size` for the `<body>` tag to **medium** (Code 7.9).
- A **small.css**: Set the `font-size` for the `<body>` tag to **x-small** (Code 7.10).

Code 7.8 The external style sheet `large.css` needs to contain only the new font size.

```
body {  
    font-size: large;  
}
```

Code 7.9 The external style sheet `medium.css` needs to contain only the new font size.

```
body {  
    font-size: medium;  
}
```

Code 7.10 The external style sheet `small.css` needs to contain only the new font size.

```
body {  
    font-size: xx-small;  
}
```

Code 7.11 The external file `fontSizeLibrary.js` holds the JavaScript that calls the appropriate style sheet, changing the font size and saving the preference in a cookie for later use.

```
Code
var fontStyles = ['small', 'medium', 'large'];
function setFontSize(newFontSize) {
    setCookie('fontSize', newFontSize);
    changeFontSize();
}
function changeFontSize() {
    if (document.getElementById) {
        var fontSize = getCookie('fontSize');
        for (var i = 0; i <
            fontStyles.length; i++) {
            styleObject = document.
                getElementById(fontStyles[i]);
            if (fontSize == fontStyles[i])
                styleObject.disabled = false;
            else
                styleObject.disabled = true;
        }
    }
}
changeFontSize();
```

5. `fontSizeLibrary.js`

Create and save the external JavaScript file `fontSizeLibrary.js` (**Code 7.11**) in a new folder called `javascript`. This file will contain the functions used to enable and disable the style sheets that control the relative font size for the Web page.

Steps 6–9 apply to this file.

6. `fontStyles = ['small', 'medium', 'large'];`

Add the array `fontStyles[]` to your JavaScript. This array holds the names of the three alternative font sizes.

7. `function setFontSize(fontSize) {...}`

Add the function `setFontSize()` to your JavaScript. This function sets the value of the cookie `fontSize`, based on the value passed to it through the variable `newFontSize`, and then runs the function `changeFontSize()`.

8. `function changeFontSize() {...}`

Add the function `changeFontSize()` to your JavaScript. This function finds the value of the cookie `fontSize` and will then loop through the array `fontStyles[]`, enabling the style sheet for the specified size (`disabled = false`) and disabling the other style sheets.

9. `changeFontSize();`

Add a function call for the function `changeFontSize()` so that the preferred style sheet is enabled when the page loads but before the content is rendered.

continues on next page

10. cookieLibrary.js

Add the file `cookieLibrary.js` to your `javascript` folder. This file is discussed in the section in Chapter 3, "Storing Data in Cookies," and you can simply copy and paste that code here.

11. index.html

Create and save files in which you want to use the font size controls. In this example, we'll add the controls to the Web page `index.html` (**Code 7.12**).

Steps 12–17 apply to this file.

12. href="css/default.css"

Add a `<link>` tag in the head of the page to the style sheet `default.css` located in the `css` folder. This has to be placed before other style sheets imported into the page so that those definitions will supersede the ones in this file.

Code 7.12 The controls are embedded in the HTML page `index.html`, linking to the default style sheet and with the other style sheets imported but disabled.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Transitional//EN" "http://www.w3.org/TR/
  xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I
      Adding Font Size Controls</title>
    <link href="css/default.css"
      rel="stylesheet" type="text/css"
      media="all" />
    <style id="small" type="text/css"
      disabled="disabled">
      @import url(css/small.css);
    </style>
    <style id="medium" type="text/css"
      disabled="disabled">
      @import url(css/medium.css);
    </style>
    <style id="large" type="text/css"
      disabled="disabled">
      @import url(css/large.css);
    </style>
    <script type="text/javascript"
      src="javascript/cookieLibrary.js"
    ></script>
    <script type="text/javascript"
      src="javascript/fontSizeLibrary.js"
    ></script>
  </head>
  <body>
    <span style="font: 10px/10px
      courier;">Set Text Size:</span>
    <a href="javascript:setFontSize
      ('small')" class="fontSizeControl"
      style="font-size: x-small">A</a>
    <a href="javascript:setFontSize
      ('default')" class="fontSizeControl"
      style="font-size: small">A</a>
```

(code continues on nextpage)

```

<a href="javascript:setFontSize-
('medium')" class="fontSizeControl"
style="font-size: medium">A</a>
<a href="javascript:setFontSize
>('large')" class="fontSizeControl"
style="font-size: large">A</a>
<br style="clear:both" />
<h1>CHAPTER III</h1>
<h2>Looking-Glass Insects</h2>
<p>Tickets,
please!' said the Guard, putting his
head in at the window...</p>
</body>
</html>

```

13. `<style id="small" type="text/css" disabled="disabled">...</style>`

For each of the external style sheets used to change the font size (`small.css`, `medium.css`, and `large.css`), add a `<style>` tag with an ID to specify which style sheet this tag will be used for and with the attribute `disabled` set to `disabled`. When initially loaded, this turns all of these style sheets off until the `changeFontSize()` function turns one on.

14. `@import url(css/small.css);`

Within the style tags set up in step 13, add an `@import` command for each external style sheet. This will bring these styles into the Web page, but they are initially disabled, so they won't affect the content.

15. `src="javascript/cookieLibrary.js"`

Add a `<script>` tag to link to the external JavaScript file `cookieLibrary.js` (step 10) located in the `javascript` folder.

16. `src="javascript/fontSizeLibrary.js"`

Add a `<script>` tag to link to the external JavaScript file `fontSizeLibrary.js` (step 5) located in the `javascript` folder.

continues on next page

```

17. <a href="javascript:
    setFontSize('small')"
    class="fontSizeControl" style=
    ="font-size: x-small">...</a>

```

For each of the font controls, add a link to trigger the function `setFontSize()`, passing the function the name of the font size to be used. This value will then be recorded by the browser in a cookie and the page will be redrawn using the new font size. Passing the function any value other than `small`, `medium`, or `large` will cause the page to revert back to the default font size (Figures 7.5, 7.6, 7.7, and 7.8).

Tip

Although used here to change the font size, this method can also be used to swap other styles and is an alternative to the style sheet–swapping method shown in Chapter 5, in the section "Switching Layouts on the Fly."



Figure 7.5 When the page first loads, the default font size is used.



Figure 7.6 Small font size.



Figure 7.7 Medium font size.



Figure 7.8 Large font size.

Creating a Scrollable Area

One common request I get from clients is for important content to load "above the fold." That is, when the page first loads, the content should be at least partially visible without having to scroll the Web page. However, often there is a lot of important content, and it can be quite lengthy, pushing everything else below it down.

To overcome this problem, we can use a simple trick that allows us to put more content into a smaller area by limiting the height of an object and then relying on scrollbars to view the remaining content (**Figure 7.9**). That way more of the content starts above the fold, so that visitors are aware of it without having to scroll the page.

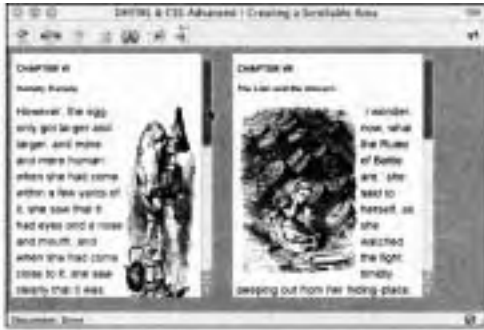
To create a scrollable area:

1. `.scrollBox { ... }`

Add the class `scrollBox` to the **CSS** in your page (**Code 7.13**). This class will be used to control the general appearance of all the independently scrollable areas on the page and set the width and height of the box, setting any overflow (extra content) to be scrolled.

continues on page 241

Figure 7.9 Two areas on the screen have been set up to be scrollable.



Code 7.13 Adding an independently scrollable area to a page is simply a matter of setting the width and height of a layer and then setting the overflow to auto so that scrollbars are displayed if necessary.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
    <title>DHTML & CSS Advanced | Creating a Scrollable Area</title>
    <style type="text/css" media="screen"><!--
body {
  background-color: #999;
```

(code continues on next page)

```

.scrollBox {
    width: 250px;
    height: 300px;
    overflow: auto;
    background-color: white;
}

#SB1 {
    position: relative;
    float: left;
    margin-right: 25px;
}

#SB2 {
    position: relative;
    float: left;
    margin-right: 25px;
}

.scrollBoxContent {
    margin-left: 5px;
    margin-right: 5px;
    color: black;
    font: 10px arial, Helvetica, Geneva, Swiss, SunSans-Regular;
}

--></style>
</head>
<body>
    <div id="SB1" class="scrollBox">
        <div class="scrollBoxContent">
            <h3>CHAPTER VI</h3>
</div>
<h4>Humpty Dumpty</h4>
<p>However,
    + the egg only got larger and larger, and more and more human...</p>
    </div>
</div>
<div id="SB2" class="scrollBox">
    <div class="scrollBoxContent">
        <h3>CHAPTER VII</h3>
        <h4>The Lion and the Unicorn</h4>
        <p>`I wonder, now, what the Rules of Battle are,' she said to herself...</p>
    </div>
</div>
</div>
</body>
</html>

```



Figure 7.10 Each area can be scrolled independently.

2. #SB1 {...}

Add an ID for each of the independently scrollable areas on the page, and set unique layout options for each. In this example, I'm floating them to the left and giving them a 25-pixel right margin.

3. .scrollBoxContent {...}

Add the class `scrollBoxContent` to your CSS. This class is primarily used to control the margins of the content within the scroll box. Theoretically, we could set these attributes in the `scrollBox` class, but remember that Internet Explorer does not handle padding correctly. This is an alternative method to using the box model hack discussed in Chapter 3.

4. <div id="SB1" class="scrollBox">...</div>

To begin setting up the scrollable area, add a `<div>` tag to your HTML with one of the unique IDs from step 2 and the `scrollBox` class from step 1.

5. <div class="scrollBoxContent">... </div>

Within the `<div>` tag from step 4, add a new `<div>` tag with the class `scrollBoxContent`. Inside this tag, you can now place any content you want displayed in the scroll box (**Figure 7.10**).

✓ Tips

You can place any content you want in this scrollable area, including lists of links or other navigation.

Of course, if the content uses an image that is too wide to be displayed, you will see horizontal scrollbars as well.

Animating Scroll Controls

The scroll controls on the browser window allow the visitor to scroll up and down or left and right on the page. However, JavaScript can also force the browser window to scroll in any direction. Generally, scrolling the window without the visitor's consent is bad form, since it can be confusing and disorienting. However, we can use the JavaScript scroll capabilities to give the visitor new scrolling controls.

In this example, we'll use the JavaScript method to scroll the browser window, but rather than instantly scrolling from point to point, we'll scroll incrementally, speeding up slowly to a maximum speed and then slowing down as we approach the end point (Figures 7.11, 7.12, and 7.13). This "zoe-trope" effect can be quite a crowd pleaser, especially in entertainment Web sites.



Figure 7.11 When the page first loads.



Figure 7.12 After clicking one of the links, the content area begins to scroll to the left while the section title (underneath) scrolls to the right.

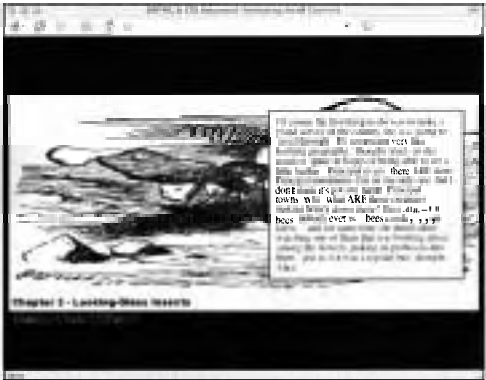


Figure 7.13 The scrolling completes and the sections come to a rest.

Code 7.14 The functions in `scrollLibrary.js` make use of the built-in JavaScript methods to automatically scroll a page from point to point. However, Internet Explorer uses a different method for scrolling than other browsers, so we have to do a bit of browser sniffing first.

```

var isIE = 0;
if (navigator.appName.indexOf('Microsoft
Internet Explorer') != -1) {isIE = 1;}
var xC = 5;
var theDelay = 0;
var frameRateMax = 48;
var frameRate = 1;
var xN = 0;
function scrollPageTo(objectID) {
    var object = document.getElementById
        (objectID);
    xN = object.offsetLeft;
    scrollPage();
}
function scrollPage() {
    if (xC >= xN) {
        xC = xC - frameRate;
        posDif = (xC - xN)/(frameRate/2);
        if (xC <= xN) { frameRate = 1; return; }
    }
    else {
        xC = xC + frameRate;
        posDif = (xN - xC)/(frameRate/2);
        if (xC >= xN) { frameRate = 1; return; }
    }
    if (isIE) {
        document.body.scrollLeft = xC;
    }
    else {
        scrollTo(xC,0);
    }
    if ((posDif >= frameRateMax) &&
        (frameRate != frameRateMax))
        frameRate = frameRate + 1;
    else if (frameRate >= 1)
        frameRate = frameRate - 2;
    setTimeout ('scrollPage()',theDelay);
}

```

To create animated scroll controls:

1. `scrollLibrary.js`

Create and save the external JavaScript file `scrollLibrary.js` (**Code 7.14**). This file will have the functions used to scroll pages horizontally.

Steps 2–5 apply to this file.

2. `if (navigator.appName.indexOf('Microsoft Internet Explorer') != -1) {isIE = 1;}`

Add browser sniffing to the file. We'll need that later to specify which method is used to scroll the page.

3. `var xC = 5;`

Initialize the following variables:

- ▲ `xC` keeps track of the page's current horizontal scroll position.
- A `xN` records the new horizontal scroll position.
- ▲ `theDelay` is a constant that sets the delay between each scroll movement. The larger the value, the slower (but jerkier) the scrolling will look
- ▲ `frameRateMax` is a constant that limits the maximum speed for the scroll.
- ▲ `frameRate` records the current speed for the scroll. This value will never exceed `frameRateMax`.

4. `function scrollPageTo(objectID) {...}`

Add the function `scrollPageTo()`. This function finds the left edge of the object that is being scrolled to and sets this as the new horizontal position (`xN`) and then runs the `scrollPage()` function.

continues on next page

5. function scrollPage() {...}

Add the function `scrollPage()`. This function incrementally scrolls the page horizontally from its current position to the new position.

It first compares the current horizontal scroll position (`xC`) to the horizontal scroll position being moved to (`xN`) and will then calculate the next position to move to based on whether the new position is before or after the current position and assign this to `xC`. If the new position is past where the browser was to scroll to, the function ends. It also calculates the current difference between the two positions (`posDi f`).

The function then uses browser detection to determine the appropriate scroll method for the visitor's browser.

The function will increase the `frameRate` speed by 1 until the halfway point between the initial and final scroll positions or until the frame rate reaches the `frameRateMax` at which time it starts to decelerate.

Finally, the function runs itself again.

6. index.html

Create and save the frameset file `index.html` (Code 7.15). This file is split into six different frames, but three of them have the same content. All of the frames should have `resizing` set to `noresize` and `scrolling` set to `no`.

- ▲ `filler`: The filler frames all use the same file, `filler.html`, and are empty except for the black background.
- ▲ `navigation`: The navigation frame, located in the bottom of the center column of frames, contains all of the links that trigger the scrolling.
- ▲ `content`: The content frame is the large frame where most of the graphics and text are located.
- ▲ `section`: This is a thin frame under the content that displays the section name.

Code 7.15 The frameset index.html is used to create three main frames (content, section, and controls) with filler frames around them to frame the design.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Frameset//EN" "http://www.w3.org/TR/xhtml1/
DTD/xhtml1-frameset.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=
      iso-8859-1" />
    <title>DHTML & CSS Advanced I
      Animating Scroll Controls</title>
  </head>
  <frameset cols="*,780,*" border="0"
    frameborder="no" framespacing="0">
    <frame name="filler1" src="filler.html"
      noresize="noresize" scrolling="no" />
    <frameset rows="*,325,30,*" border="0"
      frameborder="no" framespacing="0">
      <frame name="filler2"
        src="filler.html" noresize=
        "noresize" scrolling="no" />
      <frame name="content"
        src="content.html" noresize=
        "noresize" scrolling="no" />
      <frame name="section"
        src="section.html" noresize=
        "noresize" scrolling="no" />
      <frame name="navigation"
        src="navigation.html" noresize=
        "noresize" scrolling="no" />
    </frameset>
    <frame name="filler3" src="filler.html"
      noresize="noresize" scrolling="no" />
  </frameset>
</html>
```

Code 7.16 The controls in navigation.html will cause the pages in the frames to scroll.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
 Transitional//EN" "http://www.w3.org/TR/
 xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      > content="text/html; charset=
        iso-8859-1" />
    <title></title>
    <script>
function scrollPages(sectionName) {
  if ((top.section.frameRate == 1) &&
    == (top.content.frameRate == 1)) {
    top.content.scrollPageTo(sectionName);
    top.section.scrollPageTo(sectionName);
  }
  else return;
}
    </script>
    <style type="text/css" media="screen"><!--
body {
  color: #ffffff;
  background: #000;
  margin: 5px 0px 0px 7px;
}
a {
  color: #ffffff;
}
--></style>
</head>
<body>
  <a href="javascript:scrollPages
    ('section1');">Chapter 1</a> |
  <a href="javascript:scrollPages
    ('section2');">Chapter 2</a> |
  <a href="javascript:scrollPages
    ('section3');">Chapter 3</a>
</body>
</html>
```

7. navigation.html

Create and save the file `navigation.html` (**Code 7.16**). This file contains the navigation links that control the content and section frames.

Steps 8 and 9 apply to this file.

8. `function scrollPages(sectionName) {...}`

Add the function `scrollPages()` to your JavaScript. This function will check to make sure that the content and section frames are not scrolling and then trigger the `scrollPageTo()` functions in both to scroll to the left edge of the section indicated by the `sectionName` variable.

9. `Chapter 1`

Add links that will trigger the `scrollPages()` function, passing it the name of the section you want to scroll to.

continues on next page

10. content.html

Create and save the file `content.html` (Code 7.17).

Steps 11–16 apply to this file.

11. `src="javascript/scrollLibrary.js"`

Add a `<script>` tag in the head of the document using `scrollLibrary.js` as its source. This will embed the JavaScript from step 1 into this file.

12. `body { ... }`

Add a definition for the `<body>` tag in your CSS. It is important to set the `margin` and `padding` to `0px`.

Code 7.17 The content.html page contains three layers placed side by side. Each layer exactly fills the content frame so that only one at a time is visible.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
 Transitional//EN" "http://www.w3.org/TR/
 xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=
      iso-8859-1" />
    <title>Content</title>
    <script src="javascript/
      scrollLibrary.js"></script>
    <style type="text/css"><!--
body {
  color: #000;
  background-color: #fff;
  margin: 0px;
  padding: 0px;
}
#section1, #section2, #section3 {
  visibility: visible;
  position: absolute;
  top: 0px;
  width: 800px;
  height: 325px;
}
#section1 {
  background: url(media/bg_section01.jpg)
  no-repeat;
  left: 0px;
}
#section2 {
  background: url(media/bg_section02.jpg)
  no-repeat;
  left: 800px;
}
#section3 {
  background: url(media/bg_section03.jpg)
  no-repeat;
  left: 1600px;
```

(code continues on next page)

Code 7.17 continued

```
.sectionContent {
  font: 12px/18px;
  position: absolute;
  padding: 10px;
  top: 25px;
  right: 45px;
  width: 300px;
  background-color: #fff;
  border: 2px solid black;
}

--></style>
</head>
<body>
  <div id="section1">
    <div class="sectionContent">
      One thing was certain, that the
      WHITE kitten had had nothing to
      do with it...
    </div>
  </div>
  <div id="section2">
    <div class="sectionContent">
      'I should see the garden far better...
    </div>
  </div>
  <div id="section3">
    <div class="sectionContent">
      Of course the first thing to do was
      to make a grand survey of the
      country she was going to travel
      through...
    </div>
  </div>
</body>
</html>
```

13. #section1, #section2, #section3 {...}

Add definitions for each of the sections on the page. Each section is an absolutely positioned layer, laid out horizontally, and 20 pixels wider than the *content* frame defined in step 6. This extra space provides a buffer between sections.

14. .sectionContent {...}

Add a class to define how the content in each section should be presented. In this example, the content is absolutely positioned to the right in a white box with a solid black border.

15. <div id="section1">...</div>

Create a layer for each section using a `<div>` tag and the section's ID.

16. <div class="pageContent">...</div>

Within the layer created in step 15, add another layer using a `<div>` tag with the `pageContent` class. All content for the section goes within this layer.

continues on next page

17. section.html

Create and save the file `section.html` (**Code 7.18**). Although the content is slightly different, the process for setting up the section title page (displayed in the section frame) is almost identical to steps 11–16.

There are two important differences, detailed in steps 18 and 19.

18. The section titles are actually reversed, with the `section1` title on the extreme right of the page, so that the section titles will scroll in the opposite direction of their content above. Strictly speaking, this is not required, but it creates a nice effect.

19. `onload="scrollPageTo('section1');"`
Add an `onload` event handler to the `<body>` tag to scroll to the `section1` title. This provides a nice sense of motion when the page loads.

✓ Tip

Due to an unfathomable "feature," this technique will not work in Netscape 4 and 6. It does work fine in Netscape 7 and most versions of Mozilla and Firefox.

Code 7.18 The file `section.html` holds the section title page.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Transitional//EN" "http://www.w3.org/TR/
  xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=
        iso-8859-1" />
    <title>Section</title>
    <script src="javascript/
      scrollLibrary.js"></script>
    <style type="text/css"
      media="screen"><!--
  body {
    color: #000000;
    background: #ffffff;
    font: 16px "arial black";
    margin: 0px;
  }
  #section1, #section2, #section3 {
    top: 3px;
    width: 800px;
    position: absolute;
    padding-left: 5px;
    visibility: visible;
  }
  #section1 {
    left: 1600px;
  }
  #section2 {
    left: 800px;
  }
  #section3 {
    left: 0px;
  }
  --></style>
</head>
<body onload="scrollPageTo('section1');">
  <div id="section1">
    Chapter 1 - Looking-Glass house</div>
  <div id="section2">
    Chapter 2 - The Garden of Live
    Flowers</div>
  <div id="section3">
    Chapter 3 - Looking-Glass Insects</div>
</body>
</html>
```

Adding a Calendar Date Picker

If you are requesting that visitors enter a date into form fields, you could just hope they have a desk calendar handy or you could give them a calendar right on the screen. Even better, you could let them click a date in the calendar to automatically place the correctly formatted date into the form fields.

In this example we'll use JavaScript, CSS, and HTML to set up a fully formatted calendar (**Figure 7.14**). We'll also give it controls that allow visitors to flip through the different months (**Figure 7.15**), jump back to the current month, and click a date to fill in the date fields of a form (**Figure 7.16**).



Figure 7.14
When the calendar loads, it defaults to current month with the current day highlighted.



Figure 7.15
Clicking the controls allows you to move forward or backward in the calendar. Clicking the circle flips you back to the current month from wherever you are.

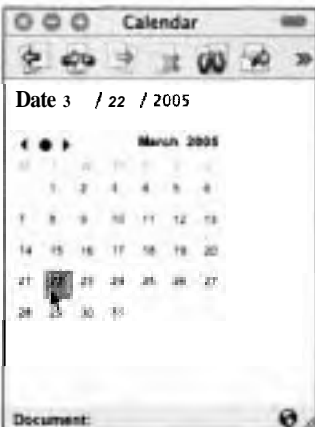


Figure 7.16
Clicking a date in the calendar will place the chosen month, day, and year into the form fields.

To add a calendar date picker:

1. calendar.js

Create and save the external JavaScript file `calendar.js` in the folder `javascript` (**Code 7.19**). This file will hold the functions used to create and control the calendar.

Steps 2–5 apply to this file.

2. `var thisDate = 1;`

Add the following variables to your JavaScript:

`thisDate` tracks current day of the month being written in the calendar.

`wordMonth` stores the names of the months.

`today` is a date object that stores the current date string.

`today'sDay` stores the numeric value (1–7) of the current day of the week.

`today'sDate` stores the current day of the month.

`today'sMonth` stores the numeric value of the current month (1–12)

`today'sYear` stores the current year.

`monthNum` tracks the month currently displayed.

`yearNum` tracks the year currently displayed

`firstDate` stores the first day of the currently displayed month.

`firstDay` tracks the numeric value (1–7) of the day of the week for the first day of the currently displayed month.

`lastDate` tracks the last date of the currently displayed month.

`direction` inputs the direction (next, prev, or return) the calendar should flip.

continues on page 252

Code 7.19 The functions used to write a standard grid calendar into the Web page are stored in `calendar.js`.

```
var thisDate = 1;
var wordMonth = new Array("january",
    "february", "march", "april", "may", "june",
    "july", "august", "september", "october",
    "november", "december");
var today = new Date();
var today'sDay = today.getDay() + 1;
var today'sDate = today.getDate();
var today'sMonth = today.getUTCMonth() + 1;
var today'sYear = today.getFullYear();
var monthNum = today'sMonth;
var yearNum = today'sYear;
var firstDate = new Date(String(monthNum)+
    "/1/"+String(yearNum));
var firstDay = firstDate.getUTCDay();
var lastDate = new Date(String(monthNum+1)+
    "/0/"+String(yearNum));
function changeMonth(direction) {
    if (direction == "prev") monthNum--;
    else if (direction == "next")
        monthNum++;
    else if (direction == "return") {
        monthNum = today'sMonth;
        yearNum = today'sYear;
    }
    if (monthNum == 8) {
        monthNum = 12;
        yearNum--;
    }
    else if (monthNum == 13) {
        monthNum = 1;
        yearNum++
    }
    lastDate = new Date(String(monthNum+1)+
        "/0/"+String(yearNum));
    numDays = lastDate.getDay();
    firstDate = new Date(String(monthNum)+
        "/1/"+String(yearNum));
    firstDay = firstDate.getDay() + 1;
    createCalendar();
    return;
}
```

(code continues on next page)

Code 7.19 *continued*

```

}

function createCalendar() {
    var writeCalendar = '';
    writeCalendar += '<a class="calControl" onmouseover="window.status=\'Previous Month\';return true" onmouseout="window.status=\'\';return true" href="javascript:changeMonth(\'prev\')"><img alt="Previous Month" data-bbox="135 145 185 165"/></a>';
    writeCalendar += '<a class="calControl" href="javascript:changeMonth(\'return\')"></a>';
    writeCalendar += '<a class="calControl" onmouseover="window.status=\'Next Month\';return true" onmouseout="window.status=\'\';return true" href="javascript:changeMonth(\'next\')"><img alt="Next Month" data-bbox="135 230 185 250"/></a>';

    writeCalendar += '<span class="monthShow">';
    writeCalendar += wordMonth[monthNum-1] + '&nbsp;&nbsp;&nbsp;';
    writeCalendar += yearNum;
    writeCalendar += '</span><br style="clear:both" />';
    writeCalendar += '<span class="calDay">M</span><span class="calDay">T</span><span class="calDay">W</span><span class="calDay">Th</span><span class="calDay">F</span><span class="calDay">S</span><span class="calDay">S</span>';
    for (var i = 1; i <= 42; i++) {
        if ((i==1)|| (i==8)|| (i==15)|| (i==22)|| (i==29)|| (i==36))
            writeCalendar += '<br style="clear:both" />';
        if ((thisDate <= numbDays) && (i >= (firstDay-1))) {
            if ((thisDate == todaysDate) && (todaysMonth == monthNum) && (todaysYear == yearNum)) {
                writeCalendar += '<a class="calDateToday" href="javascript:setDate(' + thisDate + ', ' + monthNum + ', ' + yearNum + ')">' + thisDate + '</a>';
            }
            else
                writeCalendar += '<a class="calDate" href="javascript:setDate(' + thisDate + ', ' + monthNum + ', ' + yearNum + ')">' + thisDate + '</a>';
            thisDate++;
        }
        else writeCalendar += '<span class="calEmpty">&nbsp;&nbsp;&nbsp;</span>';
    }
    var object=document.getElementById('calendar');
    object.innerHTML= writeCalendar;
    thisDate = 1;
}

function setDate(dayVal,monthVal,yearVal) {
    document.fores.dateInput.dayVal.value = dayVal;
    document.fores.dateInput.monthVal.value = monthVal;
    document.fores.dateInput.yearVal.value = yearVal;
}

```

3. function changeMonth(direction) {...}

Add the function `changeMonth()`. This function checks which direction to flip the calendar and will then change the month forward (next), backward (prev), or back to today's month (return).

If the month flipped to is 0 (December) or 13 (January), `monthNum` is reset to 12 or 1 and the year is flipped accordingly.

Finally, the `firstDate` and `lastDate` variables are reevaluated and the `createCalendar()` function is triggered.

4. function createCalendar() {...}

Add the function `createCalendar()`. This function renders the calendar and controls into the HTML page using the variable `writeCalendar` to store the HTML code, and then a single `innerHTML` to place the code into the calendar layer (see step 17). It begins with controls to move forward, backward, and return to the current month, and then writes out the month, year, and each day in the month with either the `calDate` or `calDateToday` class. Each day will be a link that passes the day, month, and year values to the `setDate()` function.

5. function setDate(dayVal, monthVal, yearVal) {...}

Add the function `setDate()`. This function places the values for the day, month, and year into the `dayVal`, `monthVal`, and `yearVal` fields of the form named `dateInput` (see step 16).

6. calendar.css

Create and save the file `calendar.css` (**Code 7.20**). This file is used to store the styles for the calendar format.

Steps 7–10 apply to this file.

Code 7.20 The styles in `calendar.css` are used to create each little date box, as well as setting the controls to use the `Webdings` font.

```
#calendar {
    font-family: Arial, Helvetica, sans-serif;
    font-size: 9px;
}

.monthShow {
    display: relative;
    float: left;
    width: 155px;
    text-align: center;
    text-transform: capitalize;
    font-size: 10px;
    font-weight: bold;
}

a.calControl {
    font-size: 16px;
    font-family: webdings;
    text-decoration: none;
    display: absolute;
    width: 15px;
    float: left;
}

a.calControl:hover {
    text-decoration: blink;
}

.calDay, .calDate, .calDateToday, .calEmpty {
    display: absolute;
    float: left;
    width: 15px;
    height: 15px;
    margin-right: 2px;
    margin-bottom: 2px;
    padding: 2px;
}

.calDay {
    color: #938e8f;
    height: 10px;
    border: 1px solid transparent;
    font-style: italic;
    margin-bottom: 0px;
```

(code continues on next page)

```

border-left: 1px solid #f0f0f0;
}
.calDate, .calDateToday {
border: 1px solid #f0f0f0;
background-color: #ffffff;
text-decoration: none;
}
.calDateToday {
border: 1px solid #FF0000;
background-color: #ffdddd;
}
.calEmpty {
border: 1px solid #f0f0f0;
background-color: #f9f9f9;
}
.calDate:hover, .calDateToday:hover {
border: 1px solid #FF9999;
background-color: #FF9999;
}

```

7. `#calendar {...}`
Add the `calendar` ID to your CSS. This defines the layer where the function `createCalendar()` will place the calendar.
8. `.monthShow {...}`
Add the class `monthShow`. This is used to style the month/year header above the calendar.
9. `a.calControl {...}`
Add the class `calControl` associated with the anchor tag. This class will be used to style the controls used to move between months. I've chosen to rely on the Web-safe font `Webdings`. This allows me to add the icon controls without having to rely on downloading extra graphics.
10. `.calDay, .calDate, .calDateToday, * .calEmpty {...}`
Add the following classes with a common rule list, and then customize each as desired:
`calDay`: Style used for the days of the week, listed as the headers of each column in the calendar (M,T,W,Th,F,S,S).
`calDate`: Style used for days in the month. Also include a hover pseudo-class so that when the visitor mouses over a date, it's highlighted.
`calDateToday`: Style used only for the current date.
`calEmpty`: Style used for empty placeholder cells in the calendar.

continues on next page

11. index.html

Create and save the file index.html (Code 7.21). This file will contain the form fields for the date and an empty layer where the calendar will be rendered by JavaScript.

Steps 12–17 apply to this file.

12. src="javascript/calendar.js"

Add a `<script>` tag to the head of your document with the source pointing to the external JavaScript file calendar.js from step 1.

13. href="css/calendar.css"

Add a `<link>` tag with the reference pointing to the external style sheet calendar.css from step 6.

14. onload="changeMonth('return')"

In the `<body>` tag add an `onload` event handler to trigger the `changeMonth()` function from step 3. This will cause the calendar layer to display the current month.

15. <form id="dateInput" action="#" method="get" name="dateInput">...</form>

Add a `<form>` tag with the ID `dateInput`.

16. <input type="text" name="monthVal" id="monthVal" value="mm" size="3" />

Add `<input>` tags for the month, day, and year using the IDs `monthVal`, `dayVal`, and `yearVal`.

17. <div id="calendar">...</div>

Add an empty layer to your HTML using `<div>` tags and the calendar ID. This will be initially blank, but is filled in by the function `createCalendar()`.

Code 7.21 The file index.html contains the date form and calls to the dynamically generated calendar.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Transitional//EN" "http://www.w3.org/TR/
  xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-Type"
      content="text/html; charset=utf-8" />
    <title>Calendar</title>
    <script type="text/JavaScript"
      language="JavaScript" src=
      "javascript/calendar.js"></script>
    <link href="css/calendar.css"
      type="text/css" rel="stylesheet" />
  </head>
  <body onload="changeMonth('return')">
    <form id="dateInput" action="#"
      method="get" name="dateInput">
      <label>Date</label>
      <input type="text" name="monthVal" id=
        "monthVal" value="mm" size="3" /> /
      <input type="text" name="dayVal" id=
        "dayVal" value="dd" size="3" /> /
      <input type="text" name="yearVal" id=
        "yearVal" value="yyyy" size="5" />
    </form>
    <p>
      <div id="calendar">
        <!-- Dynamically Filled -->
      </div>
    </body>
  </html>
```

Adding QuickTime Video Controls

Adding static images to Web page is a straightforward affair: Save the image in one of three common formats (GIF, JPEG, or PNG) and then use the `` tag. The advantage of static image formats is that they are deciphered by the browsers themselves. That is, the code used to display the images is built into the browser. Video images, on the other hand, are not natively supported by browsers, but rely on plug-ins—separate chunks of code that are used by the browser to determine how to display different MIME types. MIME (which actually stands for "multipurpose Internet mail extensions") is used to define particular media types and associate them with the particular plug-in that supports their display in the Web browser. Although this provides greater versatility, since different formats can be added without updating the browser, it also means that the browser has to have the plug-in to display particular file types.

While often thought of as a stand-alone application for displaying video, QuickTime can also be used as a plug-in to display video and audio directly in a Web browser, using a wide variety of different MIME types (Figure 7.17). Although typically associated with files that have the `.mov` extension, QuickTime can also play MPEG, Flash, and AIFF files, among many others. The other great thing about the QuickTime plug-in is that it comes preinstalled with most Web browsers and has its own built-in controls (Figure 7.18).



Figure 7.17 A video clip showing in the browser window.



Figure 7.18 QuickTime controls for volume, play, and scrub are built into the player.

Although you can use QuickTime itself to detect what it can play and to offer a link to the download site for the plug-in (using the `pluginspage` attribute), it is also helpful to use scripting (JavaScript or VBScript) to detect the plug-in and then offer alternative content to viewers who don't have it (Figure 7.19). Unfortunately, there is no "one-size-fits-all" solution. While Netscape and Internet Explorer for the Mac can use JavaScript's `plugin` object to query the browser, Internet Explorer for Windows must use VBScript and Active X to detect QuickTime version 4.1.1 or later.

To add QuickTime video controls:

1. `var gotQT = false;`

Add the variable `gotQT` to your JavaScript (Code 7.22). This variable is initially set to `false` and will be set to `true` only if the QuickTime plug-in is found using one of the detection methods. Although this is defined as a JavaScript variable, it will be treated as a global variable, accessible with VBScript.

2. `<script language="vbscript">...
 </script>`

Add the VBScript in the `<head>` of your document. This code uses VBScript to exploit ActiveX controls to determine whether the QuickTime variable is available to the browser. If it is, `gotQT` is set to `true`. This will only run in Internet Explorer for Windows, and will be ignored by other browsers.

continues on page 258



Figure 7.19 If the browser does not have access to QuickTime, you can provide alternative content, most likely a link to get QuickTime.

Code 7.22 Although you can simply embed the movie into the HTML, with a bit of JavaScript, you can make sure visitors have the correct plug-in and deliver alternative content if they don't.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
>xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced ! Adding QuickTime Video Controls</title>
    <style type="text/css" media="screen"><!--
body {
  color: black;
  background-color: #cc6666;
}
--></style>
  <script language="javascript">
    <!--
var gotQT = false;
  //-->
  </script>
  <script language="vbscript">
    <!--
On Error Resume Next
Set theObject = CreateObject("QuickTimeCheckObject.QuickTimeCheck.1")
On Error goto 0
If IsObject(theObject) Then
  If theObject.IsQuickTimeAvailable(0) Then gotQT = true
  End If
End If
  //-->
  </script>
  <script language="javascript">
    <!--
if (navigator.plugins) {
  for (i=0; i = navigator.plugins.length; i++ ) {
    if(navigator .plugins[i].name.indexOf("QuickTime") >= 0)
      { gotQT = true; }
  }
}
  //-->
  </script>
</head>
<body>
```

(code continues on next page)

3. `<script language="javascript">...
 </script>`

Add the script that follows in the head of your Web page. This code uses a loop to sort through all of the browser plug-ins, accessed through the JavaScript plug-in object (which is first checked so that this script will not run in Internet Explorer for Windows), to check whether QuickTime is available to the browser. If it is, `gotQT` is set to true.

4. `if (gotQT) {...}`

Add this code in the body of your document at the place you want the video embedded. If the variable `gotQT` is true (meaning the plug-in is available), then the QuickTime movie will be embedded and played. If not, you can include alternative content, such as a link to get QuickTime, a static image to stand in place of the video, or even an alternative version of the video file, possibly using Real or Windows Media Players. In addition, you will want to include a message in the `<noscript>` tags for any browsers not running JavaScript.

Tip

For legal reasons, some versions of Internet Explorer may display an alert to confirm the loading of the QuickTime movie before the visitor can see it.

Code 7.22 continued

```
Code
<script language="Javascript">
<!--
    if (!gotQT) {
        document.writeln('<object height="256"
            width="320">');
        document.writeln('<param name="src"
            value="media/myMovie.mov" />');
        document.writeln('<embed height="256"
            src="media/myMovie.mov" type="video/
            quicktime" width="320"></embed>');
        document.writeln('</object>');
    }
    else document.write('You do not seem
        to have <a href="http://www.apple.com/
        quicktime">QuickTime</a> installed. ');
//-->
</script>
<noscript>We are unable to detect whether
    your browser is equipped with <a href=
    "http://www.apple.com/quicktime">QuickTime<
    /a>.</noscript>
</body>
</html>
```



Figure 7.20 When the page first loads, the menu is closed.



Figure 7.21 Clicking the Menu button opens the frame menu, loading the `menu_frame.html` code.

Opening and Closing Frames

One complaint about frames is that they monopolize screen space by placing menus and titles permanently on the screen. Although this arrangement may be fine if you have a large monitor with plenty of room, people who have smaller monitors can be turned off by the experience.

Here is a technique that uses nested frame-sets and some *JavaScript* to open and close a menu in a frame (**Figures 7.20** and **7.21**). When the menu is closed, the content area of the window can use as much space as needed.

To set up collapsible frames:

1. index.html

Set up the main frameset. In this example, I've set up two frames: header and content (Code 7.23 and Figure 7.22). The frame called header will stay static (not change its source) and contain the file header.html, which holds most of the JavaScript needed to pull off this trick. The frame content starts with a nested frameset called nomenu_frames.html, but then switches to the frameset menu_frames.html to toggle the menu open.

Code 7.23 The frameset document index.html is used to create the layout of the Web site.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <frameset rows="35,*" border="0"
    frameborder="no" framespacing="0">
    <frame name="header" src="header.html"
      frameborder="no" marginwidth="2px"
      marginheight="2px" noresize="no"
      "noresize" scrolling="no" />
    <frame name="content"
      src="nomenu_frames.html"
      frameborder="no" marginwidth="10"
      marginheight="10" />
  </frameset>
</html>
```

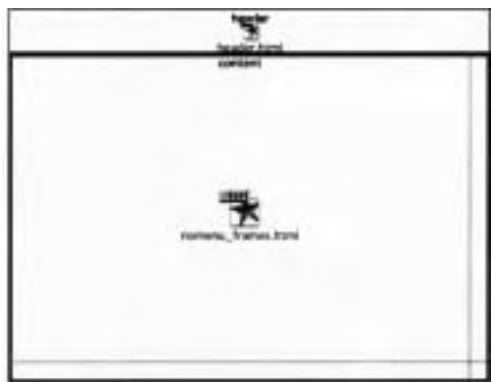


Figure 7.22 This is the frameset for index.html.

Code 7.24 The frameset `nomenu-frames.html` will be loaded into the content frame of `index.html` when the menu is closed.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Frameset//EN" "http://www.w3.org/TR/xhtml1/
DTD/xhtml1-frameset.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <frameset cols="25,*" border="0"
    = frameborder="no" framespacing="0">
    <frame name="control"
      = src="control.html" frameborder="no"
      = marginwidth="0" marginheight="0"
      = noresize="noresize" scrolling="no" />
    <frame name="content2"
      = src="content.html" frameborder="no"
      = marginwidth="10" marginheight="10"
      = scrolling="auto" />
  </frameset>
</html>
```

`nomenu_frames.html`

Create a frameset with two columns, and save it as `nomenu-frames.html` (**Code 7.24** and **Figure 7.23**). The first frame, called `control`, will house the file `control.html`; the second column, called `content2`, initially contains `content.html`.

continues on next page



Figure 7.23 This is the frameset for `nomenu_frames.html`.

menu_frames.html

Create a frameset with three columns, and save it as menu_frames.html (Code 7.25 and Figure 7.24). The second two frames in this frameset are identical to nomenu_frames.html. The first frame, however, is 150 pixels wide and displays the file menu.html. Visitors will switch back and forth between this frameset and the one you set up in step 2, depending on whether they want the menu open or closed.

Code 7.25 The frameset document menu_frames.html is loaded into the content frame of index.html when the menu is open.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Frameset//EN" "http://www.w3.org/TR/xhtml1/
DTD/xhtml1-frameset.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <frameset cols="100,25,*" border="0"
    frameborder="no" framespacing="0">
    <frame name="menu" src="menu.html"
      frameborder="no" marginwidth="18"
      marginheight="18" noresize="noresize"
      scrolling="no" />
    <frame name="control"
      src="control.html" frameborder="no"
      marginwidth="0" marginheight="0"
      noresize="noresize" scrolling="no" />
    <frame name="content2"
      src="content.html" frameborder="no"
      marginwidth="18" marginheight="18"
      scrolling="auto" />
  </frameset>
</html>
```

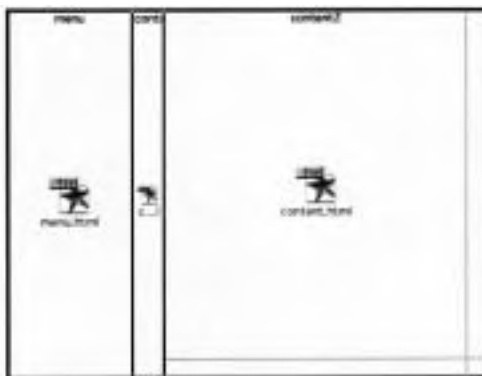


Figure 7.24 This is the frameset for menu_frames.html.

Code 7.26 The HTML document header. `html` is used in the header frame of `index.html`. It contains the function `menuToggle()` that opens or closes the frame menu when triggered.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
 Transitional//EN" "http://www.w3.org/TR/
 xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <script language="JavaScript">
      var frameState = 0;
      var contentSRC = null;
      function menuToggle() {
        if (frameState == 0) {
          contentSRC = parent.content.
            content2.location.href;
          top.content.location =
            'menu_frames.html';
          frameState = 1;
          return;
        }
        else {
          contentSRC = parent.content.
            content2.location.href;
          top.content.location =
            'nomenu_frames.html';
          frameState = 0;
          return;
        }
      }
    </script>
    <style type="text/css" media="screen"><!--
  {
    color: gray;
    font-size: 24px;
    font-family: "Trebuchet MS", Arial,
      Helvetica, Geneva, sans-serif;
    font-weight: bold;
    margin: 0;

  ody {
    background-color: #333;
    margin: 3px;

    --></style>
  </head>
  <body>
    <h2>Through the Looking Glass</h2>
  </body>
</html>
```

4. header.html

Create the HTML file that will be used in the frame header from step 1, and save it as `header.html` (**Code 7.26**). The function `menuToggle()` is the meat of this page; it is executed when a visitor clicks the `Menu` link in `control.html`. The function first checks to see which HTML document is loaded into the `content2` frame and stores that URL in the variable `contentSRC`. It then checks to see whether the menu is visible and switches the frameset to either `nomenu_frames.html` or `menu_frames.html`. The variable `frameState` records the current state of the menu: 0 (closed) or 1 (open).

continues on next page

5. menu.html

Create an HTML file to be displayed when the menu is open, and save it as menu.html (**Code 7.27**). Although I've used HTML links in this example, you can place any type of content in this file. You should target all links to the frame content2.

Code 7.27 The HTML document menu.html contains the menu to be displayed in the menu frame of menu-frames.html. This document can include any HTML content.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Transitional//EN" "http://www.w3.org/TR/
  xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <style type="text/css" media="screen"><!--
body {
    color: gray;
    font: bold 12px/24px "Trebuchet MS",
    Arial, Helvetica, Geneva, sans-serif;
    background-color: black;
}
a:link {
    color: white;
    text-decoration: none;
}
a:hover {
    color: white;
    text-decoration: underline;
}
a:active {
    color: silver;
    text-decoration: underline;
}
a:visited {
    color: white;
    text-decoration: none;
}
--></style>
<body>
  <a href="home.html" target=
  "content2">Cover</a><br />
  <a href="page1.html" target="content2">
  Chapter 1</a><br />
  <a href="page2.html" target="content2">
  Chapter 2</a><br />
  <a href="page3.html" target="content2">
  Chapter 3</a><br />
</body>
</html>
```

Code 7.28 The HTML document `control.html` is used in the control frame of `menu_frames.html` and `nomenu_frames.html`. This file contains a link that triggers the `menuToggle()` function in `header.html`.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0  
· Transitional//EN" "http://www.w3.org/TR/  
· xhtml1/DTD/xhtml1-transitional.dtd">  
<html xmlns="http://www.w3.org/1999/xhtml">  
  <head>  
    <meta http-equiv="content-type"  
      content="text/html; charset=  
      iso-8859-1" />  
    <title>DHTML for the WWW I  
      Control</title>  
  </head>  
  <body>  
    <div id="tabs">  
      <a href="javascript:top.header.  
      menuToggle();" >  
    </a></div>  
  </body>  
</html>
```

6. `control.html`

Create the HTML file that will be used in the frame `control`, and save it as `control.html` (**Code 7.28**). This file contains a link that, when clicked, triggers the `menuToggle()` function from step 4.

continues on next page

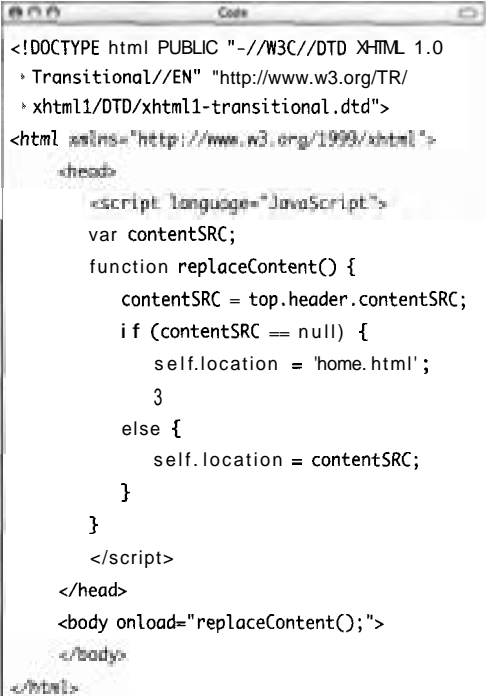
7. content.html

Create an HTML page that contains the initial page loaded into the content2 frame (Code 7.29). This document is an intermediary step and never stays on the screen long. It checks what document was loaded into the content2 frame by accessing the variable in the header frame that recorded it (contentSRC); then it reloads that document. If no previous source exists (when the file first loads, for example), it loads home.html.

8. home.html

Create the Web pages for your site (Code 7.30). All these pages will be in the content2 frame. When you navigate to a page through the menu (Figure 7.25), you can close the menu without losing the page (Figure 7.26).

Code 7.29 The file content.html initially loads into the content2 frame of menu-frames.html and nomenu-frames.html. The JavaScript in this file finds the URL of the current page before the menu was changed (recorded in contentSRC in header.html) and reloads that page into this frame.



```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Transitional//EN" "http://www.w3.org/TR/
  xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <script language="JavaScript">
      var contentSRC;
      function replaceContent() {
        contentSRC = top.header.contentSRC;
        if (contentSRC == null) {
          self.location = 'home.html';
        }
        else {
          self.location = contentSRC;
        }
      }
    </script>
  </head>
  <body onload="replaceContent();" >
  </body>
</html>
```

Code 7.30 A typical content page on the site.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
> Transitional//EN" "http://www.w3.org/TR/
.xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Page 1</title>
    <link href="css/content.css"
      rel="stylesheet" />
  </head>
  <body>
    <h1>CHAPTER I</h1>
    <h2>Looking-Glass house</h2>
    <img alt="Alice"
      height="388" width="206"
      align="right" border="0" hspace="10"
      vspace="10" />
    <p>One thing was certain,
    that the WHITE kitten had had nothing
    to do with it...</p>
  </body>
</html>
```

✓ Tips

This technique lets a site visitor open or close the navigation menu without losing her place within the site.

You do not have to use a menu in the collapsible frame. I created an intranet for an organization that used this technique to provide a calendar of events that could be popped out at any time.

Due to a security restriction in both Netscape and Internet Explorer, you cannot open or close the menu when the content2 frame contains a document from a server other than the one that hosts your Web pages, so you cannot use content from different Web sites.

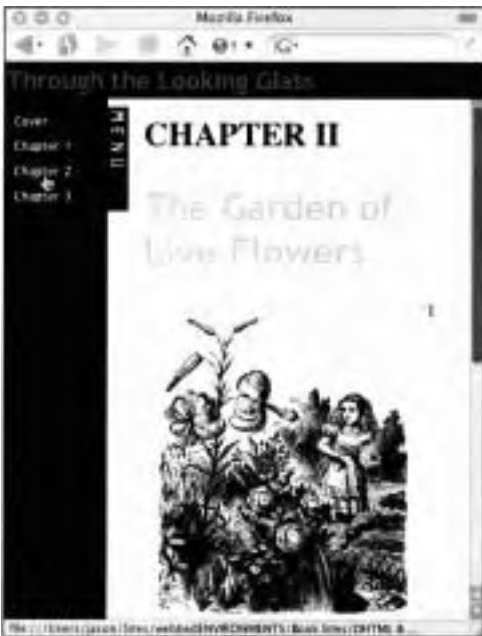


Figure 7.25 The visitor has clicked on a menu option.

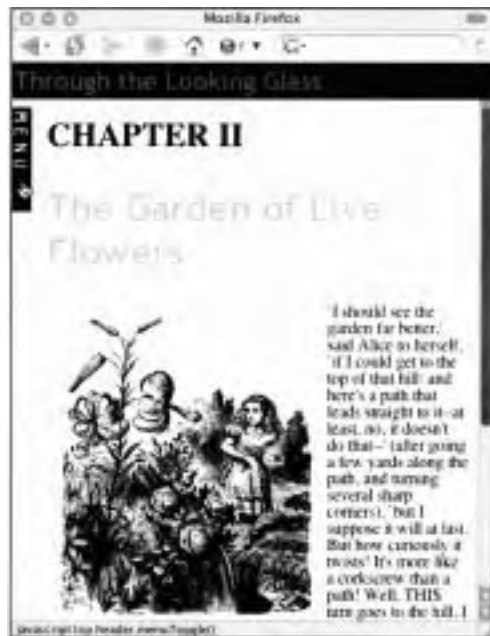


Figure 7.26 The visitor closes the menu, but the content page stays in place.

Blank

FORMS

Forms are used to allow the visitor to send information back to the server to be processed. Yet HTML forms seem to have benefited little from the great advances made in browser technologies. By and large, this is probably because they are easy to use and set up, and many Web visitors have become comfortable with their appearance on the page.

However, we can not only control the appearance of forms (text fields, select boxes, buttons, checkboxes, radio buttons) using CSS, but we can greatly enhance their functionality using DHTML:

Style: Most form elements can have different borders, colors, and fonts, allowing you to design forms that better integrate with the look and feel of your Web site.

Interaction: You can automatically select fields, hide and show parts of forms as needed, and create rollover, selected, and completed display states for text fields on the page.

Error prevention: Catch mistakes in forms before they ever make it to the server.

Beyond the desktop: Create form controls that can be used with a variety of interfaces, such as touch-screen kiosks.

Styling Forms

If left to their own devices, HTML forms are displayed according to the styles dictated by the visitor's browser and operating system. Often, these styles (which can be radically different from browser to browser) are clunky and unattractive (Figure 8.1).

Using CSS, we can specify styles that are applied directly to various form elements (Figure 8.2). However, each form element has its own idiosyncrasies when it comes to CSS, and some browsers won't allow you to change certain form elements, while others do.

Fieldset: This is an optional border around a form that also allows you to set a form legend (title). With CSS, you can define the border and background color that will go behind the form, as well as the general foreground colors and fonts to be used.

Legend: The legend is an optional part of the fieldset that allows you to add a title.

Label: Although levels for form fields can be set using any tag or even no tag at all, the `<label>` tag was specifically designed to be used with form fields.

Input: There are actually several kinds of input elements, each defined by a different type of attribute, and each with its own idiosyncrasies. It's generally not a good idea to set a universal style for the `<input>` tag, since these styles are applied to all the various kinds of inputs. Instead, create a class for each input type and apply these.

Text input: You can control the border, background, foreground color, and font for form input boxes. There are select fields, which vary drastically from browser to browser.

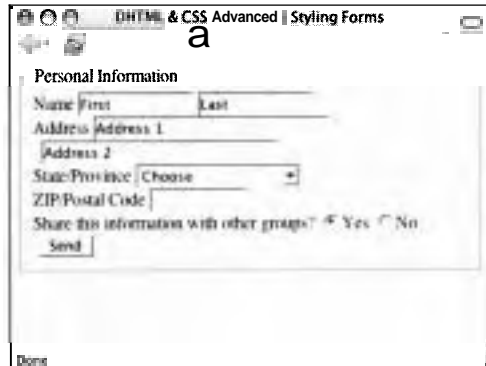


Figure 8.1 Without any styling, the form is a cluttered mess.



Figure 8.2 With styling, the form labels line up in a neat column, and the form fields don't have the clunky beveled edges, but still stand out clearly.

Code 8.1 Set up styles for different form elements to override the default styles dictated by the browser.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Transitional//EN" "http://w.w3.org/TR/
  xhtml1/DTD/xhtml1-transitional.dtd">
<html xml:lang="en" xmlns="http://w.w3.org/
  1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced |
      Styling Forms</title>
    <style type="text/css">
#fieldset {
  color: #666;
  font: 0.8em "Helvetica Neue", helvetica,
    arial, sans-serif;
  background-color: #efefef;
  border: solid 1px #d3d3d3;
}
legend {
  color: #666;
  font-family: "arial block";
  background-color: #d3d3d3;
  padding: 2px;
}
label {
  font-weight: bold;
  line-height: normal;
  text-align: right;
  display: block;
  margin-right: 10px;
  position: relative;
  width: 100px;
  float: left;
}
label.fieldLabel {
  display: inline;
  float: none;
}
input.formInputField {
  border: solid 1px #f66;
  background-color: #fee;
```

(code continues on next page)

Select input: Browsers vary greatly in what parts of the select input you can control with CSS. Most allow background, foreground color, and font. However, most browsers don't allow you to control the border. Safari and Camino don't allow you to change the background and border for the elements, overriding all style changes with the Mac OS Aqua theme.

Radio/Check input: Although styles will affect these, it is usually not with the intended result. Most browsers will simply apply the styles to the rectangular box around the element, leaving the central element unchanged.

Submit/Reset button input: Most browsers allow you to change the font and foreground color for buttons. However, Safari and Camino do not allow you to change the background and border for the buttons, overriding all style changes with the Mac OS Aqua theme.

To add styles to forms:

1. fieldset {...}

Add a definition to your CSS for the `<fieldset>` tag (**Code 8.1**). You'll want to define the background, foreground color, the border around the form, and the general font appearance.

2. legend {...}

Add a definition to your CSS for the `<legend>` tag. This will be used to present the title of your form, so set the background of the title area, the foreground color, and an appropriate font and size.

continues on next page

3. label {...}

Add a definition for the `<label>` tag within your form. These labels will appear next to form fields. We need to set these up as block elements with a specific height and float them to the left. This allows us to present a clearly structured form with labels neatly aligned on the left and form elements on the right, without having to resort to adding tables.

4. label.fieldLabel {...}

Add a class called `fieldLabel` associated with the `<label>` tag. This will be used to override the general `label` style when we need a label that sits directly next to the form element with which it is associated (as with the radio buttons).

5. select.formSelect {...}

Add the class `formSelect` associated with the `<select>` tag. This will be used to add styles to drop-down or multiselect form elements. Although I have set the border for this class, many browsers will ignore this or only place it around the existing border.

6. input.formInputField {...}

Add the class `formInputField` to your CSS associated with the `<input>` tag. This will be used to define the appearance of text input fields.

continues on page 274

Code 8.1 continued

```
color: #333;
margin-right: 5px;
margin-bottom: 5px;
padding: 2px;
height: 15px;
}

input.formInputField:hover {
    background-color: #ccffff;
    border: solid 1px #006600;
    color: #000;
}

select.formSelect {
    border: solid 1px #f66;
    background-color: #fee;
    color: #333;
    cursor: pointer;
    margin-right: 5px;
    margin-bottom: 5px;
    padding: 2px;
}

select.formSelect:hover {
    background-color: #ccffff;
    color: #000;
    cursor: pointer;
}

input.formInputButton {
    border: solid 1px #f66;
    background-color: #f99;
    background-image: url(bg_button.png);
    color: #300;
    cursor: pointer;
    font-size: 1.2em;
    font-weight: bolder;
    text-align: center;
    padding: 1px;
    margin-right: 5px;
    vertical-align: middle;
}

input.formInputButton:hover {
    background-image: url(bg_button_hover.png);
}
}
```

(code continues on next page)

```

Code
</style>
</head>
<body>
<form action="(EmptyReference!)" method="get" id="myform" name="myform">
  <fieldset>
    <legend>Personal Information</legend>
    <label for="firstName">Name</label>
    <input class="formInputField" type="text" name="firstName" id="firstName" value="First"
      size="12" maxlength="16" tabindex="1"/>
    <input class="formInputField" type="text" name="lastName" id="lastName" value="Last"
      size="12" maxlength="16" tabindex="2"/><br/>
    <label for="address1">Address</label>
    <input class="formInputField" type="text" name="address1" id="address1" value="Address 1"
      size="26" maxlength="24" tabindex="3"/><br/>
    <label>&nbsp;</label>
    <input class="formInputField" type="text" name="address2" id="address2" value="Address 2"
      size="26" maxlength="24" tabindex="4"/><br/>
    <label for="state">State/Province</label>
    <select class="formSelect" name="state" id="state" size="1" tabindex="5">
      <option selected="selected">Choose</option>
      <option value="nc">North Carolina (NC)</option>
      <option value="va">Virginia (VA)</option>
    </select><br/>
    <label for="postalCode">ZIP/Postal Code</label>
    <input class="formInputField" type="text" name="postalCode" id="postalCode" size="10"
      maxlength="18" tabindex="6"/><br/>
    <label>Share this information with other groups?</label>
    <input type="radio" name="share" id="shareYes" value="no" checked="checked"
      tabindex="7"/><label class="fieldLabel" for="shareYes">Yes</label>
    <input type="radio" name="share" id="shareNo" value="yes"
      tabindex="8"/><label class="fieldLabel" for="shareNo">No</label><br/>
    <div style="clear:both;"><label>&nbsp;</label><input class="formInputButton"
      type="submit" name="submitButtonName" id="submitButtonName" value="Send"
      tabindex="9"/></div>
  </fieldset>
</form>
</body>
</html>

```


7. `input.formInputField:hover {...}`

Add a hover pseudo-class for the `formInputField` class. This will create a simple rollover effect whenever the visitor mouses over a text field, in this example turning the box blue (**Figure 8.3**).

However, `hover` does not work in Internet Explorer for Windows.

8. `input.formInputButton {...}`

Add the class `formInputButton` associated with the `<input>` tag. Many browsers won't allow you to affect anything other than the foreground color and font of the button, but Mozilla, Netscape, and Internet Explorer will allow you to change the border and background for the button. In this example, I have added a background image to create a customized graphic button that looks very different from the standard form button.



Figure 8.3 When the visitor mouses over a form field, its color changes to indicate that the user can click it (this will not work in Internet Explorer for Windows).

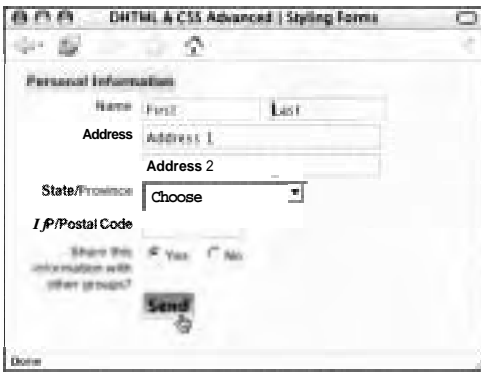


Figure 8.4 When the visitor is ready to check out, the hover pseudo-class will change the background image to make the Submit button look as if it is being pressed.

9. `input.formInputButton:hover {...}`
 Add a hover pseudo-class for the `formInputButton` class. This allows you to set a style for when the visitor hovers over the button. In this example, I simply created a new background graphic by turning the existing graphic 180 degrees, so it looks as if the button stays pressed (**Figure 8.4**).

```
10. <form action="" method="get"
    - id="myForm" name="myForm">...
    </form>
```

Add your form to the page, applying the appropriate classes where necessary.

✓ Tips

Although the hover state is an effective way to show which form element is about to be clicked, keep in mind that Internet Explorer only allows the hover pseudo-class to be used with the anchored link tag (``). This means that hover won't work with form fields. However, you can use CSS to achieve a similar result, though it takes a bit more effort.

Notice that I added a label with a nonbreaking space next to the Submit button. This will ensure that the button is directly under the form fields in a nice straight line.

Highlighting Form Fields

Most forms presented on the Web are extremely static, giving the visitor little or no feedback about what is happening. Basically, whether the form field is empty, selected, or completed, it looks pretty much the same.

In the previous example, I showed you how to use the hover pseudo-class to show when the visitor mouses over a form field. The main disadvantage, though, of using the hover pseudo-class is that it doesn't work in Internet Explorer for Windows.

The code in this example not only allows the hover state to work, but also provides other highlight states for the form fields:

Unfilled: This is the state of the form field when the page first loads (**Figure 8.5**). This is useful for showing visitors which fields they still need to fill in.

Filled: This is the state of the form field after it has information entered and it is no longer selected (**Figure 8.6**). This is useful for showing visitors which fields they have completed.

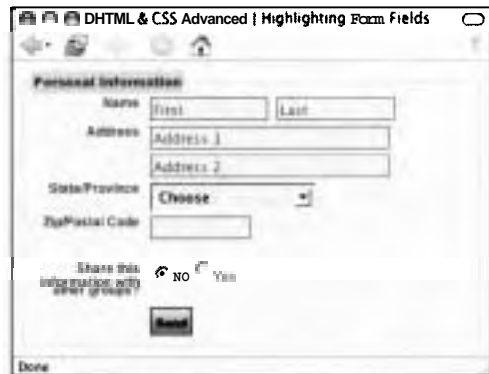


Figure 8.5 Before any of the form fields are filled in, they are grayed out.



Figure 8.6 As the form fields are filled in, they turn yellow to indicate that they have been completed.



Figure 8.7 When visitors mouse over a form field that has not been filled in, the field turns green to let them know it's ready.

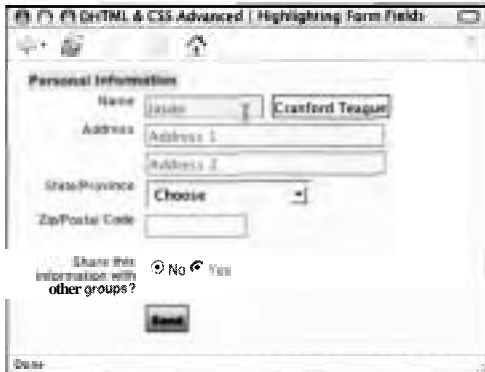


Figure 8.8 When the mouse is over a form field that has already been filled, the field turns red to warn visitors that they may not want to change it.



Figure 8.9 When a form field is selected, it has a white background and black text, making it easy to read.

Hover unfilled: This state is used when the visitor hovers over a form field before it has been filled in (**Figure 8.7**).

Hover filled: This state is used when the visitor hovers over a form field that has already been filled in (**Figure 8.8**). Having a different hover state for filled fields is a useful way to warn visitors that they don't have to make any changes.

Selected: This state is used for the active field and should allow visitors to see as clearly as possible what they are typing (**Figure 8.9**).

To create highlighted form fields:

1. function hoverField(me) {...}

Add the function `hoverField()` to your JavaScript (**Code 8.2**). When triggered by the `onmouseover` event, this function changes the class of the triggering form field for that form object—as long as it is not currently in focus (`formInputField-Focus`)—to its hover state, depending on whether it is unfilled (`formInputField`) or filled (`formInputFieldFilled`).

2. function hoverOffField(me) {...}

Add the function `hoverOffField()` to your JavaScript. When triggered by the `onmouseout` event, this function reverts the class of the form field to its default state, unless it is the form field currently in focus, in which case it leaves it alone.

3. function focusField(me) {...}

Add the function `focusField()` to your JavaScript. When triggered by the `onfocus` event, this function will clear the value of the form field, if it has not already been changed, and then switches to the Selected form class (`formInputFieldSelected`).

4. function blurField(me) {...}

Add the function `blurField()` to your JavaScript. When triggered by the `onblur` event handler, this function will set the field to the Filled state class (`formInputFieldFilled`).

5. .formInputField {...}

Add classes to your CSS for each of the states you'll need:

- `.formInputField`: for unfilled fields
- `.formInputFieldHover`: for unfilled fields being moused over
- `.formInputFieldFilled`: for filled fields
- `.formInputFieldFilledHover`: for filled fields being moused over
- `.formInputFieldSelected`: for the currently selected field

6. You aren't limited to just setting up these classes for a single `<form>` tag type. Optionally, you might set up the styles for the `<input>` tag:

```
input.formInputField {...}
```

as well as for the `<select>` tag

```
select.formInputField {...}
```

and give them different styles depending on what you need.

7. onfocus="focusField(this)"

For each form field or select element, add the following event handlers:

```
onmouseover="hoverField(this)"
```

```
onmouseout="hoverOffField(this)"
```

```
onfocus="focusField(this)"
```

```
onblur="blurField(this)"
```

Keep in mind that you don't have to (and may not even want to) add all of the different states for each form element.

✓ Tip

Coupled with error detection, you could use this technique to highlight form fields that have not been properly completed.

Code 8.2 JavaScript is used in place of the `hover` pseudo-class so that the hover effect works in Internet Explorer for Windows. In addition, you can use the JavaScript to highlight the selected field and fields that have already been filled.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xml:lang="en" xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <title>DHTML &amp; CSS Advanced | Highlighting Form Fields</title>
    <script language="javascript" type="text/javascript">
function hoverField(me) {
    if (me.className != 'formInputFieldSelected') {
        if (me.className == 'formInputField') me.className='formInputFieldHover';
        else me.className = 'formInputFieldFilledHover';
    }
}
function hoverOffField(me) {
    if (me.className != 'formInputFieldSelected') {
        if (me.className == 'formInputFieldHover') me.className='formInputField';
        else me.className = 'formInputFieldFilled';
    }
}
function focusField(me) {
    if ((me.className == 'formInputFieldHover') || (me.className == 'formInputField')) me.value='';
    me.className = 'formInputFieldSelected';
}
function blurField(me) {
    me.className = 'formInputFieldFilled';
}
    </script>
    <style type="text/css">
fieldset {
    border: solid 1px #d3d3d3;
    background-color: #efefef;
    color: #666;
    font: .8em "helvetica neu", helvetica, arial, sans-serif;
}
legend {
    color: #666;
    font-family: "arial black";
    background-color: #d3d3d3;
    padding: 2px;
}
table {
    font-weight: bold;
    line-height: normal;
    text-align: right;
```

(code continues on next page)

Code 8.2 continued

```

display: block;
margin-right: 10px;
position: relative;
width: 100px;
float: left;
}
label.fieldLabel {
display: inline;
float: none;
}
input.formInputField {
border: solid 1px #666666;
background-color: #999999;
color: #666;
margin-right: 5px;
margin-bottom: 5px;
padding: 2px;
height: 15px;
}
input.formInputFieldHover {
background-color: #ccffff;
border: solid 1px #006600;
color: #666;
margin-right: 5px;
margin-bottom: 5px;
padding: 2px;
height: 15px;
}
input.formInputFieldFilled {
border: solid 1px #000;
background-color: #ffc;
color: #000;
margin-right: 5px;
margin-bottom: 5px;
padding: 2px;
height: 15px;
}
input.formInputFieldFilledHover {
background-color: #fcc;
border: solid 1px #006600;
color: #666;
margin-right: 5px;
margin-bottom: 5px;
padding: 2px;
height: 15px;
}

```

(code continues in next column)

Code 8.2 continued

```

}
input.formInputFieldSelected {
background-color: #fff;
border: solid 1px #66c;
color: #000;
margin-right: 5px;
margin-bottom: 5px;
padding: 2px;
height: 15px;
}
select.formInputField {
background-color: #999999;
cursor: pointer;
margin-right: 5px;
margin-bottom: 5px;
padding: 2px;
}
select.formInputFieldSelected {
background-color: #fff;
cursor: pointer;
margin-right: 5px;
margin-bottom: 5px;
padding: 2px;
}
select.formInputFieldFilled {
background-color: #ffc;
cursor: pointer;
margin-right: 5px;
margin-bottom: 5px;
padding: 2px;
border: solid 1px #666;
}
input.formInputButton {
background-image: url(bg_button.png);
color: #300;
font-size: 0.9em;
font-family: "arial black";
background-color: #f99;
text-align: center;
cursor: pointer;
margin-right: 5px;
padding: 1px;
height: 25px;
vertical-align: middle;
border: solid 1px;
}

```

(code continues on next page)

Code 8.2 *continued*

```

}
input.formInputButton:hover {
    background-image: url(bg_button_hover.png);
}

</style>
</head>
<body>
    <fieldset>
        <legend>Personal Information</legend>
        <form id="FormName" action="" method="get" name="FormName">
            <label>Name</label>
            <input class="formInputField" onfocus="focusField(this)" onblur="blurField(this)"
                onmouseover="hoverField(this)" onmouseout="hoverOffField(this)" type="text"
                name="firstName" value="First" size="12" maxlength="16" tabindex="1"/>
            <input class="formInputField" onfocus="focusField(this)" onblur="blurField(this)"
                onmouseover="hoverField(this)" onmouseout="hoverOffField(this)" type="text"
                name="lastName" value="Last" size="12" maxlength="16" tabindex="2"/><br/>
            <label>Address</label>
            <input class="formInputField" onfocus="focusField(this)" onblur="blurField(this)"
                onmouseover="hoverField(this)" onmouseout="hoverOffField(this)" type="text"
                name="address1" value="Address 1" size="26" maxlength="24" tabindex="3"/><br/>
            <label>&nbsp;</label>
            <input class="formInputField" onfocus="focusField(this)" onblur="blurField(this)"
                onmouseover="hoverField(this)" onmouseout="hoverOffField(this)" type="text"
                name="address2" value="Address 2" size="26" maxlength="24" tabindex="4"/><br/>
            <label>State/Province</label>
            <select class="formInputField" onfocus="focusField(this)" onblur="blurField(this)"
                name="state" size="1" tabindex="5">
                <option selected="selected"="selected">Choose</option>
                <option value="nc">North Carolina (NC)</option>
                <option value="va">Virginia (VA)</option>
            </select><br/>
            <label>Zip/Postal Code</label><input class="formInputField" onfocus="focusField(this)"
                onblur="blurField(this)" onmouseover="hoverField(this)" onmouseout="hoverOffField
                (this)" type="text" name="postalCode" size="10" maxlength="10" tabindex="6"/>
            <p><label>Share this information with other groups?</label><input type="radio"
                name="share" value="no" checked="checked" tabindex="7"/>No<input type="radio"
                name="radiogroup" value="yes" tabindex="8"/>Yes</p>
            <p style="clear:both;"><label>&nbsp;</label><input class="formInputButton" type="submit"
                name="submitButtonName" value="Send" tabindex="9"/></p>
        </form>
    </fieldset>
</body>
</html>

```


Auto-Focusing Form Fields

If visitors will use your form often, it helps if you go ahead and place the cursor into the first form field once the page loads, so that they can start typing (**Figure 8.10**). Then, beyond setting the stage with the first form field, you can use JavaScript to automatically pop visitors between form fields. For example, if they're entering a phone number, you can have the cursor tab automatically from the area code to the prefix to the number (**Figure 8.11**).

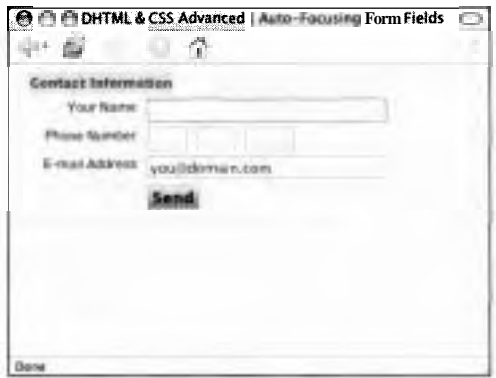


Figure 8.10 The first field in the form is automatically in focus as soon as the page loads.



Figure 8.11 As the visitor types their phone number, the cursor automatically skips to the next field.

Code 83 The function `focusField()` will focus a particular form object on the page. We'll use it to focus on the first form field.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
 Transitional//EN" "http://www.w3.org/TR/
 xhtml1/DTD/xhtml1-transitional.dtd">
<html xml:lang="en" xmlns="http://www.w3.org/
 1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I
      Auto-Focusing Form Fields</title>
    <script language="javascript"
      type="text/javascript">
function focusField(me) {
  if (me.className == 'formInputField')
    me.value='';
  me.className = 'formInputFieldSelected';
}
function blurField(me) {
  me.className = 'formInputFieldFilled';
}
function firstField(formName,formField) {
  document[formName][formField].focus();
}
function nextField(me,formName) {
  if (me.value.length == me.maxLength) {
    var next = me.tabIndex;
    if (next <= formName.elements.length)
      formName.elements[next].focus();
  }
}
    </script>
    <style type="text/css">
fieldset {
  color: #666;
  font: 0.8em "Helvetica Neue", helvetica,
    arial, sans-serif;
  background-color: #efefef;
  border: solid 1px #d3d3d3;
}
legend {
  color: #666;
  font-family: "arial black";
  background-color: #d3d3d3;
  padding: 2px;
}
  </head>
  <body>
    <div style="border: 1px solid #d3d3d3; padding: 5px; width: 50%; margin: 0 auto;">
      <div style="border: 1px solid #d3d3d3; padding: 5px; width: 80%; margin: 0 auto;">
        <input type="text" value=" " />
      </div>
    </div>
  </body>
</html>
```

(code continues on next page)

To set up auto-focus fields:

1. `function focusField(me) {...}`
`function blurField(me) {...}`
Add the functions `focusField()` and `blurField()` to your JavaScript (**Code 8.3**). These functions are identical to those presented in the previous section, "Highlighting Form Fields."
2. `function firstField(formName, formField) {...}`
Add the function `firstField()` to your JavaScript. When triggered by an `onload` event handler, this function will focus the specified form field (`formField`) in the specified form (`formName`).
3. `function nextField(me, form) {...}`
Add the function `nextField()` to your JavaScript. When triggered by a `keyup` event handler, this function will check to see if the form text field currently in focus has reached its full capacity. If it has, the function will automatically jump to the next field in the form.
4. `onload="firstField('myForm', 'name')"`
In the `<body>` tag, add an `onload` event handler to trigger the `firstField()` function, passing to it the name of the form (`myForm`) and its first element (name).

continues on next page

5. `<form id="myForm" name="myForm" action="" method="get">...</form>`

Set up your form, making sure to give it both a name and an ID.

6. `onfocus="focusField(this);"`
`onblur="blurField(this);"`

Add the `onfocus` and `onblur` event handlers to text form fields so that the styles will be switched to highlight the current field in focus.

7. `onkeyup="nextField(this, this.form);"`

In text form fields where you want the cursor to automatically tab upon hitting the maximum character length, add the `onkeyup` event handler. Although you can apply this to any field, it's particularly effective with fields where the info will always be a fixed length like ZIP codes and phone numbers.

Tip

One annoyance I have found with auto-focusing is that if the page is your home page (the first page that loads in the browser window) the focus is taken off of the browser's URL field and put into the form field. If you are used to being able to type a URL as soon as you open a new browser window then you may start typing a URL, only to realize you're entering it in the Name field of the form.

Code 8.3 continued

```
Code
Label {
    font-weight: bold;
    line-height: normal;
    text-align: right;
    display: block;
    margin-right: 10px;
    position: relative;
    width: 100px;
    float: left;
}
label.fieldLabel {
    display: inline;
    float: none;
}
input.formInputField {
    border: solid 1px #f66;
    background-color: #fee;
    color: #333;
    margin-right: 5px;
    margin-bottom: 5px;
    padding: 2px;
    height: 15px;
}
input.formInputField:hover {
    background-color: #ccffff;
    border: solid 1px #006600;
    color: #000;
}
input.formInputFieldFilled {
    border: solid 1px #000;
    background-color: #ffc;
    color: #000;
    margin-right: 5px;
    margin-bottom: 5px;
    padding: 2px;
    height: 15px;
}
input.formInputFieldSelected {
    background-color: #fff;
    border: solid 1px #66c;
    color: #000;
    margin-right: 5px;
    margin-bottom: 5px;
    padding: 2px;
    height: 15px;
}
```

(code continues on next page)

Code 8.3 continued

```

input.formInputButton {
    border: solid 1px #f66;
    background-color: #f99;
    background-image: url(bg_button.png);
    color: #300;
    cursor: pointer;
    font-size: 1.2em;
    font-weight: bolder;
    text-align: center;
    padding: 1px;
    margin-right: 5px;
    vertical-align: middle;
}
input.formInputButton:hover {
    background-image: url(bg_button_hover.png);
}

</style>
</head>
<body onload="firstField('myForm','name')">
  <fieldset>
    <legend>Contact Information</legend>
    <form id="myForm" name="myForm" action="" method="get">
      <label for="name">Your Name</label>
      <input class="formInputField" onfocus="focusField(this);" onblur="blurField(this);"
        type="text" name="name" id="name" size="26" maxlength="50" tabindex="1"/><br/>
      <label for="phoneArea">Phone Number</label>
      <input class="formInputField" onfocus="focusField(this);" onblur="blurField(this);"
        onkeyup="nextField(this, this.form);" type="text" name="phoneArea" id="phoneArea"
        size="3" maxlength="3" tabindex="2"/>
      <input class="formInputField" onfocus="focusField(this);" onblur="blurField(this);"
        onkeyup="nextField(this, this.form);" type="text" name="phonePrefix" id="phonePrefix"
        size="3" maxlength="3" tabindex="3"/>
      <input class="formInputField" onfocus="focusField(this);" onblur="blurField(this);"
        onkeyup="nextField(this, this.form);" type="text" name="phoneNum" id="phoneNum"
        size="4" maxlength="4" tabindex="4"/><br/>
      <label for="email">E-mail Address</label>
      <input class="formInputField" onfocus="focusField(this);" onblur="blurField(this);"
        type="text" name="email" id="email" value="you@domain.com" size="26" maxlength="50"
        tabindex="5"/><br/>
      <label>&nbsp;</label><input class="formInputButton" type="submit"
        name="submitButtonName" id="submitButtonName" value="Send" tabindex="6"/>
    </form>
  </fieldset>
</body>
</html>

```

Performing Form Validation

There is no such thing as a foolproof Web page, especially when the page relies on information submitted by the visitor. Form information can be confirmed after it is submitted to the server, but this requires that the information be sent, processed, and then sent back for corrections if needed.

Using DHTML, however, we can go a long way in speeding this process up by checking the information before it is ever sent and asking the user to fill in missing information or to correct information that is obviously inaccurate.

To validate a form before sending it:

1. `function validateForm(formName) {...}`
Add the function `validateForm()` to your JavaScript (**Code 8.4**). This function checks through all of the data in the form for possible mistakes.

Form fields: The function loops through all of the text fields to make sure that there's something in them (**Figure 8.12**). If one is empty, an alert appears telling the visitor to finish entering the data.

Radio buttons: The function checks that one option is selected for each radio button field (**Figure 8.13**). If not, an alert is displayed asking the visitor to select one of the options.

Select fields: The function checks to make sure that an option has been chosen from select menus (**Figure 8.14**). If not, a JavaScript alert pops up to ask the visitor to select an option.

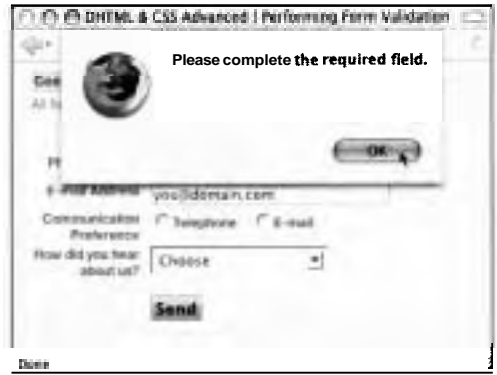


Figure 8.12 Submitting incomplete forms generates an error message.

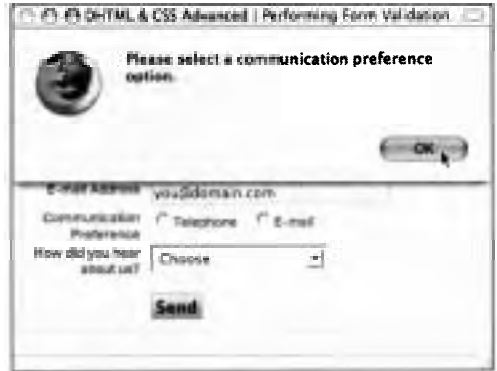


Figure 8.13 If you leave the radio buttons untouched, a message alerts you to choose one option.

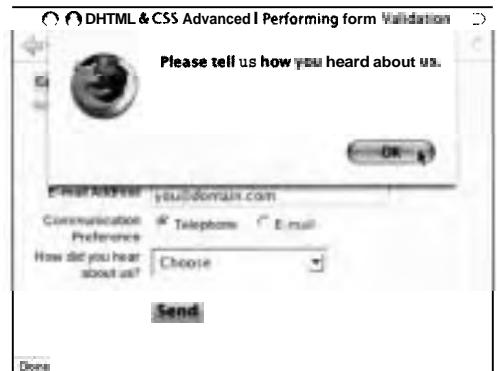


Figure 8.14 If you don't select one of the drop-down options, shame on you.



Figure 8.15 The page checks to make sure that your email address is at least plausible.

Email: The previous checks were general checks to make sure information had been entered, but the final part of the function checks the email address field to make sure that the information in it could be an email address (**Figure 8.15**). There's no way to verify from this page that the email address entered really works, but we can at least make sure that its format conforms with that of a valid email address.

2. `onsubmit="return validateForm(this);"`
In the `<form>` tag, add an `onsubmit` event handler to trigger the validation function before the form information is sent to the server. If there is an error, the submission will be aborted until the data passes validation.

Code 8.4 The `validateForm()` function checks to make sure that all of the form fields have been properly completed.

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xml:lang="en" xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced | Performing Form Validation</title>
    <script language="javascript" type="text/javascript">
function validateForm(formName) {
  var fields = formName.elements;
  for (var i = 0; i <= fields.length; i++) {
    if (!fields[i].value) {
      alert("Please complete the required field.");
      fields[i].focus();
      return false;
    }
  }
  var radioCheck = false;
  for (i = 0; i <= formName.pref.length; i++) {
    if (formName.pref[i].checked) radioCheck = true;
  }
  if (!radioCheck) {
    alert("Please select a communication preference option.");
    formName.pref[0].focus();
    return false;
  }
}
    </script>
  </head>
  <body>
    <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: auto; text-align: center;">
      <img alt="Globe icon" style="vertical-align: middle; height: 30px;"/>
      <span style="font-weight: bold; font-size: 1.2em; display: inline-block; margin-left: 10px;">Please enter a valid e-mail address (e.g. user@domain.com).
      <input type="button" value="Send" style="margin-left: 20px; border: none; background-color: #ccc; padding: 5px 15px; border-radius: 5px; cursor: pointer;"/>
    </div>
    <hr style="border: 0.5px solid #ccc; margin: 10px 0;"/>
    <div style="display: flex; justify-content: space-between; align-items: center;">
      <span style="font-size: 0.8em; font-weight: normal;">Communication Preference
      <span style="font-size: 0.8em; font-weight: normal;">
        <input checked="" type="radio"/> Telephone
        <input type="radio"/> E-mail
      </span>
    </div>
    <div style="margin-top: 10px;">
      <span style="font-size: 0.8em; font-weight: normal;">How did you hear about us?
      <span style="font-size: 0.8em; font-weight: normal; margin-left: 20px;">
        <input type="radio"/> Print, Radio, or TV Ad
      </span>
    </div>
  </body>
</html>

```

(code continues on next page)

```

}
if (formName.referrer.selectedIndex == 0) {
    alert("Please tell us how you heard
        about us.");
    formName.referrer.focus();
    return false;
}
var emailFilter = /^.+@.+\..{2,4}$/;
if (!(emailFilter.test(formName.email.
    value))) {
    alert("Please enter a valid email
        address (e.g., user@domain.com).");
    formName.email.focus();
    return false;
}
else {
    var illegalChars=
        "/[\(\)\<>|,;\:\:\\\"'\[\]\/
if (formName.email.value.match
    (illegalChars)) {
    alert("The email address contains
        illegal characters. Please enter a
        valid email address
        (e.g., user@domain.com).");
    formName.email.focus();
    return false;
}
}
return true;
}
</script>
<style type="text/css">

```

```

fieldset {
    color: #666;
    font: 0.8em "Helvetica Neue", helvetica,
        arial, sans-serif;
    background-color: #efefef;
    border: solid 1px #d3d3d3;
}
legend {
    color: #666;
    font-family: "arial black";
    background-color: #d3d3d3;
    padding: 2px;
}

```

(code continues in next column)

```

label {
    font-weight: bold;
    line-height: normal;
    text-align: right;
    display: block;
    margin-right: 10px;
    position: relative;
    width: 100px;
    float: left;
}
label.fieldLabel {
    display: inline;
    float: none;
}
input.formInputField {
    border: solid 1px #f66;
    background-color: #fee;
    color: #333;
    margin-right: 5px;
    margin-bottom: 5px;
    padding: 2px;
    height: 15px;
}
input.formInputField:hover {
    background-color: #ccffff;
    border: solid 1px #006600;
    color: #000;
}
select.formInputField {
    background-color: #fee;
    color: #333;
    cursor: pointer;
    margin-right: 5px;
    margin-bottom: 5px;
    padding: 2px;
}
input.formInputButton {
    border: solid 1px #f66;
    background-color: #f99;
    background-image: url(bg_button.png);
    color: #300;
    cursor: pointer;
    font-size: 1.2em;
    font-weight: bolder;
    text-align: center;
}

```

(code continues on next page)

Code 8.4 continued

```

padding: 1px;
margin-right: 5px;
vertical-align: middle;
}
input.formInputButton:hover {
background-image: url(bg_button_hover.png);
}

</style>
</head>
<body>
  <fieldset>
    <legend>Contact Information</legend>
    <p>All fields are required. Please complete all of the fields below.</p>
    <form id="FormName" action="" method="get" name="FormName" onsubmit="return validateForm(this);">
      <label for="name">Your Name</label>
      <input class="formInputField" type="text" name="name" id="name" size="26" maxlength="50"
        † tabindex="1"/><br/>
      <label for="phoneArea">Phone Number</label>
      <input class="formInputField" type="text" name="phoneArea" id="phoneArea" size="3"
        † maxlength="3" tabindex="2"/>
      <input class="formInputField" type="text" name="phonePrefix" id="phonePrefix" size="3"
        † maxlength="3" tabindex="3"/>
      <input class="formInputField" type="text" name="phoneNum" id="phoneNum" size="4"
        † maxlength="4" † tabindex="4"/><br/>
      <label for="email">Email Address</label>
      <input class="formInputField" type="text" name="email" id="email" value="you@domain.com"
        † size="26" maxlength="50" † tabindex="5"/><br/>
      <label>Communication Preference</label>
      <input type="radio" name="pref" id="prefPhone" value="byPhone" tabindex="6"/><label
        † class="fieldLabel" for="prefPhone">Telephone</label>
      <input type="radio" name="pref" id="prefEmail" value="byEmail"
        † tabindex="7"/><label class="fieldLabel" for="prefEmail">Email</label><br/>
    <br />
    <div style="clear:both;"><label>How did you hear about us?</label>
    <select class="formInputField" name="referrer" id="referrer" size="1" tabindex="8">
      <option selected="selected">Choose</option>
      <option value="advertisement">Print, Radio, or TV Ad</option>
      <option value="person">Friend or colleague</option>
      <option value="search">Search Engine</option>
    </select></div><br />
    <div style="clear:both;"><label>&nbsp;</label><input class="formInputButton" type="submit"
      † name="submitButtonName" id="submitButtonName" value="Send" tabindex="9"/></div>
  </form>
</fieldset>
</body>
</html>

```


Creating Contextual Forms

Forms are used to enter information on a Web page. This information is sent back to the Web server for a multitude of purposes. Often, however, the form's creator doesn't know exactly what information a particular visitor must enter.

If I'm registering a product online (**Figure 8.16**), I may be registering it for my own use (**Figure 8.17**) or for business (**Figure 8.18**). These two options may require different forms. The common solutions to this problem are to either put both forms on a common page and tell the visitor to fill out the appropriate one, or to use a different page for each form. With DHTML, you can fit both forms on a common page and simply have one or the other appear, depending on the visitor's input.



Figure 8.16 When the page loads, both the personal and the business form are hidden.



Figure 8.17 Clicking the Personal option shows the personal version of the form.



Figure 8.18 Clicking the Business option shows the business version of the form.

Code 8.5 The function `swapForm()` is not too different from other functions used to show and hide content, but is specifically tailored for showing and hiding forms.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
 Transitional//EN" "http://www.w3.org/TR/
 xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=
 iso-8859-1" />
    <title>DHTML &amp; CSS Advanced :
 Creating Contextual Forms</title>
    <script language="JavaScript">
var oldObject = null;
function swapForm(objectID){
  var object = document.getElementById
    (objectID);
  if (oldObject) oldObject.style.display =
    'none';
  object.style.display = 'block';
  oldObject = object;
}
    </script>
    <style type="text/css"
      media="screen"><!--
fieldset {
  border: solid 1px #d3d3d3;
  background-color: #efefef;
  color: #666;
  font: .8em "helvetica neu", helvetica,
    arial, sans-serif;
}
legend {
  color: #666;
  font-family: "arial black";
  background-color: #d3d3d3;
  padding: 2px;
}
label {
  font-weight: bold;
  line-height: normal;
  text-align: right;
  display: block;
  margin-right: 10px;

```

(code continues on next page)

To create a contextual form:

1. `var oldObject = null;`
Initialize the variable `oldObject` to `null` (**Code 8.5**). This step allows the `swapForm()` function in the next step to run the first time without an error.
2. `function swapForm(objectID){...}`
Add `swapForm()` to the JavaScript. This function hides the form that is currently displayed and displays the form selected by the visitor.
3. `#formStack {...}`
Set up an ID for the stack of forms. This ID is a relatively positioned object that contains a stack of absolutely positioned forms.
4. `#personal, #business {...}`
Set up an ID for each form, making them absolutely positioned and hidden. Because all the forms have the same definitions, you can include them in a common list.
5. `onclick="swapForm('personal')"`
In the body of your document, set up a form with no action and a radio button input for each form in the stack. Include in each button an `onclick` event handler that executes `swapForm()` and passes it the name of the form to be revealed.

continues on next page

6. <div id="formStack">...</div>

Add a layer for the stack of forms. This layer will have the form layers nested within it and allows you to position the forms on the page easily.

7. <div id="personal">...</div>

Within the layer you added in step 6, add a separate layer for each form to be used.

8. <form name="personalForm" action="submit.html" method="get">...</form>

Within the layers you added in step 7, add the form code. Each form stands apart from other forms on the page and includes its own Submit and Clear buttons.

Code 8.5 continued

```
position: relative;
width: 100px;
float: left;
}
label.fieldLabel {
display: inline;
float: none;
}
input.formInputField {
border: solid 1px #666666;
background-color: #ccc;
color: #666;
margin-right: 5px;
margin-bottom: 5px;
padding: 2px;
height: 15px;
}
input.formInputField:hover {
background-color: #ccffff;
border: solid 1px #006600;
color: #666;
margin-right: 5px;
margin-bottom: 5px;
padding: 2px;
height: 15px;
}
input.formInputButton {
background-image: url(bg_button.png);
color: #300;
font-size: 0.9em;
font-family: "arial black";
background-color: #f99;
text-align: center;
cursor: pointer;
margin-right: 5px;
padding: 1px;
height: 25px;
vertical-align: middle;
border: solid 1px
}
input.formInputButton:hover {
background-image: url(bg_button_hover.png);
}
#formStack {
position: relative;
```

(code continues on next page)

Code 8.5 continued

```

width: 400px;
visibility: visible;
}
#personal,#business {
padding: 5px;
border: solid 1px gray;
position: relative;
top: 0px;
left: 0px;
display: none;
}

--></style>
</head>
<body>
<fieldset>
<legend>The Registration is for?</legend>
<form method="get" name="myForm">
<input onclick="swapForm('personal')" type="radio" name="context" />Personal <input onclick=
="swapForm('business')" type="radio" name="context" />Business
</form>
<div id="formStack">
<div id="personal">
<form action="submit.html" method="get" name="personalForm">
<label>Name</label> <input class="formInputField" type="text" name="name" size="24" /><br />
<label>Age</label><input class="formInputField" type="text" name="age" size="3" /><br />
<label>SS #</label> <input class="formInputField" type="password" name="ss#" size="9" /><br />
<p style="clear:both;"><label>&nbsp;</label><input class="formInputButton"
type="submit" name="submitPeronal" value="Submit Peronal" />
</form>
</div>
<div id="business">
<form action="submit.html" method="get" name="businessForm">
<label>Company</label> <input class="formInputField" type="text" name="company"
size="24" /><br />
<label>Location</label><input class="formInputField" type="text" name="location"
size="24" /><br />
<label>Tax ID #</label><input class="formInputField" type="password" name="tax#"
size="9" /><br />
<p style="clear:both;"><label>&nbsp;</label><input class="formInputButton"
type="submit" name="submitBusiness" value="Submit Business" />
</form>
</div>
</div>
</fieldset>
</body>
</html>

```

Creating Contextual Form Data

Forms are often cluttered or long, or they stretch over multiple pages. This is because if you don't know exactly what information the visitor is interested in, the answers to one question may change the options for another. One way to cut down on the clutter in forms is with context-sensitive form data that will change as the visitor answers questions.

In this example, we'll set up two selection lists with search criteria. The first list allows visitors to select the type of medium they are searching for (**Figure 8.19**). Depending on which option they choose, the second drop-down will present a different list of options to further refine the search (**Figures 8.20, 8.21, and 8.22**).

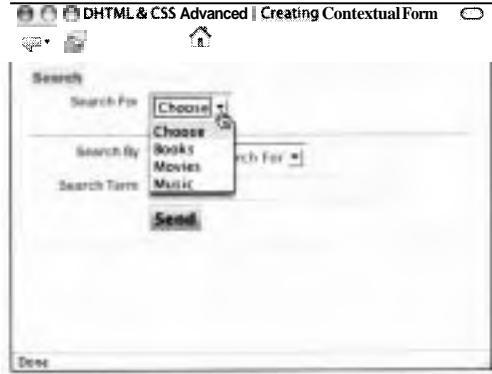


Figure 8.19 Choose the Search For option first, and the options below will change.

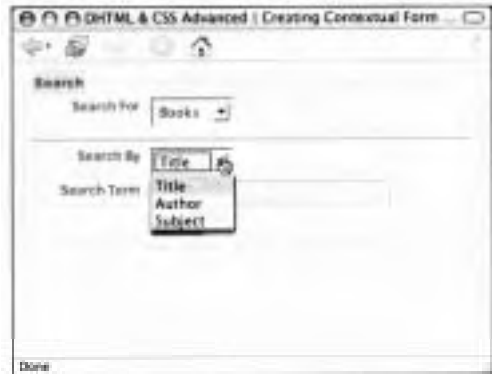


Figure 8.20 These are the options for Books.

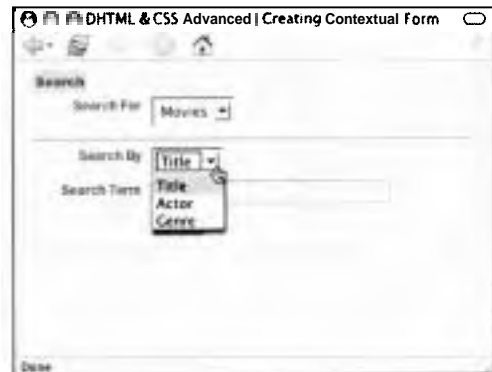


Figure 8.21 These are the options for Movies.

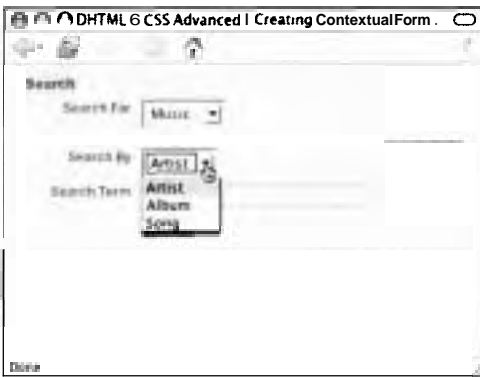


Figure 8.22 These are the options for Music.

Code 8.6 The function `changeSearch()` adds new selection options to the Search By drop-down menu.

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
 Transitional//EN" "http://www.w3.org/TR/
 xhtml1/DTD/xhtml1-transitional.dtd">
<html xml:lang="en" xmlns="http://www.w3.org/
 1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I
      Creating Contextual Form Data</title>
    <script language="javascript"
      type="text/javascript">
function changeSearch(me, formName) {
  if (me == '') return;
  else {
    var search = formName.searchBy;
    switch (me) {
      case "books":
        search.options[0] = new Option
          ("Title", "title");
        search.options[1] = new Option
          ("Author", "author");
        search.options[2] = new Option
          ("Subject", "subject");
        break;
      case "movies":

```

(code continues on next page)

To add contextual data to a form:

1. function `changeSearch(me, formName)`
 `{ ... }`

Add the function `changeSearch()` to your JavaScript (**Code 8.6**). When triggered by an `onchange` event handler in a selection form element, this function will look at the option chosen and then change the values for the `searchBy` selection list using the new `Option()` method, where the first value is the label and the second is the value used for the selection list.

2. `onchange="changeSearch(this.value, this.form);"`

In your form, add an `onchange` event handler to the `<select>` tag. This triggers the function `changeSearch()` and passes it the value selected and the name of the current form.

3. `<option value="books">Books</option>`

Add your `<option>` tags for each choice, making sure that you have a corresponding option in the `changeSearch()` function.

4. `<select class="formSelect" name="searchBy" id="searchBy" size="1" tabindex="2">...</select>`

Add the `searchBy` selection menu. Its only option should be an instruction to choose an option from the first selection list.

Code 8.6 continued

```

search.options[0] = new Option
  ("Title", "title");
search.options[1] = new Option
  ("Actor", "actor");
search.options[2] = new Option
  ("Genre", "genre");
break;
case "music":
search.options[0] = new Option
  ("Artist", "artist");
search.options[1] = new Option
  ("Album", "album");
search.options[2] = new Option
  ("Song", "song");
break;
default:
break;
}
search.focus();
}
</script>
<style type="text/css">
fieldset {
color: #666;
font: 0.8em "Helvetica Neue", helvetica,
arial, sans-serif;
background-color: #efefef;
border: solid 1px #d3d3d3;
}
legend {
color: #666;
font-family: "arial black";
background-color: #d3d3d3;
padding: 2px;
}
label {
font-weight: bold;
line-height: normal;
text-align: right;
display: block;
margin-right: 10px;

```

(code continues in next column)

Code 8.6 continued

```

position: relative;
width: 100px;
float: left;
}
input.formInputField {
border: solid 1px #f66;
background-color: #fee;
color: #333;
margin-right: 5px;
margin-bottom: 5px;
padding: 2px;
height: 15px;
}
input.formInputField:hover {
background-color: #ccffff;
border: solid 1px #006600;
color: #000;
}
select.formSelect {
background-color: #fee;
color: #333;
cursor: pointer;
margin-right: 5px;
margin-bottom: 5px;
padding: 2px;
}
select.formSelect:hover {
background-color: #ccffff;
color: #000;
cursor: pointer;
}
input.formInputButton {
border: solid 1px #f66;
background-color: #f99;
background-image: url(bg_button.png);
color: #300;
cursor: pointer;
font-size: 1.2em;
font-weight: bolder;
text-align: center;
padding: 1px;

```

(code continues on next page)

Code 8.6 continued

```
margin-right: 5px;
vertical-align: middle;
}
input.formInputButton:hover {
background-image: url(bg_button_hover.png);
}
</style>
</head>
<body onload="document.FormName.searchType.focus();">
  <fieldset>
    <legend>Search</legend>
    <form id="FormName" action="" method="get" name="FormName">
      <label for="searchType">Search For</label>
      <select class="formSelect" name="searchType" id="searchType" size="1" tabindex="1"
        onchange="changeSearch(this.value, this.form);">
        <option selected="selected">Choose </option>
        <option value="books">Books</option>
        <option value="movies">Movies</option>
        <option value="music">Music</option>
      </select><br/>
      <hr/>
      <label for="searchBy">Search By</label>
      <select class="formSelect" name="searchBy" id="searchBy" size="1" tabindex="2">
        <option selected="selected">Choose Search For</option>
      </select>
      <br/>
      <label for="searchTerm">Search Term</label>
      <input class="formInputField" type="text" name="searchTerm" id="searchTerm" size="26"
        maxlength="50" tabindex="3"/><br/>
      <label>&nbsp;</label><input class="formInputButton" type="submit" name="submitButtonName"
        id="submitButtonName" value="Send" tabindex="4"/>
    </form>
  </fieldset>
</body>
</html>
```


Restricting a Form Field's Content

Another way to validate a form is to check the data as soon as it's typed in the field and restrict what you allow the visitor to type. In this example, as soon as the visitor finishes typing in a user name or password, the page will check whether the string is the right length and only contains alphanumeric characters (Figure 8.23).

To restrict the content of a form field:

1. `function checkLogin(me) {...}`

Add the function `checkLogin()` to your JavaScript (Code 8.7). When triggered by the `onchange` event handler, this function has two similar parts that check both the user name and password.

Proper length: The user name must be 4–12 characters long, and the password must be at least 6 characters long (Figure 8.24).

Bad characters: The user name can contain letters, numbers, and underscores (`_`) while the password can only contain letters and numbers. The permissible characters for the fields are stored in the `usrBadChars` and `pwdBadChars` variables, allowing only letters, numbers, and the underscore (Figure 8.25). Although they look cryptic, the code in these variables assigns a list of the alphabetical and numeric characters for comparison.

`[\W\d_]` defines any characters that aren't letters (`\W`), numbers (`\d`), or the underscore (`-`). If either condition is met (wrong length or bad characters), an alert appears with instructions. After the visitor clicks OK, the alert closes and the offending form field is selected automatically.

2. `onchange="return checkLogin(this);"`

Add an `onchange` event handler to the form fields that you will want to check passing it the `this` object, which passes information to the function about *this* form field.

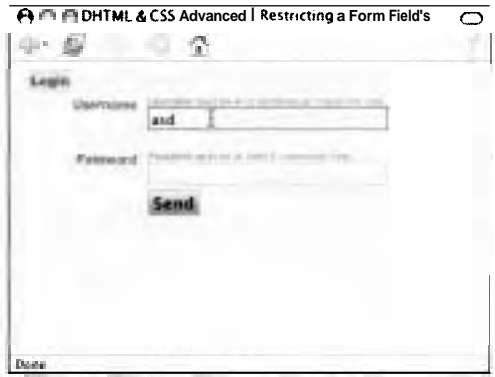


Figure 8.23 When the page first loads, the Username input box is selected.



Figure 8.24 If the visitor enters too few or too many characters, an alert appears.



Figure 8.25 If the visitor enters something other than letters, numbers, or an underscore, an alert appears.

Code 8.7 The function `checkLogin()` performs error-checking even before the Submit button is clicked by checking the information entered as soon as the visitor is done typing.

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
 Transitional//EN" "http://www.w3.org/TR/
 xhtml1/DTD/xhtml1-transitional.dtd">
<html xml:lang="en" xmlns="http://www.w3.org/
 1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I
      Restricting a Form Field's
      Content</title>
    <script language="javascript"
      type="text/javascript">
function checkLogin(me) {
  switch (me.name) {
    case "username":
      var usrBadChars = /[^\w\d]/;
      if ((me.value.length < 4) ||
        (me.value.length > 12)) {
        alert("Please enter a username
          that is 4-12 characters long.");
        document.FormName.username.
          focus();
        return false;
      } else if (usrBadChars.test
        (me.value)) {
        alert("Your username contains
          non-alphanumeric characters.");
        document.FormName.username.
          focus();
        return false;
      }
      break;
    case "password":
      var pwdBadChars = /[^\w\d]/;
      if (me.value.length < 6) {
        alert("Please enter a password
          that is at least 6 characters
          long.");
        document.FormName.password.
          focus();
        return false;
      }
      break;
  }
}

```

(code continues in next column)

Code 8.7 continued

```

} else if (pwdBadChars.
  test(me.value)) {
  alert("Your password contains
    non-alphanumeric characters.");
  document.FormName.password.
    focus();
  return false;
}
break;
default:
  break;
}
return true;
}
</script>
<style type="text/css">
em {
  font-size: 0.75em;
  font-family: arial;
  font-style: normal;
}
fieldset {
  color: #666;
  font: 0.8em "Helvetica Neue", helvetica,
    arial, sans-serif;
  background-color: #efefef;
  border: solid 1px #d3d3d3;
}
legend {
  color: #666;
  font-family: "arial black";
  background-color: #d3d3d3;
  padding: 2px;
}
label {
  font-weight: bold;
  line-height: normal;
  text-align: right;
  display: block;
  margin-right: 10px;
  position: relative;
  width: 100px;
  float: left;
}
input.formInputField {

```

(code continues on next page)

Code 8.7 continued

```
border: solid 1px #f66;
background-color: #fee;
color: #333;
margin-right: 5px;
margin-bottom: 5px;
padding: 2px;
height: 15px;
```

```
input.formInputField:hover {
background-color: #ccffff;
border: solid 1px #006600;
color: #000;
```

```
input.formInputButton {
border: solid 1px #f66;
background-color: #f99;
background-image: url(bg_button.png);
color: #300;
cursor: pointer;
font-size: 1.2em;
font-weight: bolder;
text-align: center;
padding: 1px;
margin-right: 5px;
vertical-align: middle;
```

```
input.formInputButton:hover {
background-image: url(bg_button_hover.png);
```

```
</style>
</head>
<body onload="document.FormName.username.focus();">
<fieldset>
<legend>Login</legend>
<form id="FormName" action="" method="get" name="FormName">
<label for="username">Username</label><em>Username must be 4-12 alphanumeric characters
long.</em><br />
<input class="formInputField" onchange="return checkLogin(this);" type="text"
name="username" id="username" size="26" maxlength="12" tabIndex="1"/> <br/><br/>
<label for="password">Password</label><em>Password must be at least 6 characters
long.</em><br/>
<input class="formInputField" onchange="return checkLogin(this);" type="password"
name="password" id="password" size="26" maxlength="50" tabIndex="2"/> <br/>
<label>&nbsp;</label><input class="formInputButton" type="submit" name="submitButtonName"
id="submitButtonName" value="Send" tabIndex="3"/><br/>
</form>
</fieldset>
</body>
</html>
```

Disabling Form Controls

One important rule of effective user interface design is "don't let them do anything impossible." For example, don't let the visitor submit a form until all of the questions have been answered.

In this example, we'll look at how to disable form elements, including the Submit button, until the visitor has checked a box agreeing to the terms of service (**Figure 8.26**). Once that box is checked the rest of the form is then made available (**Figure 8.27**).



Figure 8.26 When the page first loads, the form fields are disabled and shown as being grayed out.



Figure 8.27 After the visitor checks to accept the terms, the form fields become active and look as if they can be typed in.

To disable form controls until needed:

1. function enableFields(formName) {...}

Add the function `enableFields()` to your JavaScript (**Code 8.8**). When triggered by an `onchange` event handler, this function cycles through all of the form objects in the specified form (`formName`), checking their `disabled` attribute and switching it between `true` and `false`. In addition, the function switches style classes so that when the fields are disabled, they look disabled.

2. .formInputFieldDeactivated {...} .formInputButtonDeactivated {...}

In addition to the other styles we created earlier in the "Styling Forms" section, add two styles to set the appearance of the form fields and buttons when they are deactivated. I recommend a blend of low-contrast grays to make them look as uninviting as possible. Alternatively, you can simply set the `display` class to `none`, which removes them from the page completely.

3. onchange="enableFields(this.form);"

Set up a checkbox input with an `onchange` event handler to trigger the `enableFields()` function, passing it the `this.form` object, which will be used as the `formName` in the function.

Code 8.8 The function `enableFields()` does double duty. It will make disabled fields active or disable active fields.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Transitional//EN" "http://www.w3.org/TR/
  xhtml1/DTD/xhtml1-transitional.dtd">
<html xml:lang="en" xmlns="http://www.w3.org/
  1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I
      Disabling Form Controls</title>
    <script language="javascript"
      type="text/javascript">
function enableFields(formName) {
  var fields = formName.elements;
  for (var i = 1; i <= fields.length; i++) {
    if (fields[i].disabled == true) {
      fields[i].disabled = false;
      if (fields[i].className ==
        'formInputFieldDeactivated')
        fields[i].className = 'formInput
        Field';
      else fields[i].className =
        'formInputButton';
    }
    else {
      fields[i].disabled = true;
      if (fields[i].className ==
        'formInputField') fields[i].
        className = 'formInputField
        Deactivated';
      else fields[i].className = 'formInput
        ButtonDeactivated';
    }
  }
}
</script>
<style type="text/css">
fieldset {
  color: #666;
  font: 0.8em "Helvetica Neue", helvetica,
  arial, sans-serif;

```

(code continues on next page)

Code 8.8 *continued*

```

        background-color: #efefef;
        border: solid 1px #d3d3d3;
    }
    legend {
        color: #666;
        font-family: "arial black";
        background-color: #d3d3d3;
        padding: 2px;
    }
    label {
        font-weight: bold;
        line-height: normal;
        text-align: right;
        display: block;
        margin-right: 10px;
        position: relative;
        width: 100px;
        float: left;
    }
    label.fieldLabel {
        display: inline;
        float: none;
    }
    .formInputFieldDeactivated {
        color: #666;
        background-color: #ccc;
        cursor: wait;
        margin-right: 5px;
        margin-bottom: 5px;
        padding: 2px;
        height: 15px;
        border: solid 1px;
    }
    .formInputButtonDeactivated {
        color: #666;
        background-color: #ccc;
        font-size: 1.2em;
        font-weight: bolder;
        text-align: center;
        cursor: wait;
        border: solid 1px;
    }

```

(code continues in next column)

Code 8.8 *continued*

```

    }
    input.formInputField {
        border: solid 1px #f66;
        background-color: #fee;
        color: #333;
        margin-right: 5px;
        margin-bottom: 5px;
        padding: 2px;
        height: 15px;
    }
    input.formInputField:hover {
        background-color: #ccffff;
        border: solid 1px #006600;
        color: #000;
    }
    input.formInputButton {
        border: solid 1px #f66;
        background-color: #f99;
        background-image: url(bg_button.png);
        color: #300;
        cursor: pointer;
        font-size: 1.2em;
        font-weight: bolder;
        text-align: center;
        padding: 1px;
        margin-right: 5px;
        vertical-align: middle;
    }
    input.formInputButton:hover {
        background-image: url(bg_button_hover.png);
    }
</style>
</head>
<body>
    <fieldset>
        <legend>Software Registration</
        legend>
        <form id="FormName" action=""
        method="get" name="FormName"
        onsubmit="return
        validateForm(this);">

```

(code continues on next page)

```
<input onchange="enableFields
  (this.form);" type="checkbox"
  name="agree" id="agree"
  value="yes" tabindex="1">
<label class="fieldLabel" for="agree">I agree to the <a href="#">terms of service</a>
  outlined by Company X.</label><br style="clear:both"/>
<hr/>
<label for="name">Your Name</label>
<input class="formInputFieldDeactivated" disabled="disabled" type="text" name="name"
  id="name" size="26" maxlength="50" tabindex="2"/><br/>
<label for="email">Email Address</label>
<input class="formInputFieldDeactivated" disabled="disabled" type="text" name="email"
  id="email" value="you@domain.com" size="26" maxlength="50" tabindex="3"/><br/>
<label>&nbsp;</label><input class="formInputButtonDeactivated" disabled="disabled"
  type="submit" name="submitButtonName" id="submitButtonName" value="Send" tabindex="4"/>
</form>
</fieldset>
</body>
</html>
```



Figure 8.28 The two pseudo form-buttons (unchecked and checked).

Creating Graphic Form Controls

The radio and checkbox form controls are basically graphics built into the browser that save you time and make forms look consistent. A checkbox on one site will look the same on other sites. The big drawback to these controls, though, is that you can't change them.

If you're designing a touch-screen kiosk, for example, the little tiny radio button controls are probably not going to be big enough for even the smallest fingers to select. However, with a bit of DHTML, we can create our own checkboxes or radio buttons with any design or size we like (**Figure 8.28**).

To do this, we'll use links as our actual form input mechanism, and then put the form data into a hidden form to be submitted.

To create graphic form controls:

1. checkbox-check.png
checkbox.png

To have graphic buttons, you first have to create the graphics. In this example, I used ImageReady to create two nice, finger-sized buttons, one for the unchecked state, and one for the checked state. I saved these images as checkbox-check.png and checkbox.png.

continues on next page


```
2. function checkField(me, hidden,
  › data) {...}
```

Add the function `checkField()` to your JavaScript (Code 8.9). When triggered by an `onclick` event handler, this function will change the class of the object clicked between `selected` and `notSelected` (depending on what it was when clicked), changing the custom form graphic. It will then place the data to be submitted with the form into hidden form fields.

```
3. div a.notSelected {...}
  div a.selected {...}
```

Add two anchor tag pseudo-classes: `notSelected` (Figure 8.29) and `selected` (Figure 8.30). The two classes are nearly identical except for the background graphic they use. Please note that you have to allow the proper padding on the left (in this example, 55 because the graphic is 50 pixels wide) so that the text doesn't run over the image. You also need to make sure the height will accommodate the button graphic.

```
4. <input type="hidden"
  › name="question1" id="question1"
  -value=" " />
```

For each form element, include a hidden form field, with a blank value and a unique name/ID. This form field will be filled in by the `checkField()` function when the corresponding link (masquerading as a form field) is clicked.

```
5. <a href=" javascript:void( "" )"
  › class="notSelected"
  › onclick="checkField(this,
  › 'question1', 'true');">...</a>
```

Now add a link for each checkbox or radio button in the form, using the `onclick` event handler to trigger the `checkField()` function, passing it the name of the form (*this*), the name of the hidden form field to be filled in, and the value to assign to it.

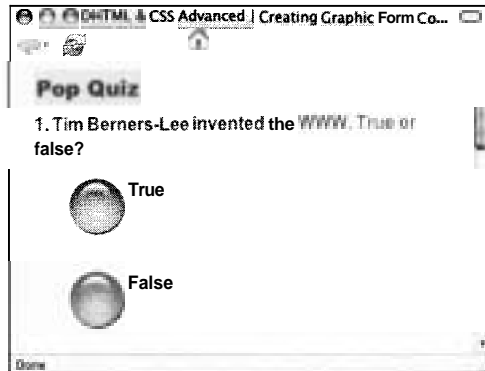


Figure 8.29 Before any pseudo form-buttons are clicked, they're all unchecked.

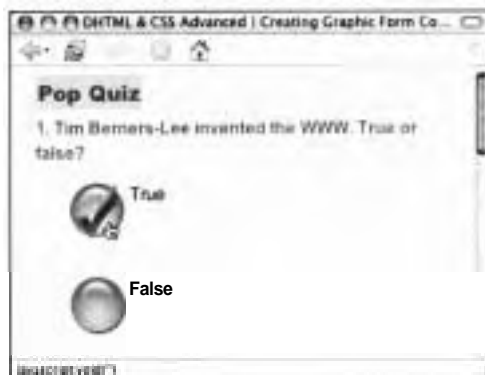


Figure 8.30 After the pseudo form-button has been clicked, it looks checked.

Code 8.9 The function `checkField()` will change the class for the triggering link, changing its background graphic so it looks as if the customized form button has gone from unchecked to checked.

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
· Transitional//EN" "http://www.w3.org/TR/
· xhtml1/DTD/xhtml1-transitional.dtd">
<html xml:lang="en" xmlns="http://www.w3.org/
· 1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      · content="text/html;charset=utf-8" />
    <title>DHTML & CSS Advanced I
      · *Creating Graphic Form Controls</
      · title>
    <script language="javascript"
      · type="text/javascript">
function checkField(me,hidden,data) {
  var hiddenField = eval("document.
    · FormName." + hidden);
  if (me.className == 'notSelected') {
    me.className = 'selected';
    hiddenField.value = data;
  }
  else {
    me.className = 'notSelected';
    hiddenField.value = '';
  }
}
    </script>
    <style type="text/css">
fieldset {
  color: #666;
  font: 1.25em "Helvetica Neue", helvetica,
    arial, sans-serif;
  background-color: #fff;
  border: solid 1px #d3d3d3;
}
legend {
  color: #666;
  font-family: "arial black";
  background-color: #d3d3d3;
  padding: 2px;
}
label {
  font-weight: bold;

```

(code continues in next column)

Code 8.9 continued

```

  line-height: normal;
  text-align: right;
  display: block;
  margin-right: 10px;
  position: relative;
  width: 100px;
  float: left;
}
dtv {
  font-size: 0.8em;
  font-family: arial;
  font-style: normal;
  margin-bottom: .3em;
  line-height: 1.4em;
}
dtv a.notSelected {
  background-image: url(checkbox.png);
  background-repeat: no-repeat;
  margin: 2px 2px 2px 2em;
  padding-left: 55px;
  display: block;
  color: black;
  text-decoration: none;
  height: 60px;
}
dtv a.selected {
  background-image: url(checkbox_check.png);
  background-repeat: no-repeat;
  margin: 2px 2px 2px 2em;
  padding-left: 55px;
  display: block;
  color: black;
  text-decoration: none;
  height: 60px;
}
p.question {
  font-weight: bold;
}
input.formInputButton {
  border: solid 1px #f66;
  background-color: #f99;
  background-image: url(bg_button.png);
  color: #300;
  cursor: pointer;
  font-size: .9em;

```

(code continues on next page)

```

font-weight: bold;
text-align: center;
padding: 1px;
margin-right: 5px;
vertical-align: middle;
}
input.formInputButton:hover {
background-image: url(bg_button_hover.png);
}
</style>
</head>
<body>
<fieldset>
<legend>Pop Quiz</legend>
<form id="FormName" action="" method="get" name="FormName">
<div><p class="question">1. Tim Berners-Lee invented the WWW. True or false?</p>
<input type="hidden" name="question1" id="question1" value=" "/>
<a href="javascript:void(')" class="notSelected" onclick="checkField
(this, 'question1','true');">True</a>
<br/>
<a href="javascript:void(')" class="notSelected" onclick="checkField
(this, 'question1','false');">False</a><br/>
</div>
<div><p class="question">2. Which of the following are HML elements? Select all that
"apply.</p>
<input type="hidden" name="question2_a1" id="question2_a1" value=" "/>
<input type="hidden" name="question2_a2" id="question2_a2" value=" "/>
<input type="hidden" name="question2_a3" id="question2_a3" value=" "/>
<input type="hidden" name="question2_a4" id="question2_a4" value=" "/>
<a href="javascript:void(')" class="notSelected" onclick="checkField
(this,'question2_a1','strong');">Strong</a><br/>
<a href="javascript:void(')" class="notSelected" onclick="checkField
(this,'question2_a2','css');">CSS</a><br/>
<a href="javascript:void(')" class="notSelected" onclick="checkField(
this,'question2_a3','link');">Link</a><br/>
<a href="javascript:void(')" class="notSelected" onclick="checkField
(this,'question2_a4','span');">Span</a><br/>
</div>
<input class="formInputButton" type="button" name="submitButtonName"
id="submitButtonName" value="Grade Your Quiz" tabindex="6"/>
</form>
</fieldset>
</body>
</html>

```

SPECIAL EFFECTS

You don't need to have boring Web pages that sit there on the screen, waiting for the visitor to do something. In this chapter, I'll show you some of the fun effects you can achieve with CSS and DHTML.

Several of these effects may come under the label "stupid Web tricks," but they are all cool, and some may even be useful. A word of caution, though: Use special effects judiciously, lest you annoy your visitors.

While planning your Web site, keep in mind these pointers on using special effects:

- **Enhance layout:** Special effects can be used to create effects that aren't possible with standard CSS in order to create a more visually compelling layout.
- **Call attention to important elements:** Use special effects to subtly guide the visitor's eye around the page to important elements.
- **Illustrate concepts:** Some ideas need movement and action to properly explain them. You can use special effects like GIF animation to create more compelling illustrations.
- **Add atmosphere:** Some special effects are great for just creating a general mood or feeling for your Web site.

Creating Transparent Layers

One of the great frustrations of Web design is how to integrate text into graphics without having to resort to "baking" the text directly into the graphic—that is, using a program like Photoshop to create graphic text. Of course, using CSS you can place HTML text directly over a graphic. But text over graphics is often difficult to read, especially if the graphic has widely varying contrasts (Figure 9.1), and simply covering the image with a solid color to underlie the text is equally unsatisfying since it obscures part of the image (Figure 9.2).

To overcome this problem we need to place a solid-color "slug" behind the text to mute the image enough so that the text is easier to read. However, the slug needs to be transparent enough to allow the image to show through (Figure 9.3).

Most modern browsers now recognize the opacity style, which allows you to set an opacity value for a layer from 0.0 (clear) to .99 (solid). Internet Explorer for Windows doesn't recognize this style, but it does allow you to use a proprietary ActiveX filter to achieve the same effect.



Figure 9.1 The black text over the black-and-white background is difficult to read.



Figure 9.2 The text is now clearly legible, but half of the image is obscured.



Figure 9.3 The image is still visible, but now it is muted enough so that the text is easily readable.

Code 9.1 To place text on top of a transparent slug requires three layers: the background (`mainStoryParent`), the slug (`mainStorySlug`), and the foreground (`mainStoryCopy`).

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Transitional//EN" "http://www.w3.org/TR/
  xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=
        ISO-8859-1" />
    <title>DHTML & CSS Advanced |
      Creating Transparent Layers</title>
    <style type="text/css" media="screen"><!--
body {
  font: .75em helvetica, arial, sans-serif;
  margin: 10% auto;
  width: 360px;
}
h1 {
  font-size: 1.75em;
  font-family: Zapfino, "trebuchet MS",
    sans-serif;

```

(code continues on nextpage)

In this example, we'll use the opacity style to create a slug layer over a background image with text on top of it.

To create transparent layers:

1. `div.mainStoryParent` {...}

Add the class `mainStoryParent` applied to the `<div>` tag (**Code 9.1**). This class is used to set up the layer's background, so you'll want to use the background style to load a background image.

2. `div.mainStorySlug` {...}

Add the class `mainStorySlug` applied to the `<div>` tag. This class will be used to create the solid-color slug over the image and behind the text. Give it the following styles:

- ▲ `background-color`: This can be whatever works best for your design, but should allow optimal reading with the text color.
- ▲ `position`: Set this to `absolute` so the slug layer floats behind the text layer.
- ▲ `top`, `right`, `bottom`, or `left`: Set the position of the slug. For this example, we'll stick with the top-left corner.
- ▲ `width` and `height`: Set the width according to the image and amount of text. Set the height to 188%.
- ▲ `filter`: For Internet Explorer (Windows), set an opacity value using the `filter` attribute.
- ▲ `opacity`: For other browsers, set the opacity attribute directly. It's best to make the `filter` and opacity values the same.

continues on page 313

Code 9.1 continued

```
font-style: normal;
font-variant: normal;
font-weight: normal;
}
div.mainStoryParent {
    background: #ccc url(images/9.1.jpg) no-repeat;
    position: relative;
    width: 360px;
    height: 268px;
}
div.mainStorySlug {
    background-color: white;
    position: absolute;
    top: 0px;
    left: 0px;
    width: 175px;
    height: 268px;
    border: solid 1px #ccc;
    filter: progid:OXImageTransform.Microsoft.BasicImage(opacity=.7);
    opacity: .7;
}
div.mainStoryCopy {
    position: absolute;
    top: 0px;
    left: 0px;
    width: 165px;
    height: 100%;
    overflow: auto;
    padding: 0px 5px;
    opacity: .99;
}
--></style>
</head>
<body>
    <div class="mainStoryParent">
        <div class="mainStorySlug"></div>
        <div class="mainStoryCopy">
            <h1>Queen Alice</h1>
            <p>'Well, this IS grand!' said Alice...</p>
            <a href="#">More &gt;</a></div>
        </div>
    </body>
</html>
```

3. `div.mainStoryCopy { ... }`

Add the `mainStoryCopy` class associated with the `<div>` tag. This class is used to hold the actual content. Set the following:

- ▲ *position*: Set this to *absolute* so that the layer floats above the slug.
- ▲ *top*, *right*, *bottom*, or *left*: Set the position of the text layer. This should match the position set for the slug.
- ▲ *width* and *height*: Although the height should be 100% (the same as `mainStorySlug`), the width generally needs to be a bit less, or the content will run to the edge of the slug.
- ▲ *overflow*: Generally, set this to *auto*, which forces a scrollbar if there isn't enough room to display all of the content. Although a scrollbar isn't optimal, it's better than losing content.
- A *padding*: Set padding around the content so that it doesn't bump against the edges.
- ▲ *opacity*: Some browsers (such as Safari) will inherit the opacity of whatever layers are behind them, even if they aren't directly within the layer. To overcome that, set *opacity* here to `.99`.

4. `<div class="mainStoryParent">...`

In your HTML, set up the enclosing layer using the class `mainStoryParent`.

5. `<div id="mainStorySlug"></div>`

Within the `mainStoryParent` layer, add a layer using the `mainStorySlug` class. This layer will have no content within its tags.

6. `<div class="mainStoryCopy">... </div></div>`

Still within the `mainStoryParent` layer, add a layer using the `mainStoryContent` class. Within this layer, place all of the content you want displayed.

✓ Tips

Notice that the `mainStoryCopy` layer is not within the `mainStorySlug` layer. Conceivably this could make life easier because then we wouldn't have to worry about positioning the copy layer since it would simply follow the slug layer. However, many browsers will force nested layers to use the same opacity as their parent, even if you try to override the opacity directly.

- Older versions of the Mozilla browsers used the style `-moz-opacity`. If you need to support older browsers, just add a rule using that style immediately after the *opacity* and *filter* styles.

ActiveX controls do not work in Internet Explorer for the Macintosh; thus you cannot control opacity for that browser.

Older versions of Safari do not support any method for setting opacity. Instead, the slug would just be a solid color.

Creating Text Drop Shadows

Drop shadows are often used to add emphasis to text or graphics by making the object seem to float over the screen. However, in Web design, drop shadows have historically only been achievable by placing them into graphics or using CSS hacks that place one text layer over another, slightly offset.

The good news is that now CSS includes the text-shadow property that allows you to quickly create a variety of drop shadow (or drop glow) effects (Figures 9.4 and 9.5). The bad news is that the style is not widely supported. At the time of this writing, only Safari supports this style.

In this example (Code 9.2) we'll create a drop shadow for header level 1 (h1) text that is offset 6 pixels by 6 pixels with a slight 3-pixel blur (Figure 9.6).

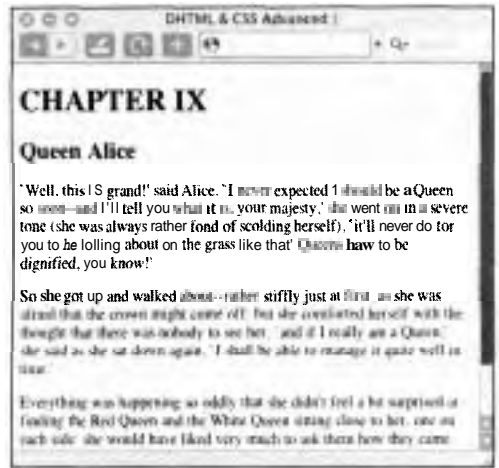


Figure 9.4 Without drop shadows the headers sit lifelessly on the page.

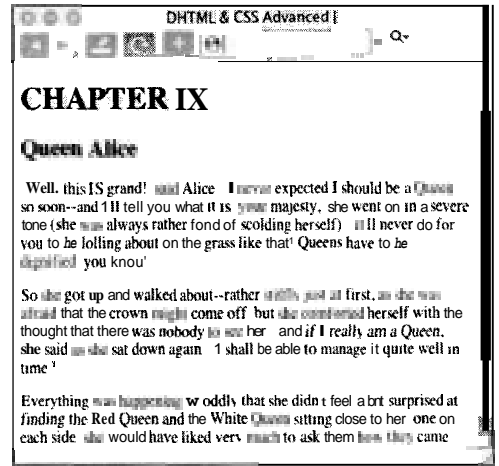


Figure 9.5 With drop shadows, the titles now seem to pop right off the page.



Figure 9.6 Examples of drop shadows and drop glows.

Code 9.2 Although the `text-shadow` attribute is defined in the CSS Level 2 specification, only a few browsers have opted to support it so far.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
 Transitional//EN" "http://www.w3.org/AR/
 xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=utf-8" />
    <title>DHTML & CSS Advanced I
      Creating Drop Shadows</title>
    <link href="default.css"
      type="text/css" rel="stylesheet"
      media="all" />
    <style type="text/css" media="screen">{
h1 {
  font-size: 2em;
  font-weight: bold;
  margin: 0.67em 0;
  text-shadow: #ccc 6px 6px 3px;
}
h2 {
  font-size: 1.5em;
  font-weight: bold;
  margin: 0.83em 0;
  text-shadow: #999 3px 3px 1px;
}
--</style>
</head>
<body>
  <h1>CHAPTER IX</h1>
  <h2>Queen Alice</h2>
  <p>'Well, this IS grand!' said Alice.
  'I never expected I should be a Queen
  so soon...<br />
  </body>
</html>
```

To create a drop shadow effect:

1. `text-shadow`:
Type the property name `text-shadow` followed by a colon (:).
2. `#ccc`
After the colon, enter a color value for the shadow. For a stronger shadow, use values closer to black
3. `6px`
Type a space and then a size value for the horizontal offset of the shadow. Positive values push the shadow to the right, negative values push the shadow to the left.
4. `6px`
Type a space and then a size value for the vertical offset of the shadow. Positive values move the shadow down, negative values move the shadow up.
5. `3px;`
Type a space and then a value for the shadow blur, followed by a semicolon (;). The larger the value, the more diffuse the shadow will appear.

✓ Tips

Generally, you'll want to keep the offsets proportional between different shadow effects on the screen, otherwise the shadows will look unrealistic and unconvincing.

Although I used pixel values in this example, any length value will work (see "Introduction").

To create a drop glow, simply use a dark background color for the page or layer that the text is on and then a lighter color value for **text-shadow**. Generally, glows don't require a lot (if any) offset to look convincing (**Figure 9.7**).

You can add multiple drop shadows to a single element by simply adding a comma after the blur value and then another set of values:

```
text-shadow: #000 2px 2px 5px,  
             #999 -5px 3px 8px;
```

Add as many shadow effects as you want, separated by commas. But before you get too excited, know that this doesn't even work in Safari, which only allows a single shadow effect per object.



Figure 9.7 The text shadow attribute can also be used to set up a drop glow, giving the illusion that there is a light source behind the text.

Floating Objects

This technique is a real eye-catcher. The objects spin around on the screen, speeding up and slowing down. The effect doesn't accomplish much, but it looks cool (**Figure 9.8**).



Figure 9.8 The letters swirl around on the page until the visitor leaves.

To set up floating objects:

1. `var tall = 200;`

In the JavaScript, initialize the following variables (**Code 9.3**):

- ▲ `tall` sets the height of the area in which the objects will float. The objects can float this many pixels to the left or right of the `Xpos`. You can change this value. The larger the number, the farther objects move above or below the center.
- ▲ `wide` sets the width of the area in which the objects will float. The objects can float this many pixels above or below the `Ypos`. You can change this value. The larger the number, the farther objects move to the left or right of the center.
- ▲ `step` sets the increment for the `nextStep` variable. You can change this value. The smaller the number, the slower the objects will float.
- ▲ `delay` controls the delay when the function loops. You can change this value. The larger the number, the slower the objects will float.
- ▲ `nextStep` controls the current speed of the objects.
- ▲ `numObjects` sets the number of objects being floated. Set this variable to the number of objects you will be controlling.
- ▲ `Xpos` sets the position from the left of the page around which the objects will center. You can change this value.
- ▲ `Ypos` sets the position from the top of the page around which the objects will center. You can change this value.

2. `function objectsFloat() {...}`

Add `objectsFloat()` to the JavaScript. This function repositions each object, using a formula that causes them to appear to spin in an elliptical orbit around `Xpos/Ypos`. The function loops recursively until the page is unloaded.

3. `onload="objectsFloat()"`

In the `<body>` tag of the Web page, add an `onload` event handler to trigger the `objectsFloat()` function.

4. `<div id="letters1" style="position: absolute; left: 300; top: 140; *visibility: visible; font: 136px times; color: #000000;">...</div>`

Set up the CSS layers for the objects that you will be floating, using a `<div>` tag defined with an ID and style attributes.

The style defines the object as being positioned absolutely, includes an initial top and left position, and includes any other definitions desired.

✓ Tips

Although this example triggered the floating objects with the `onload` event handler, you can trigger the floating from any event handler you choose.

You can place any content in the CSS layer, but the larger the object, the chopier the animation will look

I placed the style definitions within the `<div>` tag for each of the six layers. But I could have just as easily have set up six different IDs in a `<style>` tag or used an external CSS file.

Code 93 The objectsFloat() function cycles through the objects whose ID starts with "letters," moving each incrementally to create the illusion that they are swirling around the page.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"> <head>
  <meta http-equiv="content-type" content="text/html; charset=iso-8859-1">
  <title>DHTML & amp; CSS Advanced : Floating Objects</title>
  <script language=JavaScript>
var tall = 200;
var wtde = 200;
var step = .25;
var delay = 50;
var nextStep = 0;
var numObjects = 6;
var Xpos = 200;
var Ypos = 200
function objectsFloat() {
  for(var xx = 1 ; xx <= (numObjects + 1) ; xx++ ) {
    var objectID = 'letters' + xx;
    var object = document.getElementById(objectID);
    object.style.top = Ypos + Math.cos((28*Math.sin(nextStep/(30+xx))+xx*78)*tall +
      (Math.sin(10+nextStep/10)+0.2)*Math.cos((nextStep+ xx*55)/10));
    object.style.left = Xpos + Math.sin((28*Math.sin(nextStep/30)+xx*78)*wide"
      (Math. sin(10+nextStep/(10+xx))+0.2)*Math.cos((nextStep+ xx*55)/10));
  }
  nextStep += step;
  setTimeout('objectsFloat()', delay) ;
}
  </script>
</head>
<body onload="objectsFloat()">
  <div id="letters1" style="position: absolute; left: 300; top: 140; vtstbtlty: vtstble;
    font: 136px tmes; color:#000000;">a</div>
  <div id="letters2" style="position: absolute; left: 690; top: 240; visibilty: vtstble;
    font: 136px times; color:#333333;">L</div>
  <div id="letters3" style="position: absolute; left: 400; top: 340; visibilty: vistble;
    font: 136px tmes; color:#666666;">i</div>
  <div id="letters4" style="position: absolute; left: 400; top: 340; vtstbtlty: visible;
    font: 136px tmes; color:#999999;">C</div>
  <div id="letters5" style="position: absolute; left: 400; top: 340; vtstbtlty: vtstble;
    font: 136px tmes; color:#CCCCC;">e</div>
  <div id="letters6" style="position: absolute; left: 400; top: 340; visibility: visible;
    font: 136px tmes; color:#FFFFFF;">!</div>
</body>
</html>
```

Adding a GIF Animation

Remember when you got your first paperback dictionary as a kid? The first thing I did with mine (it was 1977) was create flip movies of X-wing and TIE fighters battling in the margins. I would draw one small picture in the right margin of the first page, draw another picture just like it but moved up or down slightly on the next page, and so on for several dozen pages. When I flipped through the pages of the dictionary, I had my movie of the X-wing swooping in and destroying the TIE fighter.

The graphic interchange format—GIF, to its friends—allows you to set up your own flip-book animation. Although the tools, the medium, and the method of distribution have radically changed since my dictionary flip book, the technique for creating animation is pretty much the same: You draw a picture on one layer, draw the same picture on the next layer but move it just slightly, and so on. When you play back these images, the object appears to be moving.

A variety of tools allow you to create GIF images and animations, but Macromedia Fireworks and Adobe ImageReady both include drawing and animation functions in the same package.

To create a GIF animation in Fireworks:

1. In Fireworks, open a new document at 72 dpi, and set the canvas to the desired size (Figure 9.9).

As with all graphics, larger canvas sizes mean larger files and longer download times.

2. In the Optimize palette, choose Animated GIF from the top drop-down menu (Figure 9.10).



Figure 9.9 The New Document dialog box in Fireworks. Make sure the new document's resolution is set to 72 dpi.

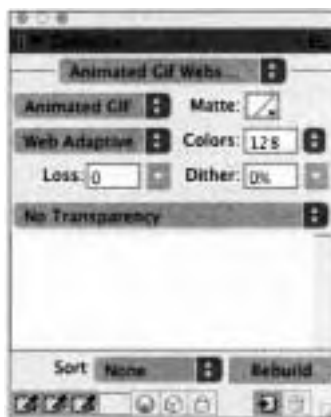


Figure 9.10 The image type needs to be set to Animated GIF.

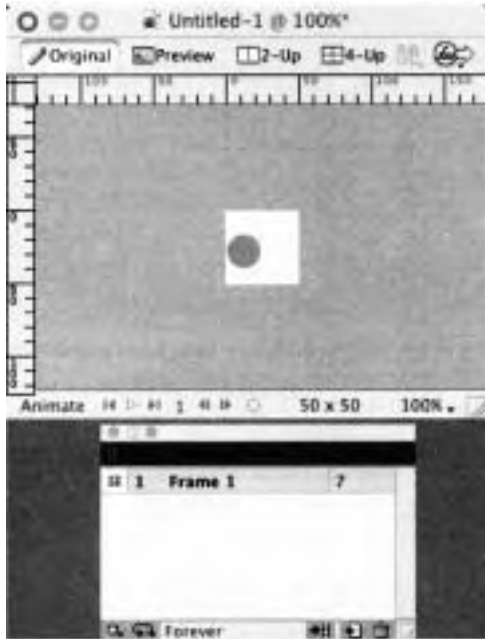


Figure 9.11 In the first animation frame, the ball is at the bottom of the image.

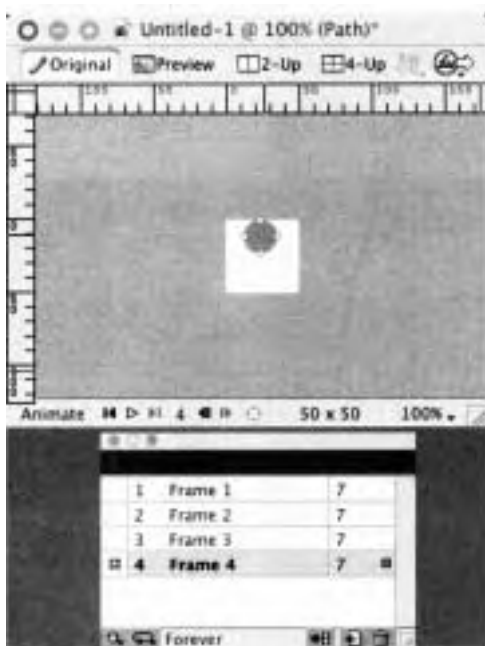


Figure 9.12 In the last frame, the ball is on the left side of the screen, after having traveled to the right and top in previous frames.

3. Add an image to the first frame of the animation (Figure 9.11).
4. Add more frames, with images in each frame.

The easiest way to do this is to simply duplicate the first frame and then move the object from one frame to the next (Figure 9.12). You can also use the Frame palette to set the animation to play once or loop.
5. When you finish creating your animation, choose File > Export.

The Export dialog box opens.
6. Type the name of the image (making sure to preserve the .gif extension), navigate to the folder in which you want to save the image, and click Save.

To create a GIF animation in ImageReady:

1. In ImageReady, open a new document and set the canvas to the desired size (**Figure 9.13**).

As with all graphics, larger canvas sizes mean larger files and longer download times.

2. Add an image to the first frame of the animation (**Figure 9.14**).

3. Add more frames, with images in each frame.

The easiest way to do this is to simply duplicate the first frame and then move the object from one frame to the next (**Figure 9.15**). You can also use the Frame palette to set the animation to play once or loop.

4. When you have finished creating your animation, choose File > Save Optimized. The Save Optimized dialog box opens.

5. Type the name of the image (making sure to preserve the .gif extension), navigate to the folder in which you want to save the image, and click Save.



Figure 9.13 The New Document dialog box in ImageReady. The program assumes 72 dpi for all images.

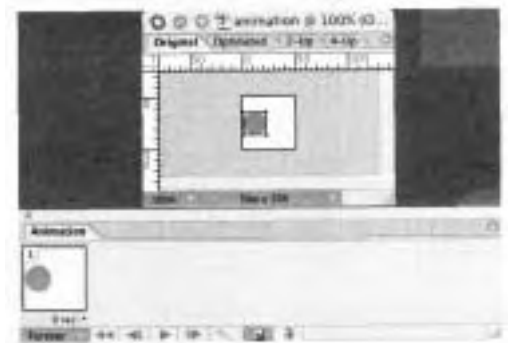


Figure 9.14 In the first animation frame, the ball is at the bottom of the image.

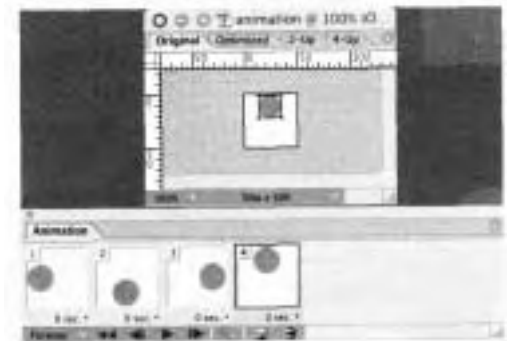


Figure 9.15 In the last frame, the ball is on the left side of the screen, after having traveled to the right and top in previous frames.

Code 9.4 In the HTML code, an `` link to the animated GIF is identical to an `` link to any GIF.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
 Transitional//EN" "http://www.w3.org/TR/
 xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=
        iso-8859-1" />
    <title>DHTML & CSS Advanced I
      Adding GIF Animation</title>
  </head>
  <body>
    
  </body>
</html>
```



Figure 9.16 The ball starts at the bottom, moves to the right, to the top, and to the left, and then starts all over again.

To add a GIF animation to your Web page:

```

```

Adding a GIF animation to a Web page (**Code 9.4**) works just like adding any other GIF image (**Figure 9.16**).

✓ Tips

If you don't already own ImageReady or Fireworks, an assortment of free or inexpensive GIF animation tools are available on the Web. For the Mac, I recommend GIFBuilder, and for Windows, I recommend GIF Construction Set. You can download both of these programs from Download.com.

Remember that although an animated GIF may occupy only as much space as a similar static GIF, each frame of the GIF is an entire image, adding that much more to the file size and download time.

What Is GIF Animation Good For?

GIF animations have been with us for a while—you have no doubt seen them gaping or blinking at you on pages around the Web. Yes, they add noise to more Web pages than I care to think about, but that doesn't mean they can't be used to make Web pages more informative, attractive, and dynamic. You just need to know how to use them effectively.

I have placed examples of each of the following ideas online—check out www.webbedenvironments.com/dhtml_css_advanced/gifanimate to see them in action.



Figure 9.17 This example acts like a normal JavaScript rollover, but the new image is an animated GIF that flashes on and off.

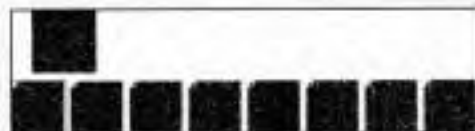


Figure 9.18 Each frame contains visual noise—which, when tiled in a background, looks like television static.

JavaScript rollovers. Most Web sites use JavaScript rollovers to change a button's graphic when the visitor places the mouse over it (**Figure 9.17**). This technique gives visitors visual feedback about the button they are about to click. Because these graphics are GIFs, you can use an animated GIF for the rollover. I created two graphics, one for the button's off state and one for the ready state (when the mouse rolls over it). The off state is a static arrow; the ready state flashes the arrow on and off.

Backgrounds. For a long time, animated GIFs didn't work as background graphics. Or rather, they wouldn't animate: You'd see either the first or last frame of the animation (depending on the browser) as the tiling background graphic, but nothing else happened. Most browsers, however, now support animated background GIFs (**Figure 9.18**). Animation in a background can be highly distracting, though, so use this feature sparingly.

continues on next page

What Is GIF Animation Good For? (continued)

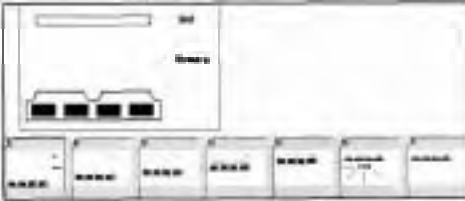


Figure 9.19 The illustration shows the movement needed to snap the memory chip into the slot.



Figure 9.20 These are some snapshots I took in England that I turned into a GIF slide show.

Illustrations. One of the most obvious, yet overlooked, uses for animated GIFs is to improve illustrations (**Figure 9.19**). For example, illustrating a step-by-step process (such as how to install RAM in your computer) is difficult with static images. You can demonstrate the process far more clearly by using GIF animation to show the actual movement required to insert a RAM module and snap it into place.

Slide shows. If you want to display several images in a limited amount of space, you can use GIF animation to set up a slide show (**Figure 9.20**). Rather than show fluid motion, as in the preceding examples, you simply place each image in its own frame and then set how much time you want to pause before the next image appears. If you loop this setup, the slide show will continue infinitely. Depending on the images you are using, though, looping can lead to large files that take forever to download. Remember that each frame of a GIF animation is a GIF, and the file size adds up as you include more frames.

Adding Ambient Sound

Most modern browsers come with built-in sound capabilities, making it easy to embed in a Web page sound files that can play in the background (Figure 9.21), or that users can control.

This section shows you how to place the sound file in a Web page by using two tags—`<object>` and `<embed>`—simultaneously. Although the `<embed>` tag is more universally recognized by browsers, the `<object>` tag is the standardized tag and will be used increasingly in the future. Adding both versions now shouldn't interfere with playback and should ensure the greatest compatibility.

To add ambient sound to a Web page:

1. `ambient.wav`

First, you need a sound file to be played (Figure 9.22). The file can be in a variety of formats, including WAV (.wav), MIDI (.mid), AU (.au), and QuickTime (.mov).



Figure 9.21 Although no visual cues let you know that a sound is playing on this page, you would hear the sound of distant alien winds in the background.

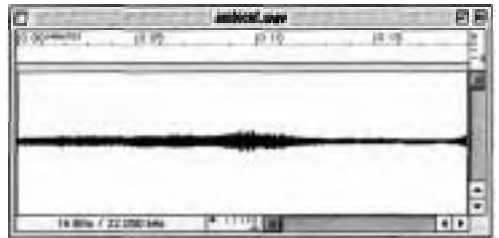


Figure 9.22 A visual representation of a sound file. This is what you would see if you opened the file in a sound editing program.

Table 9.1

Important Sound Attributes		
PARAMETER NAME	VALUES	WHAT IT DOES
src	{url}	The location of the sound file
hide	true false	Whether the controls show up on the screen
loop	true false	Whether the sound repeats
autostart	true false	Whether the sound plays immediately after the page loads
pluginpage	{url}	Location of page to get the sound plug-in

Code 9.5 To make sure that the sound file is loaded in older browsers, you need to use both the `<embed>` and `<object>` tags together.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=
        iso-8859-1" />
    <title>DHTML for the WWW I
      Adding Ambient Sound</title>
    <style type="text/css">
body {
  background-linage: url(mars1_pathfinder.jpg);
  color: red;
}
</style>
  <body>
    <h1>Welcome to my world...</h1>
    <object height="100" width="100">
      <param name="htdden" value="true" />
      <param name="loop" value="true" />
      <param name="autostart" value="true" />
      <param name="src" value="ambient.wav" />
      <embed height="100" hidden="true"
        src="ambient.wav" width="100"
        autostart="true" loop="true">
    </embed>
    </object>
  </body>
</html>
```

2. `<object>`

In the body of your HTML document, add the `<object>` tag to open an object container (**Code 9.5**).

3. `<param name="hidden" value="true">`

Within the object container, add the parameters for the object, using the `<param>` tag. **Table 9.1** lists common parameters and values to define for sound.

4. `<embed src="ambient.wav" autostart="true" loop="true" hidden="true">`

Within the object container, add the `<embed>` tag, defining its attributes the same way you defined the parameters in step 3.

5. `</object>`

Close the object container.

✓ Tips

The download times for sound files are high even for a short clip. (The 10-second loop I used in this example is larger than 800 KB.) You can further compress the sound in other ways, such as by using MP3 files, but not all browsers support newer, better sound formats.

Not only could you annoy your visitors with long download times, but to many people, sound files are the aural equivalent to the `<blink>` tag. If you use sound, make sure it's for a good reason.

Creating Transparent Graphics in PNG Format

The most promising alternative to GIF is PNG (portable network graphics), a standardized, unpatented graphic format supported by the World Wide Web Consortium (W3C). This format has several features that make it a superior choice over GIF.

Most important, this format allows you to create transparent graphics. Rather than being solid, the graphics' colors are translucent, allowing content below them to show through—something that GIF cannot do (Figure 9.23). The following tasks describe how to create PNG graphics using either Adobe ImageReady or Macromedia Fireworks.

To create a PNG graphic in ImageReady:

1. Open a new graphic document in RGB mode at 72 dpi, and set the background to transparent (Figure 9.24).
Allow extra working space in defining the initial size of the graphic.



Figure 9.23 Both images are the same, but the left graphic was saved as a GIF and the right as a PNG. Notice that the GIF image covers the text even in transparent areas, while the text shows through the PNG.

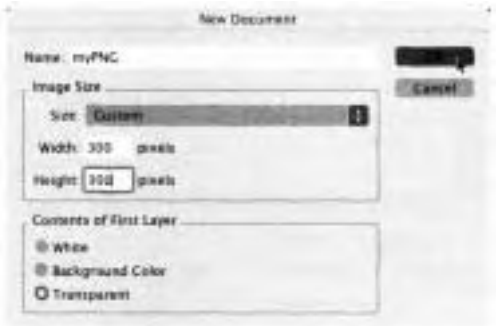


Figure 9.24 In ImageReady's New Document dialog box set the new image to be 72 dpi, RGB, with a transparent background.

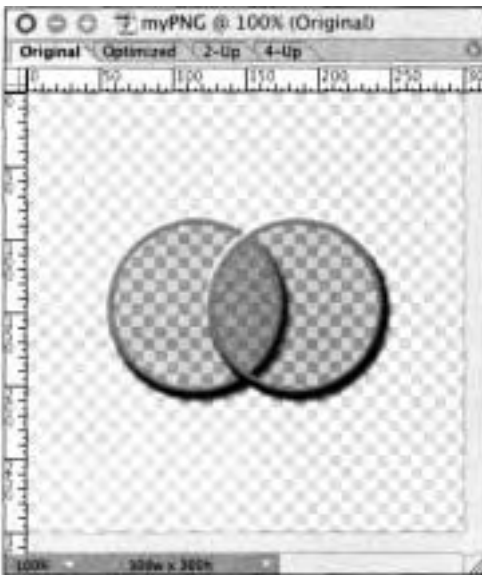


Figure 9.25 The image in ImageReady: two intersecting rings with their interior opacity set to 25%. The rings also have a slight drop shadow.

2. Create a graphic.

The graphic can use drop shadows and other transparency effects, and different layers can have different opacities (**Figure 9.25**). When the image is complete, crop it or use the slice tool to get the desired size.

3. From the Preset drop-down menu in the Optimize palette, choose PNG-24, check the Transparency checkbox, and click OK (**Figure 9.26**).

4. Choose Save ▸ Save Optimized As. The Save Optimized As dialog box opens.

5. Type the name of the image (preserving the .png extension), navigate to the folder in which you want to save the image, and click Save (**Figure 9.27**).

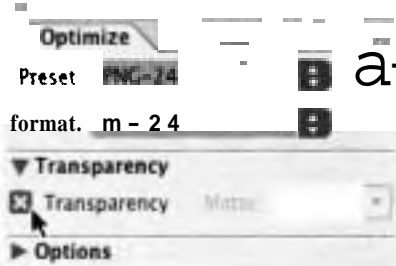


Figure 9.26 ImageReady's Optimize palette.



Figure 9.27 ImageReady's Save Optimized As dialog box.

To create a PNG graphic in Fireworks:

1. Open a new graphic document at 72 dpi, and set the background to Transparent (**Figure9.28**).

Allow extra working space in defining the initial size of the graphic.

2. Create a graphic.

The graphic can use drop shadows and other transparency effects, and different layers can have different opacities (**Figure9.29**). When the image is complete, crop it or use the slice tool to get the desired size.



Figure 9.28 In Fireworks's New Document dialog box, set the new image to be 72 dpi with a transparent background.

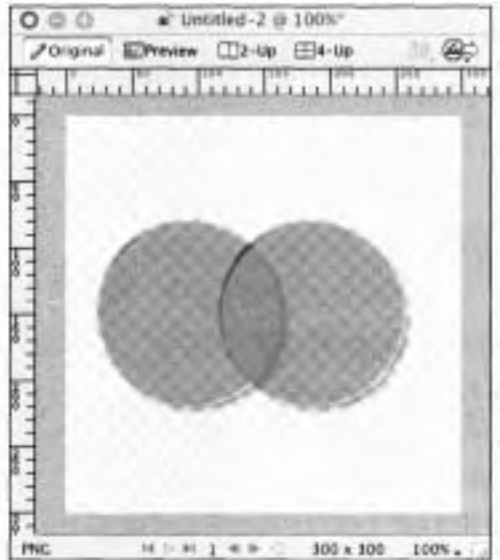


Figure 9.29 The image in Fireworks: two intersecting rings with their interior opacity set to 25%. The rings also have a slight drop shadow.

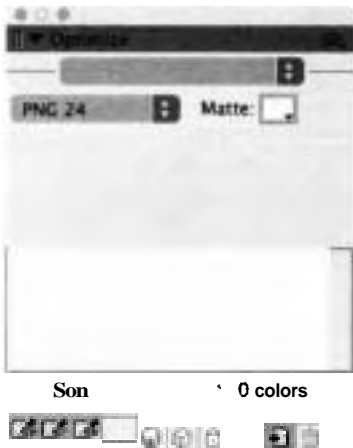


Figure 9.30 In Fireworks's Optimize control panel, set the image format to PNG 24.

3. In the Optimize control panel, set the format to PNG 24 (Figure 9.30). If you don't see the Optimize control panel, select **Window** » **Optimize** and it should appear.
4. Choose **File** » **Export Preview**.
The Export Preview dialog box opens (Figure 9.31).
5. Make sure the Format drop-down menu is set to PNG 24, then click **Export**.
The Export dialog box opens.
6. Type the name of the image (preserving the **.png** extension), navigate to the folder in which you want to save the image, and click **Save** (Figure 9.32).



Figure 9.31 In Fireworks's Export Preview dialog box, you can set the image format to PNG 24 if you forgot to do it earlier.



Figure 9.32 Fireworks's Export dialog box.

To use a PNG graphic on a Web page:

- ``

Add the `` tag, and use the URL of the PNG image you created in the preceding exercises as the source (**Code 9.6**).

Because PNG is just another image format, you can use the `` tag to include PNG graphics. Since some browsers do not support this format fully—or even at all—you may want to include `alt` text.

✓ Tips

Unfortunately, the PNG format has two major drawbacks that have prevented it from wide adoption: (1) It cannot create animations, and (2) Internet Explorer for Windows does not support most of its best features, such as transparency.

With PNG, you no longer have to worry about the background color you set when you created the image. The antialiased pixels adapt to any background.

Code 9.6 To add a PNG image to your Web page, simply use the standard `` tag.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
 Transitional//EN" "http://www.w3.org/TR/
 xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=
 iso-8859-1" />
    <title>DHTML & CSS Advanced I
      PNG Graphic Format</title>
    <style type="text/css"><!--
body {
  color: black;
  font-size: 14px;
  font-family: times, "Times New Roman",
    Georgia, serif;
  line-height: 16px;
  background-color: white;
}
#overlay {
  visibility: visible;
  position: absolute;
  top: 25px;
  left: 25px;
}
--></style>
</head>
<body>
  <div id="overlay">
    
    
    </div>
    <p>Quamquam mihi semper frequens
      > conspectus vester multo
      > iucundissimus...</p>
  </body>
</html>
```

Special Effects in Internet Explorer for Windows

In addition to the opacity filter presented earlier in this chapter, Internet Explorer for Windows has several other special effects that are unique to it. It's important to remember that these effects won't work in any other browsers, so you should avoid using them unless their absence will not interfere with visitors' use of your Web site.

Adding effects filters (IE Windows only)

Blurring can make an element—text or graphic—look as though it's moving by simulating the blurring caused by motion (**Figure 9.33**).



Figure 9.33 The griffon blurs across the screen.

To blur an element:

1. `function blurOn(currStrength, currDirection) {...}`
Add the `blurOn()` function to the JavaScript in the head of your document. This function recursively applies the blur filter to the image that has the `blurMe` ID until it reaches a strength of 360. The strength attribute specifies how far the element should be blurred (**Code 9.7**).
2. `<body onLoad="blurOn(15,15) ">`
Add an `onLoad` event handler that runs the `blurOn()` function from step 1.
3. ``
In the body of your document, add the image you want to blur. In the `` tag, add the `style` attribute with the blur filter. Set its initial strength to 0; set direction to 0 as well.

Code 9.7 The `blurOn()` function causes the image to blur across the screen slowly.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Transitional//EN" "http://www.w3.org/TR/
  xhtml1/DTD/xhtml1-transitional.dtd">
<!-- INTERNET EXPLORER FOR WINDOWS ONLY -->
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=
      iso-8859-1" />
    <title>DHTML & CSS Advanced I
      Making an Element Blur</title>
    <script>!--
function blurOn(currStrength,
  currDirection){
  if (document.all && currStrength < 360)
  {
    currStrength += 1;
    currDirection += 1;
    document.all.blurMe.style.filter =
    "blur(strength=" + currStrength +
    " , " + currDirection + ")";
    setTimeout("blurOn(" + currStrength +
    + " , " + currDirection + ")",100);
  }
}
// --></script>
</head>
<body onLoad="blurOn(15,15)">
  The blur will only work in Internet
  Explorer 4 and above for Windows.
  
  </body>
</html>
```

Fading between objects (IE Windows only)

By far the most popular way to show a transition between two elements is the fade. For Internet Explorer for Windows, you can set up two images (or a pair of any screen elements, for that matter) and fade one out as the other fades in (Figure 9.34).



Figure 9.34 One image fades in as the other fades out.

To fade elements:

1. `var isFade = 0;`

In your JavaScript, Initialize the variable `isFade` (Code 9.8), which will keep track of whether the objects have done their transition yet (`isFade=1`).

2. `function fadeElement(){...}`

Add the function `fadeElement()` to the JavaScript in the head of your document. This function applies the `blendTrans` filter between the current source image for `image1` (`alice04.gif`, in the example) and its replacement, `alice05.gif`.

3. ``
`onclick="fadeElement()">>`

In the style attribute for `image1`, add a call to Internet Explorer's `blendTrans` filter, and set its fade duration to 3 (seconds). This filter will not be acted upon until the image is clicked and the `fadeElement()` function is executed.

Code 9.8 The `fadeElement()` function, when activated causes `image1` to fade from its initial source (`alice04.gif`) to its new source (`alice05.gif`).

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
 Transitional//EN" "http://www.w3.org/TR/
 xhtml1/DTD/xhtml1-transitional.dtd">
<!-- INTERNET EXPLORER FOR WINDOWS ONLY -->
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=
        iso-8859-1" />
    <title>DHTML & CSS Advanced I
      Fading between Elements</title>
    <script><!--
var isFade = 0;
function fadeElement() {
  if (document.all && isFade == 0) {
    isFade = 1;
    alice.filters.blendTrans.Apply();
    alice.src = "alice05.gif";
    alice.filters.blendTrans.Play();
  }
}
// --></script>
</head>
<body>
  This fade will only work in Internet
  Explorer 4 and above on Windows.
  <img id="alice" style="filter:
  >blendTrans(duration=3)"
  >onclick="fadeElement()"
  >src="alice04.gif" />
<br />
</body>
</html>
```

Adding page transitions (IE Windows only)

When you're jumping between Web pages, it's always a bit disconcerting when the first page blinks out and is slowly replaced by the next page, piece by piece. Using the `RevealTrans()` filters, however, you can produce transitions between Web pages that are more cinematic (**Figure 9.35**).

Table 9.2 lists the transition-filter effects that are available.



Figure 9.35 The page is shown in mid-transition as the new page with the King and Queen spins in.

Table 9.2

IE Transition Filters	
TRANSITION	REFERENCE #
Box In	0
Box Out	1
Circle In	2
Circle Out	3
Wipe Up	4
Wipe Down	5
Wipe Right	6
Wipe Left	7
Vertical Blinds	8
Horizontal Blinds	9
Checkerboard Across	10
Checkerboard Down	11
Random Dissolve	12
Split Vertical In	13
Split Vertical Out	14
Split Horizontal In	15
Split Horizontal Out	16
Strips Left Down	17
Strips Left Up	18
Strips Right Down	19
Strips Right Up	20
Random Bars Horizontal	21
Random Bars Vertical	22
Random	23

To set up a transition between Web pages:

1. `<meta http-equiv="Page-Enter"`

Set up a `<meta>` tag in the head of your document (**Code 9.9**). Set the `http-equiv` attribute to `Page-Enter`.

2. `content="RevealTrans`

`> (Duration=20, Transition=2)" />`

Set the `content` attribute for the `<meta>` tag in step 1 to execute the `RevealTrans()` filter, and set its duration to 20 (seconds). Then add the name of a transition from Table 9.2. For this example, I chose the Circle In effect, to have the page spiral in.

3. `<meta http-equiv="Page-Exit"`

`> content="RevealTrans
> (Duration=20, Transition=3)" />`

To set a filter to run when the visitor leaves this page, type another `<meta>` tag for `Page-Exit`. For this one, I selected the Circle Out filter.

Code 9.9 Here, the `<meta>` tag is used to execute transition filters when the document is opened and when it is closed.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
 Transitional//EN" "http://www.w3.org/TR/
 xhtml1/DTD/xhtml1-transitional.dtd">
<!-- INTERNET EXPLORER FOR WINDOWS ONLY -->
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Page-Enter" content="
      RevealTrans(Duration=20,
      Transition=2)" />
    <meta http-equiv="Page-Exit"
      content="RevealTrans
      (Duration=20, Transition=3)" />
    <meta http-equiv="content-type"
      content="text/html; charset=
      iso-8859-1" />
    <title>DHTML & CSS Advanced I
      Transitions Between Pages</title>
  </head>
  <body style="background-color:red">
    This transition will only work in
    Internet Explorer 4 and above for
    Windows.
    <center>
      <a href="page2.html">
    <br />
    Next Picture --> </a></center>
  </body>
</html>
```

DATABASES

Few Web sites today are straight DHTML and CSS. They depend on back-end data sources to power what is displayed on the page. The DHTML and CSS that has been covered thus far in this book will empower you to take raw data and turn it into an appealing user interface.

Often it feels like there are two conflicting goals in Web site creation: design and development. With design your concern is to create rich and intuitive interfaces. With development your goal is pure function and creating a somewhat manageable interface for data. The true magic happens when you bring these two concepts together and develop Web sites that are both rich in content and intuitive in interface.

In this chapter I'll show you how to become the magician who turns a well-designed static Web site into a dynamic data-driven Web site using XML. Once you have a good understanding of XML, I'll show you the magic that occurs when you use PHP with the database MySQL.

Understanding XML

Extensible markup language (XML) is a versatile data-storage format. XML can hold complex data structures while maintaining a rather simple format (plain text). XML is completely customizable to your needs, or can follow a predetermined format specified by an XML Schema Definition (XSD). A full reference to XML and XSD can be found at the W3 Consortium's Web site.

In this section we'll create an XML document that will be used in the following sections to create a dynamic blog. (See the sidebar "What Is a Blog?" for more information about blogs.)

To create an XML document:

1. `<?xml version="1.0" encoding="iso-8859-1" ?>`

Add an XML document tag (Code 10.1). The document tag contains the XML specification you are using (`version="1.0"`) and the encoding of the text contained within the XML document (`encoding="iso-8859-1"`). See Table 10.1 for a list of common XML encoding values.

Code 10.1 This XML document contains a parent `<Blog>` tag (or node) and two child `<Entry>` nodes.

```
Code a
<?xml version="1.0" encoding="iso-8859-1"?>
<Blog>
  <Entry>
    <Title>Where do you come from?</Title>
    <Date>6/2/2004</Date>
    <Time>10:32 AM</Time>
    <Author>Red Queen</Author>
    <Content>Where do you come from?
      And where ore you going? Look up,
      speak nicely, and don't twtddle your
      fingers all the time.</Content>
  </Entry>
  <Entry>
    <Title>The Garden of Ltve Flowers</
    Title>
    <Date>5/15/2004</Date>
    <Time>9:24 AM</Time>
    <Author>Alice</Author>
    <Content>Does she ever come out here?
      </Content>
  </Entry>
</Blog>
```

Table 10.1

XML Encoding Types	
ENCODING	DESCRIPTION
UTF-8	Unicode Transform Format 8 was created by the Unicode Consortium (www.unicode.org) as a standard for encoding characters. Most English characters can be encoded this way.
UTF-16	The Unicode Consortium created this larger encoding standard to handle the most common characters (over 60,000 total) for a wide variety of languages.
ISO-8859-1	Also known as ASCII and Latin-1, this is the basic character set known to most people. It includes all the characters on a US keyboard, as well as some European keyboards (Spanish, French, etc.). ISO-8859-1 includes only ■-byte characters.

What Is a Blog?

"Blog" is short for "Web log." A blog is the modern-day diary. Unlike the original diary (a personal, private journal), most blogs are publicly viewable with no registration required.

The format for a blog is a dated list of journal entries, sometimes with links and pictures contained within each entry. Most blogs put the most recent entry at the top, with a title and author name, and they often provide a method for the reader to leave comments or read others' comments. In fact, blogs have become so common now that many people read 5–10 different blogs each day for updates or news.

Many blogs now also offer Really Simple Syndication (RSS) feeds for their content. RSS is an XML-based format for sharing Web content. RSS takes blogs to a whole new level. Various companies and some individuals in the open-source community have created RSS aggregators that allow you to view all your favorite RSS feeds in one simple interface.

Some of my favorite blogs

The Laporte Report
tir.leoville.com

Kevin Rose
www.kevinrose.com

webbedENVIRONMENTS
www.webbedenvironments.com
The Rantings of Daniel C. Rymer
zhunzi.blogspot.com

2. <Blog>...</Blog>

Add the <Blog> node. At the highest level of an XML document, there is only one node (or tag in XHTML terminology). This is referred to as the root node. The <html> tag inside an XHTML document is also a root node because there are no tags that exist outside of it.

3. <Entry>...</Entry>

Add two <Entry> nodes to the <Blog> root node. Each <Entry> node will contain information pertaining to that specific node. Similarly, the <table> tag in XHTML contains information (rows, cells, and so on) specific to that table.

4. <Title>...</Title>

Add the <Title> tag, along with <Date>, <Time>, <Author>, and <Content> tags. Each one of these nodes will contain specific content about the entry.

Accessing XML Data in Internet Explorer for Windows

XML data islands provide you with a (relatively) simple way to embed data in your Web page. Microsoft Internet Explorer 5.0 and above provide some nice functionality to bind that data directly into a table for display. Unfortunately these features are not cross-browser compatible and only work in Internet Explorer 5.0 and above. Still, XML data islands are a useful technology if Internet Explorer for Windows is your only concern.

Basically an XML data island is an embedded data store. Similar to a cascading style sheet that can be embedded in the page or linked to externally, an XML data island can be embedded into the page or linked to as an external source file. As you know, cascading style sheets contain information about how to present or format information. XML data islands can provide the actual information to be presented and formatted.

In this example, we will create an XML data island using the XML document we created in the last section (Code 10.1). Using a style sheet created for this blog (Code 10.2), the XML data island will fill a table dynamically in the user's browser to display our blog entries (Code 10.3).

When you look at the resulting page (Figure 10.1) you'll notice not only the two blog entries are read from the first XML data island, but also that the book title, chapter, and section are read from the second data island. To do this, we will be binding the data directly to the first table in the HTML code (`datasrc="#header"`).

Code 10.2 This simple CSS file adds a little splash of color to the blog. It will be used throughout this chapter.

```
body {
    font-family: arial, helvetica, sans-serif;
    font-size: 12px;
}
.header {
    background-color: #e0edfd;
    border: 1px solid #6699cc;
    font-weight: bold;
}
.blog {
    background-color: #e0edfd;
    border: 1px solid #6699cc;
}
.title {
    font-weight: bold;
}
```

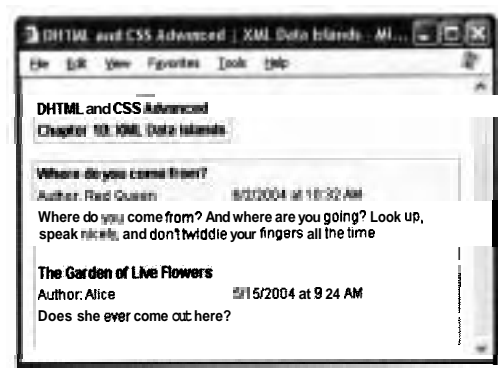


Figure 10.1 The two blog entries displayed from the XML data island.

Code 10.3 Two embedded XML data islands are added to a simple XHTML page.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
  = xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <title>DHTML & CSS Advanced I XML Data Islands</title>
  <link rel="stylesheet" type="text/css" href="blog.css" />
</head>
<body>
  <xml id="blogData" src="blog.xml"></xml>
  <xml id="header">
    <HeaderDoc>
      <Book>DHTML and CSS Advanced</Book>
      <Chapter>18</Chapter>
      <Section>XML Data Islands</Section>
    </HeaderDoc>
  </xml>
  <table class="header" id="tblHeader" border="0" datasrc="#header">
    <tr>
      <td><span datafid="Book"></span></td>
    </tr>
    <tr>
      <td>
        Chapter <span datafid="Chapter"></span>:
        <span datafid="Section"></span>
      </td>
    </tr>
  </table>
  <br />
  <table class="blog" id="tblBlog" border="0" datasrc="#blogData">
    <tr>
      <td colspan="2"><span class="title" datafid="Title"></span></td>
    </tr>
    <tr>
      <td>Author:&#160;<span datafid="Author"></span></td>
      <td><span datafid="Date"></span>&#160;at&#160;<span datafid="Time"></span></td>
    </tr>
    <tr>
      <td colspan="2"><span datafid="Content"></span><br /><br /></td>
    </tr>
  </table>
</body>
</html>
```

To create an XML data island:

1. Create an external style sheet for your Web page, including styles to be used for the XML. Call the file `blog.css` (Code 10.2).

2. `<link rel="stylesheet" type="text/css" href="blog.css" />`

Start a new XHTML file (Code 10.3) and include a link to the style sheet from step 1 (Code 10.2).

3. `<xml id="blogData" src="blog.xml">`
`</xml>`

To call the external XML file, add an `<xml>` tag with an ID attribute (`id=". . ."`) to give the XML a unique ID. This is required so that Internet Explorer can locate the XML data island. Add a source attribute (`src=" . . ."`) to tell the browser where the XML file is located. If the source attribute is not included, Internet Explorer will expect the XML code to exist between the opening and closing `<xml>` tags.

4. `<table class="blog" id="tblBlog" border="0" datasrc="#blogData">`

Create a table. Add the data source attribute (`datasrc=" . . ."`) to tell Internet Explorer where the XML data source is located. The name must match the ID given to the XML data island. Do not forget to put the number sign (#) in front of the name; this is required.

5. ``

Inside the table, create a template that will be repeated using the data field attribute (`datafld=" . . ."`) of the `` tags. The XHTML tags between the opening and closing table tags will be repeated for each XML child node (`<Entry>`) of the parent node (`<Blog>`).

Tip

The source attribute (`src="http://MyServer/ReturnXml.php"`) is not limited to a static XML file as shown in the example. Using a fully qualified URL allows you to obtain XML data from an XML Web service, or any variety of other network resources that return XML.

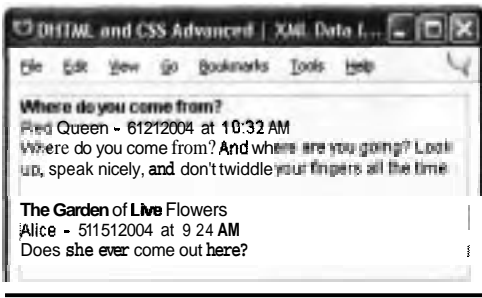


Figure 10.2 The two blog entries displayed from the XML data island using JavaScript.

Code 10.4 In this XHTML page, the embedded XML data island is rendered using the DOM.

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Transitional//EN" "http://www.w3.org/TR/
  xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xml:lang="en" lang="en">
<head>
  <title>DHTML and CSS Advanced :
    XML Data Islands</title>
  <link rel="stylesheet" type="text/css"
    href="blog.css" />
  <script language="javascript">
    function bindBlogData(divId, template,
      xmlIslandId) {
      var xmlIsland = document.
        getElementById(xmlIslandId);
      var blogEntries = xmlIsland.
        getElementsByTagName("entry");
      var template = document.
        getElementById(template);
      var blogDiv = document.
        getElementById(divId);
      for (var entry=0; entry<blogEntries.
        length; entry++) {
        var tempDiv=createEntry
          (blogEntries[entry], template)
        blogDiv.appendChild(tempDiv);
      }
    }
  </script>

```

(code continues on next page)

Accessing XML Data with Mozilla and Internet Explorer

In the previous section I showed you how to embed an XML data island, and then access it using propriety techniques in Internet Explorer. In this section I'll show you how to access the XML data with JavaScript. The techniques explained in this section will work in both Internet Explorer and Mozilla.

These are two completely different (but similar) techniques. IE includes built-in functionality, but if you want to use both Mozilla and Internet Explorer you have to write your own JavaScript to handle it all. Unfortunately, the cross-browser compatibility with Opera and Safari is disappointing. It seems most people shy away from it because you are currently limited to Internet Explorer and Mozilla.

In this example, we'll write JavaScript code (**Code 10.4**) to access the embedded XML data using the Document Object Model (DOM). The output looks very similar to that produced by the previous section (**Figure 10.2**).


```

function createEntry(currentEntry,template) {
    var tempDiv=template.cloneNode(true);
    var spanTags = tempDiv.getElementsByTagName("span");
    for (var i=0; i<spanTags.Length; i++) {
        var xmlvalue=spanTags[i].getAttribute("xmlvalue");
        if (xmlvalue!=null && xmlvalue!="") {
            var value=currentEntry.getElementsByTagName(xmlvalue);
            var node=document.createTextNode(value[0].firstChild.nodeValue);
            spanTags[i].appendChild(node);
        }
    }
    tempDiv.style.display='inline';
    return(tempDiv);
}
</script>
<style>
    xml {display: none}
</style>
</head>
<body onload="bindBlogData('divBlog', 'blogTemplate', 'blogData')">
    <xml id="blogData">
        <blog>
            <entry>
                <entrytitle>Where do you come from?</entrytitle>
                <date>6/2/2004</date>
                <time>10:32 AM</time>
                <author>Red Queen</author>
                <content>Where do you come from? And where are you going? Look up, speak nicely,
                    and don't twiddle your fingers all the time.</content>
            </entry>
            <entry>
                <entrytitle>The Garden of Live Flowers</entrytitle>
                <date>5/15/2004</date>
                <time>9:24 AM</time>
                <author>Alice</author>
                <content>Does she ever come out here?</content>
            </entry>
        </blog>
    </xml>
    <div id="divBlog" class="blog" style="display:block">
        <div id="blogTemplate" style="display:none">

```

```

<div class="title"><span
- xmlvalue="entrytitle"
- ></span></div>
<div>
  <span xmlvalue="author"></span>
  &#160;-&#160;
  <span xmlvalue="date"></span>
  &#160;at&#160;
  <span xmlvalue="time"></span>
</div>
<div>
  <span xmlvalue="content"></span>
  <br /><br />
</div>
</div>
</body>
</html>

```

To access XML with JavaScript:

1. `<xml id="blogData">`

One of the first things you will notice in this example is that the source (`src="blog.xml"`) is not on the XML data island; instead the XML is included in the page code (Code 10.4). This is because that property is specific to Internet Explorer. Most browsers do not know what to do with that `<xml>` tag, so you have to add a style to hide it from display. You will also notice that the case has changed on the XML to match the XHTML specification. In the previous section you may have noticed that the tags were initially capitalized, but in this section they are all in lower case. Since this browser thinks this is just XHTML, we need to conform to that standard.

2. `<div id="divBlog" class="blog" style="display:block">`

Add the main `<div>` tag that will hold our blog entries.

3. `<div id="blogTemplate">`

Add a second `<div>` tag to the main `<div>` tag to specify a template for the blog. We will clone this template in the JavaScript code to generate multiple entries in the main `<div>` tag.

4. ``

Add a custom attribute (`xmlvalue`) to the `` tags that will be used to place the appropriate XML value into the cloned template.

5. `<link rel="stylesheet"`

`· type="text/css" href="blog.css" />`
Add an external style sheet reference to reuse the same style sheet from the previous section (Code 10.2).

6. `<script language="javascript">`

Add the `<script>` tag for the JavaScript code. The most complex piece of code is the JavaScript, which renders the XML data into the main `<div>` tag (`<div id="divBlog">`).

7. `function bindBlogData`

Add the `bindBlogData()` function. It locates the data island, the template, and the main `<div>` tag. It then loops through the data island entries and inserts them into that main `<div>` tag using a clone of the template.

8. `createEntry(blogEntries[entry], template)`

Add the `createEntry()` function. The `bindBlogData()` function calls the `createEntry` function for each entry, which makes a copy of the `blogTemplate <div>` and copies the values from the XML entry into the `` tags.

9. `blogDiv.appendChild(tempDiv);`

Append the modified copy of the template to the main `<div>` tag.

10. `<body onload="bindBlogData ('divBlog', 'blogTemplate', 'blogData')">`

Add the `onload` event to your `<body>` tag. When the page loads, it calls the JavaScript function to start the process.

Tips

Try to avoid tags in your XML data that conflict with XHTML tags such as `<title>`. Sometimes they create problems that are very hard to debug.

The `xmlvalue` attribute can be called anything, but it is wise to avoid common XHTML attribute names such as `value`, `id`, or `name`.

Understanding MySQL

Although XML data islands (covered in the previous sections) can be powerful, they are limited by browser compatibility. The best way to avoid browser compatibility problems is to write code for a database standard that works on all browsers.

How can you write compliant code and still have a powerful data-driven Web site when visitors may have Internet Explorer for the Mac, Netscape for Windows, or any number of other browsers? The answer is to have the server do the dirty work for you using a database. The first thing you have to figure out is what technology is available on your server (you'll probably need to ask your server administrator).

Table 10.2

Basic SQL Commands	
COMMAND	DESCRIPTION
CREATE DATABASE	Used to create a database on the server
USE	Tells the SQL command processor which database you are working with
CREATE TABLE	Adds a new table to the database with the specified fields
ALTER TABLE	Used to change a table's schema or layout (number of fields, type of fields, etc.)
DROP TABLE	Deletes a table from the database
SELECT	Used to return records from the specified table(s)
INSERT	Creates records in a specified table
UPDATE	Updates values in the specified fields of a specified table
DELETE	Deletes record(s) from a specified table

Although there are a wide variety of databases on the market that you can use for the Web, I'll be using MySQL because it is an open-source database, free to download, and available on a wide range of platforms (Linux, Windows, Solaris, FreeBSD, Mac OS X, and many others). For more information about MySQL downloads and platforms, please visit the Web site www.mysql.com.

MySQL is a *relational database*, meaning it uses relationships to link groupings of data together. These groupings are known as *tables*. An example of a table would be a blog entry. It has a title field, an author field, a timestamp field, a field for the content, and some sort of unique identifier known as a *primary key* that distinguishes one entry from another. Another table is a comments table. This table has an author field, a timestamp field, a unique identifier (primary key), and something known as a *foreign key* that relates a specific comment to a specific record in the entry table. This type of relationship is what makes up a relational database.

MySQL uses a language known as Structured Query Language (SQL) to access data. You can use SQL to insert records, update records, delete records, and of course view (or select) records. This same language is what you use to create, change (or alter), and delete (or drop) the tables that hold the data. In future versions of MySQL you will be able to use SQL to write user functions (functions that you create in SQL) and stored procedures (another type of function in which you can store a complex query).

See **Table 10.2** for a listing of basic SQL commands, and a description of what they do. As you will see in the next section, none of these commands are case sensitive.

Creating Database Objects in MySQL

In this section we'll create the database and tables needed for a blog using MySQL. In a later section we'll use PHP to read the data from these tables and render XHTML.

There are several graphical tools that make creating MySQL easier; however they are not available on all platforms. These tools are available for download on the MySQL Web site. In this section, I'll cover the MySQL console, which is available to all platforms, and is included with the basic MySQL download.

To create the blog database:

1. CREATE DATABASE blog;

Create the database using the CREATE DATABASE command (**Code 10.5**).

2. USE blog

Tell MySQL that you want to use the blog database.

3. CREATE TABLE Entry (...)

Create a table for blog entries. The CREATE TABLE command creates a table named Entry in our blog database. This table has five fields (entry-id, title, author, content, and insert-date). Following each field name is the data type (INT, VARCHAR, TEXT, and TIMESTAMP).

The data type tells MySQL what type of data is to be stored.

A INT is an integer.

A VARCHAR is a variable-length character field with the size in parentheses.

A TEXT is similar to VARCHAR, but is designed to handle larger character strings (longer than 255 characters).

A TIMESTAMP is a date and time field populated with the current date and time if you do not specify a value.

Code 10.5 This is the log from the MySQL console that is used to create our blog database and insert some initial values.

```
mysql> CREATE DATABASE blog;
Query OK, 1 row affected (0.00 sec)

mysql> USE blog
Database changed

mysql> CREATE TABLE Entry (
    -> entry-id INT NOT NULL auto-increment
    = PRIMARY KEY,
    -> title VARCHAR(50) NOT NULL,
    -> author VARCHAR(50) NOT NULL,
    -> content TEXT NOT NULL,
    -> insert-date TIMESTAMP NOT NULL);
Query OK, 0 rows affected (0.06 sec)

mysql> CREATE TABLE Comment (
    -> comment-id INT NOT NULL
    *auto-increment PRIMARY KEY,
    -> entry-id INT NOT NULL,
    -> author VARCHAR(50) NOT NULL default '',
    -> content TEXT NOT NULL,
    -> insert-date TIMESTAMP NOT NULL,
    -> FOREIGN KEY (entry-id) REFERENCES
    = Entry(entry_id)
    -> ON DELETE CASCADE);
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO Entry
    -> (title, author, content, insert-date)
    -> VALUES (
    -> 'The Garden of Live Flowers',
    -> 'Alice',
    -> 'Does she ever come out here?',
    -> 2004.0515092400);
Query OK, 1 row affected (0.02 sec)

mysql> INSERT INTO Entry
    -> (title, author, content, insert-date)
    -> VALUES (
    -> 'Where do you come from?',
    -> 'Red Queen',
    -> 'Where do you come from? And where
    + are you going? Look up, speak nicely,
    + and don't twiddle your fingers all
    + the time.',
```

(code continues on next page)

Code 10.5 continued

```
mysql> INSERT INTO Comment
  -> (entry-id, author, content, insert-date)
  -> VALUES (
  -> 1,
  -> 'Rose',
  -> 'I daresay you'll see her soon.
  = She's one of the thorny kind.',
  -> 20040516191800);

Query OK, 1 row affected (0.0T sec)

mysql> INSERT INTO Comment
  -> (entry_id, author, content, insert-date)
  -> VALUES (
  -> 2,
  -> 'Alice',
  -> 'Only wanted to see what the garden
  was like, your Majesty--',
  -> 20040602110700);

Query OK, 1 row affected (0.00 sec)
```

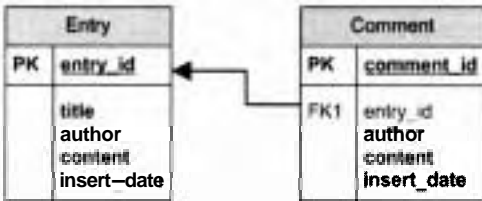


Figure 10.3 Diagram of a blog database showing the relationship between the two tables.

Tips

I often draw out my database schema (design) before I start writing my scripts. This provides a visual plan of what I am creating, and often reduces the time it takes to write the SQL scripts (**Figure 10.3**).

The visual tools that MySQL provides can be very helpful when you're first learning SQL. MySQL Control Center (available on the MySQL Web site) makes most MySQL tasks simple.

Following the data type are a variety of statements that tell MySQL a bit more about this field:

- **NOT NULL** tells MySQL that this is a required field and the record cannot be saved without some value in there. However, if you specify a default value and nothing is entered, with **NOT NULL** set, MySQL will automatically enter the default value.
- **AUTO-INCREMENT** tells MySQL to create a unique number to place in this field. It will start with 1, then 2, and continue on until the maximum value of the data type is reached.
- **PRIMARY KEY** tells MySQL that this field is the main key field, and should not have any duplicates.

4. CREATE TABLE Comment (...)

Create a table for blog comment entries. In the Comment table, there is an additional statement (**FOREIGN KEY**) that links the two tables together. The **FOREIGN KEY** command tells MySQL that the `entry-id` in the Comment table is the same as the `entry-id` in the Entry table. **ON DELETE CASCADE** tells MySQL to delete entries in this table automatically if they are removed from the Entry table. This lets MySQL do the work of cleaning things up after we remove an entry, rather than our having to take care of it manually.

5. INSERT INTO Entry (...) VALUES (...)

INSERT INTO Comment (...) VALUES (...)

To finish it off, add two records to each table so that you can have some sample data to work with. The **INSERT** statements tell MySQL to insert data into the tables in specified column order. The **VALUES** keyword tells MySQL that these are the values to go into the specified columns. We will cover this further in later sections.

Understanding PHP

"PHP" stands for "PHP: hypertext preprocessor." When a user's browser requests a Web page, it contacts the Web server, which then gets the requested page and sends it to the user's browser. PHP acts as a layer between the Web server and the actual file on the disk (hence, preprocessor). When a user's browser requests a PHP Web page, it contacts the Web server. The Web server contacts the PHP engine. The PHP engine reads the `file` from disk and scans it for PHP-specific code. It processes any code it finds, and then returns the resulting data stream to the Web server, which sends it to the user's browser. Simply renaming an `.html` file to `.php` causes this process to start.

PHP is an open-source Web-server programming language. Like MySQL, PHP is widely available on many platforms, including Windows, Mac OS X, and Linux. PHP also has built-in ability to connect and access data on MySQL databases. You can readily find cheap PHP Web hosts as well, which keeps your overall costs down when creating a data-driven Web site. I have chosen to use PHP for these reasons. If you prefer, you can use a number of other technologies to get the same results we'll achieve in the next few sections, including the top commercial players: Sun J2EE and Microsoft .NET.

All PHP code is contained between open (`<?>`) and close (`?>`) tags. Any text that is not contained between these tags is ignored by the preprocessor, and sent directly as a part of the data stream it sends to the server.

PHP has all the basic structures you would expect from a programming language. If you're familiar with JavaScript, you might even notice some similarities. All lines end with a semicolon (`;`). The basic structures such as `if` statements, Boolean operators, `while` statements, and function definitions are all the same. Also, variables do not have a predefined type (string, integer, float, and so on), which is similar to JavaScript's single variant (`var`) type.

However, PHP is very different from JavaScript. Some of the differences are cosmetic: For example, the variables begin with a dollar sign (`$`). Other changes are more fundamental: The functions, for example, are significantly different. Over time you'll become used to these differences. The largest difference, though, is that PHP code is processed by the server, instead of the client browser.

The PHP engine will deliver exactly the same results to every single browser, and therefore as long as the XHTML you render is cross-browser compatible, every PHP page will work in every browser! That opens the door to much simpler code, because you aren't constantly checking first what browser the client is using before executing client-specific JavaScript. That headache is, for you, the newest PHP programmer, history!

Using PHP with MySQL

One practical use of PHP is to combine it with a MySQL database and create a Web page based on a basic template using content from the database.

In this section, I'll show you how to use PHP to access the MySQL database we created earlier in the chapter. I'll also show you how to take the data from MySQL and generate dynamic XHTML pages on the fly with some very simple server-side commands.

We'll improve the blog you created earlier in this chapter, and escape from browser compatibility problems forever. Using PHP you can write dynamic code that will work in nearly every browser. You are on the road to great things. As you can see, our output is exactly the same as before (**Figure 10.4**).

continues on next page

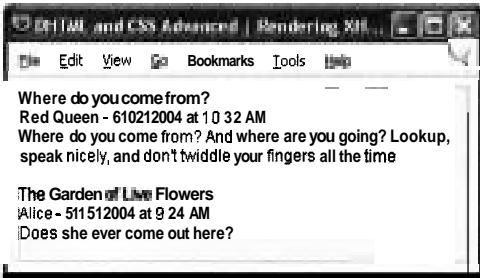


Figure 10.4 The two blog entries displayed using PHP and MySQL.

To create a PHP blog entry listing page:

1. <?

Add the opening PHP tag (**Code 10.6**). All PHP code is contained between open (<?) and close (?>) tags. The Web server looks for these tags and knows to process this code before sending the XHTML to the client's browser.

```
$connBlog = mysql_connect()
or die("Error connecting to
database: " . mysql_error());
```

Add the connection to the MySQL database. The function `mysql_connect()` attempts to connect to the local MySQL database. If an error occurs, the second option (or `die`) will run and the page will generate an error message.

3. `mysql_select_db("blog", $connBlog)`

Next, add the database selection code. Similar to the use `blog` statement we had to issue in MySQL, we need to tell PHP which database we'll be using on our MySQL server.

4. `$sql = "select title, author, *content, date_format(insert_date, '%c/%d/%Y at %l:%i %p') as insert-date from Entry order by +entry - iddesc;"`

Add the select SQL statement to a variable named `$sql`.

I like to keep my SQL statements in a variable because it makes the code easier to read. As you can see in this SQL statement, I have applied some formatting to the `insert-date` field so that it is returned as "06/02/2004 at 10:32 AM" instead of "20040602103200." I could easily have done this in the PHP code as well, but MySQL provides some nice formatting functions to handle this quickly.

Code 10.6 This page is using PHP script to access the MySQL blog database and render the data dynamically.

```
<?
$connBlog = mysql_connect()
    or die("Error connecting to database: " .
        mysql_error());
mysql_select_db("blog", $connBlog)
    or die("Error accessing database: " .
        mysql_error());
$sql = "select title, author, content,
date_format(insert_date, '%c/%d/%Y at %l:%i
%p') as insert-date from Entry order by
entry-id desc;";
$entries = mysql_query($sql)
    or die("Error accessing database: " .
        mysql_error());
$blogHTML="";
if (mysql_num_rows($entries) >= 0) {
    while ($entry = mysql_fetch_
        assoc($entries)) {
        $blogHTML = $blogHTML .
            buildDiv($entry);
    }
}
mysql_free_result($entries);
mysql_close($connBlog);
function buildDiv($entry) {
    $div="";
    $div=$div . '<div class="title">';
    $div=$div . $entry["title"];
    $div=$div . '</div>';
    $div=$div . '<div>';
    $div=$div . $entry["author"];
    $div=$div . '&#160; -&#160;';
    $div=$div . $entry["insert_date"];
    $div=$div . '</div>';
    $div=$div . '<div>';
    $div=$div . $entry["content"];
    $div=$div . '</div><br />';
    return $div;
}
?>
```

(code continues on next page)

Code 10.6 continued

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
 Transitional//EN" "http://www.w3.org/TR/
 xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
 xml:lang="en" lang="en">
<head>
  <title>DHTML and CSS Advanced I
  Rendering XHTML with PHP</title>
  <link rel="stylesheet" type="text/css"
 href="blog.css" />
</head>
<body>
  <div id="divBlog" class="blog"
 style="display:block">
    <?= $blogHTML ?>
  </div>
</body>
</html>
```

Tips

Proper error trapping can really change the way an application flows. Any time you connect to a remote data source, it's good practice to check for errors.

The more data manipulation you can do in PHP, the less you have to worry about browser compatibility. This is because you'll typically have much less JavaScript running in the client browser.

- `$entries = mysql_query($sql)`
Execute the SQL statement and store the results in a variable named `entries`. You'll notice that I have also added some error-trapping functionality to this code.
- `if (mysql_num_rows($entries) > 0) {`
`while ($entry = mysql_fetch_`
`assoc($entries)) {`
Add some code to check that you received at least one entry back from the query. Then add the code to loop through the entries. Notice that inside the loop I just call a function to handle each entry.
- `function buildDiv($entry)`
Add the `buildDiv()` function, which creates a string that represents the XHTML `<div>` tag for the entry. This function reads the values from the query results and places them into the appropriate spots within the XHTML code.
- `mysql_free_result($entries);`
`mysql_close($connBlog);`
Add the code to free up the results from the database, and close the connection to MySQL.
- `<?= $blogHTML ?>`
Place the generated `<div>` tags into the appropriate place in the XHTML output. The equal sign (=) is a PHP shortcut for the keyword `echo`, which writes a value out to the XHTML output.
- `<link rel="stylesheet"`
`type="text/css" href="blog.css" />`
Add an external style sheet reference to reuse the same style sheet we created earlier (Code 10.2).

Adding Comments to the PHP Blog

Building on the comment table we created earlier in the chapter, we will now learn how to add the comments to our page. A blog is pretty boring if you can't make comments to the author. I get about as much pleasure reading comments as I do the blogs themselves!

In this section I'll show you how to take the code we created in the previous section one step further and add the comments. I'll keep them in a hidden `<div>` and show and hide them through an anchor tag (`<a>`). You'll see the link "Show/Hide 1 Comment(s)" under each blog entry in **Figure 10.5**. In the next section, I will show you how to add entries through a Web input form.

How to add comments to the blog:

1. Start with the code from the previous section (Code 10.6).
2. `$div=$div . buildComments($entry["entry-id"]);`
Modify the `buildDiv` function and add the line to call the `buildComments` function (**Code 10.7**).
3. `function buildComments($entryId)`
Add the `buildComments()` function and have it take an `entryId` value as an input. This `entryId` is the `entry-id` column from the database.
4. `$commentSql= "select ...`
Write the SQL statement to return all comments for a given entry. I've written it with the same timestamp formatting that was used before.

continues on page 359

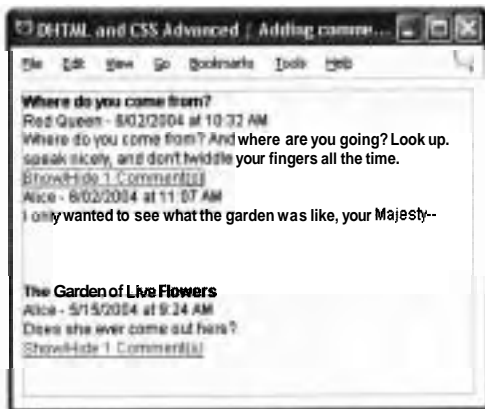


Figure 10.5 The dynamic comments link has been added and the first link has been clicked.

Code **10.7** This page uses PHP to display the blog entries as well as the comments. The differences between this code and the previous are highlighted.

```
<?
$connBlog = mysql_connect()
    or die("Error connecting to database: " . mysql_error());
mysql_select_db("blog",$connBlog)
    or die("Error accessing database: " . mysql_error());
$entrySql = "select entry-id, title, author, content, date_format(insert_date,'%c/%d/%Y at %l:%i %p')
    a s insert-date from Entry order by entry-id desc;";
$entries = mysql_query($entrySql)
    or die("Error accessing database: " . mysql_error());
$blogHTML="";
if (mysql_num_rows($entries) >= 0) {
    while ($entry = mysql_fetch_assoc($entries)) {
        $blogHTML = $blogHTML . buildDiv($entry);
    }
}
mysql_free_result($entries);
mysql_close($connBlog);
function buildDiv($entry) {
    $div="";
    $div=$div . '<div class="title">';
    $div=$div . $entry["title"];
    $div=$div . '</div>';
    $div=$div . '<div>';
    $div=$div . $entry["author"];
    $div=$div . '&#160;-&#160;';
    $div=$div . $entry["insert_date"];
    $div=$div . '</div>';
    $div=$div . '<div>';
    $div=$div . $entry["content"];
    $div=$div . '</div>';
    $div=$div . buildComments($entry["entry_id"]);
    $div=$div . '<br />';
    return $div;
}
function buildComments($entryId) {
    $commentSql = " select author, content, date_format(insert_date,'%c/%d/%Y at %l:%i %p')
        a s insert-date from Comment where entry-id= " . $entryId . " order by comment-id desc;";
    $comments = mysql_query($commentSql)
        or die("Error accessing database: " . mysql_error());
    $commentHTML="";
    if (mysql_num_rows($comments) >= 0) {
```

(code continues on next page)

Code 10.7 continued

```

$commentHTML='<a href="javascript:showHideComments(\'comments\' , $entryId , \'' .
-mysql_num_rows($comments) . ' Comment(s)' , '</a><br />';
$commentHTML=$commentHTML . '<div id="comments\' . $entryId . '" style="display:none">';
while ($comment = mysql_fetch_assoc($comments)) {
    $commentHTML=$commentHTML . '<div>';
    $commentHTML=$commentHTML . $comment["author"];
    $commentHTML=$commentHTML . '&#160;-&#160;';
    $commentHTML=$commentHTML . $comment["insert_date"];
    $commentHTML=$commentHTML . '</div>';
    $commentHTML=$commentHTML . '<div>';
    $commentHTML=$commentHTML . $comment["content"];
    $commentHTML=$commentHTML . '</div><br />';
}
$commentHTML=$commentHTML . "</div><br />";
}
return $commentHTML;
}
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/
DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<title>DHTML and CSS Advanced I Adding comments to a PHP blog</title>
<link rel="stylesheet" type="text/css" href="blog.css" />
<script language="javascript">
    function showHideComments(divId) {
        var div=document.getElementById(divId);
        if (div.style.display=='none') {
            div.style.display='inline';
        }
        else {
            div.style.display='none';
        }
    }
</script>
</head>
<body>
<div id="divBlog" class="blog" style="display:block">
<?=$blogHTML ?>
</div>
</body>
</html>

```

5. `if (mysql_num_rows($comments) > 0)`
Add the code to check whether any records were returned. If none were returned, then the comments link simply will not appear.

```
6. $commentHTML='<a href="javascript:
  showHideComments(\'comments\'
  . $entryId . '\')">Show/Hide '
  . mysql_num_rows($comments) . '
  Comment(s)' . '</a><br />';
```

The first thing you want to add in the XHTML code is the anchor tag that will show or hide the comments.

This creates an anchor tag that calls a JavaScript function named `showHideComments` and sends a string value. This string value is going to be the `id` value of the `<div>` tag in which the comments are stored. I've also added the number of rows returned in the link text, so the user will know how many comments exist before clicking the link.

```
7. $commentHTML=$commentHTML .
  '<div id="comments' . $entryId . "'
  . style="display:none">';
```

Add the `<div>` tag with a dynamic `id` (`commentsN` where `N` is the `entryId`) and set the current display style to `none`. The JavaScript function `showHideComments()` will look for that `id` and change the style attribute based on its current status.

```
8. while($comment = mysql_fetch_
  + assoc($comments)) {
```

Add the code through which all returned comments will loop. For each comment, create the `<div>` tags to put the values in. For the blog entries, I put this in a separate function. That can be done here as well.

```
9. function showHideComments(divId
```

Finally, add the JavaScript function to show and hide the comments. Make sure that you put this code in the XHTML section and not between the PHP open and close tags (`<? ?>`).

Adding a Comments Input Form

So far we have displayed blog entries and comments from the database. You may now be wondering how a visitor can enter comments for blog entries. Obviously, they don't have access to your database. They need a form of some sort to enter their comments. In this section we'll create a form for adding comments, and I'll show you exactly what needs to happen in PHP and MySQL for those comments to show up.

The process will involve three PHP pages. The first one you're familiar with: It displays the blog entries and comments and is what we have worked on until now (Figure 10.6). The second page contains a form into which visitors can enter their comments (Figure 10.7). The final PHP page doesn't display anything. It simply takes the form values and inserts them into the MySQL database and then returns the visitor to the first page, with the updated comments displayed (Figure 10.8).

To add a comments input form:

1. blog.php

Reuse the `blog.php` code we created in the previous section (Code 10.7).

```
2. $div=$div . '<a href="addComment.php?entryId=' . $entry["entry_id"] . "'>Add a comment</a>';
```

Add a line to the `buildDiv` function in `blog.php` to create a link to a new page called `addComment.php` (Code 10.8). You need to pass the `entry-id` (from the result set) to this page so that it knows which blog entry the comments will be applied to.

continues on page 363

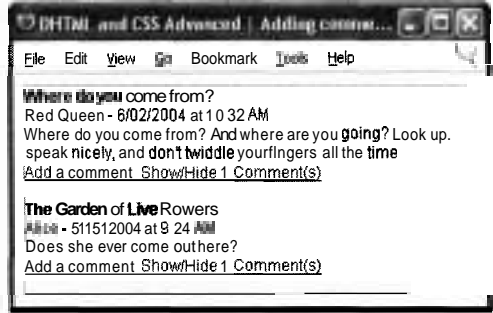


Figure 10.6 The "Add a comment" link has been added to `blog.php`.



Figure 10.7 Data entered from this form will be inserted into the MySQL blog database.

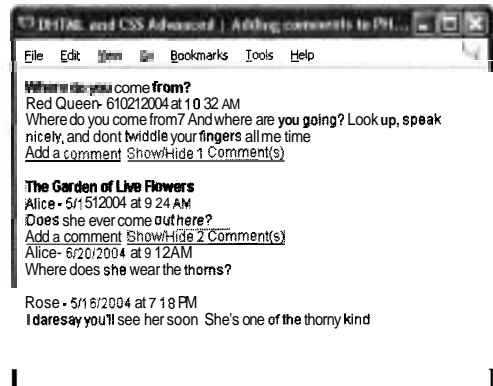


Figure 10.8 This is `blog.php` after a comment has been added using the Add Comment input form.

Code 10.8 As you can see in the highlighted code, I have added a link for each entry to the `addComment.php` page. For this example to work, name this file `blog.php`.

```
<?
$connBlog = mysql_connect()
    or die("Error connecting to database: " . mysql_error());
mysql_select_db("blog",$connBlog)
    or die("Error accessing database: " . mysql_error());
$entrySql = "select entry-id, title, author, content, date_format(insert_date,'%c/%d/%Y at %l:%i %p')
a s insert-date from Entry order by entry-id desc;";
$entries = mysql_query($entrySql)
    or die("Error accessing database: " . mysql_error());
$blogHTML="";
if (mysql_num_rows($entries) >= 0) {
    while ($entry = mysql_fetch_assoc($entries)) {
        $blogHTML = $blogHTML . buildDiv($entry);
    }
}
mysql_free_result($entries);
mysql_close($connBlog);
function buildDiv($entry) {
    $div='';
    $div=$div . '<div class="title">';
    $div=$div . $entry["title"];
    $div=$div . '</div>';
    $div=$div . '<div>';
    $div=$div . $entry["author"];
    $div=$div . '&#160;-&#160;';
    $div=$div . $entry["insert_date"];
    $div=$div . '</div>';
    $div=$div . '<div>';
    $div=$div . $entry["content"];
    $div=$div . '</div>';
    $div=$div . '<a href="addComment.php?entryId=' . $entry["entry_id"] . '">Add a comment</a>&#160;&#160;';
    $div=$div . buildComments($entry["entry_id"]);
    $div=$div . '<br /><br />';
    return $div;
}
function buildComments($entryId) {
    $commentSql = "select author, content, date_format(insert_date,'%c/%d/%Y at %l:%i %p')
a s insert-date from Comment where entry-id= " . $entryId . " order by comment-id desc;";
    $comments = mysql_query($commentSql)
        or die("Error accessing database: " . mysql_error());
    $commentHTML="";
```

(code continues on next *page*)

Code10.8 continued

```

if (mysql_num_rows($comments) > 0) {
    $commentHTML='<a href="javascript:showHideComments(\'comments\' . $entryId . '\')">Show/Hide ' .
        mysql_num_rows($comments) . ' Comment(s)' . '</a>';
    $commentHTML=$commentHTML . '<div id="comments\' . $entryId . '" style="display:none">';
    while($comment = mysql_fetch_assoc($comments)) {
        $commentHTML=$commentHTML . '<div>';
        $commentHTML=$commentHTML . $comment["author"];
        $commentHTML=$commentHTML . '%#160;-##160;';
        $commentHTML=$commentHTML . $comment["insert_date"];
        $commentHTML=$commentHTML . '</div>';
        $commentHTML=$commentHTML . '<div>';
        $commentHTML=$commentHTML . $comment["content"];
        $commentHTML=$commentHTML . '</div><br />';
    }
    $commentHTML=$commentHTML . "</div>";
}
return $commentHTML;
}
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<title>DHTML and CSS Advanced | Adding comments to a PHP blog</title>
<link rel="stylesheet" type="text/css" href="blog.css" />
<script language="javascript">
    function showHideComments(divId) {
        var div=document.getElementById(divId);
        if (div.style.display=='none') {
            div.style.display='inline';
        }
        else {
            div.style.display='none';
        }
    }
</script>
</head>
<body>
<div id="divBlog" class="blog" style="display:block">
    <?=$blogHTML ?>
</div>
</body>
</html>

```

Code 10.9 This is the `addComment.php` page. It takes the `entryId` from the query string and inserts it into a hidden form variable.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
 Transitional//EN" "http://www.w3.org/TR/
 .xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
 xml:lang="en" lang="en">
<head>
  <title>DHTML and CSS Advanced I
  Adding a comments form</title>
  <link rel="stylesheet" type="text/css"
 href="blog.css" />
  <script language="javascript">
    function validate(frm) {
      var errors=new Array();
      if (frm.author.value=='') {
        errors.push("Your Name is a required
        field.");
      }
      if (frm.content.value=='') {
        errors.push("Comment is a required
        field.");
      }
      if(errors.length>0) {
        var msg="The following error(s)
        occurred:\n";
        for (var i=0; i<errors.length; i++) {
          msg+=errors[i] + "\n";
        }
        alert(msg);
        return(false);
      }
      else return(true);
    }
  </script>
</head>
<body>
  <div style="font-size: 18px;
  font-weight:bold;">
  Add Comment<br /><br />
</div>
```

3. `addComment.php`

Create a page called `addComment.php`. This file will contain the PHP code to read the passed `entryId` and well as the XHTML code for a comment input form (**Code 10.9**).

```
4. <form name="frmComment" action=
  "processAddComment.php"
  method="post" onsubmit="return
  validate(this)">
```

Create an XHTML page and add a form that will post to `processAddComment.php`. You'll also want to add some JavaScript validation to make sure both fields have some value. If you leave off the JavaScript validation, someone could enter a blank comment with nothing but a timestamp.

```
5. <input type="text" name="author"
  style="width:300px" />
```

Add an input field for the author field of the database. You may want to make the text beside it user friendly. I used the text "Your Name:" so the visitor knows exactly what to enter there.

```
6. <textarea name="content" rows="5"
  style="width:370px" ></textarea>
```

Add a multiline text area where visitors can enter comments. This will be stored in the content field of the database. You can make the text area larger or smaller depending on how long you expect comments to be.

```
7. <input type="hidden" name="entry_id"
  value="<?=$_REQUEST['entryId']?>" />
```

Add a hidden input field that pulls the `entryId` off the query string. This data will be stored in the `entry-id` field of the database.

continues on next page

(code continues on next page)

8. `<input type="submit" name="save" value="Save" />`

Add a Submit button for the form.

When clicked this will start the form validation. If the form passes validation, then the form fields will be posted to `processAddComment.php`.

9. `<input type="button" value="Cancel" onclick="history.go(-1);" />`

Add a Cancel button that will take the visitor back to the main blog page. It is important to always have a smooth flow through your Web interface. Your visitors should never have to use the Back button to return to a certain page.

10. `function validate(frm)`

Add a JavaScript function that checks the form field values for blanks. If any field is blank, stop the form submission process (by returning false) and alert the user of the required fields. If the form is complete, let the submission process continue (by returning a true value).

11. `processAddComment.php`

Create a page called `processAddComment.php`. This file will contain the PHP code to read the form field values from `addComment.php`, store them in the MySQL database, and then return the visitor to the main blog page (**Code 10.10**).

Code 10.9 continued

```
<form name="frmComment" action=
  "processAddComment.php" method="post"
  onsubmit="return(validate(this))">
  Your Name: <input type="text"
    name="author" style="width:300px"
  /><br />
  Comment:<br />
  <textarea name="content" rows="5"
    style="width:370px" ></textarea>
  <br />
  <input type="submit" name="save"
    value="Save" />
  &#160;&#160;
  <input type="button" value="Cancel"
    onclick="history.go(-1);" />
  <input type="hidden" name="entry_id"
    value="<?=$_REQUEST['entryId']?>" />
</form>
</body>
</html>
```

Code 10.10 This is the processAddComment.php page. It takes the form fields and stores them in the database. Once it is complete, it forwards the user back to blog.php where they will see the new comment posted.

```
Code
<?
$sql="insert into Comment (author, content,
    entry_id) ";
$sql=$sql . "values ('" . $_REQUEST['author'];
$sql=$sql . "','" . $_REQUEST['content'];
$sql=$sql . "','" . $_REQUEST['entry_id'] .
    "')";
$connBlog = mysql_connect()
    or die("Error connecting to database: " .
        mysql_error());
mysql_select_db("blog",$connBlog)
    or die("Error accessing database: " .
        mysql_error());
mysql_query($sql)
    or die("Error accessing database: " .
        mysql_error());
mysql_close($connBlog);
header("Location:blog.php");
?>
```

12. `$sql="insert into Comment (author, content, entry-id) ";`
Add an `insert` statement to insert the values for `author`, `content`, and `entry-id` in the `Comments` table.

13. `$_REQUEST['author']`
Use the `request` function to retrieve values from the form. This is the same function that retrieves values from the query string.

14. `mysql_query($sql)`
Use the same code as you used in the main blog entry list page to execute the query. This query won't return any records from the database, so you don't need to set it equal to a variable. This also means that you don't have to execute the `mysql_free_result` method.

15. `header("Location:blog.php");`
Add a call to the `header` function to change the location to `blog.php`. Setting the location header is known as a *redirect*. It causes the visitor's browser to change its location bar to the given URL.

Tips

The `header` function in PHP can be a big help in handling the workflow of your application. It will accept a relative path (such as a file in the same folder), or a fully qualified URL (such as `header("Location:http://www.peachpit.com/")`).

You can create a similar page that will add blog entries to the `Entry` table instead. You simply need to add a `title` field and change the table name in the processing page.

Adding an Administrator Page

Now that I have you all excited to have a blog, here are some things to think about. Are you going enter blog entries manually into MySQL? What about removing blog entries or removing unwanted comments?

Of course not. What you want now is an administrator page (**Figure 10.9**). This page will be secured by some sort of user name and password (**Figure 10.10**), and any pages after it will need to check to make sure the user is authorized before processing any information entered.

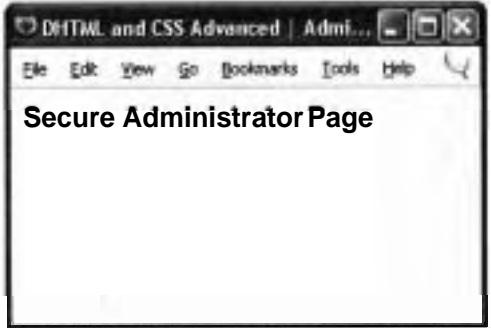
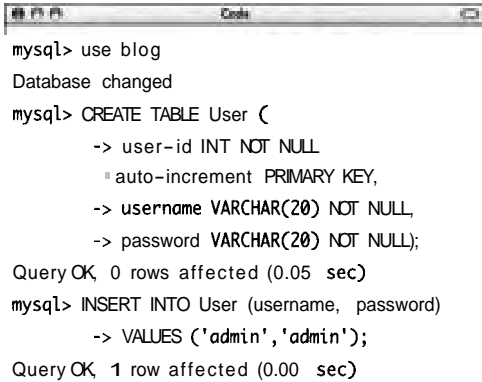


Figure 10.9 This is the placeholder page for our secure administrator page. We'll add a link to this page in the next section.



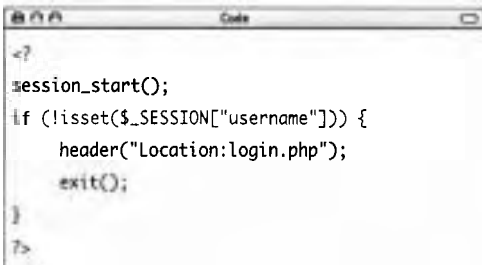
Figure 10.10 This is the login. php page. Any page that includes the securePage. php file will forward the visitor here if the user name is not in the session.

Code 10.11 These are the SQL commands for creating a new User table and adding an admin user.



```
mysql> use blog
Database changed
mysql> CREATE TABLE User (
  -> user-id INT NOT NULL
  ->   auto-increment PRIMARY KEY,
  -> username VARCHAR(20) NOT NULL,
  -> password VARCHAR(20) NOT NULL);
Query OK, 0 rows affected (0.05 sec)
mysql> INSERT INTO User (username, password)
  -> VALUES ('admin','admin');
Query OK, 1 row affected (0.00 sec)
```

Code 10.12 This is the `securePage.php` file that we'll include in all pages that need to be secured by a login screen.



```
<?
session_start();
if (!isset($_SESSION["username"])) {
    header("Location:login.php");
    exit();
}
?>
```

To add an administrator page:

1. Add a table to the blog database named user, with columns `username` and `password` to store login credentials (**Code 10.11**). Also add a user name of `admin` with the password of `admin`.
2. `securePage.php`
Create a file named `securePage.php` (**Code 10.12**). This page will be used to secure all pages in the administrator area.
3. `session-start()`
Add code to initialize the session. This will not erase any variables if the session is already started. See the sidebar titled "What Is a Session?" for more information.
4. `isset($_SESSION["username"])`
Check to see if the user name has been set. You can use the PHP `isset` function to let you know whether the user name has been set in the session. If the session is not set, you need to direct the visitor to your login page for authentication.
5. `exit()`
Add the `exit()` function after you have set the header location. This code tells PHP to stop processing any commands. If you left this out, PHP would continue to process additional commands after you set the header. You want to stop all processing once you have determined that the visitor is not authorized for this page.

continues on page 369

Code 10.13 This is the login page (login.php). It's a basic input form.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <title>DHTML and CSS Advanced I Administrator Page</title>
  <link rel="stylesheet" type="text/css" href="blog.css" />
  <script language="javascript">
    function validate(frm) {
      var errors=new Array();
      if (frm.username.value=='') {
        errors.push("User name is a required field.");
      }
      if (frm.password.value=='') {
        errors.push("Password is a required field.");
      }
      if (errors.length>0) {
        var msg="The following error(s) occurred:\n";
        for (var i=0; i<errors.length; i++) {
          msg+=errors[i] + "\n";
        }
        alert(msg);
        return(false);
      }
      else return(true);
    }
  </script>
</head>
<body>
  <div style="font-size: 18px; font-weight:bold;">
  Administrator Login<br /><br />
  </div>
  <form name="frmComment" action="processLogin.php" method="post" onsubmit="return(validate(this))">
    User name:<br />
    <input type="text" name="username" /><br />
    Password:<br />
    <input type="password" name="password" /><br />
    <input type="submit" name="save" value="Login" />
  </form>
</body>
</html>
```

Code 10.14 This is the `processLogin.php` page, which looks up the user name and password in the blog database and either transfers the user to the `administrator.php` page or back to the `login.php` page.

```
#!/?
$sql="select * from user where username='" .
$_REQUEST["username"] . "' and password='" .
$_REQUEST["password"] . "'";
$connBlog = mysql_connect()
    or die("Error connecting to database: " .
mysql_error());
mysql_select_db("blog",$connBlog)
    or die("Error accessing database: " .
mysql_error());
$users=mysql_query($sql)
    or die("Error accessing database: " .
mysql_error());
if (mysql_num_rows($users) >= 0) {
    session_start();
    $_SESSION["username"]=$_REQUEST["username"];
    header("Location:administrator.php");
}
else {
    header("Location:login.php");
}
mysql_free_result($users);
mysql_close($connBlog);
?>
```

6. `login.php`

Create a login page that has form input fields for a user name and a password (**Code 10.13**). It should post to a page called `processLogin.php`.

7. `<input type="password" name="password" />`

When adding the password input field, make sure you set the **type** to **password**. This causes the characters entered to show up as asterisks.

8. `processLogin.php`

Create a page named `processLogin.php` that will look up the user name and password posted from `login.php` (**Code 10.14**).

9. `$sql="select * from user..`

Write a SQL statement to return all users matching the user name and password submitted. Theoretically there should be only one match, but we didn't put any restrictions on our `user` table in MySQL to force only one occurrence of each user.

continues on next page

What Is a Session?

When you connect to a Web site and session state is enabled, the Web server sends your browser a "session cookie." This cookie is stored in the browser's memory instead of on disk like normal cookies. Because the cookie is stored only in memory, as soon as you close your browser, that cookie is erased. Thus, a session is only active for a specific instance of a browser. Typically sessions time out after 20 to 30 minutes (depending on the server settings).

Session cookies contain a unique identifier for the browser instance. Using this unique identifier, the Web server can store various pieces of data about the visitor for this particular session. Many applications use sessions, including shopping carts and just about any site that you have to log into, especially those that provide specific information for your user name (Web-based email, online banking, and so forth).

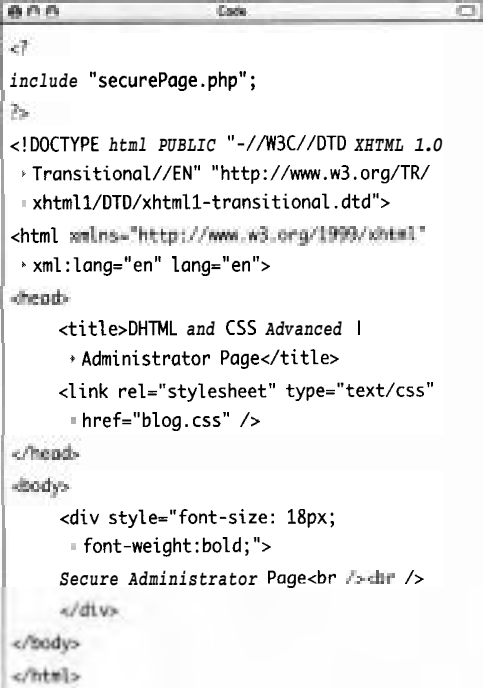
10. `if(mysql_num_rows($users) > 0)`
Check to see if at least one row was returned. If so, the user must exist in the database. If the user exists, send them to the administrator.php page; otherwise send them back to the login.php page.

11. `$_SESSION["username"]=$_REQUEST["username"];`
If the user is valid, set a session variable called username equal to the posted user name.

12. administrator.php
Create a page called administrator.php. This will be a placeholder for our secure links (**Code 10.15**).

13. `include "securePage.php";`
Include a copy of the securePage.php. This will prompt the PHP engine to process all code in the securePage.php file. This must be put at the top of the page so that it is processed before anything is sent to the browser. If some data has already been sent to the client browser, the header code you have in securePage.php will not work.

Code 10.15 This is the administrator.php page, which will contain our secured Web pages.



```
<?php
include "securePage.php";
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
 Transitional//EN" "http://www.w3.org/TR/
 xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
 xml:lang="en" lang="en">
<head>
<title>DHTML and CSS Advanced I
 Administrator Page</title>
<link rel="stylesheet" type="text/css"
 href="blog.css" />
</head>
<body>
<div style="font-size: 18px;
 font-weight:bold;">
Secure Administrator Page<br /></div>
</body>
</html>
```

Tip

If you have access to an HTTPS (SSL) server, it is best to put your administrator pages (including login.php) on that server. Even though the pages are secured by a user name and password, that information is sent over the Internet using clear text. But if you use SSL, all data is transmitted in an encrypted format. This is probably not nearly as important for your blog administrator pages as it is for an e-commerce site, but you should always keep this in mind.

Creating a Secure Blog Entry Input Form

In the previous section you created a secure administrator page to contain links to pages that required authentication before use. To finish this chapter off, we'll now look at an example of a secure blog entry input form.

We'll create a link to the entry input form (Figure 10.11). Then we'll create an entry input form that is secured behind our login page (Figure 10.12). Finally, we'll use the code created earlier in the chapter to view the entry that was added (Figure 10.13).

continues on next page

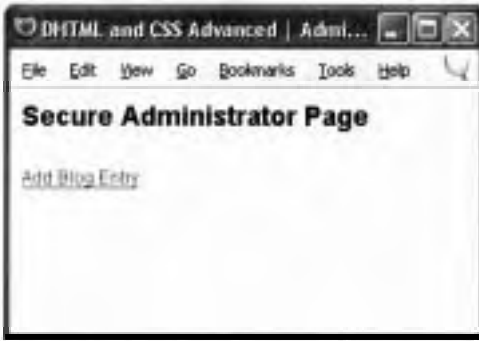


Figure 10.11 This is the secure administrator page with a link added to the entry input form.



Figure 10.12 This is the entry input form. It's similar to the comment input form, but this form is secured by a user name and password.

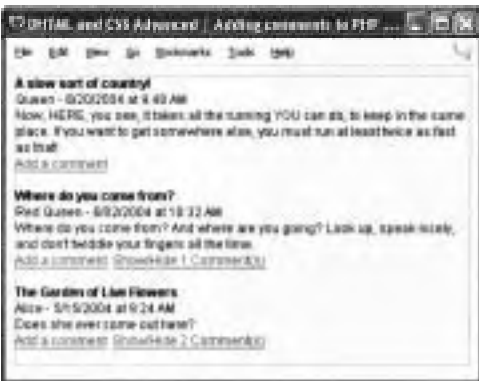


Figure 10.13 The main blog page now shows the data entered using the entry input form.

To add a secure entry input form:

1. `Add Blog
Entry`

Modify the administrator.php page from the previous section (Code 10.15) to add a link to the blog entry input form (Code 10.16).

2. `addEntry.php`

Create a page called `addEntry.php` (Code 10.17). You can copy and paste from `addComment.php`, which we created earlier (Code 10.9), and you'll only need to modify a few lines of code.

3. `include "securePage.php";`

Add an include to the security page. This will keep unauthorized users out of this page.

4. `<input type="text" name="title"
style="width:309px" />`

Add input fields for author, title, and content. If you copied from `addComment.php`, you'll need to add an input field for title and remove the hidden input field for entry-id.

5. `if (frm.title.value=='')`

Add JavaScript validation for the form to make sure the title field isn't empty. If you copied from `addComment.php`, you'll need to add this JavaScript validation.

6. `action="processAddEntry.php"`

Set the action of the input form to `processAddEntry.php`. If you copied from `addComment.php`, you'll need to change the action.

Code 10.16 This is administrator.php, which has a link to the Add Entry input form.

```
Code
<?
include "securePage.php";
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xml:lang="en" lang="en">
<head>
<title>DHTML and CSS Advanced I
Administrator Page</title>
<link rel="stylesheet" type="text/css"
href="blog.css" />
</head>
<body>
<div style="font-size: 18px;
font-weight:bold;">
Secure Administrator Page<br /><br />
</div>
<a href="addEntry.php">Add Blog
Entry</a><br />
</body>
</html>
```

continues on page 374

Code 10.17 This is the addEntry.php page that displays the Add Entry input form.

```
<?
include "securePage.php";
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <title>DHTML and CSS Advanced I Adding an entry form</title>
  <link rel="stylesheet" type="text/css" href="blog.css" />
  <script language="javascript">
    function validate(frm) {
      var errors=new Array();
      if (frm.author.value=='') {
        errors.push("Your Name is a required field.");
      }
      if (frm.title.value=='') {
        errors.push("Entry Title is a required field.");
      }
      if (frm.content.value=='') {
        errors.push("Content is a required field.");
      }
      if (errors.length>0) {
        var msg="The following error(s) occurred:\n";
        for(var i=0; i<errors.length; i++) {
          msg+=errors[i] + "\n";
        }
        alert(msg);
        return(false);
      }
      else return(true);
    }
  </script>
</head>
<body>
  <div style="font-size: 18px; font-weight:bold;">
    Add Entry<br /><br />
  </div>
  <form name="frmComment" action="processAddEntry.php" method="post" onsubmit="return(validate(this))">
    Your Name: <input type="text" name="author" style="width:300px" /><br /><br />
    Entry Title: <input type="text" name="title" style="width:300px" /><br /><br />
    Content:<br />
    <textarea name="content" rows="5" style="width:370px" ></textarea><br />
    <input type="submit" name="save" value="Save" />
    &#160;&#160;
    <input type="button" value="Cancel" onclick="history.go(-1);" />
  </form>
</body>
</html>
```

7. processAddEntry.php

Create a page called `processAddEntry.php` (Code 10.18). You can copy the code from `processAddComment.php` (Code 10.10) and modify a few lines.

8. include "securePage.php";

Add an include to the security page. Even though this is simply a processing page, technically anyone can post to it and save records in your database unless you include the security page.

9. \$sql="insert into Entry (title, author, content) ";

Add a SQL statement that inserts new entries into the Entry table of the blog database. Write the code (or copy from `processAddComment.php`) to execute this SQL statement.

10. header("Location:administrator.php");

Change the location back to the `administrator.php` page. This keeps a smooth workflow and takes the visitor back to the main menu of the secure administrator area.

Code 10.18 This is `processAddEntry.php`, the page that adds the entry data to the blog database.

```
<?
include "securePage.php";
$sql="insert into Entry(title, author, content) ";
$sql=$sql . "values ('" . $_REQUEST['title'];
$sql=$sql . "','" . $_REQUEST['author'];
$sql=$sql . "','" . $_REQUEST['content'] . "')";
$connBlog = mysql_connect(
    or die("Error connecting to database: "
        =mysql_error());
mysql_select_db("blog",$connBlog)
    or die("Error accessing database: "
        =mysql_error());
mysql_query($sql)
    or die("Error accessing database: "
        =mysql_error());
mysql_close($connBlog);
header("Location:administrator.php");
```

CSS QUICK REFERENCE



Although this appendix can't replace a comprehensive examination of cascading style sheets, I wanted to give you a way to quickly check the anatomy of the most common CSS code. I'm also providing a compatibility table for the most common CSS attributes.

Adding Styles

Styles can be added to a Web page in a variety of ways:

Inline

Inline styles are applied directly to HTML tags through the style attribute (Figure A.1). They are the last styles considered in the cascade order, so they are useful for overriding other styles.

Internal

Internal styles are set in the <style> tag, generally located in the head of an HTML document (Figure A.2). Styles set internally in a document will only affect that Web page.

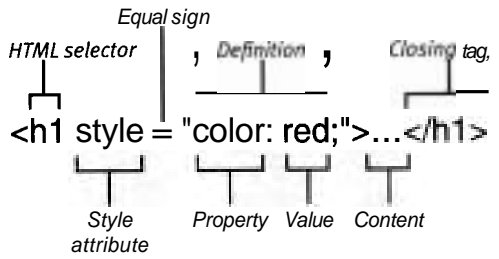


Figure A.1 Anatomy of an inline style.

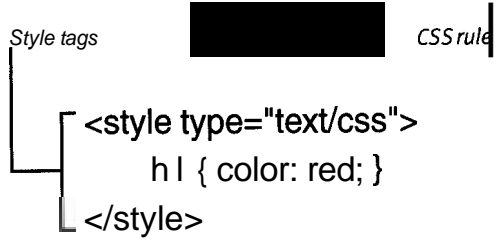


Figure A.2 Anatomy of an internal style.

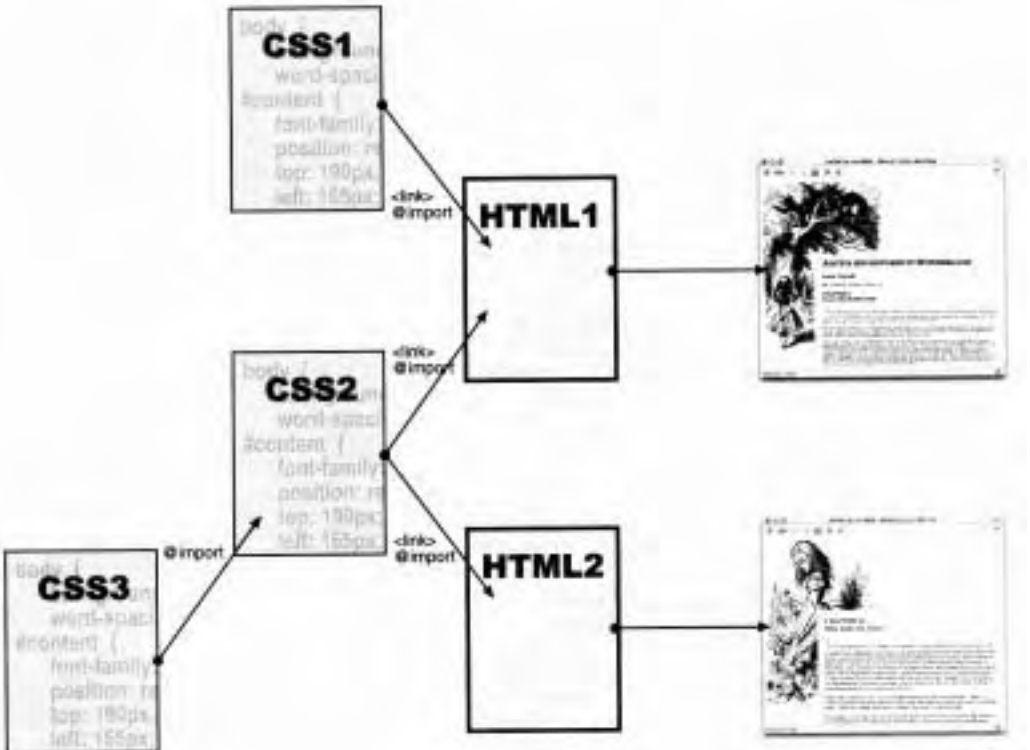


Figure A.3 External CSS files can not only be used in multiple HTML files using `<link>` or `@import`, they can also be imported into other CSS files using `@import`.

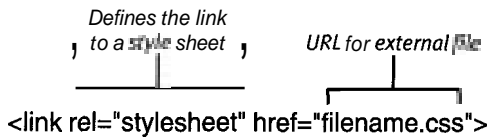


Figure A.4 Anatomy of the `<link>` tag.

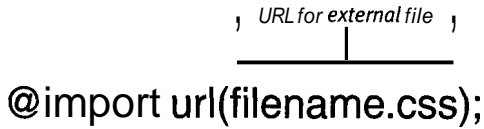


Figure A.5 Anatomy of `@import`.

External

External style sheets are text files, generally saved with a `.css` extension (for example, `myStyles.css`), which are then either imported or linked into HTML pages (**Figure A.3**). External CSS files can use `@import` to be placed into either other CSS files or HTML files (**Figure A.4**). The `<link>` tag can only be used to import CSS files into HTML files (**Figure A.5**). Styles set in an external style sheet can be applied to any Web page throughout your site or the entire Web.

Selectors: HTML, Classes, IDs

There are three basic types of HTML selectors used to redefine styles on a Web page:

HTML

To redefine the styles of a particular HTML tag, use that tag's selector, followed by its new (or revised) attributes (Figure A.6). This will change the appearance of all HTML tags with this selector.

Class

Classes can be used to selectively redefine tags on the page, by creating a selector using a keyword with a period (.) in front of it, followed by a list of rules (Figure A.7). Classes can also be associated with a particular HTML tag, so that the definitions will only be applied if the class is applied to that particular tag (Figure A.8).

ID

As with the class selector, ID selectors are created using a keyword (Figure A.9). However, the keyword is preceded by a number sign (#) and, unlike a class, an ID should only be used once on a Web page. Although the overall effect of an ID is like a class, it serves as a unique identifier for an object on the page.

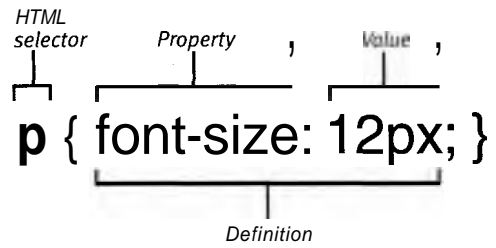


Figure A.6 Anatomy of an HTML selector.

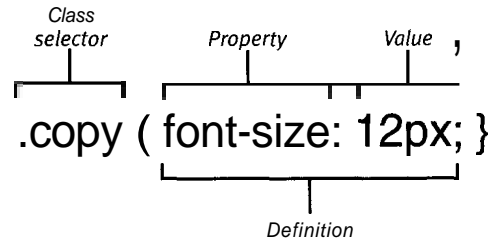


Figure A.7 Anatomy of a class selector.

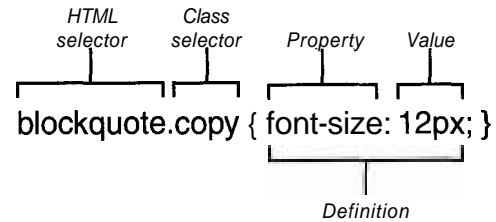


Figure A.8 Anatomy of a dependent class selector.

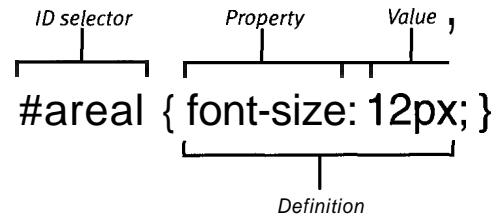


Figure A.9 Anatomy of an ID selector.

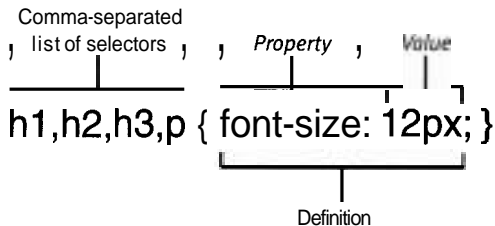


Figure A.10 Anatomy of a same definition selector.

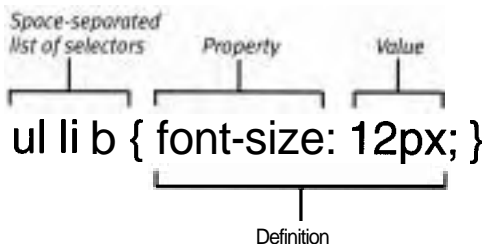


Figure A.11 Anatomy of a contextual selector.

Grouping Styles

You can group styles together in order to combine definitions or limit the effects of a definition.

NOTE: See Chapter 2 in this book for additional CSS grouping types.

Same definitions

If you need to apply the same rules to more than one selector, rather than retyping them for each, you can group them together, separated by commas (**Figure A.10**). This saves space and allows you to make changes more quickly.

Contextual definitions

You can specify the styles applied to a selector based on its parent elements by grouping the selectors, separated by spaces (**Figure A.11**). The final selector in the group will receive the styles, but only if it's within selectors for each of the previous selectors in order.

Properties and Values

Browsers Legend

- N** Netscape
- IE** Internet Explorer
- S** Safari
- O** Opera

In addition to the browsers listed, remember that Netscape 6 and above is based on the Gecko rendering engine. Therefore, generally speaking, Mozilla 1 will have the same compatibility as Netscape 6. Mozilla 1.3 and 1.5, Firefox, and Camino will have roughly the same compatibility as Netscape 7.

Keep in mind that each browser has several versions, even within a single version number. There is not a Netscape 4, for example, but several versions (4.06, 4.5, and 4.7), with slight differences among them. The information presented in this appendix generally should be correct, but if you want to test the CSS capabilities of your own browser, check out the World Wide Web Consortium's (W3C) test suite at www.w3.org/Style/CSS/Test/. This utility will help you confirm which properties work in your browser.

Compatibility Legend

- Mac and Windows
- Neither
- W** Windows only
- M** Mac only
- P** Problems
- All Property can be applied to any HTML tag
- Block Property can be applied only to block-level tags
- Inline Property can be applied only to inline tags

Boldface Indicates the default value for that property

Properties marked with a **P** in the browser columns are partially implemented or buggy in one or both operating systems. I generally recommend avoiding these properties.

Table A.1

Basics								
NAME	N4	N6	N7	IE4	IE5	IE6	S1	O7
<style>	■	■	■	■	■	■	■	■
<link>	■	■	■	■	■	■	■	■
@import	○	■	■	■	■	■	■	■
@media	○	■	■	○	■*	■	■	■
Inheritance	■	■	■	■	■	■	■	■
Contextual	■	■	■	■	■	■	■	■
Comments	■	■	■	■	■	■	■	■
!important	P	■	■	P	■	■	■	■
Media: Print	○	■	■	○	■	■	■	■

*IE5.5 for Windows

Table A.2

Pseudo-Classes and Pseudo-Elements											
NAME	VALUE	APPLIES TO	INHERITED	IE4	N6	N7	IE4	IE5	IE6	S1	O7
:link	—	Anchor	Yes	■	■	■	■	■	■	■	■
:active	—	Anchor	Yes	○	■	■	■	■	■	■	■
:visited	—	Anchor	Yes	○	■	■	■	■	■	■	■
:hover	—	All*	Yes	○	■	■	■	■	■	■	■
:first-line	—	Block	No	○	■	■	○	■**	■	■	■
:first-letter	—	Block	No	○	■	■	○	■**	■	■	■

*Applies only to Anchor tags in IE for Windows
 **IE5.5 for Windows

Table A.3

Font Controls

NAME	VALUE	APPLIES TO	IMPLEMENTED	84	85	87	104	105	106	51	07
font-family	<family-names> serif sans-serif cursive fantasy monospace	All	Yes	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
font-style	normal italic oblique	All	Yes	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
font-variant	normal small-caps	All	Yes	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
font-weight	normal bold bolder lighter 100-900*	All	Yes	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
font-size	<lengths> <percentages> smaller larger xx-small x-small small medium large x-large xx-large	All	Yes	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
font	<font-styles> <font-variants> <font-weights> <font-size> / <lineweights> <font-family>	All	Yes	<input type="radio"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

* Requires the writer's computer to have display-weighted fonts available

Table A.5

X1 set Controls

NAME	VALUE	APPLIES TO	INHERITS	INITIAL	MIN	MAX	EF	ET	IES	6	S1	O7
margin-top, -right, -bottom, -left	<length> <percentage> auto	pl	No	No	1	1	1	1	1	1	1	1
margin	<length> <percentage> auto	pl	No	No	1	1	1	1	1	1	1	1
padding-top, -right, -bottom, -left	<length> <percentage> <length> <percentage> auto	pl	No	No	1	1	1	1	1	1	1	1
padding	<length> <percentage> <length> <percentage> auto	pl	No	No	1	1	1	1	1	1	1	1
border-color	inherit		No	No	1	1	1	1	1	1	1	1
border-style	inherit		No	No	1	1	1	1	1	1	1	1
border-top, -right, -bottom, -left, width	medium <length> thin thick medium <length> thin thick	pl	No	No	1	1	1	1	1	1	1	1
border-top, -right, -bottom, -left	<border-width> <border-style> <color> <border-width> <border-style> <color>	pl	No	No	1	1	1	1	1	1	1	1
border	<border-width> <border-style> <color> <border-width> <border-style> <color>	pl	No	No	1	1	1	1	1	1	1	1
-moz-border-radius, -bottomleft, -bottomright, -topleft, -topright, width	<length> <percent g> Auto <length> <percentage g>	Block	No	No	1	1	1	1	1	1	1	1

Table A.5 continued

Element Controls

NAME	VALUE	APPLIES TO	INHERIT	IE3	IE4	N7	IE5	N4	IE6	S1	O7
height	auto <length>	All	No	■	○	■	■	○	■	■	■
max-width, min-width, max-height, max-width	<length> <percentage> auto none left right none left right both block inline list-item table table-cell table-footer-group table-row-group table-row-group none <color> transparent none url(url) repeat repeat-x repeat-y no-repeat scroll fixed <percentage> <length> top center (vertical) bottom left center (horizontal) right <background-color> <background-image> <background-repeat> <background-attachment>	All	No	○	○	■	■	○	■	■	■
background-color	<color>	All	No	■	■	■	■	■	■	■	■
background-image	url(url) repeat repeat-x repeat-y no-repeat	All	No	■	○	■	■	■	■	■	■
background-repeat	repeat repeat-x repeat-y no-repeat	All	No	■	○	■	■	■	■	■	■
background-attachment	scroll fixed <percentage> <length>	All	No	■	■	■	■	■	■	■	■
background-position	top center (vertical) bottom left center (horizontal) right <background-color> <background-image> <background-repeat> <background-attachment> <background-position>	Block	No	■	■	■	■	■	■	■	■
background	<background-color> <background-image> <background-repeat> <background-attachment> <background-position>	All	No	■	■	■	■	■	■	■	■

Table A.6

Element Positioning Controls

NAME	VALUE	APPLIES TO	INHERITED	84	86	87	104	105	106	51	07
position	static	All	No	■	■	■	■	■	■	■	■
	absolute			■	■	■	■	■	■	■	■
	relative			■	■	■	■	■	■	■	■
left	fixed	All**	No	■	○	■	○	M*	■	■	■
	auto			■	■	■	■	■	■	■	■
top	<length>			■	■	■	■	■	■	■	■
	<percentage>	All**	No	■	■	■	■	■	■	■	■
bottom	auto			○	○	■	○	■	■	■	■
	<length>			○	○	■	○	■	■	■	■
right	<percentage>	All**	No	○	○	■	○	■	■	■	■
	auto			○	○	■	○	■	■	■	■
z-index	<length>			○	○	■	○	■	■	■	■
	<percentage>	All	No	○	○	■	○	■	■	■	■
	number			■	■	■	■	■	■	■	■

* Mac only; not available in Windows

** The position property for the element must also be set to absolute or relative.

Table A.7

Element Visibility Controls

NAME	VALUE	APPLIES TO	INHERITED	84	86	87	104	105	106	51	07
clip	auto	All*	No	■	■	■	■	■	■	■	■
	<shape>			■	■	■	■	■	■	■	■
	visible			■	■	■	■	■	■	■	■
	hidden			■	■	■	■	■	■	■	■
	scroll			■	■	■	■	■	■	■	■
visibility	auto	All	Yes**	■	■	■	■	■	■	■	■
	Inherit			■	■	■	■	■	■	■	■
	visible			■	■	■	■	○	○	○	○
-moz-opacity	hidden			■	■	■	■	○	○	○	○
	hide			■	■	■	■	○	○	○	○
	show			■	■	■	■	○	○	○	○
	<0.0-1.0>			○	■	■	○	○	○	○	

* The position property for the element must also be set to absolute or relative.

** [visibility] is set to inherit.

List, Table, and Interface Controls

NAME	VALUE	APPLIES TO	INHERITED	IE8	IE6	IE7	IE4	IE5	IE6	IE5	IE4	IE7	IE6	IE5	IE4	DT	
list-style-type	disc	All*	Yes														
	circle																
list-style-image	square	All*	Yes														
	decimal																
list-style-position	lower-roman	All*	Yes														
	upper-roman																
list-style	lower-alpha	All*	Yes														
	upper-alpha																
border-collapse	none	All*	Yes														
	url(<url>)																
caption-side	outside	Table	No														
	inside																
cursor	<list-style-type>	All	Yes														
	<list-style-position>																
	<list-style-image>																
cursor	collapse	All	Yes														
	separate																
	top																
	left																
	bottom																
	right																
	auto																
	crosshair																
	hand**																
	pointer																
	move																
	n-resize																
	ne-resize																
	e-resize																
se-resize																	
s-resize																	
sw-resize																	
w-resize																	
nw-resize																	
text																	
wait																	
help																	

* In Netscape 4.0 and IE 4 & 5, applies only to the <list> tag. In standard CSS, these properties can be applied only to tags that include display: list-item, in the definition.

** IE only. Same as pointer.

*** IE 5 for Windows.

Blank

DHTML



QUICK REFERENCE

DHTML is the combination of CSS, JavaScript, DOM, and XHTML. Although I don't have enough space here to teach you all of those technologies, I wanted to include a quick reference of important concepts to help refresh your memory if you're away from a more comprehensive resource.

The DOM

The Document Object Model (DOM) describes a path starting with the window itself, down through the various objects on the Web page, each element representing a node within the document. For example, **Code B.1** is broken down into nodes as shown in **Figure B.1**.

Code B.1 This is a simple Web page with its node structure broken down, as shown in **Figure B.1**.

```
Code
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Transitional//EN">
<html>
  <head>
    <title>DHTML & CSS for the WWW I
      Understanding the DOM</title>
  </head>
  <body>
    <form action="" method="get">
      <input type="text" size="24" />
    </form>
    <div>
      
      Your Message Here
    </div>
  </body>
</html>
```

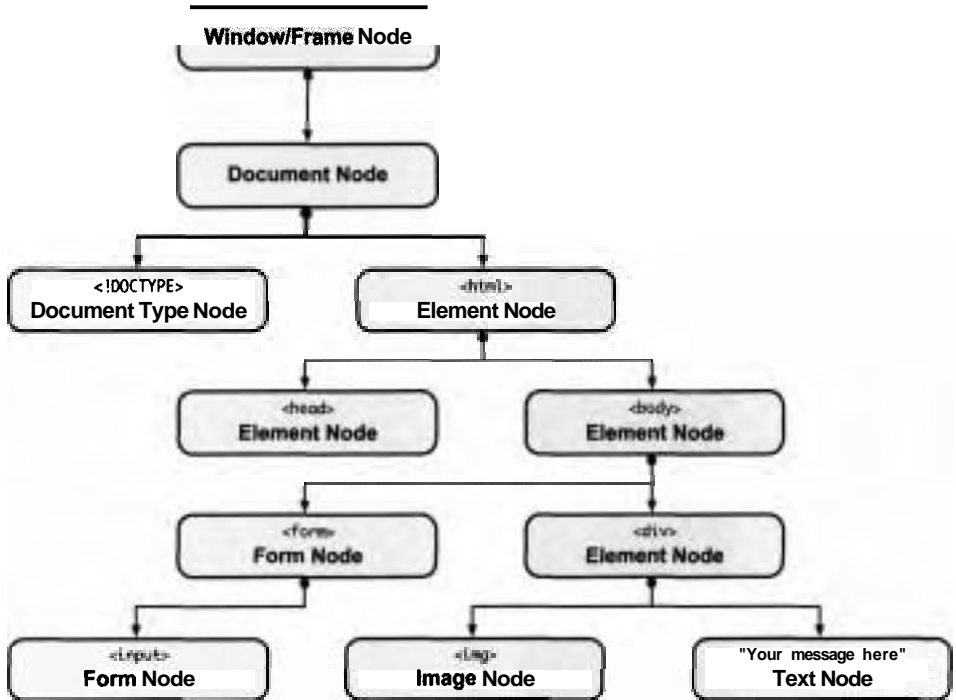


Figure B.1 The Web page node starts at the top with the window, moving down to each individual element on the page.

Events

An event handler—which is the event name with the word on at the beginning (for example, onload)—allows you to define what should happen when a particular event is detected (**Figure B.2**).

```
onmouseover = "toggle(); if (a==b) {x=y; alert('don\'t tread on me!')}";
```

Figure B.2 Here's an example of an event handler.

Table B.1

Common Event Handlers		
NAME	WHEN IT HAPPENS	APPLIES TO
onload	After an object is loaded	Documents and images
onunload	After the object is no longer loaded	Documents and images
onfocus	When an element is selected	Documents and forms
onblur	When an element is deselected	Documents and forms
onmouseover	When the mouse pointer passes over an area	All*
onmouseout	When the mouse pointer passes out of an area	All*
onclick	When the mouse button is clicked over an area	All*
onmousedown	While the mouse button is pressed	All*
onmouseup	When the mouse button is released	All*
onmousemove	As the mouse is moved	Document
onkeydown	While a keyboard key is pressed	Forms and document
onkeyup	When a keyboard key is released	Forms and document
onkeypress	When a keyboard key is pressed and immediately released	Forms and document
onresize**	When the browser window or a frame is resized	Document
onmove	When the browser window is moved	Document

*Available only for anchor links and images in Netscape 4

**Not supported by Internet Explorer 4

Properties and Values

JavaScript includes several methods for finding properties and values in relation to the browser window and Web page. The following tables present those methods. Where there are multiple methods for getting the same information, the method in **bold** text is the preferred method.

Table B.2

Finding Objects		
TO FIND	METHOD	COMPATIBILITY
Object	document.getElementById(objectID) document.all[objectID] document.layers[objectID]	IE4, N6, S1, O5 IE4, O5 N4 (Only)

Table B.3

System Properties			
TO FIND	METHOD	VALUE TYPE	COMPATIBILITY
Operating system	navigator.appVersion	<string>	IE3, N2, S1, O5
Screen width (total)	screen.width	<pixels>	IE4, N4, S1, O5
Screen height (total)	screen.height	<pixels>	IE4, N4, S1, O5
Screen width (live)	screen.availWidth	<pixels>	IE4, N4, S1, O5
Screen height (live)	screen.availHeight	<pixels>	IE4, N4, S1, O5
Number of colors	screen.colorDepth	<number>	IE4, N4, S1, O5

Table B.4

Browser Properties			
TO FIND	METHOD	VALUE TYPE	COMPATIBILITY
Browser name	navigator.appName	<string>	IE3, N2, S1, O5
Browser version	parseInt(navigator.appVersion)	<number>	IE3, N2, S1, O5
Browser window width	window.outerWidth	<pixels>	N4, S1, O5
Browser window height	window.outerHeight	<pixels>	N4, S1, O5

Table B.5

Page Properties			
TO FIND	METHOD	VALUE TYPE	COMPATIBILITY
URL	self.location	<string>	IE3, N2, S1, O5
Title	document.title	<string>	IE3, N2*, S1, O5
Visible width	window.innerWidth document.body.clientWidth	<pixels> <pixels>	N4, S1, O5 IE4, N7, S1, O5
Visible weight	window.innerHeight document.body.clientHeight	<pixels> <pixels>	N4, S1, O5 IE4, N7, S1, O5
Scroll position left	window.pageXOffset document.body.scrollLeft	<pixels> <pixels>	N4, S1, O5 IE4, N7, S1, O5
Scroll position top	window.pageYOffset document.body.scrollTop	<pixels> <pixels>	N4, S1, O5 IE4, N7, S1, O5

*Buggy in Mac version of Netscape 4; returns file name instead of title.

Table B.6

Object Properties			
TO FIND	METHOD	VALUE TYPE	COMPATIBILITY
Object ID	evt.target.id evt.srcElement.id	<string> <string>	N4, S1, O5 IE4, O5
Width	offsetWidth style.width*	<length> <length>	IE4, N6, S1, O5 IE4, N4, S1, O5
Height	offsetHeight style.height*	<length> <length>	IE4, N6, S1, O5 IE4, N4, S1, O5
Left position	offsetLeft pixelLeft style.left*	<length> <length> <length>	N6, S1, O5 IE4, N6, S1, O5 IE4, N4, S1, O5
Top position	offsetTop pixelTop style.top*	<length> <length> <length>	N6, S1, O5 IE4, N6, S1, O5 IE4, N4, S1, O5
Z-index	style.zindex*	<number>	IE4, N4, S1, O5
Visibility	style.visibility*	visible hidden show hide	IE4, N4, S1, O5 IE4, N4, S1, O5 N4 (only) N4 (only)
Clip area	style.clip[]** style.clipBottom style.clipLeft style.clipRight style.clipTop	<array> <length> <length> <length> <length>	IE4, N6, S1, O5 IE4 (Win) IE4 (Win) IE4 (Win) IE4 (Win)

*Requires that value be set using JavaScript before it can be read.

**The most reliable way to find the clip area is by querying the clip array.

Table B.7

Event Properties			
To FIND	METHOD	VALUE TYPE	COMPATIBILITY
Event type	evt.type	<string>	IE4, N4, S1, O5
Key pressed	evt.charCode evt.keyCode	<number> <number>	N4 IE4, N6
Shift key	evt.shiftKey	<boolean>	IE4, N6, O5, S1
Control key	evt.ctrlKey	<boolean>	IE4, N6, O5, S1
Alt/Option key	evt.altKey	<boolean>	IE4, N6, O5, S1
Command key	evt.metaKey	<boolean>	N6
Mouse button pressed	evt.button	<string>	IE4, N6, S1, O5
Left mouse position (screen)	evt.screenX	<length>	IE4, N4, S1, O5
Top mouse position (screen)	evt.screenY	<length>	IE4, N4, S1, O5
Left mouse position (window)	evt.clientX	<length>	IE4, N6, S1, O5
Left mouse position (screen)	evt.clientY	<length>	IE4, N6, S1, O5

Reserved Words

When you are creating names for a CSS class, CSS ID, or JavaScript variable, keep in mind that the browser has dibs on certain words. I recommend not using these.

That said, it's OK to combine different words to form compound words, even if both words are on the reserved list. For example, although *new* and *label* would not make good variable names, `newLabel` would be fine.

JavaScript and Java reserved words

The following words are part of the JavaScript or Java language and should be avoided at all costs:

<i>abstract</i>	<i>false</i>	<i>private</i>
<i>boolean</i>	<i>final</i>	<i>protected</i>
<i>break</i>	<i>finally</i>	<i>public</i>
<i>byte</i>	<i>float</i>	<i>return</i>
<i>case</i>	<i>for</i>	<i>short</i>
<i>catch</i>	<i>function</i>	<i>static</i>
<i>char</i>	<i>goto</i>	<i>super</i>
<i>class</i>	<i>if</i>	<i>switch</i>
<i>comment</i>	<i>implements</i>	<i>synchronized</i>
<i>const</i>	<i>import</i>	<i>this</i>
<i>continue</i>	<i>in</i>	<i>throw</i>
<i>debugger</i>	<i>instanceOf</i>	<i>throws</i>
<i>default</i>	<i>int</i>	<i>transient</i>
<i>delete</i>	<i>interface</i>	<i>true</i>
<i>do</i>	<i>label</i>	<i>try</i>
<i>double</i>	<i>long</i>	<i>typeof</i>
<i>else</i>	<i>native</i>	<i>var</i>
<i>enum</i>	<i>new</i>	<i>void</i>
<i>export</i>	<i>null</i>	<i>while</i>
<i>extends</i>	<i>package</i>	<i>with</i>

Other words to avoid

Although not officially on the reserved list, these words are used by JavaScript and will cause problems if you use them.

Remember that Netscape is case-sensitive, so capital letters make a difference. For example, *history* is not the same as *History*.

<i>alert</i>	<i>evt</i>	<i>location</i>	<i>pageXoffset</i>	<i>status</i>
<i>Anchor</i>	<i>FileUpload</i>	<i>Location</i>	<i>pageYoffset</i>	<i>statusbar</i>
<i>Area</i>	<i>find</i>	<i>locationbar</i>	<i>parent</i>	<i>stop</i>
<i>arguments</i>	<i>focus</i>	<i>Math</i>	<i>parseFloat</i>	<i>String</i>
<i>Array</i>	<i>Form</i>	<i>menubar</i>	<i>parseInt</i>	<i>Submit</i>
<i>assign</i>	<i>Frame</i>	<i>MimeType</i>	<i>Password</i>	<i>sun</i>
<i>blur</i>	<i>frames</i>	<i>moveBy</i>	<i>personalbar</i>	<i>taint</i>
<i>Boolean</i>	<i>Function</i>	<i>moveTo</i>	<i>Plugin</i>	<i>Text</i>
<i>Button</i>	<i>getClass</i>	<i>name</i>	<i>print</i>	<i>Textarea</i>
<i>callee</i>	<i>Hidden</i>	<i>NaN</i>	<i>prompt</i>	<i>toolbar</i>
<i>caller</i>	<i>hide</i>	<i>navigate</i>	<i>prototype</i>	<i>top</i>
<i>captureEvents</i>	<i>history</i>	<i>navigator</i>	<i>Radio</i>	<i>toString</i>
<i>Checkbox</i>	<i>History</i>	<i>Navigator</i>	<i>ref</i>	<i>unescape</i>
<i>clearInterval</i>	<i>home</i>	<i>netscape</i>	<i>RegExp</i>	<i>untaint</i>
<i>clearTimeout</i>	<i>Image</i>	<i>Number</i>	<i>releaseEvents</i>	<i>unwatch</i>
<i>close</i>	<i>Infinity</i>	<i>Object</i>	<i>Reset</i>	<i>valueOf</i>
<i>closed</i>	<i>innerHeight</i>	<i>onblur</i>	<i>resizeBy</i>	<i>watch</i>
<i>confirm</i>	<i>innerWidth</i>	<i>onerror</i>	<i>resizeTo</i>	<i>window</i>
<i>constructor</i>	<i>isFinite</i>	<i>onfocus</i>	<i>routeEvent</i>	<i>Window</i>
<i>Date</i>	<i>isNaN</i>	<i>onload</i>	<i>scroll</i>	
<i>defaultStatus</i>	<i>java</i>	<i>onunload</i>	<i>scrollBars</i>	
<i>document</i>	<i>JSONArray</i>	<i>open</i>	<i>scrollBy</i>	
<i>Document</i>	<i>JavaClass</i>	<i>opener</i>	<i>scrollTo</i>	
<i>Element</i>	<i>JavaObject</i>	<i>Option</i>	<i>Select</i>	
<i>escape</i>	<i>JavaPackage</i>	<i>outerHeight</i>	<i>self</i>	
<i>eval</i>	<i>length</i>	<i>outerWidth</i>	<i>setInterval</i>	
<i>event</i>	<i>Link</i>	<i>Packages</i>	<i>setTimeout</i>	

WAI ACCESSIBILITY CHECKLIST



Accessibility is quickly becoming a vital concern for all professional Web sites. The following reprints the World Wide Web Consortium's Checklist of Checkpoints for Web Content Accessibility Guidelines (version 1.0) available at www.w3.org/TR/WAI-WEBCONTENT/full-checklist.html

Each checkpoint in the list references one of the W3C Accessibility Guidelines (version 1.0) available at www.w3.org/TR/WAI-WEBCONTENT

Priorities

Each checkpoint has a priority level assigned by the Working Group based on the checkpoint's impact on accessibility.

Priority 1

A Web content developer must satisfy this checkpoint. Otherwise, one or more groups will find it impossible to access information in the document. Satisfying this checkpoint is a basic requirement for some groups to be able to use Web documents.

Priority 2

A Web content developer should satisfy this checkpoint. Otherwise, one or more groups will find it difficult to access information in the document. Satisfying this checkpoint will remove significant barriers to accessing Web documents.

Priority 3

A Web content developer may address this checkpoint. Otherwise, one or more groups will find it somewhat difficult to access information in the document. Satisfying this checkpoint will improve access to Web documents.

Some checkpoints specify a priority level that may change under certain (indicated) conditions.

Priority 1 Checkpoints

Priority 1

In General

GUIDELINE	CHECKPOINT
1.1	Provide a text equivalent for every nontext element (e.g., via <code>alt</code> , <code>longdesc</code> , or in-element content). This includes images, graphical representations of text (including symbols), image-map regions, animations (e.g., animated GIFs), applets and programmatic objects, ASCII art, frames, scripts, images used as list bullets, spacers, graphical buttons, sounds (played with or without user interaction), stand-alone audio files, audio tracks of video, and video.
2.1	Ensure that all information conveyed with color is also available without color, for example from context or markup.
4.1	Clearly identify changes in the natural language of a document's text and any text equivalents (e.g., captions).
6.1	Organize documents so they may be read without style sheets. For example, when an HTML document is rendered without associated style sheets, it must still be possible to read the document.
6.2	Ensure that equivalents for dynamic content are updated when the dynamic content changes.
7.1	Until user agents allow users to control flickering, avoid causing the screen to flicker.
14.1	Use the clearest and simplest language appropriate for a site's content.

Priority 1

And if you use images and image maps

GUIDELINE	CHECKPOINT
1.2	Provide redundant text links for each active region of a server-side image map.
9.1	Provide client-side image maps instead of server-side image maps, except where the regions cannot be defined with an available geometric shape.

Priority 1

And if you use tables

GUIDELINE	CHECKPOINT
5.1	For data tables, identify row and column headers.
5.2	For data tables that have two or more logical levels of row or column headers, use markup to associate data cells and header cells.

Priority 1

And if you use frames

GUIDELINE	CHECKPOINT
12.1	Title each frame to facilitate frame identification and navigation.

Priority 1

And if you use applets and scripts

GUIDELINE	CHECKPOINT
6.3	Ensure that pages are usable when scripts, applets, or other programmatic objects are turned off or not supported. If this is not possible, provide equivalent information on an alternative, accessible page.

Priority 1

And if you use multimedia

GUIDELINE	CHECKPOINT
1.3	Until user agents can automatically read aloud the text equivalent of a visual track, provide an auditory description of the important information of the visual track of a multimedia presentation.
1.4	For any time-based multimedia presentation (e.g., a movie or animation), synchronize equivalent alternatives (e.g., captions or auditory descriptions of the visual track) with the presentation.

Priority 1

And if all else fails

GUIDELINE	CHECKPOINT
11.4	If, after best efforts, you cannot create an accessible page, provide a link to an alternative page that uses W3C technologies, is accessible, has equivalent information (or functionality), and is updated as often as the inaccessible (original) page.

Priority 2 Checkpoints

Priority 2

In General	
GUIDELINE	CHECKPOINT
2.2	Ensure that foreground and background color combinations provide sufficient contrast when viewed by someone having color deficits or when viewed on a black-and-white screen [Priority 2 for images, Priority 3 for text].
3.1	When an appropriate markup language exists, use markup rather than images to convey information.
3.2	Create documents that validate to published formal grammars.
3.3	Use style sheets to control layout and presentation.
3.4	Use relative rather than absolute units in markup language attribute values and style sheet property values.
3.5	Use header elements to convey document structure and use them according to specification.
3.6	Mark up lists and list items properly.
3.7	Mark up quotations. Do not use quotation markup for formatting effects such as indentation.
6.5	Ensure that dynamic content is accessible or provide an alternative presentation or page.
7.2	Until user agents allow users to control blinking, avoid causing content to blink (i.e., change presentation at a regular rate, such as turning on and off).
7.4	Until user agents provide the ability to stop the refresh, do not create periodically auto-refreshing pages.
7.5	Until user agents provide the ability to stop auto-redirect, do not use markup to redirect pages automatically. Instead, configure the server to perform redirects.
10.1	Until user agents allow users to turn off spawned windows, do not cause pop-ups or other windows to appear and do not change the current window without informing the user.
11.1	Use W3C technologies when they are available and appropriate for a task and use the latest versions when supported.
11.2	Avoid deprecated features of W3C technologies.
12.3	Divide large blocks of information into more manageable groups where natural and appropriate.
13.1	Clearly identify the target of each link.
13.2	Provide metadata to add semantic information to pages and sites.
U.3	Provide information about the general layout of a site (e.g., a site map or table of contents).
13.4	Use navigation mechanisms in a consistent manner.

Priority 2

And if you use tables	
GUIDELINE	CHECKPOINT
5.3	Do not use tables for layout unless the table makes sense when linearized. Otherwise, if the table does not make sense, provide an alternative equivalent (which may be a linearized version).
5.4	If a table is used for layout, do not use any structural markup for the purpose of visual formatting.

Priority 2

And if you use frames

GUIDELINE	CHECKPOINT
12.2	Describe the purpose of frames and how frames relate to each other if it is not obvious by frame titles alone.

Priority 2

And if you use forms

GUIDELINE	CHECKPOINT
10.2	Until user agents support explicit associations between labels and form controls, for all form controls with implicitly associated labels, ensure that the label is properly positioned.
12.4	Associate labels explicitly with their controls.

Priority 2

And if you use applets and scripts

GUIDELINE	CHECKPOINT
6.4	For scripts and applets, ensure that event handlers are input device-independent.
7.3	Until user agents allow users to freeze moving content, avoid movement in pages.
8.1	Make programmatic elements such as scripts and applets directly accessible or compatible with assistive technologies [Priority 1 if functionality is important and not presented elsewhere, otherwise Priority 2].
9.2	Ensure that any element that has its own interface can be operated in a device-independent manner.
9.3	For scripts, specify logical event handlers rather than device-dependent event handlers.

Priority 3 Checkpoints

Priority 3

In General	
GUIDELINE	CHECKPOINT
4.2	Specify the expansion of each abbreviation or acronym in a document where it first occurs.
4.3	Identify the primary natural language of a document.
9.4	Create a logical tab order through links, form controls, and objects.
9.5	Provide keyboard shortcuts to important links (including those in client-side image maps), form controls, and groups of form controls.
10.5	Until user agents (including assistive technologies) render adjacent links distinctly, include nonlink, printable characters (surrounded by spaces) between adjacent links.
11.3	Provide information so that users may receive documents according to their preferences (language, content type, etc.).
13.5	Provide navigation bars to highlight and give access to the navigation mechanism,
13.6	Group related links, identify the group (for user agents), and, until user agents do so, provide a way to bypass the group.
13.7	If search functions are provided, enable different types of searches for different skill levels and preferences.
13.8	Place distinguishing information at the beginning of headings, paragraphs, lists, etc.
13.9	Provide information about document collections (i.e., documents comprising multiple pages).
13.10	Provide a means to skip over multiline ASCII art.
14.2	Supplement text with graphic or auditory presentations where they will facilitate comprehension of the page.
14.3	Create a style of presentation that is consistent across pages.

Priority 3

And if you use images and image maps	
GUIDELINE	CHECKPOINT
1.5	Until user agents render text equivalents for client-side image map links, provide redundant text links for each active region of a client-side image map.

Priority 3

And if you use tables	
GUIDELINE	CHECKPOINT
5.5	Provide summaries for tables.
5.6	Provide abbreviations for header labels.
10.3	Until user agents (including assistive technologies) render side-by-side text correctly, provide a linear text alternative (on the current page or some other) for all tables that lay out text in parallel, word-wrapped columns.

And if you use forms

GUIDELINE

10.4

CHECKPOINT

Until user agents handle empty controls correctly, include default, place-holding characters in edit boxes and text areas.



BROWSER-SAFE FONTS

You can use the *font-family* attribute to specify the font(s) to be used with your Web page:

font-family: times, "times new roman", serif;


Of course, which fonts you can use depends directly on which fonts are available on the visitors' machines. A visitor who doesn't have the font installed won't see the design exactly as you intended it. To keep their designs as consistent as possible, many Web designers stick to using Times for the serifed font, Arial/Helvetica for the sans-serif fonts, and Courier for the monospace font. However, most machines have many dozen fonts preinstalled—the trick is knowing which fonts are likely to be installed on which computers.

The following tables present the fonts that are preinstalled on Windows and Macintosh computers as they come out of the box. In addition, you'll find the list of the Microsoft Core Web fonts, which are installed by Internet Explorer. The list also includes which styles (bold, bold italic, or italic) are available for the fonts, the generic family to which the font belongs, and an example of the font.

To use these fonts, either pick one that's available for both Mac and Windows or choose similar fonts and list both of them in the font list. Remember that multiword font names should be in quotes (for example, "Andale Mono").

If you want a PDF version of this appendix to print out for quick reference, visit the support Web site, www.webbedevirontments.com/dhtml_css_advanced.

Table D.1

Microsoft Core Web Fonts			
NAME	WEIGHTS & STYLES	GENERIC FAMILY	EXAMPLE
Adobe Minion Web		Sans-serif	ABCDEFGHIJKLMNPOQRSTUVWXYZ abcdefghijklmnopqrstuvwxy 1234567890
Andale Mono*		Monospace	ABCDEFGHIJKLMNPOQRSTUVWXYZ abcdefghijklmnopqrstuvwxy 1234567890
Arial	bold, italic, bold italic	Sans-serif	ABCDEFGHIJKLMNPOQRSTUVWXYZ abcdefghijklmnopqrstuvwxy 1234567890
Arial Black		sans-serif	ABCDEFGHIJKLMNPOQRSTUVWXYZ abcdefghijklmnopqrstuvwxy 1234567890
Comic Sans MS	bold	Cursive	<i>ABCDEFGHIJKLMNPOQRSTUVWXYZ</i> <i>abcdefghijklmnopqrstuvwxy 1234567890</i>
Courier New	bold, italic, bold italic	Monospace	ABCDEFGHIJKLMNPOQRSTUVWXYZ abcdefghijklmnopqrstuvwxy 1234567890
Georgia	bold, italic, bold italic	Serif	ABCDEFGHIJKLMNPOQRSTUVWXYZ abcdefghijklmnopqrstuvwxy 1234567890
Impact		sans-serif	ABCDEFGHIJKLMNPOQRSTUVWXYZ abcdefghijklmnopqrstuvwxy 1234567890
Times New Roman	bold, italic, bold italic	Serif	ABCDEFGHIJKLMNPOQRSTUVWXYZ abcdefghijklmnopqrstuvwxy 1234567890
Trebuchet MS*	bold, italic, bold italic	sans-serif	ABCDEFGHIJKLMNPOQRSTUVWXYZ abcdefghijklmnopqrstuvwxy 1234567890
Verdana	bold, italic, bold italic	sans-serif	ABCDEFGHIJKLMNPOQRSTUVWXYZ abcdefghijklmnopqrstuvwxy 1234567890
Webdings		Fantasy	

* Previously named *Monotype.com*

Table D.2

Mac OS			
NAME	WEIGHTS & STYLES	GENERIC FAMILY	EXAMPLE
American Typewriter*	bold	Monospace	ABCDEF GH IJKLMNOPQRSTUVWXYZ abcdef gh ijklmnopqrstuvwxyz 1234567890
Andale Mono**		Monospace	ABCDEF GH IJKLMNOPQRSTUVWXYZ abcdef gh ijklmnopqrstuvwxyz 1234567898
Apple Chancery		Cursive	ABCDEF GH IJKLMNOPQRSTUVWXYZ abcdef gh ijklmnopqrstuvwxyz 1234567890
Apple Symbols*		Fantasy	ABXΔEΦΓΗθKAMNOΠΘΡΣΤΥϚΩΨΖ αβγδεφγηθκλμνοπρστυωξψζ 1234567890
Arial	bold, italic, bold italic	Sans-serif	ABCDEF GH IJKLMNOPQRSTUVWXYZ abcdef gh ijklmnopqrstuvwxyz 1234567890
Arial Black		Sans-serif	ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz 1234567890
Arial Narrow*	bold, italic, bold italic	Sans-serif	ABCDEF GH IJKLMNOPQRSTUVWXYZ abcdef gh ijklmnopqrstuvwxyz1234567890
Arial Rounded MT Bold*		Sans-serif	ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz1234567890
Baskerville*	bold, italic, bold italic	Serif	ABCDEF GH IJKLMNOPQRSTUVWXYZ abcdef gh ijklmnopqrstuvwxyz 1234567890
Big Caslon*		Serif	ABCDEF GH IJKLMNOPQRSTUVWXYZ abcdef gh ijklmnopqrstuvwxyz.1234567890
Brush Script MT*		Cursive	<i>ABCDEFGHIJKLMNOPQRSTUVWXYZ</i> <i>abcdefghijklmnopqrstuvwxyz 1234567890</i>
Capitals**		Sans-serif	ABCDEFGHIJKLMNOPQRSTUVWXYZ ABCDEFGHIJKLMNOPQRSTUVWXYZ 1234567890
Charcoal**		Sans-serif	ABCDEF GH IJKLMNOPQRSTUVWXYZ abcdef gh ijklmnopqrstuvwxyz 1234567890
Chicago**		Sans-serif	ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz 1234567898
Cochin*	bold, italic, bold italic	Serif	ABCDEF GH IJKLMNOPQRSTUVWXYZ abcdef gh ijklmnopqrstuvwxyz 1234567890
Comic Sans MS	bold	Cursive	ABCDEF GH IJKLMNOPQRSTUVWXYZ abcdef gh ijklmnopqrstuvwxyz 1234567890
Copperplate*	bold	Sans-serif	ABCDEFGHIJKLMNOPQRSTUVWXYZ ABCDEFGHIJKLMNOPQRSTUVWXYZ 1234567890
Courier**	bold, oblique, bold oblique	Monospace	ABCDEF GH IJKLMNOPQRSTUVWXYZ abcdef gh ijklmnopqrstuvwxyz 1234567890
Courier New	bold, italic, bold italic	Monospace	ABCDEF GH IJKLMNOPQRSTUVWXYZ abcdef gh ijklmnopqrstuvwxyz 1234567890
Didot*	bold, italic	Sans-serif	ABCDEF GH IJKLMNOPQRSTUVWXYZ abcdef gh ijklmnopqrstuvwxyz 1234567890
Futura*	Sans-serif	Serif	ABCDEF GH IJKLMNOPQRSTUVWXYZ abcdef gh ijklmnopqrstuvwxyz 1234567890
Gadget**		Sans-serif	ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz 1234567890
Geneva		Sans-serif	ABCDEF GH IJKLMNOPQRSTUVWXYZ abcdef gh ijklmnopqrstuvwxyz 1234567890
Georgia	bold, italic, bold italic	Serif	ABCDEF GH IJKLMNOPQRSTUVWXYZ abcdef gh ijklmnopqrstuvwxyz1234567890
Gill Sans*	bold, italic, bold italic	Sans-serif	ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz 1234567890

(table continues on next page)

Table D.2 continued

Mac OS			
NAME	WEIGHTS & STYLES	GENERIC FAMILY	EXAMPLE
Helvetica	bold, oblique, bold oblique	Sans-serif	ABCDEFGHIJKLMN OPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz1234567890
Helvetica Neue*	bold, italic, bold italic	Sans-serif	ABCDEFGHIJKLMN OPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz1234567890
Herculanum*		Cursive	<i>ABCDEFGHIJKLMN</i> <i>OPQRSTUVWXYZ</i> ABCDEFGHIJKLMNOPQRSTUVWXYZ 1234567890
Hoefler Text	bold, italic, bold italic	Serif	ABCDEFGHIJKLMN OPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz 1234567890
Impact		Serif	ABCDEFGHIJKLMN OPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz1234561890
Lucida Grande*	bold	Sans-serif	ABCDEFGHIJKLMN OPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz 1234567890
Marker Felt*		Fantasy	ABCDEFGHIJKLMN OPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz 1234567890
Monaco		Monospace	ABCDEFGHIJKLMN OPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz 1234567890
New York**		Serif	ABCDEFGHIJKLMN OPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz 1234567890
Optima*	bold, italic, bold italic	Sans-serif	ABCDEFGHIJKLMN OPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz1234567890
Palatino**	bold, italic, bold italic	Serif	ABCDEFGHIJKLMN OPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz1234567890
Papyrus*		Cursive	<i>ABCDEFGHIJKLMN</i> <i>OPQRSTUVWXYZ</i> abcdefghijklmnopqrstuvwxyz 1234567890
Sand**		Fantasy	ABCDEFGHIJKLMN OPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz 1234567890
Skia		Sans-serif	ABCDEFGHIJKLMN OPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz 1234567890
Symbol		Fantasy	ABCDEFGHIJKLMN OPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz 1234567890
Techno*		Fantasy	ABCDEFGHIJKLMN OPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz 1234567890
Textile**		Cursive	<i>ABCDEFGHIJKLMN</i> <i>OPQRSTUVWXYZ</i> abcdefghijklmnopqrstuvwxyz 1234567890
Times	bold, italic, bold italic	Serif	ABCDEFGHIJKLMN OPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz 1234567890
Times New Roman	bold, italic, bold italic	Serif	ABCDEFGHIJKLMN OPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz 1234567890
Trebuchet MS	bold, italic, bold italic	Sans-serif	ABCDEFGHIJKLMN OPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz1234567890
Verdana*	bold, italic, bold italic	Sans-serif	ABCDEFGHIJKLMN OPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz 1234567890
VT100*		Monospace	ABCDEFGHIJKLMN OPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz 1234567890
Webdings**		Fantasy	
Zapf Dingbats*		Fantasy	
Zapfino*		Cursive	<i>ABCDEFGHIJKLMN</i> <i>OPQRSTUVWXYZ</i> abcdefghijklmnopqrstuvwxyz 1234567890

*as of OS X; **Only installed in OS X if Classic is installed

*As of Mac OS 8.5

Blank

INDEX

A

- absolute font size, 37
- absolute length values, xii
- accessibility
 - Checklist of Checkpoints for Web Content, 397
 - dynamic site checklist, 8–9
- accessing arrays, 58–60
- action delays, 91–93
- activities, dynamic site criteria, 4
- adaptability, dynamic site criteria, 4
- `addIframeElement()` function, 116–117
- `addImageElement()` function, 112–113
- `addTextElement()` function, 114–115
- adjacent selectors, 50
- administrator pages, databases, 366–370
- Adobe GoLive, 25
- Adobe ImageReady
 - columns with graphic backgrounds, 137
 - creating GIF animation, 322
 - framed drop shadows, 158–161
 - PNG graphics, 328–329
- Adobe Photoshop
 - columns with graphic backgrounds, 137–140
 - framed drop shadows, 158–161
- after pseudo element, 51
- airbrush-stroke graphics, 176–178
- ALTER TABLE command, SQL (Structured Query Language), 349
- ambient sounds, 326–327
- Amnesty International Web site, 10
- animated scroll controls, 243–248
- `appendChild()` function, 113, 115
- `appendChild()` method, 111
- arrays
 - accessing, 58–60
 - making changes to, 61–63
 - setting up, 56–57
 - sorting, 64–68
- attributes
 - autostart, 326
 - background, 41
 - background-color, link menus, 189
 - border, 41
 - font, 41
 - hide, 326
 - list-style, 41
 - loop, 326
 - margin, 41
 - padding, 41
 - pluginpage, 326
 - sounds, 326
 - src, 326
 - text-shadow, link menus, 189
- audience, defining site, 18
- author Web site, xiv
- auto-focusing form fields, 282–285
- autostart attribute, 326

B

- background attribute, 41
- background-color attribute, link menus, 189
- background-image property, differentiating link icons, 178–181
- backgrounds
 - balanced columns, 137–140
 - GIF animation, 324
- balanced columns, 133–137
- before pseudo element, 51
- `bgLinkActive` variable, 194
- `bgLinkColor` variable, 194
- `bgLinkHover` variable, 194
- `blindBlogData()` function, 345–348
- blogs, 341
 - adding comments, 356–359
 - creating entry listing page, 353–355
 - secure entry input form, 371–374
- `blurField()` function, 276–285
- `blurOn()` function, 334
- blurring elements, Internet Explorer for Windows, 333–334
- bookmarks, HTML page in framesets
 - automatically, 162–165
- border attribute, 41
- border-bottom attribute, 41
- border-bottom property, customizing link styles, 174–175
- border-left attribute, 41
- border-right attribute, 41
 - link menus, 189
- border-top attribute, 41
- borders
 - balanced columns, 133–137
 - box model hacks, 46–47
 - curving, 150–153
- box model hacks, browser inconsistencies, 46–47
- `browserButton` class, 224–227

- browsers
 - common fonts, 34, 405–409
- CSS (cascading style sheets)
 - availability issues, 28
 - shorthand recognition, 42
- customized control buttons, 224–227
- inconsistencies
 - box model hacks, 46–47
 - setting page origin, 45–46
- `<link>` tag set up, 220–221
- `<meta>` tag set up, 222
- properties, 392
- `buildDiv()` function, 353–355
- button class, 185
- `buttonCapLeft` class, 185
- buttons
 - customized browser control, 224–227
 - HTML text graphics, 182–185
- `buttonText` class, 185

C

- `calControl` class, 249–254
- `calDateToday` class, 249–254
- `calDay` class, 249–254
- `calEmpty` class, 249–254
- calendar date picker, 249–254
- `callDate` class, 249–254
- cascading style sheets. See CSS
- centering layouts, 145–146
- `changeArray()` function, 61
- `changeFontSize()` function, 233–238
- `changeMonth()` function, 249–254
- `changeSearch()` function, 294–297
- `changeStyle()` function, 167–168
- check input, 271
- `checkField()` function, 306–308
- Checklist of Checkpoints for Web Content Accessibility Guidelines, 397–404
- `checkLogin()` function, 298–300
- child selectors, 48–49

- class selectors, 32
 - redefining Web page tags, 378
- `clearClipMenu()` function, 203–211
- `clearClipMenuRead()` function, 203–211
- `clearInterval()` method, 93
- `clearTimeout()` method, 91
- `clipLink()` function, 203–211
- `clipLinkStyle` class, 203–211
- clipping menus, 203–211
- `clockObject` variable, 126
- clocks, adding to page, 125–127
- `clockStyle` class, 127
- codes
 - creating, 25
 - viewing hidden external content, 104–108
- collapsible frames, 259–267
- colors
 - CSS (cascading style sheets) shorthand, 42
 - links
 - customizing underline, 174–175
 - menus, 189
 - values, xii
- columns
 - balanced, 133–137
 - contextual layouts, 141–144
 - creating simple, 130–132
 - graphic backgrounds, 137–140
- commands, SQL (Structured Query Language), 349
- comments input form, 360–365
- content
 - adding clock to page, 125–127
 - border curving, 150–153
 - collecting, 17–18
 - columns
 - balanced, 133–137
 - creating simple, 130–132
 - graphic backgrounds, 137–140
 - contextual layouts, 141–144
 - CSS (cascading style sheets), 43–44
 - curved text wrapping, 147–149
 - drop shadows
 - adding to frames, 158–161
 - creating, 154–157
 - external
 - iframes, 100–101
 - JavaScript, 104–105
 - PHP, 109–110
 - server-side includes, 102–103
 - viewing hidden code, 104–108
 - highlighting table rows, 169–172
 - HTML page in framesets, 162–165
 - inserting new elements, 111–113
 - iframe element, 116–117
 - new tags with text, 114–115
 - multiple pages in single HTML page, 122–124
 - preloading images, 98–99
 - preparation, 25
 - random, 120–121
 - removing element, 118–119
 - replacement methods, 111
 - swapping style sheets, 166–168
 - type differentiation, 36
 - Web layouts, 5
- `contentSRC` variable, 259–267
- contextual forms
 - creating, 290–293
 - data creation, 294–297
- contextual layouts, 141–144
- controls, 10–12, 223
 - animating scroll controls, 243–248
 - calendar date picker, 249–254
 - collapsible frames, 259–267
 - customized browser buttons, 224–227
 - elements, 384–385
 - fonts, 382
 - size, 233–238
 - forms
 - disabling, 301–304
 - graphic, 305–308
 - interface, 387

controls (continued)

lists, 387

QuickTime video, 255–258

scrollable areas, 239–241

sortable tables, 228–232

table, 387

text, 383

cookies

data storage, 84–92

sessions, 369

corporation design standards, 33

countThis() function, 125–127

CREATE DATABASE command, SQL (Structured Query Language), 349

CREATE TABLE command, SQL (Structured Query Language), 349

createCalendar() function, 249–254

createElement() function, 113

createEntry() function, 345–348

cross-browsers, CSS (cascading style sheets)
availability, 28–29

CSS (cascading style sheets), ix

adding styles, 376–377

browser inconsistencies

box model hacks, 46–47

setting page origin, 45–46

building

adding notes, 32

master style sheets, 39–40

style order, 30–32

context, 43–44

grouping, 43–44, 379

Internet Explorer 6

adjacent selectors, 50

child selectors, 48–49

dynamic pseudo classes, 52–53

pseudo elements, 51

myths, 28–29

printing improvement, 37–38

properties and values, 380–387

shorthand properties, 41–42

typography

common safe fonts, 34

content differentiating, 36

readability, 35

text in graphics, 33

typeface selection, 34

visual message, 36

CSS Zen Garden Web site, fixed height Web
layouts, 6

curveText class, 147–149

curving

borders, 150–153

text wrapping, 147–149

D

data

arrays

accessing, 58–60

making changes to, 61–63

setting up, 56–57

sorting, 64–68

contextual forms, 294–297

cookies, 84–92

creating objects, 69–70

frames, 71–77

sortable tables, 228–232

URL storage, 78–83

data islands, Internet Explorer for Windows,
342–344

databases, 339

administrator page, 366–370

comments input form, 360–365

MySQL, 349

creating objects, 350–351

using with PHP, 353–359

PHP, 352

adding comments to blog, 356–359

creating blog entry listing page, 353–355

secure blog entry input form, 371–374

- XML (extensible markup language)
 - accessing with JavaScript, 345–348
 - creating document, 340–341
 - data islands, 342–344
- `defineObject()` function, creating data objects, 69–70
- defining sites
 - collecting content, 17–18
 - establish goals, 18–19
 - feature planning, 19
 - graphic design style, 19
 - know audience, 18
- delay functions, 91–93
- delay* variable, 317–319
- DELETE** command, SQL (Structured Query Language), 349
- deploying dynamic sites, 26
- designs, dynamic sites, 15–16
 - defining, 17–20
 - layout grid, 22
 - look and feel, 24
 - outline, 20
 - site map, 21
 - storyboard, 23
- development, dynamic sites, 24–26
- DHTML and CSS for the World Wide Web*, xi
- DHTML (dynamic HTML), ix
 - DOM (Document Object Model), 390
 - event handlers, 391
 - JavaScript, finding properties and values, 392–394
- digitalClock* variable, 125
- direction* variable, 249–254
- `displayArray()` function
 - accessing arrays, 59
 - making changes to array, 61–63
 - sorting arrays, 64–68
- `displayImage()` function, 69
- `displayTable()` function, 228–232
- Document Object Model (DOM), 3, 390
- DOM (Document Object Model), 3, 390

- downloads
 - graphics, 33
 - preloading images, 98–99
- drop shadows
 - adding to frames, 158–161
 - creating, 154–157
 - text, 314–316
- DROP TABLE** command, SQL (Structured Query Language), 349
- dynamic HTML. *See* DHTML
- dynamic pseudo classes, 52–53
- dynamic sites, 1–2
 - accessibility checklist, 8–9
 - controls, 10–12
 - design, 15–16
 - defining, 17–20
 - deployment, 26
 - development, 24–26
 - layout grid, 22
 - look and feel, 24
 - outline, 20
 - site map, 21
 - storyboard, 23
 - hypertext, 10–12
 - layout, 5–7
 - navigation, 10–11
 - dos and don'ts, 12–14

E

- Edwards, Dean, IE7, 49
- effects, 309
 - ambient sounds, 326–327
 - creating GIF animation
 - adding to Web page, 323
 - benefits, 324–325
 - Fireworks, 320–321
 - ImageReady, 322
 - CSS (cascading style sheets), 33
 - floating objects, 317–319

effects (*continued*)

Internet Explorer for Windows

blurring element, 333–334

fading, 335–336

page transitions, 337–338

PNG graphics

Fireworks, 330–331

ImageReady, 328–329

use on Web page, 332

text drop shadows, 314–316

transparent layers, 310–313

elements

controls, 384–385

creating data, 69–70

curved borders, 150–153

finding, 392

floating, 317–319

images

curved text wrapping, 147–149

preloading, 98–99

positioning controls, 386

properties, 393

removing element, 118–119

visibility controls, 386

embedded content, iframes, 100–101

`enableFields()` function, 302–304

error handling, 94–95

`escape()` method, 82

events

handlers, 391

properties, 394

external classes, differentiating link icons,
178–181

external content

adding with iframes, 100–101

JavaScript, 104–105

PHP, 109–110

server-side includes, 102–103

viewing hidden code, 104–108

external JavaScript *versus* internal, 189

external styles, 377

F

`fadeOutElement()` function, 336

fading objects, Internet Explorer for Windows,
335–336

features, defining sites, 19

`fieldLabel` class, 270–275

fields, forms

auto-focusing, 282–285

content restriction, 298–300

highlighting, 276–281

fieldsets, form styles, 270

filled highlight state, forms, 276–281

filler, Web layouts, 5

filters, Internet Explorer for Windows

blurring element, 333–334

fading, 335–336

page transitions, 337–338

finding objects, 392

`findScrollTop()` function, 200–202

Fireworks

columns with graphic backgrounds, 137

creating GIF animation, 320–321

framed drop shadows, 158–161

PNG graphics, 330–331

`firstDate` variable, 249–254

`firstDay` variable, 249–254

`firstField()` function, 282–285

firstletter pseudo element, 51

firstline pseudo element, 51

fixed drop-down menus, 192–199

fixed height Web layouts, 6

fixed size Web layouts, 7

fixed width Web layouts, 6

flexibility, dynamic site criteria, 4

floating layers

columns

balanced, 133–137

creating simple, 130–132

graphic backgrounds, 137–140

contextual layouts, 141–144

floating menus, 200–202

- floating objects, 317–319
- `focusField()` function, 276–281, 282–285
- font attribute, 41
- fonts
 - browser-safe, 34, 405–409
 - controls, 382
 - size control, 233–238
 - sizes, 37
 - versus* typeface, 34
- `fontSizeControl` class, 233–238
- foreign keys, relational databases, 349
- `formInputButton` class, 270–275
- `formInputField` class, 270–275
- forms, 269
 - contextual
 - creating, 290–293
 - data creation, 294–297
 - controls
 - disabling, 301–304
 - graphic, 305–308
 - fields
 - auto-focusing, 282–285
 - content restriction, 298–300
 - highlighting, 276–281
 - styling, 270–275
 - validation, 286–289
- `formSelect` class, 270–275
- `frameName` variable, 194
- `frameRate` variable, 243–248
- `frameRateMax` variable, 243–248
- frames
 - adding drop shadows, 158–161
 - collapsable, 259–267
 - placing HTML page automatically, 162–165
 - storing data, 71–77
- framesets
 - data storing, 71–77
 - placing HTML page automatically, 162–165
- functions, delay, 91–93

G

- `getClipMenu()` function, 203–211
- `getDataCookie()` function, 87, 167
- `getDataURL()` function, 83
- `getElementById()` function, 118
- `getElementById()` function, 115
- GIF animation
 - benefits, 324–325
 - creating
 - adding to Web page, 323
 - Fireworks, 320–321
 - ImageReady, 322
 - goals, defining site, 18–19
- `goBack()` function, 224–227
- `goForward()` function, 224–227
- `gotQT` variable, 255–258
- graphics
 - column backgrounds, 137–140
 - defining site, 19
 - forms controls, 305–308
 - hypertext links, 176–178
 - text, 33
 - HTML buttons, 182–185
 - transparent, PNG (portable network graphics), 328–332
- Greek pattern, 176–178
- grid layouts, 22, 24
- grouping, CSS (cascading style sheets), 43–44, 379
- gutters, 130

H

- `handleError()` function, 94–95
- hexadecimal notation, 42
- hidden code, viewing external content, 104–108
- hidden frames, 71–77
- hide attribute, 326
- `hideDropMenu()` function, 197
- `hideImage` class, 98–99
- highlighting fields, forms, 276–281
- `highlightOffEven` class, 169

- `highlightOffOdd` class, 169
- `highlightOn` class, 169
- `highlightRow()` function, 169–172
- horizontal centering, layouts, 145–146
- hover airbrush-stroke graphics, 176
- hover filled highlight state, forms, 276–281
- hover unfilled highlight state, forms, 276–281
- `hoverField()` function, 276–281
- `hoverOffField()` function, 276–281
- HTML, ix
 - embedded content, iframes, 100–101
 - multiple pages in single page, 122–124
 - placing in framesets automatically, 162–165
 - selectors, 31
 - redefining Web page styles, 378
 - text graphic button, 182–185
- hypertext, 3, 10–12, 16
 - links
 - customizing underline colors, 174–175
 - differentiating type icons, 178–181
 - graphics, 176–178

I

- iCab Web site, 220
- icons, differentiating link types, 178–181
- ID selectors, 31
 - redefining Web page tags, 378
- IE7, 49
- `<iframe>` tag, 100
- `iframeResize()` function, 100
- iframes
 - external content import, 100–101
 - inserting new element, 116–117
- illustrations, GIF animation, 325
- ImageReady
 - columns with graphic backgrounds, 137
 - creating GIF animation, 322
 - framed drop shadows, 158–161
 - PNG graphics, 328–329

- images
 - curved text wrapping, 147–149
 - preloading, 98–99
- `@import` tag, 39
- `<include>` tag, 39
- `initDropDownMenu()` function, 195, 197
- inline styles, 376
- input, form styles, 270–271
- `INSERT` command, SQL (Structured Query Language), 349
- `insertBefore()` method, 111, 115
- interactivity
 - controls, 223
 - animating scroll controls, 243–248
 - calendar date picker, 249–254
 - collapsible frames, 259–267
 - customized browser buttons, 224–227
 - font size, 233–238
 - QuickTime video, 255–258
 - scrollable areas, 239–241
 - sortable tables, 228–232
 - defining sites, 19
 - dynamic site criteria, 3
- interface controls, 387
- internal classes, differentiating link icons, 178–181
- internal JavaScript *versus* external, 189
- internal styles, 376
- Internet Explorer, accessing XML (extensible markup language), 345–348
- Internet Explorer 6, CSS (cascading style sheets)
 - adjacent selectors, 50
 - child selectors, 48–49
 - dynamic pseudo classes, 52–53
 - pseudo elements, 51
- Internet Explorer for Windows
 - special effects
 - blurring element, 333–334
 - fading, 335–336
 - page transitions, 337–338
 - XML data islands, 342–344
- ISO-8859-1 encoding, XML (extensible markup language), 340

J

Java, reserved words, 395–396

JavaScript, ix

- accessing XML (extensible markup language), 345–348

- character interpretations, 104

- external content, 104–105

- finding properties and values, 392–394

- internal *versus* external, 189

- reserved words, 395–396

- rollovers, 324

jump menus, 212–214

jumpTo() function, 212–214

K-L

labels, form styles, 270

Laporte Report Web site, 341

lastDate variable, 249–254

layers

- columns

 - balanced, 133–137

 - creating simple, 130–132

 - graphic backgrounds, 137–140

- contextual layouts, 141–144

- transparent, 310–313

layouts, 129

- border curving, 150–153

- centering, 145–146

- columns

 - balanced, 133–137

 - creating simple, 130–132

 - graphic backgrounds, 137–140

- contextual, 141–144

- CSS (cascading style sheets) handling, 29

- curved text wrapping, 147–149

- drop shadows

 - adding to frames, 158–161

 - creating, 154–157

- dynamic sites, 5–7

- grids, 22, 24

 - highlighting table rows, 169–172

 - HTML page in framesets, 162–165

 - swapping style sheets, 166–168

- legacy browsers, CSS (cascading style sheets)

 - availability issues, 28

 - shorthand recognition, 42

- legends, form styles, 270

- <link> tag, 39, 220–221

- linkActive variable, 194

- linkColor variable, 194

- linkHover variable, 194

- links, styles

 - customizing underline colors, 174–175

 - differentiating type icons, 178–181

 - graphics, 176–178

- list controls, 387

- list- style attribute, 41

- logos, 33

- loop attribute, 326

M

Mac OS, core Web fonts, 407–408

Macromedia Dreamweaver, 25

Macromedia Fireworks

- columns with graphic backgrounds, 137

- creating GIF animation, 320–321

- framed drop shadows, 158–161

- PNG graphics, 330–331

- mainStoryCopy class, 310–313

- mainStorySlug class, 310–313

- margin attribute, 41

- master style sheets, building, 39–40

- menuDrop class, 192–199

- menuHeader class, 197

- menuHover() function, 195

- menuInside class, 200–202

- menuLink class, 192–199

- menuOut() function, 195

- menus
 - adding, 189–191
 - clipping, 203–211
 - fixed drop-down, 192–199
 - floating, 200–202
 - jump, 212–214
 - pop-up, 215–219
- menuToggle() function, 259–267
- <meta> tag, 220,222

- metadata, 220

- Microsoft, core Web fonts, 406

- monthNum variable, 249–254

- monthShow class, 249–254

- moveMenuBar() function, 200–202

- Mozilla, accessing XML (extensible markup language),345–348

- MySQL, 349

- creating objects, 350–351

- PHP, 352

- adding comments to blog, 356–359

- creating blog entry listing page, 353–355

- mysql_connect() function, 353–355

- myths, CSS (cascading style sheets), 28–29

N

- navigation, 10–11

- creating tabs, 186–188

- dos and don'ts, 12–14

- HTML text graphic button, 182–185

- link styles

- customizing underline colors, 174–175

- differentiating type icons, 178–181

- graphics, 176–178

- <link> tag set up, 220–221

- menus

- adding, 189–191

- clipping, 203–211

- fixed drop-down, 192–199

- floating, 200–202

- jump, 212–214

- pop-up, 215–219

- <meta> tag set up, 222

- recommendations, 173

- Netscape 4, CSS (cascading style sheets)
 - recognition, 42

- nextField() function, 282–285

- nextStep variable, 317–319

- nodes, Document Object Model (DOM),390

- notes, style sheets, 32

- numbers, converting from strings, 90

- numDropMenu variable, 194

- numObjects variable, 317–319

O

- objectBeGone() function, 118–119

- objects

- creating data, 69–70

- finding, 392

- floating, 317–319

- images

- curved text wrapping, 147–149

- preloading, 98–99

- properties, 393

- removing element, 118–119

- objectsFloat() function, 317–319

- objNavMenu variable, 194

- objPopUp variable, 215–219

- oldObject variable, 290–293

- outlines, site design, 20

P

- padding

- attribute, 41

- box model hacks, 46–47

- padding-bottom property, 175

- pages

- breaks, printing Web pages, 37

- properties, 393

- transitions, Internet Explorer for Windows, 337–338

- percentages, xii
- Photoshop
 - columns with graphic backgrounds, 137–140
 - framed drop shadows, 158–161
- PHP
 - databases, 352
 - adding comments to blog, 356–359
 - creating blog entry listing page, 353–355
 - external content, 109–110
- `placeDataURL(url)` function, 78–83
- planning, dynamic site design, 16
- pluginpage** attribute, 326
- PNG (portable network graphics)
 - Fireworks, 330–331
 - `ImageReady`, 328–329
 - use on Web page, 332
- pop-up menus, 215–219
- `popHide()` function, 215–219
- `popup()` function, 215–219
- `popUpLink` class, 215–219
- portable network graphics (PNG)
 - Fireworks, 330–331
 - `ImageReady`, 328–329
 - use on Web page, 332
- positioning controls, elements, 386
- Powell's Books Web site, unrestrained Web layouts, 5
- preloading images, 98–99
- `prevObjDropDownMenu` variable, 194
- `prevObjNavMenu` variable, 194
- primary keys, relational databases, 349
- printing, improving appearance, 37–38
- `printPage()` function, 224–227
- properties
 - browser, 392
 - CSS (cascading style sheets), 380–387
 - CSS (cascading style sheets) shorthand, 41–42
 - events, 394
 - finding with JavaScript, 392–394
 - objects, 393
 - system, 392

- pseudo-class selectors, 32, 381
- pseudo elements, 51, 381

Q-R

- QuickTime video controls, 255–258
- radio input, 271
- random content, 120–121
- `randomRange()` function, 120
- readability, typeface, 35
- relational databases, 349
- relative font size, 37
- relative length values, xii
- `reloadPage()` function, 224–227
- `removeChild()` function, 118
- `removeChild()` method, 111
- `replaceChild()` function, 117
- `replaceChild()` method, 111
- reset button input, 271
- Rose, Kevin Web site, 341
- rows, highlighting in table, 169–172
- Rymer, Daniel Web site, 341

S

- Salon Web site, fixed width Web layouts, 6
- `saveClipMenu()` function, 203–211
- `scrollBox` class, 239–241
- `scrollBoxContent` class, 239–241
- `scrollPage()` function, 243–248
- `scrollPages()` function, 243–248
- `scrollPageTo()` function, 243–248
- scrolls
 - animated controls, 243–248
 - area controls, 239–241
- SELECT** command, SQL (Structured Query Language), 349
- select inputs, form styles, 271
- selected highlight state, forms, 276–281

- selectors, 31
 - adjacent, 50
 - child, 48–49
 - class, 32
 - redefining Web page tags, 378
- HTML, 31
 - redefining Web page styles, 378
- ID, 31
 - redefining Web page tags, 378
- pseudo-class, 32
- server-side includes, external content, 102–103
- sessions, 369
- `setAttribute()` function, 113
- `setDate()` function, 249–254
- `setFontSize()` function, 233–238
- `setInterval()` method, 93
- `setTimeout()` method, 91, 93
- shorthand properties, CSS (cascading style sheets), 41–42
- `showDropMenu()` function, 196
- `showPage()` function, 122
- Site Being Updated page, 26
- site maps, design, 21
- slide shows, GIF animation, 325
- sortable tables, 228–232
- `sortByArray()` function, 64–68, 228–232
- `sortBy` variable, 228–232
- `sortByGuestName()` function, 228–232
- `sortByOrderNum()` function, 228–232
- `sortByPolitics()` function, 228–232
- sorting arrays, 64–68
- sounds, ambient, 326–327
- special effects, 309
 - ambient sounds, 326–327
 - creating GIF animation
 - adding to Web page, 323
 - benefits, 324–325
 - Fireworks, 320–321
 - ImageReady, 322
- CSS (cascading style sheets), 33
 - floating objects, 317–319

- Internet Explorer for Windows
 - blurring element, 333–334
 - fading, 335–336
 - page transitions, 337–338
- PNG graphics
 - Fireworks, 330–331
 - ImageReady, 328–329
 - use on Web page, 332
- text drop shadows, 314–316
- transparent layers, 310–313
- SQL (Structured Query Language), 349
- `src` attribute, 326
- `startAlert()` function, 93
- step variable, 317–319
- `stopAlert()` function, 93
- storyboards, site design, 23
- strings, converting to numbers, 90
- structures, separating from style, 32
- styles
 - adding, 376–377
 - forms, 270–275
 - grouping, 379
 - order, 30–32
 - separating from structure, 32
 - swapping style sheets, 166–168
 - text in graphics, 33
- submit button input, 271
- `swapForm()` function, 290–293
- synchronicity, dynamic site criteria, 3
- system properties, 392

T

- tables
 - controls, 387
 - highlighting rows, 169–172
 - relational databases, 349
 - sortable, 228–232
- tabs, creating, 186–188
- tags (HTML), redefining Web page styles, 378
- `ta11` variable, 317–319
- Tantek Celik, box model hack, 47

templates, creating, 25

testing, 26

text

controls, 383

curved wrapping, 147–149

drop shadows, 314–316

form input, 270

graphics, 33

inserting new tag with, 114–115

text-shadow attribute, link menus, 189

text-shadow property, 314–316

theDelay variable, 243–248

this keyword, 70

thisDate variable, 249–254

today variable, 249–254

todayDay variable, 249–254

todayDate variable, 249–254

todayMonth variable, 249–254

todayYear variable, 249–254

transition filters, 337–338

transparent colors, printing Web pages, 37

transparent graphics, PNG (portable network graphics)

Fireworks, 330–331

ImageReady, 328–329

use on Web page, 332

transparent layers, 310–313

triangle graphic, 176–178

typeface

versus fonts, 34

readability, 35

selection, 34

visual message, 36

typography, CSS (cascading style sheets)

common safe fonts, 34

content differentiating, 36

readability, 35

text in graphics, 33

typeface selection, 34

visual message, 36

U

underlines, customizing link styles, 174–175

unescape() method, 82

unfilled highlight state, forms, 276–281

uniform resource locators (URLs), xiii

units, xii

unrestrained Web layouts, 5

UPDATE command, SQL (Structured Query Language), 349

URLs (uniform resource locator), xiii

data storage, 78–83

USE command, SQL (Structured Query Language), 349

users, 20

UTF-8 encoding, XML (extensible markup language), 340

UTF-16 encoding, XML (extensible markup language), 340

V

validateForm() function, 286–289

validating forms, 286–289

values, xii

CSS (cascading style sheets), 380–387

finding with JavaScript, 392–394

variables

arrays

accessing, 58–60

making changes to, 61–63

setting up, 56–57

sorting, 64–68

creating data objects, 69–70

vertical centering, layouts, 145–146

visibility

element controls, 386

property, 196

visible property, 196

visitors, 20

visual messages, typeface, 36

W

WAI (Web Accessibility Initiative), 8

Checklist of Checkpoints for Web Content,
397–404

W3C

Checklist of Checkpoints for Web Content
Accessibility Guidelines, 397–404

recommendations, 7

Web accessibility initiative, 8

Web Accessibility Initiative (WAI), 8

Checklist of Checkpoints for Web Content,
397–404

Web sites

Amnesty International, 10

author, xiv

CSS Zen Garden, fixed height Web layouts, 6

dynamic, 1–2

accessibility checklist, 8–9

controls, 10–12

criteria, 3–4

deployment, 26

design, 15, 20–24

development, 24–26

hypertext, 10–12

layout, 5–7

navigation, 10–14

planning, 16–26

Edwards, Dean, 49

iCab, 220

Laporte Report, 341

Powell's Books, unrestrained Web layouts, 5

Rose, Kevin, 341

Rymer, Daniel, 341

Salon, fixed width Web layouts, 6

webbedENVIRONMENTS, 341, xiv

WonderWorld, fixed size Web layouts, 7

webbedENVIRONMENTS Web site, 341, xiv

white-space property, 215–219

wide variable, 317–319

Windows OS, core Web fonts, 409

WonderWorld Web site, fixed size Web layouts, 7

wordMonth variable, 249–254

wrapping text, 147–149

wri teMenu variable, 192–199, 212–214

WYSIWYG applications, 25

X

xC variable, 243–248

XML (extensible markup language)

databases

accessing with JavaScript, 345–348

creating document, 340–341

data islands, 342–344

encoding types, 340

xN variable, 243–248

Xpos variable, 317–319

Y-Z

yearNum variable, 249–254

Ypos variable, 317–319

Master Cascading Style Sheets

Learn advanced DHTML techniques

Create interactive Web interfaces, dynamic content, and layout without tables

Work with XML, PHP, and MySQL to add database information to your Web pages

BOOK LEVEL



beginning

✓ intermediate

✓ advanced

DHTML AND CSS ADVANCED

Need to take your DHTML and CSS skills to the next level fast? Try a Visual QuickPro!



Takes a visual, task-based approach to guiding you through advanced topics and applying what you learn.



Works like a reference book—you look up what you need and then get straight to work.



Concise, straightforward steps and explanations offer the fastest way to learn tasks and concepts.



Companion Web site at www.webbedenvironments.com/dhtml_css_advanced includes working sample files, online resources, updates, and more.



Jason Cranford Teague wrote three best-selling editions of *DHTML and CSS for the World Wide Web: Visual QuickStart Guide* as well as *Final Cut Pro 3* and *the Art of Filmmaking*, and contributed to *Dreamweaver MX Magic*. He has also written for *Macworld* and *Computer Arts* magazine, and has appeared on TechTV's "The Screen Savers." He has been working as a Web designer for more than 10 years, creating real-world solutions for clients including Coca-Cola, Virgin, CNN, Kodak, and WebMD. Jason writes about technology and culture issues in his weblog, [webbedENVIRONMENTS \(www.webbedenvironments.com\)](http://webbedENVIRONMENTS.com).



Peachpit Press

1249 Eighth Street Berkeley, CA 94710
800 283-9444 • 510 524-2178 • fax 510 524-2221
www.peachpit.com

COVERS: DHTML and CSS for Microsoft Internet Explorer 5 or higher;
Netscape 6 or higher (including Mozilla, Firefox, and Camino);
Safari; Opera 6 or higher

COMPUTER BOOK SHELF CATEGORY: Web Design / World Wide Web



USA \$29.99 Canada \$42.99 UK £21.99

ISBN 0-321-26691-1



7 85342 26691 7

9 780321 266910

5 2999